Dynamic Degree Constrained Network Design: A Genetic Algorithm Approach

Chao-Hsien Chu
300 Carver Hall
Iowa State Univ.
Ames, Iowa 50011

G. Premkumar
300 Carver Hall
Iowa State Univ.
Ames, Iowa 50011

Carey Chou
Sprint Corp.
7628 W. 95th St., #8
Overland Park, KS 66212

Jianzhong Sun
Division of Mining
Handan, Hebei 056031
P. R. of China

Abstract

The design and development of network infrastructure to support mission-critical operations has become a critical and complicated issue. In this study we explore the use of genetic algorithms (GA) for the design of a degree constrained minimal spanning tree (DCMST) problem with varied degrees on each node. The performance of GA was compared with two popular heuristics. The results indicate that GA provide better solution quality compared to heuristics, but is worse than heuristics in terms of computation time.

1 INTRODUCTION

Over the last decade the design and development of network infrastructure to support mission-critical operations has become a critical and complicated issue. Researchers are exploring newer models and methodologies for network design. While traditional approaches used operations research model and simple heuristics, newer approaches are focusing on meta-heuristics (e.g. Tabu search) and GA.

The design of a backbone network infrastructure can be considered as a network topology design problem. Communication networks are typically considered as trees with specific properties. A tree is a connected graph containing no cycles. A spanning tree is a tree that spans all the nodes of an undirected network. Tree optimization problems arise in many forms. The most common form is: Given a graph $G = (V, E)$ with a node set V, edge set E and a weight defined on each edge $e \in E$, find a tree T in G that optimizes the total weight of the edges in T. There might be other constraints imposed such as number of nodes in a sub tree, degree constraints on nodes, flow and capacity constraints on any edge or node, type of services available on node or edge etc.

Minimum Spanning Tree (MST) is one of the best-known network optimization problems. The problem attempts to find a minimum cost tree that connects all the nodes of the network. The links or edges have cost associated with them which could be based on their distance, capacity, quality of line etc. Graham and Hall (1985) provide a comprehensive survey of the development of MST. The complexity of the problem increases significantly as the number of nodes increases, making it impractical to use traditional mathematical models to solve mid-to-large size problems. Hence, many heuristic solutions were developed to solve large problems, prominent among them being Kruskal (1956), Prim (1957) and Dijkstra (1959) algorithms.

Several variations to the basic problem have also been identified under different design contexts, including capacitated MST, probabilistic MST, degree constrained MST etc. (Ahuja, Magnanti, and Orlin, 1993). DCMST is one of the popular variations of MST problem. The MST algorithm, sometimes, may generate a solution where all the links connect to one or two nodes when generating a MST. This may not be a good solution from a reliability perspective since the entire network will become dependent on this node. Also, the costs associated with supporting many links in a node and the technology to support a large number of links may become a significant constraint. Hence, DCMST was developed, which adds additional constraints restricting the number of edges that can be connected to a node.

The traditional mathematical programming approach to solve DCMST problems has been problematic due to exponential increase in the number of constraints with increase in network size. Researchers have developed many heuristic algorithms to solve this class of problems (Obruca, 1968; Narula and Ho, 1980). Although these heuristics solve experimental size problems ($n = 100$), the performance deteriorates when the problem size gets larger. In addition, the problems proposed in these studies dealt with same degree constraints on each node across the entire network. In practical network design problems we are more likely to have different nodes with different levels of fault tolerance requirements and traffic handling characteristics. A realistic application of DCMST would require each node to have different levels of lower and upper degree constraints. Upper degree constraint is used to control resources for each node and make the network manageable from a reliability and operation perspective. Lower degree constraint is used for fault tolerance in the network (Harary and Hayes, 1996; Ku and Hayes, 1996).

GA have been effectively used to optimize telecommunication design problems (Coombs and Davis, 1987; Charddaire, Kapsalis, Mann, Rayward-Smith, and Smith, 1995; Esbensen, 1995; Oyman and Solvinf 1994). Todate, only Zhou and Gen (1996) have examined the use of GA for solving DCMST problem with a fixed degree constraint for all nodes. The main purpose of this study is to apply GA to solve the dynamic DCMST problems.

2 MATHEMATICAL MODEL

The mathematical formulation shown below extends the traditional DCMST (Narula and Ho, 1980) by allowing each node to have both lower and upper degree constraints. The following notations are used:

Indices:
i, j: Index of nodes, $i, j = 1, \ldots, n$
V: Set of vertices in the spanning tree
Parameters:
C_{ij}: The cost to link node i to node j
Ud_i: The upper degree limitation of node i
Ld_i: The lower degree limitation of node i
$|N|$: The number of a subset of nodes in V
$|V|$: The number of the nodes in V
Decision Variables:
$X_{ij}=$ 1 if the link exists; 0, otherwise

Minimize:

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

Subject to:

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

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

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

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

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

Equation (1) is the objective function which seeks to minimize the total connecting cost between nodes. The total cost could be distance cost, material cost or customers' requirement cost. Constraint set (2) specifies that each node has less number of the edges intercepted than the upper limitation Ud_i. Constraint set (3) indicates that each node has more number of the edges intercepted than the lower limitation Ld_i. Constraint set (4) is an anti-cycle constraint based on the theory of spanning tree. Constraint set (5) indicates that the number of edges in a spanning tree is equal to the number of nodes minus one. Constraint set (6) expresses the binary requirements of the decision variables. Although popular integer programming software such as LINDO and CPLEX can be used to solve this problem, constraint (5) increases exponentially with network node size thereby making it impractical for large size problems.

3 RELATED LITERATURE

The earliest heuristic algorithm for DCMST was proposed by Obruca in 1968. Narula and Ho (1980) proposed three heuristic algorithms - primal, dual, and branch and bound to solve the DCMST problem. Savelsbergh and Volgenant (1985) introduced an edge exchanges algorithm, which was found to perform better than the algorithms described above. Many researchers have addressed various other versions of the constrained MST (Camerini, Galbiati, and Maffioli, 1980).

In recent years, GA have been extensively used to solve telecommunications problems. GA, introduced by John Holland (1975), refers to a class of adaptive search procedures based on principles derived from natural evolution or genetics. Palmer and Kershenbaum (1995) compared GA to traditional heuristics to solve the optimal communication spanning tree (OCST) problem and found that the solutions generated by GA were equal or better than the heuristics. Zhou and Gen (1996) used GA to solve DCMST problems. Kapsalis, Rayward-Smith, and Smith, (1993) developed a GA approach to solve Steiner minimal tree problem in graphs, and compared the results of GA with four other heuristics. GA found the optimal solution in all the eighteen problems. Esbensen (1995) used GA to solve the Steiner-tree problem in a graph (SPG) and found that GA-based solutions were 1% from a global optimum in 93% of the time and compared favorably with other SPG approaches in terms of computation time. These results indicate that GA are an attractive approach to solve network design problems within reasonable computation time.

4 PROPOSED GA PROCEDURE

Our proposed procedure starts with encoding the problem, follows by randomly initializing a population of chromosomes. New repair methods are used to repair illegal and infeasible chromosomes. Then, the fitness function evaluates each chromosome and assigns fitness value. Based on the fitness value, the selection operator selects superior chromosomes for the new mating pool. Crossover and mutation operators are employed to identify new solutions based on the chromosomes in the new mating pool. Thereafter, a repair method is used to repair infeasible chromosomes and send them to the evaluation module. The process continues on for the next generation.

4.1 PROBLEM REPRESENTATION

Earlier study of DCMST problem (Zhou and Gen 1996) used Prufer encoding method (Prufer, 1918). However, due to lack of locality, even a slight change in a gene in Prufer encoding can cause a completely different spanning tree compared to the original spanning tree (Abuali et al., 1995). Therefore, determination encoding is used in this study.

Determination encoding is a simple indirect encoding strategy that was first proposed by Abuali et al. (1995) to improve the bottlenecks of Prufer encoding. Similar to Prufer, determination encoding is a node based encoding where the alleles in the chromosomes represent nodes rather than edges. The length of the chromosomes is $N-1$, where N is the number of vertices in a given graph G. The decoding algorithm treats each allele as 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. Although determination encoding is an indirect encoding strategy, the decoding algorithm is very simple, it may generate infeasible trees that needs to be repaired. Figure 1 shows an example of such an encoding, which resulted in an illegal chromosome.

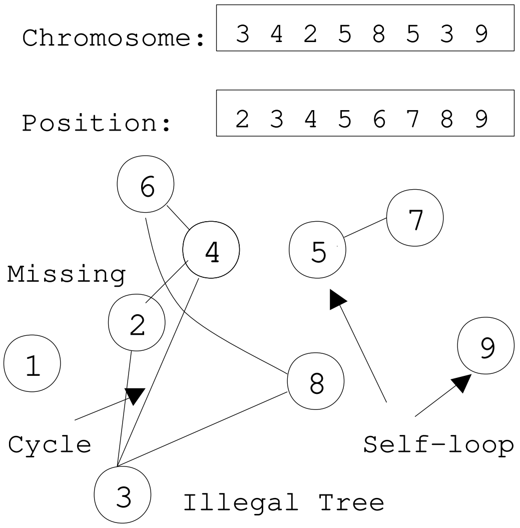

Figure 1: Example of Determination Encoding

4.2 POPULATION INITIALIZATION

The second step in GA implementation is to generate a set of initial solutions for exploration. The number of solutions is dependent on the complexity of the problem to be solved. There are two ways to generate the initial population - random initialization and hybridized initialization (Goldberg, 1989; Holland, 1975). In our experiment, we use random integer initialization method. The initial chromosomes are not required to be legal and may generate a feasible tree.

4.3 REPAIR FUNCTIONS

According to Holland (1975), best result is often located between legal and illegal solutions, and feasible and infeasible solutions. Therefore, a pure repair or penalty function is not an appropriate solution for most problems. We propose a mixed strategy that incorporates both repair and penalty functions.

4.3.1 Repair 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 allele that has a value 1. Since the fixed position starts from 2 the generated spanning tree will not contain 1 and therefore will not span all the nodes (see Figure 1). Self-loop occurs when the value of an allele is equal to its correspondent node position. For example, the chromosome in Figure 1 has self-loops at nodes 5 and 9. The value of the allele is 5 and its correspondent node position is 5, which causes an illegal self-loop connection (5-5). Cycle occurs if a sub-set of links connect in a loop, returning to the original node, in which case one of the links may be unnecessary. The algorithm proposed below describes a strategy to solve all the three situations that lead to illegal spanning trees.

Let N represents the number of nodes and $C(x)$ represents the allele of the x fixed position in chromosome C, where x starts from 2 to $|N|$. The algorithm works as follows:

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 node 1. Set $C(x) = 1$. If there is a tie, randomly select a position.

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 selects a position.

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 $A(x)$ is 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 $1 \leq j \leq |N|$. If $A(j) = A(x)$, set $A(j) = A(C(x))$. Set $j = 1$ and Go to **S4**.

S4: Repairing cycle. Exam $B(j)$, if $j > |N|$, Stop; else if $B(j) = Null$, $j = j + 1$ 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**.

For illustration, let us examine an examine of network with 9 nodes. Table 1 depicts the distance and degree requirements for the tree in Figure 1. To repair missing node 1, nodes 2, 3, and 6 are qualified as they are closest to node 1. We randomly select node 6 because they are tie. The chromosome becomes 3 4 2 5 1 5 3 9). To repair the self-loops of nodes 5 and 9, select the node that is the closest to nodes 5 and 9. Node 4 was connected to node 5 and node 7 to node 9 because they are the closest. The chromosome becomes (3 4 2 4 1 5 3 7). To repair the cycle, we delete the connect between nodes 2 and 4 and connect nodes 4 to node 6 as node 6 is closer to node 4 than node 1. Figure 2 shows the repaired tree, which is legal but infeasible because nodes 4 and 5 violate the degree requirements.

Table 1: Data for Numerical Illustration

(a) Cost Data:

	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	$*$

(b) Degree Constraints Data:

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

4.3.2 Repair for Infeasible Chromosomes

There are two choices to deal with infeasible solutions in GA. One is to repair and the other is to assign a penalty value. In our study, we introduce a method that combines both of them. The proposed repair scheme is derived from the concept of neighborhood proposed in various heuristics such as Prim (Prim, 1957). The algorithm (see below) selects the longest neighbor node to disconnect and the closest neighbor node to connect so as to meet the degree requirements.

S1: For the given encoding string C with length of $|N| - 1$ where N represents nodes. A vector A is used to store the degree statues of all nodes in a graph. Let $A(i)$ represent the value in the ith

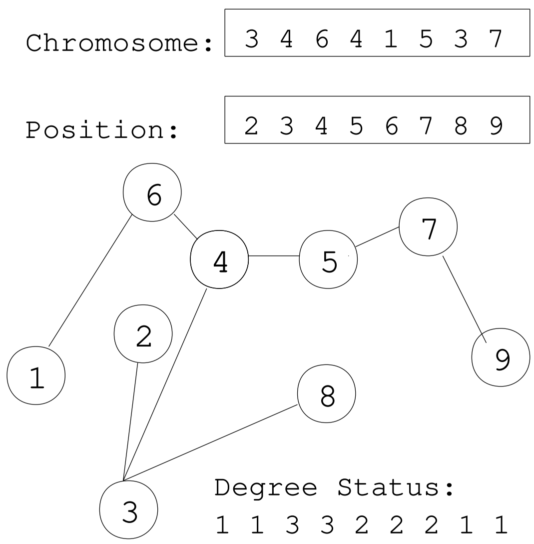

Figure 2: Legal Tree After Repair

position in vector A and $C(x)$ represent 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 $A(s) = d_s$. Set $s = s + 1$, if $s > |N|$, set $s = 1$ and go to **S3**, otherwise go back **S2**.

S3: Compare $A(s)$ with its upper and lower degree limitation 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 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 **S3**. Set $s = s+1$, if $s > |N|$, **Stop**, otherwise go back **S3**.

To repair the tree, first, we check the cost data to identify the highest cost links, (3-4), and (5-4), connected to node 4. Since it is tie we randomly break one of them, say link (3-4). We then check the cost table to 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 continue the process until the degree constraints of all nodes are met. The finalized DCMST is shown in Figure 3.

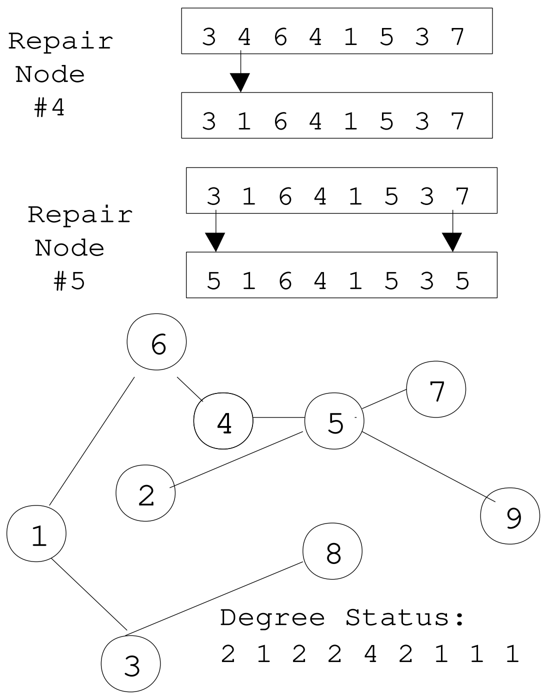

Figure 3: Feasible Tree After Repair

4.4 PENALTY FUNCTIONS

Since the network generated after repairing the degree constraints could now be illegal, 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 violate the degree constraints again. A penalty function is used to penalize chromosomes that violate the constraints. The penalty function adds a negative value to the fitness value. The process of embedding penalty values into fitness values is very critical.

Since increase in chromosome length causes the cost range in the population to increase, we need to add weight to increase penalty with increase in chromosome length. The strategy to consider both penalty and chromosome length can be represented as:

$$P(x) = \sum_{i \in x} |d_i(x) - D_i(x)| \tag{7}$$

$$g'(x) = g(x) + (P(x) + L(x)) * W \tag{8}$$

Where, for each chromosome x, $g'(x)$ and $g(x)$ are the new and original cost values respectively. $P(x)$ is the penalty value. $L(x)$ is the length of chromosome. W is a weight given by users. $d_i(x)$ is the degree of node i in chromosome x and $D_i(x)$ is the lower or upper degree limitation of node i. The penalty value is based on the differences of each allele and its correspondent limitation in a chromosome. The fitness function will adjust the impact of both the lengths of the chromosomes and the penalty.

4.5 FITNESS FUNCTION

Fitness value indicates how good is the objective value relative to others. Fitness value is not an independent but a relative value, relative to a given population. Based on fitness value, we can get the ranking of each chromosome in the population. The chromosome with the largest fitness value represents the best value in the current population. Most GA programs use the mapping scheme proposed by Goldberg (1989). Chromosomes are retained in the next generation if they have higher fitness values. In this problem, we consider the minimum values, since the objective function is a cost function.

4.6 SELECTION METHODS

There are many methods to select the initial population and each has its advantages and disadvantages (Baker, 1985; Back and Hoffmeister, 1991). Researchers prefer to use the enlarged sampling approach in optimization problems since it reduces the possibility of duplicate chromosomes entering the population during selection (Gen and Cheng, 1997). Typically, there are two enlarged sampling strategies, $(\mu + \lambda)$ and (μ, λ), originating from evolution strategies (ES). While in $(\mu + \lambda)$ the new population is selected from a population of parents and offsprings; in (μ, λ), k offspring are generated from each parent in the current population ($\lambda = k\mu$ for some $k > 1$) and the best μ offspring are selected for retention. In our study we used the stochastic $(\mu + \lambda)$ method, where the stochastic algorithm will reassemble the pool of new population.

4.7 STOP CRITERIA

Several stop criteria such as number of total generations, computing time, or fitness convergence are available. Fitness convergence occurs when all the chromosomes in the population have the same fitness value, which is cost in this case. Most studied have used generation as the stop-criteria. In this study fitness convergence is selected as the stop-criteria due to its objectivity.

4.8 GA OPERATORS - CROSSOVER

Crossover is one of the key elements in GA implementation. To explore new solution space, crossover methods are designed to incorporate the interactions between pairs of chromosomes. Traditionally, standard crossovers such as one-point, two-point, and uniform are used in GA models (Holland, 1975; Goldberg, 1989). In this study we use the uniform crossover method. This is a dynamic and non-deterministic method since the algorithm does not decide on how many or what positions to replace. All the replacements are decided randomly. Uniform crossover starts from generating a set of position, called masks, within the length of chromosome. Then a pair of chromosome exchange their alleles with each other based on the generated positions. Since all the random positions may not be neighbors, the algorithm often replaces genes in non-continuous positions.

4.9 GA OPERATORS - MUTATION

Mutation, like crossover, is designed to prevent premature closure, and explore new solution space. However, unlike crossover, it makes changes within an individual chromosome rather than across a pair of chromosomes. The simple mutation method, by Holland (1975), flips a random bit on or off. For combinatorial problem, two mutation methods are normally used, namely, insert and exchange mutation. in our study we used exchange mutation. It randomly selects two positions in a given chromosome and exchanges both genes. The remaining genes are kept intact.

5 EXPERIMENTAL DESIGN

To evaluate the performance of GA and compare its performance with two popular heuristics for DCMST, we use two performance measures - value of the objective function (solution quality) and computation time.

In this study we generated networks with node sizes of 20, 40, 60, 80, 100, 120, and 140 for experiments. Five data sets were generated for each network size. The coordinates of the nodes in the network were generated from a 500x500 Euclidean Plane using random number seed scaled from 0.1 to 0.9. Initially, the experiment was conducted for a fixed degree constraint where all the nodes had the same upper bound degree constraint ($d = 3$). In the second setup we used dynamic degree constraints where the upper bound degree constraints for all nodes could vary ($3 \leq d \leq 5$). We used a random number generator to determine the upper bound constraints for all the nodes. All the GA parameters were controlled for all the experiments. We used uniform crossover, exchange mutation, and stochastic $(\mu + \lambda)$ method for all our experiments. The initial population size was fixed at 100. Crossover and mutation rates were fixed at 1.0 due to use of enlarged selection method. The experiments were conducted on a Gateway Pentium II 266 MHZ. Computer.

6 COMPUTATIONAL RESULTS

The results of the experiment comparing GA and heuristics are shown in Tables 2 and 3. The value of each cell reflects the average from five data sets. The first set of columns in Table 2 provides the relative percentage of solution quality (with 1 as the best) for the fixed degree constraint for the three methods, namely, GA, Primal, and the edge exchange heuristic (AH). The second set of columns provides the relative performance for the dynamic degree constraint. Table 3 provides the actual computation time in the same format for the experiments.

Table 2: Solution Quality of Heuristics and GA

Network	*Fixed Degree*			*Dynamic Degree*		
(N X N)	*GA*	*Primal*	*AH*	*GA*	*Primal*	*AH*
20x20	1	1.49	1.13	1	1.11	1.03
40x40	1	1.10	1.06	1	1.09	1.05
60x60	1	1.24	1.19	1	1.22	1.17
80x80	1	1.31	1.20	1	1.27	1.20
100x100	1	1.20	1.18	1	1.18	1.15
120x120	1	1.33	1.28	1	1.25	1.22
140x140	1	1.34	1.26	1	1.29	1.21

N: Number of nodes.

Table 3: CPU Time of Heuristics and GA (in second)

Network	*Fixed Degree*			*Dynamic Degree*		
(N X N)	*GA*	*Primal*	*AH*	*GA*	*Primal*	*AH*
20x20	2.0	0.00	0.00	1.3	0.01	0.01
40x40	13.9	0.01	0.03	9.6	0.01	0.02
60x60	45.9	0.01	0.07	20.2	0.01	0.07
80x80	81.8	0.04	0.21	57.9	0.05	0.19
100x100	291.9	0.08	0.37	135.5	0.09	0.36
120x120	428.0	0.14	0.65	339.1	0.14	0.66
140x140	565.4	0.21	1.14	526.7	0.23	1.11

The results in Tables 2 and 3 indicate that there is significant difference in solution quality and computation time between GA and heuristics in both experiments, i.e. both static and dynamic degree constraints. As shown, when the network size is small (less than 40 nodes), though the solution quality of GA is better than those of heuristics, the numbers are not much different. However, once the network size increases (more than 100 nodes), these differences become more significant. While GA provides better solution quality compared to heuristics, it is worse than heuristics in terms of computation time, especially when the network size increases.

7 CONCLUSION

In this study we have explored the use of GA for the design of dynamic DCMST problem - a classic network topology design problem. To evaluate the performance of the GA we compared its performance in terms of solution quality and computation time with two popular heuristics.

The results indicate that GA provides better solution quality compared to heuristics, but is worse than heuristics in terms of computation time. While the computation time is comparable for smaller networks, the differences are significant for larger networks.

The use of hybrid algorithms, a combination of heuristics and GA, has been suggested as an area with significant potential (Ahuja and Orlin, 1997). While GA is 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 its search. The combination of these two approaches in a hybrid algorithm provides an algorithm that would be better than the two, independently. The heuristic could be used during the evolution by applying it selectively, or after each generation for generating good starting solutions. This study used a heuristic algorithm to repair infeasible chromosomes and found it to be very successful. Future research could examine the heuristic in other aspects of GA as well.

References

F. N. Abuali, R. L. Wainwright, and D. A. Schoenefeld (1995). Determinant factorization: a new encoding scheme for spanning trees applied to the probabilistic minimum spanning tree problem. *Proceedings of The Sixth International Conference on Genetic Algorithms* 470-477.

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin (1993). *Network Flow: Theory, Algorithms, Applications.* NJ: Prentice Hall.

R. K. Ahuja and J.B. Orlin (1997). Developing fitter genetic algorithms. *INFORMS Journal of Computing* **9**(5):251-253.

T. Back and F. Hoffmeister (1991). Extended selection mechanism in genetic algorithms. *Proceedings of the forth International Conference on Genetic Algorithms* 92-99.

J. E. Baker (1985). Adaptive selection methods for genetic algorithms. In J. J. Grefenstette (ed.), *Proceedings of an international conference on Genetic Algorithms* 101-111.

P. M. Camerini, G. Galbiati, and Maffioli (1980). Complexity of spanning tree problems: Part I. *European Journal of Operational Research* **5**:346-352.

P. Charddaire, A. Kapsalis, J. W. Mann, V. J. Rayward-Smith, and G. D. Smith (1995). Applications of genetic algorithms in telecommunications. *Proceedings of the Second International Workshop on Applications of Neural Networks to Telecommunications* 290-299.

S. Coombs and L. Davis (1987). Genetic algorithms and communication link speed design: constraints and operators. *Proceedings of the Second International Conference on Genetic Algorithms and their Applications* 257-260.

E. W. Dijkstra (1959). A Node on two problems in connection with graphs. *Numerical Mathematics* **1**:269-271.

H. Esbensen (1995). Computing near-optimal solutions to the Steiner problem in a graph using genetic algorithm. *Networks* **26**(4):173-185.

M. Gen and R. Cheng (1997). *Genetic Algorithms and Engineering Design.* NY: Wiley Interscience Publication.

D. E. Goldberg (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* MA: Addison-Wesley Publishing Company, Inc.

R. L. Graham and P. Hall (1985). On the history of minimum spanning tree problem. *Annals of History of Computing* **7**(1):43-57.

F. Harary and J. P. Hayes (1996). Node fault tolerance in graphs. *Networks* **27**(1):19-2.

J. Holland (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor: University of Michigan Press.

A. Kapsalis, V. J. Rayward-Smith, and G. D. Smith (1993). Solving the graphical Steiner tree problem using genetic algorithms. *Journal of Operational Research Society* **44**(4):397-406.

A. Kershenbaum (1997). When genetic algorithms work best. *INFORMS Journal on Computing* **9**(3) 254-255.

J. B. Kruskal (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of American Mathematics Society* **7**:48-50.

H. K. Ku and J. P. Hayes (1996). Optimally edge fault-tolerant trees. *Networks* **27**(3):203-214.

E. Minieka (1978). *Optimization Algorithms for Network and Graphs.* IA: Marcel Dekker, Inc.

S. C. Narula and C. A. Ho (1980). Degree-constrained minimum spanning tree. *Computers and Operations Research* **7**:239-249.

A. K. Obruca (1968). Spanning tree manipulation and the travelling-salesman problem. *Computer Journal* **10**:374-377.

A. I. Oyman and C. E. Solvinf (1994). Concentrator location-problems using genetic algorithms. *oceedings of the Seventh Mediterranean Electrotechnical Conference* 3, 1341-1344.

C. C. Palmer and A. Kershenbaum (1995). An approach to a problem in network design using genetic algorithms. *Networks* **26**(3):151-163.

R. C. Prim (1957). Shortest connection networks and some generalizations. *Bell Systems Technical Journal* **36**:1389-1401.

H. Prufer (1918). Neuer beweis eines satzes uber permutation. *Arch. Math. Phys.* **27**:742-744.

M. Savelsbergh and T. Volgenant (1985). Edge exchanges in the degree-constrained spanning tree problem. *Computers and Operations Research* **12**:341-348.

G. Zhou and M. Gen (1996). A note in genetic algorithms for degree constrained spanning tree problems. *Networks.* **30***(2):91-97.*