

# Transportation Network Design Using a Cumulative Genetic Algorithm and Neural Network

YIHUA XIONG AND JERRY B. SCHNEIDER

Currently available algorithms for finding optimal solutions to the discrete transportation network design problem are deficient in two ways. First, their computing time requirements are very large, which makes them infeasible for processing large networks. Second, they cannot process multiple criteria simultaneously—that is, only one objective value can be optimized in one run and, therefore, only one final solution can be obtained. A neural network in the optimal solution search process to replace the trip assignment algorithm for the computation of total travel time is employed. Before a neural network is used, it must be trained and tested with solutions obtained from a user-equilibrium trip assignment model. Experiments show that the trained neural network can predict total travel times quickly and accurately. Next, this neural network is used in combination with a genetic algorithm to search for optimal network designs. The original genetic algorithm did not work well for the problem. However, an analysis of its results suggested improvements that led to the creation of a very powerful search algorithm: the cumulative genetic algorithm. Experiments show that the cumulative genetic algorithm can seek and find system optimal designs extremely fast, using two criteria simultaneously. A full set of optimal solutions can be obtained to construct a trade-off curve for the two criteria. This trade-off curve, composed of optimal solutions, is the boundary of one side of the entire solution space.

The discrete transportation network design problem (DTNDP) involves the selection of new facilities (links) to add to a transportation network or to determine a set of capacity enhancements for some existing links so that the system performance and capital investment costs are optimal. Unfortunately, DTNDP is very difficult to solve because it is NP-hard (1). Currently available methods for finding optimal solutions are deficient in two ways. First, their computing time requirements are very large, which seriously limits the size of the network that can be processed (2). Some methods, such as branch-and-bound, are effective but can handle only very small networks. Heuristic algorithms can handle larger networks and have been used in many applications (3), but their computing requirements are still high. The final solutions are often locally but not globally optimal. Second, none of the currently available methods can handle multiple objectives. They try to minimize either the areawide total travel time on the network under a nonadjustable budget limit or the construction cost in relation to a specific total travel time goal. A third approach is to define first a trade-off relation between the total travel time and cost and then minimize the objective

function defined as a weighted sum of the two (4,5). Using these methods, the overall relationships among the various performance values associated with the optimal solutions cannot be determined, because only one optimal or near-optimal solution can be generated. Many optimal solutions should be seen to obtain a general overview of the problem's solution space and to assist the search for a preferred network development plan.

These difficulties can be greatly reduced by using some new analytical techniques that have become available only recently. The first problem can be addressed using a neural network. An improved genetic algorithm that uses the neural network is used to address the second problem.

Previous studies have shown that the user-equilibrium trip assignment technique can produce a network flow pattern that matches actual trip flows quite well (6). Most network design studies have used this technique to compute the total travel times. However, because the total travel must be calculated for every possible solution encountered in the search process and because the solution space is extremely large, the assignment algorithm must be invoked again and again as alternative network designs are generated and compared with previous solutions. This multiple trip assignment computing requirement is a major bottleneck in the search for optimal designs for networks of reasonable size.

To address this problem, a short list of some possible solutions is generated, and a user-equilibrium trip assignment model [the Frank-Wolfe (F-W) algorithm] is used to calculate the total travel time for each of them. Then, this list is used as a training set to train a neural network. Because a neural network can respond much faster than the F-W algorithm, it replaces the algorithm in the network design optimization process. The tests of the trained neural network have shown that its total travel time predictions become very accurate after a reasonably long training period.

Next, using the neural network that had been trained and tested, an improved genetic algorithm is employed to conduct the network design process; it has been found capable of processing multiple optimization criteria and reaching a set of optimal solutions very quickly. By examining these solutions, the overall relationship between the total travel time and the project costs as well as the distribution of the optimal solutions in the solution space can be understood clearly. Thus, a better understanding of the basic properties of DTNDP and the identification of a preferred network development plan become possible.

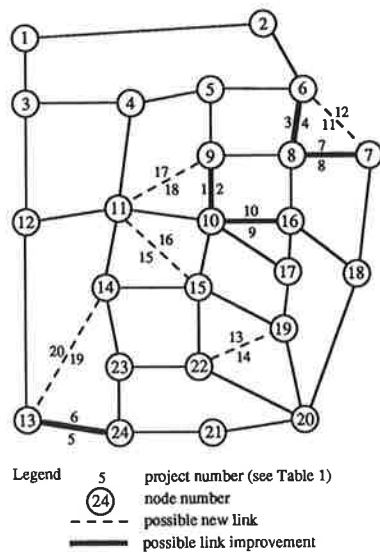


FIGURE 1 Test network.

### DATA SET

Throughout this study, a synthetic network, with a fixed travel demand matrix, and 20 proposed improvement projects, each with a specific cost, have been used (Figure 1). This network has been used as the test network in several previous papers (7,8). The travel time along each link is computed using FHWA's travel time delay function. That is, if the traffic volume on the link  $l$  is  $v_l$ , its travel time from the starting

node to the end node of the link  $l$  will be

$$t_l(v_l) = t_{0,l} \cdot \left[ 1 + 0.15 \cdot \left( \frac{v_l}{c_l} \right)^4 \right] = a_l + b_l \cdot v_l^4 \quad (1)$$

where  $t_{0,l}$  is link  $l$ 's travel time under free-flow condition.

LeBlanc gives the network link parameters ( $a$ 's and  $b$ 's) and the travel demand between each possible node pair (7).

There are 20 proposed projects for this network: 10 are new link constructions and 10 are existing-link improvements. Table 1 presents their assumed (new) parameters and expected construction costs.

### DEVELOPMENT OF NEURAL NETWORK TO REPLACE USER-EQUILIBRIUM TRIP ASSIGNMENT ALGORITHM

The neural network is a recently developed analytical technique that mathematically simulates the connections of the biological neural system in the human brain with respect to how it reacts to changes in the outside environment. A neural network can be trained by giving it some examples (inputs) and the corresponding responses (outputs) required. During the training process, the neural network's interior connections are adjusted so that the network can gradually predict the correct responses. The fundamentals of neural networks are described in several books (9). To obtain the total travel time for any network solution quickly, a neural network is used in the search process as a replacement of the trip assignment model. The neural network can respond much faster than the F-W algorithm. Experimental results show that the trained neural network performs very well in this role.

TABLE 1 Link Parameters and Costs for Proposed Projects

Proj no.	Link	Parameters		Cost (×\$1000)	New?
		$a(×10^{-2})$	$b(×10^{-5})$		
1	9-10	1.60	.0037	625	No
2	10-9	1.60	.0037	625	No
3	6-8	1.30	.1562	650	No
4	8-6	1.30	.1562	650	No
5	13-24	2.20	.2678	850	No
6	24-13	2.20	.2678	850	No
7	7-8	1.50	.0355	1000	No
8	8-7	1.50	.0355	1000	No
9	10-16	2.70	.3240	1200	No
10	16-10	2.70	.3240	1200	No
11	6-7	3.00	.0321	1500	Yes
12	7-6	3.00	.0321	1500	Yes
13	19-22	1.00	.0042	1650	Yes
14	22-19	1.00	.0042	1650	Yes
15	11-15	1.50	.0411	1800	Yes
16	15-11	1.50	.0411	1800	Yes
17	9-11	2.14	.0028	1950	Yes
18	11-9	2.14	.0028	1950	Yes
19	13-14	1.00	.0160	2100	Yes
20	14-13	1.00	.0160	2100	Yes

### Neural Network Structure

To ensure a high level of accuracy, three layers in the neural network have been used. There are 20 proposed projects, so the neural network has 20 input variables, each of which can be either 0 (not constructed or improved) or 1 (constructed or improved). The first layer is the input layer: it distributes each of the 20 input variables to each neuron in the second layer. The second layer affects the neural network's capacity the most. Therefore, this layer includes 61 neurons, which may be more than necessary. And, since there is only one output (total travel time), the third layer (output layer) contains only one neuron and its output is just the neural network's output. Each variable in the first layer is an input to every neuron (except the last one) in the second layer, and each neuron's output in the second layer is an input to the neuron in the third layer. The last neuron in the second layer has no input; it is used to correct the bias on the output (10). Altogether, there are 62 neurons and 1,261 weights in this neural network (Figure 2).

In the neural network, if a neuron's inputs are  $x_i$  ( $i = 1, \dots, n$ ) and their corresponding weights are  $w_i$ , its output will be

$$z = \frac{1}{1 + \exp(-y)} = z(y) \quad (2)$$

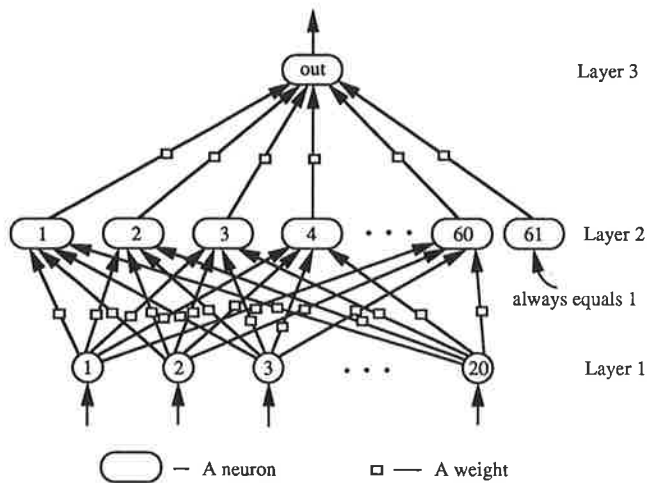


FIGURE 2 Neural network structure used to make total travel time predictions.

where

$$y = \sum_{i=1}^n x_i w_i \quad (3)$$

The output value of the neuron in the third layer is restricted to interval  $[0.25, 0.75]$ , the linear part in the middle of the curve  $z = z(y)$ . The transformation relation between the output of this neuron and the total travel time it predicted is set as a linear mapping:

$$[0.25, 0.75] \leftrightarrow [t_{\min}, t_{\max}] \quad (4)$$

Here,  $t_{\min}$  and  $t_{\max}$  are predefined minimum and maximum possible values of the total travel time on this network. Although the interval  $[0.25, 0.75]$  and linear mapping are used here, it is not certain if they are the best selections.

### Neural Network Training and Testing

There are 20 proposed projects, so there are  $2^{20} = 1,048,576$  combinations. Each combination of projects is a solution (or a design) in the solution space. To train the neural network, 2,500 solutions were selected at random and the F-W algorithm was used to calculate their total travel times. The first 1,000 solutions were used for training, and the other 1,500 solutions were used for testing. Although 1,000 training solutions were used here, a 100-solution training set can produce a quite accurate neural network.

Note that there are two different errors involved here. First, the F-W algorithm is an iterating procedure. The stable value  $t$  for a solution can only be obtained after a large number of iterations. But if we use the algorithm directly in the network design optimization process, we can calculate only a few iterations and will obtain an approximate value,  $t_n$ . Therefore, the first error is computed as  $e = (t_n - t)/t$ .

A second error describes the performance of the trained neural network. Once a neural network has been trained, it must be tested to evaluate its predictive accuracy. If the value

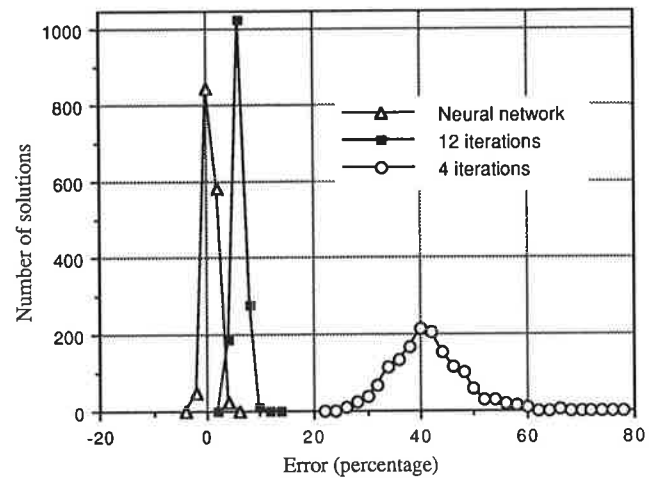


FIGURE 3 Neural network's prediction error versus F-W trip assignment algorithm's error.

it predicts is  $t_n$ , this error will be  $e = (t_n - t)/t$ . Obviously, the neural network's prediction error depends on how accurate the training set is. Therefore, to decrease the neural network's prediction error, a highly accurate training set should be used. In this study, the error for each solution in the training and testing set has been reduced to almost zero (smaller than 0.001 percent) by using a large number of iterations (more than 25). These values, therefore, can be interpreted as being true values ( $t$ ).

After the training set was ready, the neural network was trained using the back-propagation algorithm, a supervised training program widely used in many applications (9). The training algorithm applies each solution in the training set to the neural network sequentially and adjusts its weights until its prediction error is acceptable. Next, this neural network was tested using the testing set. The results from the test are shown in Figure 3, along with two other plots that represent the errors obtained from the same 1,500 solutions but by using the F-W algorithm directly with 4 and 12 iterations. The left plot shows that the errors from the neural network are very small.

### Training Time

Although the neural network's predictive errors are very small, it would not be worthwhile to devise it if the training time was longer than the time required to use the F-W algorithm in the search process directly. A neural network's training time depends on the neural network's complexity defined by the numbers of layers and neurons and their connectivity, the size of the training set, and the precision requirement to decide when to stop the training process. On an Apollo 4500 workstation (whose computing speed is much slower than a mainframe computer), we used 7,200 sec for training. If added with the training set generating time ( $3.1 \cdot 1,000$  sec) and the predicting time (1,200 sec, used in the genetic algorithm as described later), the total would be  $7,200 + 3,100 + 1,200 = 11,500$  sec.

Figure 3 shows that the neural network's prediction precision (standard deviation = 1.00 percent, mean = 0.75 per-

cent) is equivalent to 12 iterations (standard deviation = 1.07 percent, mean = 6.10 percent) of the F-W algorithm. Experiments show that each 12-iteration computation needs about 2.03 sec on an Apollo 4500. If the trip assignment algorithm had been used in the genetic algorithm directly, the total computing time would have been  $2.03 \cdot 80,000 = 162,400$  sec, which is more than 15 times longer than using the neural network. Here, 80,000 is the number of times the neural network was called by the genetic algorithm. Since the F-W algorithm computes very slowly, many previous studies have employed only four iterations (11), and the error (standard deviation = 6.69 percent, mean = 41.26 percent) for such results appears to be unacceptable.

The size of the neural network and its training time are related to the number of proposed projects, not to the transportation network size. This is a fundamental difference between a neural network and the conventional trip assignment algorithm. Therefore, neural networks may be especially useful in dealing with very large transportation networks that have only a few potential projects that need to be examined. This is often the type of problem encountered in practice.

#### DEVELOPMENT OF CUMULATIVE GENETIC ALGORITHM FOR FINDING OPTIMAL SOLUTIONS

A genetic algorithm is a general-purpose stochastic search technique applicable to a broad range of optimization problems. Its development has been derived from the study on natural evolution. During the search process, a genetic algorithm works from solution set to solution set simultaneously, not just sequentially—from one solution to another—as in conventional algorithms. Each solution set is called a generation, and population size is the number of solutions in the set. This generation-to-generation method enables the search process to escape a local optimum and eventually reach global optimum. There are three basic operations within a genetic algorithm: reproduction, crossover, and mutation. Once every operation has been applied sequentially to the current generation, a new generation consisting of offspring will be obtained. This new generation will replace the old entirely and become the current generation; the three operations will apply to it, and we will have another generation. In this repeating procedure, all generations are numbered sequentially and will be called the generation number: for example, the first generation, the second generation, and so forth. The interested reader is referred to Goldberg's book for a more detailed description (12).

##### Original Genetic Algorithm

A genetic algorithm that uses the trained neural network described previously was implemented on an Apollo 4500 [Figure 4 (*top*)]. Here, each solution is represented as a binary string with a length of 20, which is the same as the input variable format of the neural network. Each solution has a construction cost and total travel time, but in the genetic algorithm each solution can have only one objective (fitness) to represent its performance. To use these two values to define

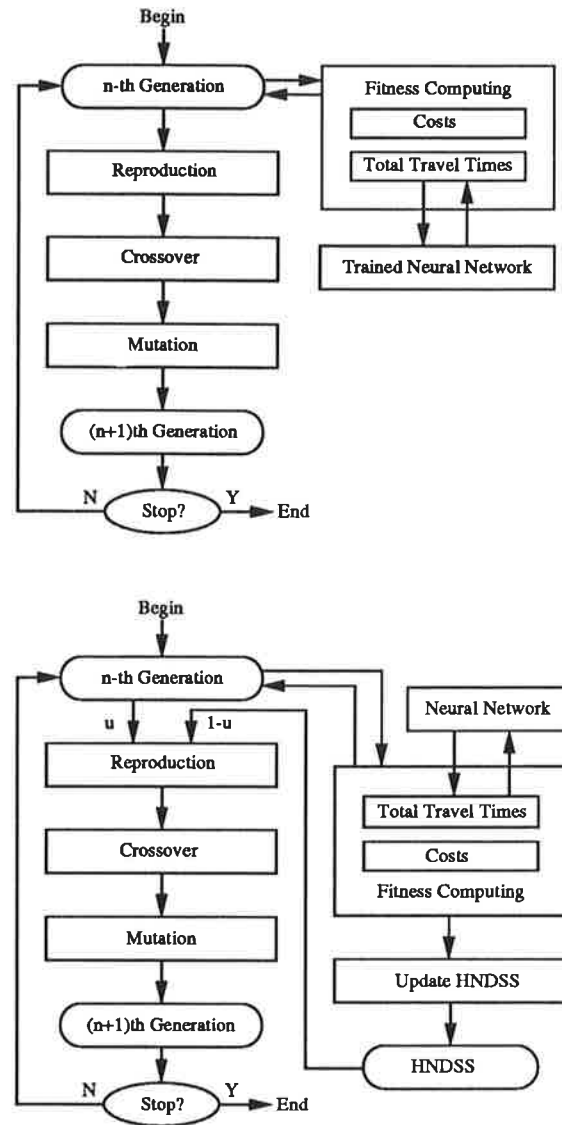
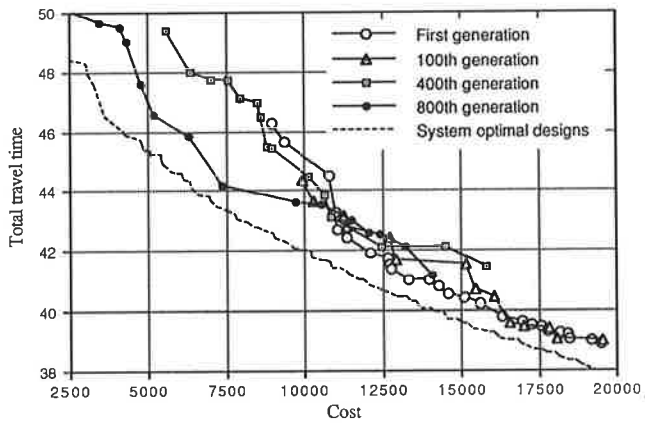


FIGURE 4 Genetic algorithms: *top*, original; *bottom*, cumulative.

a solution's fitness, the domination comparison method was used. A solution  $v$  dominates solution  $w$  if  $v$ 's performance values are not worse than  $w$ 's for both criteria. If a solution is not dominated by any other solution, it is nondominated. For each solution in a generation, a count is made of the number of times that it is dominated by the other solutions in the same generation. The more times it is dominated, the lower its fitness value will be. Therefore, each solution's fitness value is calculated as

$$\text{fitness} = C - \text{number of times dominated} \quad (5)$$

where  $C$  is the largest "number of times dominated" among all the solutions in that generation. Obviously, the nondominated solutions will have a fitness  $C$ . It is hoped that the nondominated solution subsets within each generation improve as the evolution proceeds and that the subset of non-



**FIGURE 5** Solutions generated by original and cumulative genetic algorithms.

dominated solutions from the last generation will be the final optimal solutions.

However, this algorithm's performance was unsatisfactory. Figure 5 shows all the nondominated solutions in the 1st, 100th, 400th, and 800th generations, using a population size of 100. We can see that many "best" solutions in the 100th, 400th, or even 800th generation are even worse than those in the 1st generation. Even the 800th generation is not a clear improvement on the 1st generation. This tells us that the algorithm did not make progress during its search for optimal solutions. Given such results, it is unknown whether the computing process can find the optimal solutions or when the search process can be stopped.

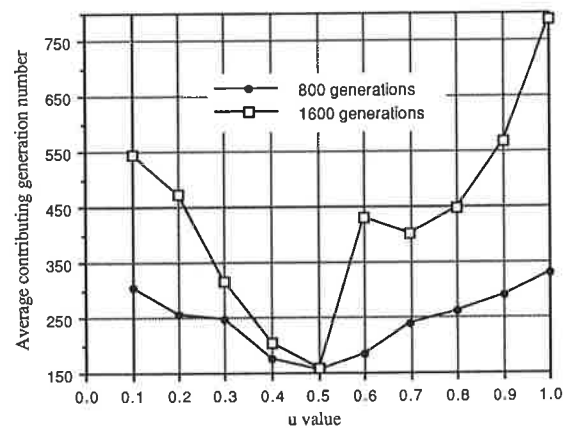
It was determined that this poor performance was due to the fact that each generation passed some good chromosomes to its offspring, but not all of them. And, after a number of generations, some good chromosomes were found to have totally disappeared. To solve this problem, the original genetic algorithm was improved and a cumulative genetic algorithm (CGA) was devised. The solutions obtained by using CGA are clearly superior to the solutions from the original genetic algorithm (Figure 5).

### Cumulative Genetic Algorithm

Transferring good chromosomes is just like transferring knowledge to children. The knowledge transfer cannot be expected to be based on oral communication only, because even if everything new is remembered, other things are often forgotten after a while. So, we often record our knowledge in book form. In this way, children will be able to learn and later review it if they forget. They can also modify the books by adding new knowledge or updating the old. Now, the children can learn either by reading the book or by asking questions of members of their parent's generation. These two options make the process of accumulating knowledge more efficient and faster; the book acts as a knowledge-accumulation device. In this algorithm, the book concept was implemented as a solution set called the historical nondominated solution set (HNDSS). At any specific moment during the search process, HNDSS includes all of the solutions that have

never been dominated (as compared with all of the solutions generated by the algorithm up to that moment). This improved genetic algorithm was named the CGA; its operation procedure is shown in Figure 4 (bottom).

In CGA, the reproduction operation was modified so that it picks up solutions randomly not only from the previous generation (parents), but also from HNDSS. The probabilities of being selected for the solutions in the previous generation are proportional to their fitnesses, whereas the probability of being selected for every solution among HNDSS is equal. Then, after the reproduction, crossover, and mutation operations, if some better solutions are found in the new generation, the algorithm will add them to HNDSS and delete all old solutions that are dominated by the newcomers. When CGA stops, the solutions in HNDSS, instead of those in the last generation, will be the final results. Here another problem is encountered: how much time should be spent on books (HNDSS) and how much on communication with parents (previous generation)? Nine possible proportion values ( $u$ ) have been tested using a population size of 100. Since each solution in HNDSS was contributed by a specific generation in the search history, we call its generation number the solution's contributing generation number. The term "average contributing generation number" is defined as the averaged value of the contributing generation numbers of all solutions in the final HNDSS. When  $u = 1.0$ , the algorithm becomes the original genetic algorithm, and, as shown in Figure 6, the average contributing generation numbers are closely related to the total number of generations carried out. This reveals that the solutions in HNDSS were almost evenly contributed by all generations and thus the process cannot be stopped after 800 generations. On the other hand, the best value is clearly located around 0.5. Its average contributing generation number is the lowest, which represents the shortest computing time. Also, the average contributing generation numbers are the same for both 800 and 1,600 generations. This means that the second 800 generations have contributed no solutions to HNDSS; all the optimal solutions have been generated by the first 800 generations. Therefore,  $u = 0.5$  (half from the previous generation and half from HNDSS) has produced the best results for this problem.



**FIGURE 6** Contributing generation number for different  $u$ -values.

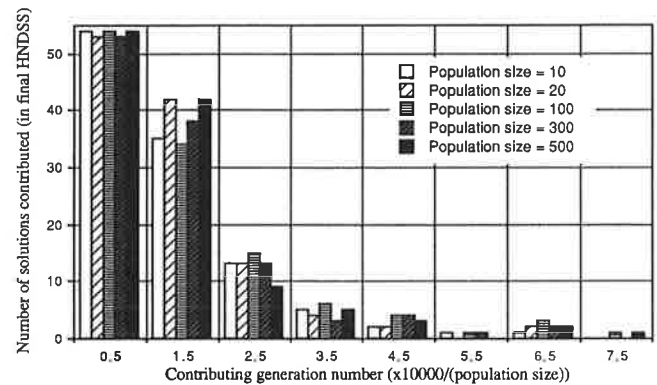
### Population Size Versus Number of Generations Needed

To implement CGA for a DTNDP, there are still two parameters yet to be determined: population size ( $p$ ) and the number of generations ( $g$ ) needed. To find the most suitable parameters for our network, population sizes of 10, 20, 40, 100, 200, 300, 500, and 2,000 have been tested sequentially; their associated performance values are presented in Table 2. We can see that the  $p$  has almost no impact on the algorithm's performance and that  $g$  is inversely proportional to the population size ( $p$ ). This indicates that for a DTNDP, the computing time ( $g \cdot p$ ) is constant.

As shown in Figure 7, the distribution of the number of final solutions contributed by every generation matches a Poisson distribution well, with their means and medians inversely proportional to population size. This shows again that after a certain number of generations, most of the optimal solutions have been generated and further searching becomes unnecessary. For this problem, most of the final optimal solutions are contributed by the first half of the entire search

**TABLE 2 Various Population Sizes and Their Contributing Generation Numbers Using CGA**

Pop-size	Contributing generation numbers		
	Mean	Median	Std.dev
10	1335	1006	1108
20	685.4	542.5	568
40	373.2	308	313.6
100	156.9	116	144.3
200	64.1	41	60.28
300	46.5	34	41.65
500	27.5	21	25.61
2000	7.9	7	5.33



**FIGURE 7 Final optimal solutions in HNDSS: number from every generation.**

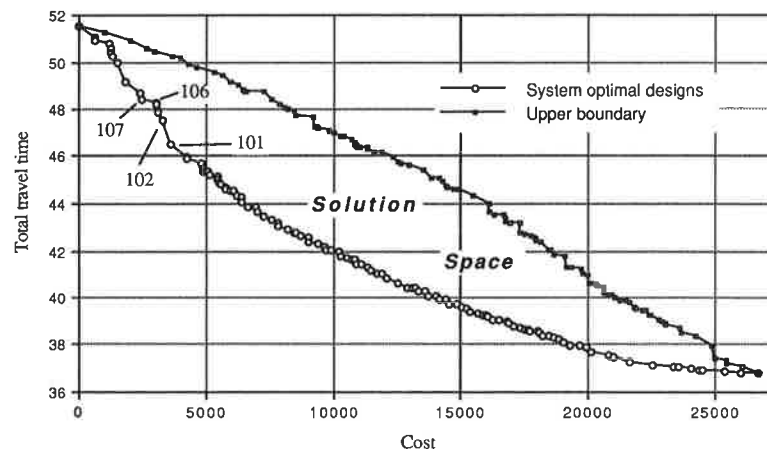
process; that is, if the population size is  $p$ ,  $40,000/p$  generations is enough to obtain 95 percent of the optimal solutions.

### Final Optimal Solutions for Testing Transportation Network Using Two Criteria

Using CGA, 117 optimal solutions were found in HNDSS after 800 generations with a population size of 100 (Figure 8). Furthermore, additional runs were made using different population size values and the final optimal solution sets were found to be identical. This reveals that the final solutions obtained are real system optimums, not just local optimums like those generated by heuristic algorithms.

### DISCUSSION OF RESULTS

If the original genetic algorithm had worked, a population size larger than the number of total optimal solutions would have been used, because the population size is fixed in the process and the last generation must be capable of containing all final optimal solutions. But, how can the number of op-



**FIGURE 8 Final optimal solutions in HNDSS: location in solution space as its lower boundary.**

timal solutions be known, even an approximate one, before the algorithm is run? CGA removes this question because the size of HNDSS is not fixed during the search process.

If those solutions that have the highest costs or longest total travel times can be found, a set of worst solutions using the same CGA with an inverted domination comparison can be obtained. These solutions form another curve, which shares two end points with the optimal solution curve selecting-no-project and selecting-all-projects. Together, these two curves form the entire boundary of the solution space (Figure 8).

The selection of a preferred transportation network design alternative is a complex process and will not normally be limited to only two criteria (total travel time and cost), although they are often the most important. If CGA is to be capable of processing more performance criteria, the domination comparison method can be modified and more performance comparisons included. If, for example, three criteria are used, the solution space would be a solid in three-dimensional space and the final optimal solution set will be a surface of this solid.

For the network design problem, an examination of the optimal solutions shows that the addition of some projects can reduce the total travel time more than others while costing less. These projects are the healthy chromosomes and are contained in most optimal solutions. Among the 117 final designs, Projects 15 and 16 both appear more than 100 times, and Projects 7 and 8 appear only 6 and 4 times, respectively (Figure 9). The addition of the healthy chromosomes will add vertical segments to the solution curve, and the addition of the bad chromosome will add horizontal segments. Therefore, the left part of the curve is more vertical because there are many "healthy" chromosomes left to select, and the right part is more horizontal because all the healthy chromosomes have already been included and only the bad chromosomes are left to select. Also, the curve is smooth in most but not in all locations. In Figure 8, Solution 101 is obtained by using Project 15 to replace Project 11 in Solution 102. Because Project 15 is the healthiest and Project 11 is relatively unhealthy, this replacement costs only \$300,000 but saves 1.014 hr of travel time. But, from Solution 107 to 106, the healthy Projects 3 and 16 are replaced by two relatively unhealthy projects, 11 and 12. This replacement costs \$550,000 but saves only 0.171 hr of time. Thus, the former replacement is represented by a relatively vertical line segment, and the latter replacement is very horizontal.

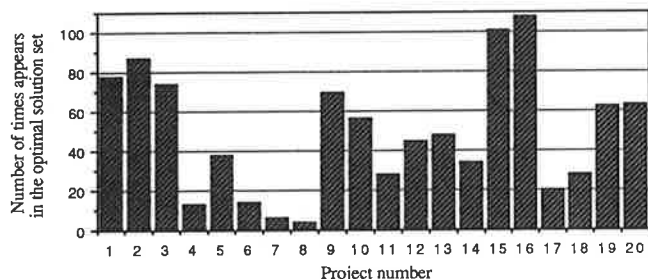


FIGURE 9 Number of times each project appears in set of final solutions.

## CONCLUSIONS AND RECOMMENDATIONS

The neural network is shown to be a technique that can speed up optimization computing dramatically, especially when the accuracy requirement (compared with the user-equilibrium trip assignment algorithm) is not very high (e.g., an error of less than 2 percent). This method also establishes a way of using other trip assignment models in the network evaluation and design process. Since a neural network can be trained with results from the F-W trip assignment model, it is certainly possible to train it using results from other trip assignment models. However, from our experience, if the requirement for the predictive accuracy is very high (e.g., an error of less than 0.2 percent), the neural network's training time would become much longer.

Although the performance of the original genetic algorithm was not satisfactory, CGA worked very well for a DTNDP, even using two criteria. This is a step forward, as multiple criteria have not been used by previous algorithms, and a set of optimal solutions, not just one, has never been generated before. Using the results from CGA, it is clearly seen how the solutions are distributed on the cost-travel time plane for any transportation network. Further, the solutions generated are system-optimal. The trade-off curve defined by these optimal solutions clearly illustrates how one performance value changes when the other changes. This trade-off information for the entire range of potential solutions can greatly assist a decision-making process designed to identify a preferred alternative.

However, this study is just a beginning. It is recommended that further work be undertaken as follows:

1. Larger transportation networks of various shapes should be used to test the performance of the method, including their neural network training times and error characteristics, CGA optimization speed, and the shape of their trade-off curves.
2. Methods for coping with various types of constraints may be applied to the solution space, because constraints are very common in practice. (For example, it might be stated that Project X is not feasible unless Projects Y and Z are both selected, or Project A and B cannot be both selected).
3. Methods should be devised for including other criteria in the genetic algorithm (e.g., the link volume-capacity ratios) and incorporating them in the analyses of the results.
4. Further efforts should be made to define the basic properties of DTNDP in relation to the specific characteristics of a range of transportation network sizes and shapes.

## REFERENCES

1. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., San Francisco, Calif., 1979.
2. T. L. Magnanti and R. T. Wong. Network Design and Transportation Planning: Models and Algorithms. *Transportation Science*, Vol. 18, No. 1, 1984, pp. 1-55.
3. B. N. Janson, L. S. Buckels, and B. E. Peterson. Network Design Programming of U.S. Highway Improvements. *Journal of Transportation Engineering*, Vol. 117, No. 4, 1991, pp. 457-478.
4. B. N. Janson and A. Husaini. Heuristic Ranking and Selection Procedures for Network Design Problems. *Journal of Advanced Transportation*, Vol. 21, No. 1, 1987, pp. 17-46.

5. B. H. Immer and P. H. Mijjer. Optimization of Transport Networks. Presented at 20th International Scientific Conference on Transport Planning and Traffic Engineering, Budapest, Hungary, April 1989.
6. D. M. Chang and G. B. Dresser. *A Comparison of Traffic Assignment Techniques*. Research Report 1153-3. Texas Transportation Institute, Texas A&M University, College Station, 1990.
7. L. J. LeBlanc, E. K. Morlok, and W. P. Pierskalla. An Efficient Approach to Solving the Road Network Equilibrium Traffic Assignment Problem. *Transportation Research*, Vol. 9, No. 5, 1975, pp. 309–318.
8. H. Poorzahedy and M. A. Turnquist. Approximate Algorithms for the Discrete Network Design Problem. *Transportation Research*, Vol. 16B, No. 1, 1982, pp. 45–55.
9. P. D. Wasserman. *Neural Computing: Theory and Practice*. Van Nostrand Reinhold, New York, N.Y., 1989.
10. P. K. Simpson. *Artificial Neural Systems: Foundations, Paradigms, Applications and Implementations*. Pergamon Press, New York, N.Y., 1990.
11. S.-I. R. Tung and J. B. Schneider. Designing Optimal Transportation Networks: An Expert Systems Approach. In *Transportation Research Record 1145*, TRB, National Research Council, Washington, D.C., 1987, pp. 20–27.
12. D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass., 1989.

---

*Publication of this paper sponsored by Committee on Transportation Programming, Planning, and Systems Evaluation.*