Local Search Genetic Algorithm for Optimization of Highly Reliable Communications Networks

Berna Dengiz and Fulya Altiparmak
Department of Industrial Engineering
Gazi University
06570 Maltepe, Ankara Turkey
berna@rorqual.cc.metu.edu.tr

Alice E. Smith
Department of Industrial Engineering
University of Pittsburgh
Pittsburgh, Pennsylvania 15261 USA
aesmith@engrng.pitt.edu

Abstract

This paper presents a genetic algorithm (GA) with specialized encoding, initialization and local search genetic operators to optimize communication network topologies. This NP-hard problem is often highly constrained so that random initialization and standard genetic operators usually generate infeasible network architectures. Compounding this infeasibility issue is that the fitness function involves calculating the all-terminal reliability of the network, a calculation which is computationally expensive. Therefore, it is imperative that the search balances the need to thoroughly explore the boundary between feasible and infeasible networks, along with calculating fitness on only the most promising candidate networks. The algorithm results are compared to optimum results found by branch and bound and also to GA results without local search operators on a suite of 79 test problems. This GA strategy of employing bounds, simple heuristic checks and problem specific repair and local search operators can be used on other highly constrained combinatorial applications where numerous fitness calculations are prohibitive.

1 INTRODUCTION

Although the topological optimization of networks is an important problem in many fields such as telephone lines, road networks, gas pipelines, it has major importance in the computer communication industry, when considering network reliability. In a communication network, *all-terminal* network reliability (also called uniform network reliability) is defined as the probability that every pair of nodes can communicate with each other (Colbourn 1987, Jan 1993). This means that the network forms at least a spanning tree. The design problem is to choose a set of links for a given set of nodes, to either maximize reliability given a cost constraint, or to minimize cost given a minimum network reliability constraint. This problem is NP-hard (Garey and Johnson 1979), and further complicating this growth is the computational intensity required to calculate all-terminal reliability.

This problem has been studied in the literature with both enumerative based methods and heuristic methods. Jan et al. (1993) developed an algorithm using decomposition based on branch and bound to minimize link costs with a minimum network reliability constraint; this is computationally tractable for fully connected networks up to 12 nodes. Using a greedy heuristic, Aggarwal et al. (1982) maximized reliability given a cost constraint for networks with differing link reliabilities and an all-terminal reliability metric. Ventetsanopoulos and Singh (1986) used a two step heuristic procedure for the problem of minimizing a network's cost subject to a reliability constraint. This algorithm first used a heuristic to develop an initial feasible network configuration, then a branch and bound approach was used to improve this configuration. Genetic algorithms (GA) have recently found their way in combinatorial optimization approaches to reliable design, mainly for series and parallel systems (Coit and Smith 1996, Ida et al. 1994, Painton and Campbell 1995). For network design, Kumar et al. (1995) developed a genetic algorithm considering diameter, average distance and computer network reliability. The same authors used this GA to expand existing computer networks (Kumar et al. 1995). Davis et al. (1993) approached a related problem considering link capacities and re-routing upon link failure using a customized GA. Deeter and Smith (1997)

presented a GA approach for a small network design problem with alternative link reliabilities.

Given the NP-hard nature of the problem, heuristics are needed to solve problems of realistic size. However, genetic algorithms have not been used as much as might be expected because of the difficulty of dealing with the feasibility issue. Highly reliable networks imply a severely constrained problem when minimum system reliability is used as a constraint. It is unknown whether a network is feasible or not until the network reliability is calculated. This calculation, if done exactly, is also NP-hard. For networks of realistic size, all-terminal reliability is accurately estimated using a Monte Carlo simulation approach. While computationally tractable for large networks, Monte Carlo is nevertheless an expensive procedure for accurate estimation, from a computational effort viewpoint.

The contributions of this paper are twofold. First, a difficult and realistic problem class is solved effectively and efficiently using a large test suite of 79 problems. Previous work, including those cited above, have demonstrated the heuristic and exact optimization procedures on small networks (10 or fewer nodes), thus the important issue of scale-up is left unanswered. The test problems in this paper range up to 20 nodes and 55 possible links. Second, a general approach to employing easily calculated fitness surrogates to minimize the actual fitness calculation is married with local search algorithms, a penalty function and a seeding strategy to encourage the production of highly fit, feasible solutions. This is a good example of customizing the GA meta-heuristic to a highly constrained combinatorial problem where the fitness calculation is difficult. Local search proves more efficient in identifying near optimal solutions, thereby minimizing the fitness calculation.

2 STATEMENT OF THE PROBLEM

A communication network can be modeled by a probabilistic graph G = (N, L, p), in which N and L are the set of nodes and edges that corresponds to the computer sites and communication links respectively and p is the link reliability. The networks are assumed to have bi-directional links and therefore are modeled by graphs with non-oriented edges. It is further assumed that the graph has no parallel (i.e. redundant) edges. Redundant links can be added to improve reliability, and the approach described in this paper could be easily modified to include redundancy. The optimization problem is:

$$\text{Minimize} \qquad Z = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} c_{ij}\, x_{ij} \qquad (1)$$

Subject to : $R(\mathbf{x}) \geq R_o$

where $x_{ij} \in \{0,1\}$ is the decision variable, c_{ij} is the cost of (i,j) link, $R(\mathbf{x})$ is the network reliability and R_o is the reliability requirement.

The following define the problem assumptions:

1. The location of each network node is given.
2. Nodes are perfectly reliable.
3. Each c_{ij} and p are fixed and known.
4. Each link is bi-directional.
5. There are no redundant links in the network.
6. Links are either operational or failed.
7. The failures of links are independent.
8. No repair is considered.

3 THE GENETIC ALGORITHM

3.1 ENCODING

A variable length integer string representation was used as done by Savic and Walters (1995) to represent a water distribution system. Thiel et al. (1994) also used it to represent the sequence of the possible insertion of objects into the knapsack problem. Every possible link is assigned an integer and the presence of that link is signaled by the presence of that integer in the ordered string. A single string can clearly define which links exist and which do not in a network design as shown in Figure 1.

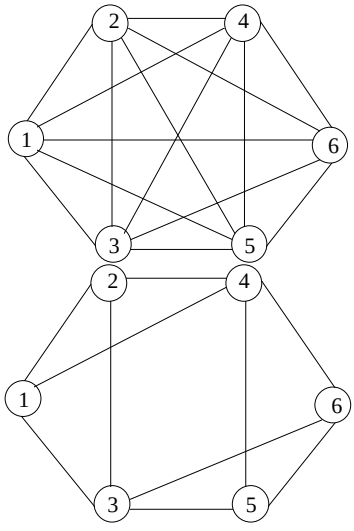

Figure 1: Two networks with six nodes.

String representations of networks given in Figure 1 are [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15] and [1 2 4 6 8 10 12 13 14 15], respectively. Node degree is defined as the number of links which emanate from a given node. For example, node 2 of the second network of Figure 1 has node degree = 3.

3.2 INITIAL POPULATION

To make the search efficient, the initial population consists of networks with the characteristics of being highly reliable. The combination of a stochastic depth-first algorithm with repair is used to generate the initial population by:

1. A spanning tree is implemented through the depth-first search algorithm by Hopcroft and Ullman (1973), which grows a tree from a randomly chosen node.
2. Links selected randomly from the co-tree set (the set of links which are not used in the tree) are added to the spanning tree to increase connectivity.
3. If the network obtained by steps 1 and 2 does not have 2-connectivity (Roberts and Wessler 1970), it is repaired by the algorithm explained in section 3.5.

3.3 OBJECTIVE FUNCTION

The objective function is the sum of the total cost for all links in the network plus a quadratic penalty function for networks which fail to meet the minimum reliability requirement. The objective of the penalty function is to lead the optimization algorithm to near-optimal, feasible solutions. It is important to allow infeasible solutions into the population because good solutions are often the result of breeding between a feasible and infeasible solution and the reproduction procedure does not ensure feasible children even if both parents are feasible, especially in highly constrained problems where the constraint is likely to be active.

The fitness function is given by,

$$Z(\mathbf{x}) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} c_{ij}\, x_{ij} + \delta\, (c_{MAX}(R(\mathbf{x})-R_o))^2 \qquad (2)$$

$$\delta = \begin{cases} 0, & \text{if } R(\mathbf{x}) \geq R_o \\ 1, & \text{if } R(\mathbf{x}) < R_o \end{cases}$$

c_{MAX} = the maximum value of c_{ij}

For computation of R(**x**), three reliability estimations are used to trade-off accuracy with computational effort. An ideal strategy would only employ the computationally intensive method of Monte Carlo simulation on the optimal network design. Since GA is an iterative algorithm, this ideal cannot be attained as many candidate networks must be evaluated during the search. Therefore, screening of candidate network designs is used. First, a connectivity check for a spanning tree is made on all new network designs using the method of (Hopcroft and Ullman 1973). Then, for networks which pass this check, the 2-connectivity measure of Robert and Wessler (1970) is made by ensuring that all nodes have at least degree 2. Finally, for networks which pass both of these preliminary checks, Jan's upper bound (1993) is used to compute the upper bound of reliability of the candidate network, $R_U(\mathbf{x})$. This upper bound is used in the calculation of the objective function (eq. 2) for all networks except those which are the best found so far ($\mathbf{x}_{BEST}$). Networks which have $R_U(\mathbf{x}) \geq R_0$ and the lowest cost so far are sent to the simulation subroutine for precise estimation of network reliability using an efficient Monte Carlo technique by Yeh et al. (1994).

3.4 THE ALGORITHM

The flow of the algorithm is as follows:

Step 1: Generate the initial population of size *s* by the method of section 3.2. Calculate the fitness of each candidate network in the population using eq. 2 and Jan's upper bound as R(**x**), except for the lowest cost network with $R_U(\mathbf{x}) \geq R_o$. For this network, $\mathbf{x}_{BEST}$, use the Monte Carlo estimation of R(**x**) in eq. 2. Generation, *g*, = 1.

Step 2: Select two candidate networks. An elitist ranking selection with stochastic remainder sampling without replacement is used.

Step 3: To obtain two children, apply crossover to the selected networks and mutation to the children.

Step 4: Determine the 2-connectivity of each new child. Use the repair algorithm on any that do not satisfy 2-connectivity.

Step 5: Calculate $R_U(\mathbf{x})$ for each child using Jan's upper bound and compute its fitness using eq. 2.

Step 6: If the number of new children < *s*-1 go to Step 2.

Step 7: Replace parents with children, retaining the best solution from the previous generation.

Step 8: Sort the new generation according to fitness. *i* = 1 to *s*.

a) If $Z(\mathbf{x}_i) < Z(\mathbf{x}_{BEST})$, then calculate the reliability of this network using Monte Carlo simulation, else go to Step 9.

b) $\mathbf{x}_{BEST} = \mathbf{x}_i$. Go to Step 9.

Step 9 : If $g = g_{MAX}$ stop, else go to Step 2 and $g = g+1$.

3.5 2-CONNECTIVITY REPAIR ALGORITHM

If any candidate network does not pass 2-connectivity (i.e., has one or more nodes with node degree = 1), the network is repaired using three different alternatives according to how many nodes fail the test.

Step 1: Determine N_k, n_k; k=1,..., max node degree in a network.

Step 2: Rank all N_k and n_k, except $N_1$and n_1, in increasing order from k=2,.....,max node degree; determine N_{min} and n_{min}.

a) If n_1=1, determine which connection between this node and the nodes in N_{min} set has min cost and connect them, stop.

b) If n_1=2,

- Compute the connection cost of the 2 nodes ($c_{m_{11},m_{12}}$) in N_1 set.
- Compute all $c_{m_{11},m_{\min j}}$ and $c_{m_{12},m_{\min j}}$ for j = 1,2,...,n_{min}.
- If $c_{m_{11},m_{12}} < \min(c_{m_{11},m_{\min j}}) + \min(c_{m_{12},m_{\min j}})$ then connect the 2 nodes in N_1 set; else connect nodes in N_1 set to other nodes in N_{min} through $\min(c_{m_{11},m_{\min j}})$, $\min(c_{m_{12},m_{\min j}})$.

c) If $n_1 > 2$,

- Randomly select 2 nodes from N_1 set,
- Apply (b) for these 2 nodes until $n_1 = 0$.

Where,

N_k	set of nodes with k degree
N_{min}	set of nodes with min degree except nodes with 1 degree
n_k	number of nodes in N_k
m_{1j}	node labels in N_1 set
m_{minj}	node labels in N_{min} set , j = 1,2,..., N_{min}

3.6 CROSSOVER OPERATOR

Crossover is a form of uniform crossover with repair to ensure each child is at least a spanning tree.

Step 1: Randomly select two candidate networks, called T1 and T2. Determine the common links = T1∩T2, other links are:

$\overline{T1} = T1 - (T1 \cap T2)$; $\overline{T2} = T2 - (T1 \cap T2)$

Step 2: Assign common links to children, T1′, T2′.

T1′ = T1∩T2; T2′ = T1∩T2

Step 3: If T1′ and T2′ are spanning trees, go to step 5, else go to step 4.

Step 4: Links from $\overline{T1}$, in cost order, are added to T1′ until T1′ is a spanning tree. Use the same procedure to obtain T2′ from $\overline{T2}$.

Step 5: Determine which links of T1 ∪ T2 do not exist in T1′ and T2′:

CT1 = T1 \ T1′; CT2 = T2 \ T2′

Step 6: T1′ = T1′ ∪ CT2; T2′ = T2′ ∪ CT1

3.7 MUTATION OPERATOR

Mutation takes the form of a local search operator. The mutation operator is applied differently according to node degrees of the network.

Step 1: Determine node degrees deg(j) of the network for j = 1,2,...,N

If deg(j) = 2 for all j; go to Step 2,
If deg(j) > 2 for all j; go to Step 3,
Else, deg(j) ≥ 2; for all j; go to Step 4.

Step 2: Randomly select an allowable link not in the network and add it, stop.

Step 3: Rank links of the network in decreasing cost order. Drop the maximum cost link from the network. If the network is still a spanning tree, stop, otherwise cancel dropping this link, and retry the procedure for the remaining ranked links until one is dropped or the list has been exhausted, stop.

Step 4: Generate u ~ U(0,1). If u <(1-DR) (where DR is the drop rate) go to step 2, otherwise go to step 3.

3.8 PARAMETER VALUES OF GA

Performance was investigated for a set of five parameters; network size (NS), population size (PS), crossover rate (CR), mutation rate (MR) and drop rate (DR). Three levels were selected for each parameter, so the experimental design included 3^5 design points. Five replications were made for each design point, resulting in 1215 observations. Statistical analysis was performed using analysis of variance and Duncan's multiple range tests. While NS, PS and MR were significant at α = 0.05 level, CR and DR were not. The F-statistic values for NS and PS were larger than that of MR, suggesting that the variations in the levels of NS and PS have greater impact on performance than MR. The best results were found for PS = 50 or 75, CR = 0.50, 0.60 or 0.70, MR = 0.20 or 0.30 and DR = 0.50, 0.60 or 0.70. In this paper, the parameters are set at PS = 50, CR = 0.70, MR = 0.30 and DR=0.60. Note that mutation is fairly active; this is a result of its local search effect which fine tunes promising search spaces identified by crossover.

4 COMPUTATIONAL RESULTS

There are two comparisons made to judge the effectiveness and efficiency of the network GA with local search, termed LS/NGA. These are the Branch and bound (B+B) technique by Jan et al. (1993) and the Network GA (NGA) that was fully investigated by the authors in (Dengiz et al. 1997). NGA uses a binary encoding, single point crossover and bit flip mutation, but is otherwise the same as LS/NGA.

The 79 randomly generated test problems are both fully connected and non-fully connected networks with N ranging from 6 to 20[1]. The available links of the non-fully connected networks were randomly generated and were 1.5 times N. The link costs for all networks were randomly generated over [1, 100]. Each problem for the GA's was run 10 times with different random number seeds.

Table 1 shows a summary of the test problems comparing the performance of the two GA approaches with the optimal solutions, in terms of nearness to optimality and computational effort. The results are averaged over each problem instance of each network size and over the 10 replications of each problem instance. It can be seen that LS/NGA does not degrade in performance with increase in problem size while NGA does. Furthermore, while computational effort grows with problem size, it is a more modest growth than for NGA and many orders of magnitude less than the exponential growth for enumerative based methods. This comparison of computational effort is more clearly seen in Table 2.

Table 3 lists complete results of the three methods for all 79 test problems. The conclusions of the results summarized in Tables 1 and 2 are confirmed. The GA's do an excellent job of finding optimal solutions at a fraction of the computational cost of branch and bound for the larger problems. Both GA formulations found the optimal solution in at least 1 of the 10 runs.

Applying statistical tests to the results gives the following. Paired t-tests between the coefficient of variation over 10 runs yields that LS/NGA is superior to NGA with a p value of 0.0000 and a mean improvement (decrease in coefficient of variation) of 0.0104. Turning to CPU time, paired t-tests confirmed that LS/NGA was superior to B+B with a p value of 0.0108 and a mean improvement of 4833 seconds. Paired t-tests of CPU between LS/NGA and NGA resulted in a p value of 0.0000 that LS/NGA is more efficient with a mean improvement of 392 seconds. Because the CPU times of the three methods have significantly different variances, a nonparametric Kruskal-Wallis ANOVA was used to simultaneously compare all three methods. This rank based procedure resulted in a significant difference with a p value of 0.0001 that LS/NGA is most efficient.

The GA described in this paper produces more efficient searches and is also less sensitive to random number seed than NGA. This combination of performance, consistency and efficiency make LS/NGA an appealing approach for optimization of network topologies for problems of realistic size.

5 CONCLUSIONS

It is not surprising that a special purpose GA is more efficient than an enumerative based method on NP-hard problems of realistic size. It is encouraging that the heuristic GA is very effective in identifying optimal solutions even in search spaces up to 10^{16}. The problem studied, while being of interest in many real applications, is not one that particularly lends itself to the GA approach at first glance. There are several major barriers which must be overcome. First, the problem when the network must be highly reliable is very constrained. This is handled initially by repairing children and mutants to ensure they at least *might be* highly reliable. For networks which might be highly reliable, but are not (identified after network reliability is calculated) the infeasibility is handled via a distance based quadratic exterior penalty function. Second, the fitness calculation is computationally burdensome, so use of bounds and repair and local search operators are used. Bounds serve as surrogates in the reliability fitness function for networks which are not the best candidates for the final solution. Repair and local search help identify networks which are particularly promising in their region of the search space.

What is of general interest is the series of steps which can be incorporated into a meta-heuristic, such as GA, which enables the efficient and effective optimization of highly constrained problems with large search spaces where the calculation of fitness is difficult. The steps used included seeding the initial population with solutions which are apt to be highly fit, crossover and mutation operators which tend to produce highly fit offspring and the judicious use of quickly calculated surrogates for fitness. These are upper bounds on network reliability and node degree as an indicator of connectivity. The incorporation of local search into the mutation algorithm improved efficiency. The

[1] All test problems are available from the authors.

performance of the GA was outstanding in terms of ability to optimize very large problems and the modest growth in computational effort as problem size grows.

References

K. K. Aggarwal, Y. C. Chopra and J. S. Bajwa, "Reliability Evaluation by Network Decomposition," *IEEE Transactions on Reliability*, **R-31**, 355-358 (1982).

D. W. Coit and A. E. Smith, "Reliability Optimization of Series-Parallel Systems Using a Genetic Algorithm," *IEEE Transactions on Reliability*, **45**, 254-260 (1996).

C. J. Colbourn, *The Combinatorics of Network Reliability*, Oxford University Press (1987).

L. Davis, D. Orvosh, A. Cox and Y. Qui, "A Genetic Algorithm for Survivable Network Design," *Proceedings of the Fifth International Conference on Genetic Algorithms*, 408-415 (1993).

D. L. Deeter and A. E. Smith, "Heuristic Optimization of Network Design Considering All-Terminal Reliability," *Proceedings of the Reliability and Maintainability Symposium*, (1997).

B. Dengiz, F. Altiparmak and A. E. Smith, "Efficient Optimization of All-Terminal Reliable Networks Using an Evolutionary Approach," *IEEE Transactions on Reliability*, **46**, in print (1997).

M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co. (1979).

J. E. Hopcroft and J. D. Ullman, "Set Merging Algorithms," *SIAM Journal of Computers*, **2**, 296-303 (1973).

K. Ida, M. Gen and T. Yokota, "System Reliability Optimization of Series-Parallel Systems Using a Genetic Algorithm," *Proceedings of the 16th International Conference on Computers and Industrial Engineering*, 349-352 (1994).

R. H. Jan, "Design of Reliable Networks," *Computers and Operations Research*, **20**, 25-34 (1993).

R. H. Jan, F. J. Hwang and S. T. Cheng, "Topological Optimization of a Communication Network Subject to a Reliability Constraint," *IEEE Transactions on Reliability*, **42**, 63-70 (1993).

A. Kumar, R. M. Pathak, Y. P. Gupta and H. R. Parsaei, "A Genetic Algorithm for Distributed System Topology Design," *Computers & Industrial Engineering*, **28**, 659-670 (1995).

A. Kumar, R. M. Pathak and Y. P. Gupta, "Genetic Algorithm Based Reliability Optimization for Computer Network Expansion," *IEEE Transactions on Reliability*, **44**, 63-72 (1995).

L. Painton and J. Campbell, "Genetic Algorithms in Optimization of System Reliability," *IEEE Transactions on Reliability*, **44**, 172-178 (1995).

L. G. Roberts and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," in *Proceedings of the Spring Joint Computing Conference*, AFIPS Conf. Proc. 36, AFIPS Press, 543-599 (1970).

D. A. Savic, and G. A. Walters, "An Evolution Program for Pressure Regulation in Water Distribution Networks," *Engineering Optimization*, **24**, 197-219 (1995).

J. Thiel, and S. Voss, "Some Experiences on Solving Multiconstraint Zero-One Knapsack Problems with Genetic Algorithms," *INFOR Journal*, **32**, 226-242 (1994).

A. N. Ventetsanopoulos and I. Singh, "Topological Optimization of Communication Networks Subject to Reliability Constraints," *Problem of Control and Information Theory*, **15**, 63-78 (1986).

M. S. Yeh, J. S. Lin and W. C. Yeh, "New Monte Carlo Method for Estimating Network Reliability," *Proceedings of 16th International Conference on Computers & Industrial Engineering*, 723-726 (1994).

Table 1: Summary of results of two GA approaches (averaged over 10 runs of each problem size).

Problem			NGA (Dengiz et al. 1997)		LS/NGA	
N	L	Search Space	Mean Solns Searched	Mean % from Optimal	Mean Solns Searched	Mean % from Optimal
6	15	3.28×10^{4}	2378	0.472	1596	0.400
7	21	2.09×10^{6}	6254	1.068	4190	0.777
8	28	2.68×10^{8}	11638	1.176	7811	0.889
9	36	6.87×10^{10}	28166	2.957	12922	1.050
10	45	3.15×10^{13}	62783	3.509	34168	1.094
11	55	3.60×10^{16}	83833	4.675	43566	0.323

Table 2: Comparison of computation time.

Problem		Mean CPU Seconds		
N	L	B+B (Jan et al. 1993)	NGA (Dengiz et al. 1997)	LS/NGA
6	15	0.514	51.313	13.216
7	21	2.859	145.741	35.775
8	28	3839.133	361.253	118.751
9	36	3903.195	588.717	203.386
10	45	4164.566	1175.533	458.937
11	55	59575.263	1532.341	472.105

Table 3: Complete results comparing performance and CPU time.

Problem						B+B (Jan et al. 1993)	NGA (Dengiz et al. 1997)		LS/NGA	
No.	N	L	p	R_O	Best Cost	CPU sec.	Coeff. Var. *	CPU sec.	Coeff. Var.*	CPU sec.
FULLY CONNECTED NETWORKS										
1	6	15	0.90	0.90	231	1.87	0.0245	57.50	0	11.97
2	6	15	0.90	0.90	239	0.01	0	41.05	0	8.28
3	6	15	0.90	0.90	227	0.04	0	38.90	0	12.30
4	6	15	0.90	0.90	212	0.17	0	46.32	0	12.60
5	6	15	0.90	0.90	184	0.28	0	52.39	0.0233	13.72
6	6	15	0.90	0.95	254	0.11	0	69.39	0.0217	19.48
7	6	15	0.90	0.95	286	0.00	0	50.17	0	13.04
8	6	15	0.90	0.95	275	0.06	0	48.37	0	12.40
9	6	15	0.90	0.95	255	0.06	0	59.32	0	14.36
10	6	15	0.90	0.95	198	0.01	0	53.65	0.0121	21.51
11	6	15	0.95	0.95	227	3.90	0.0357	57.98	0.0023	14.08
12	6	15	0.95	0.95	213	0.11	0.0235	47.83	0.0193	10.03
13	6	15	0.95	0.95	190	0.00	0.0280	42.32	0	10.09
14	6	15	0.95	0.95	200	0.44	0.0238	57.54	0.0173	13.04
15	6	15	0.95	0.95	179	0.66	0.0193	46.97	0.0256	11.36
16	7	21	0.90	0.90	189	11.26	0.0177	130.71	0.0175	21.77
17	7	21	0.90	0.90	184	0.17	0	76.74	0	18.80
18	7	21	0.90	0.90	243	0.50	0.0167	135.98	0.0202	26.93
19	7	21	0.90	0.90	129	1.21	0.0121	122.46	0.0195	28.91
20	7	21	0.90	0.90	124	0.05	0	83.45	0	23.77
21	7	21	0.90	0.95	205	0.83	0.0406	301.41	0.0337	71.40
22	7	21	0.90	0.95	209	0.06	0	4	0	37.06
23	7	21	0.90	0.95	268	0.06	0.0310	255.73	0.0187	56.39
24	7	21	0.90	0.95	143	0.17	0.0264	280.26	0.0193	78.72
25	7	21	0.90	0.95	153	0.01	0	160.43	0	52.93
26	7	21	0.95	0.95	185	22.85	0.0333	112.26	0.0111	28.89
27	7	21	0.95	0.95	182	1.27	0.0046	81.78	0.0035	16.99
28	7	21	0.95	0.95	230	1.76	0.0090	109.47	0.0072	26.64
29	7	21	0.95	0.95	122	2.31	0.0265	112.62	0.0259	27.82
30	7	21	0.95	0.95	124	0.39	0	74.49	0	19.64
31	8	28	0.90	0.90	208	21.9	0.0211	260.86	0.0161	79.55
32	8	28	0.90	0.90	203	20.37	0	175.06	0	75.37
33	8	28	0.90	0.90	211	140.66	0.0149	198.80	0.0119	79.67
34	8	28	0.90	0.90	291	173.01	0.0204	210.95	0.0108	83.66
35	8	28	0.90	0.90	178	159.34	0.0112	230.70	0	67.34

* Over 10 runs.

Table 3 continued: Complete results comparing performance and CPU time.

Problem						B+B (Jan et al. 1993)	NGA (Dengiz et al. 1997)		LS/NGA	
No.	N	L	p	R_O	Best Cost	CPU sec.	Coeff. Var.*	CPU sec.	Coeff. Var.*	CPU sec.
FULLY CONNECTED NETWORKS										
36	8	28	0.90	0.95	247	10162.53	0.0152	611.28	0.0140	168.79
37	8	28	0.90	0.95	247	15207.83	0.0274	808.94	0.0183	226.08
38	8	28	0.90	0.95	245	12712.21	0.0124	663.99	0.0034	184.31
39	8	28	0.90	0.95	336	9616.80	0.0169	743.39	0.0177	303.50
40	8	28	0.90	0.95	202	9242.10	0.0231	629.13	0.0235	266.47
41	8	28	0.95	0.95	179	0.11	0	133.32	0	43.81
42	8	28	0.95	0.95	194	2.69	0.0053	202.57	0.0033	40.56
43	8	28	0.95	0.95	197	26.97	0.0052	173.74	0.0080	58.04
44	8	28	0.95	0.95	276	20.76	0.0133	187.02	0.0100	50.64
45	8	28	0.95	0.95	173	72.78	0.0190	189.02	0.0206	53.51
46	9	36	0.90	0.90	239	8.02	0.0105	324.38	0.0066	98.19
47	9	36	0.90	0.90	191	23.78	0.0277	365.31	0.0081	153.77
48	9	36	0.90	0.90	257	702.05	0.0301	530.37	0.0171	176.79
49	9	36	0.90	0.90	171	0.82	0.0255	292.01	0	81.18
50	9	36	0.90	0.90	198	12.36	0.0228	378.91	0	90.49
51	9	36	0.90	0.95	286	8321.87	0.0821	1215.28	0.0325	404.93
52	9	36	0.90	0.95	220	14259.48	0.0330	998.79	0.0309	358.28
53	9	36	0.90	0.95	306	9900.87	0.0313	1256.82	0.0163	560.89
54	9	36	0.90	0.95	219	17000.04	0.0457	865.38	0.0226	340.13
55	9	36	0.90	0.95	237	7739.99	0.0760	1024.77	0.0778	391.52
56	9	36	0.95	0.95	209	4.95	0.0576	274.83	0	59.24
57	9	36	0.95	0.95	171	21.75	0.0137	293.43	0.0092	99.98
58	9	36	0.95	0.95	233	525.03	0.0375	372.18	0.0268	97.95
59	9	36	0.95	0.95	151	0.99	0.0471	252.71	0	65.78
60	9	36	0.95	0.95	185	25.92	0.0381	385.59	0	71.67
61	10	45	0.90	0.90	131	4623.19	0.0518	1047.60	0.0231	375.14
62	10	45	0.90	0.90	154	2118.75	0.0651	794.83	0.0223	214.63
63	10	45	0.90	0.90	267	1860.74	0.0142	999.01	0.0061	415.53
64	10	45	0.90	0.90	263	1466.73	0.0126	678.02	0	171.04
65	10	45	0.90	0.90	293	2212.70	0.0329	1093.36	0.0182	488.12
66	10	45	0.90	0.95	153	5712.97	0.0257	1718.45	0.0150	982.98
67	10	45	0.90	0.95	197	7728.21	0.0203	1689.51	0.0177	726.31
68	10	45	0.90	0.95	311	8248.16	0.0367	1967.61	0.0136	984.30
69	10	45	0.90	0.95	291	6802.16	0.0404	1529.61	0.0244	825.45
70	10	45	0.90	0.95	358	12221.39	0.0276	2662.34	0.0048	1071.99
71	10	45	0.95	0.95	121	3492.17	0.0563	793.22	0.0124	177.31
72	10	45	0.95	0.95	136	1125.89	0.0291	615.29	0.0185	81.87
73	10	45	0.95	0.95	236	987.64	0.0276	781.68	0.0160	139.53
74	10	45	0.95	0.95	245	2507.89	0.0369	632.11	0	98.31
75	10	45	0.95	0.95	268	1359.91	0.0513	630.37	0.0120	131.55
76	11	55	0.90	0.90	246	59575.49	0.0499	1532.34	0	472.11
NON FULLY CONNECTED NETWORKS										
77	14	21	0.90	0.90	1063	23950.01	0.0129	7293.97	0.0079	1672.75
78	16	24	0.90	0.95	1022	131756.43	0.0204	2699.38	0.0185	2334.15
79	20	30	0.95	0.95	596	#	0.0052	5983.24	0.0152	4458.81

* Over 10 runs.

\# Optimum solution taken from (Jan et al. 1993). CPU time unknown.