A GENETIC ALGORITHM FOR THE MINIMUM TOLLBOOTH PROBLEM

R. CORBAN HARWOOD, CHRISTOPHER J. KOLLMANN, MATTHEW T. STAMPS

Abstract. This paper considers the minimum tollbooth problem (MINTB) for determining a tolling strategy in a transportation network that requires the least number of toll locations, and simultaneously causes the most efficient use of the network. The paper presents a genetic algorithm to solve MINTB, and reports numerical results on small networks.

1. Introduction

In the United States, one third of all road travel is held under congested conditions [2]. These traffic delays result in wasted time which translates to wasted money. Since the effectiveness of the economy depends on an efficient transportation system, it is important to tackle congestion problems rigorously. While road expansion has proven to provide little (if any) relief, tolling systems have recently been implemented in large cities such as London. Although roadway tolls are generally seen as either a revenue source or a hindrance, transportation engineers view them as tools of efficiency. Tolls are used to deter travelers from roadways with high congestion, thereby allowing the systems to run more efficiently.

Hearn and Ramana [10] show that, under mild conditions, the set of all valid tolls can be characterized by linear equalities and inequalities. Thus, toll pricing problems of finding the optimal tolls are either linear programs or linear mixed-integer programs. In [10], examples of linear toll pricing problems include minimizing the total toll revenues and minimizing the maximum toll on a network. Additionally, [10] proposes the minimum tollbooth problem (MINTB) as a linear mixed-integer program, which has remained relatively unexplored.

The primary goal of toll pricing is to change the traffic flow to run efficiently on a network. Still, there may be multiple sets of toll locations that optimize the system, where each location necessitates the cost of setting up an additional toll plaza. Minimizing the number of toll locations while optimizing the system flow is the goal of MINTB. An investigation of MINTB in Bai's dissertation [3] provides the following results. MINTB is NP-complete, when subsidized tolls are allowed. In fact, general purpose solvers for mixed-integer programs, such as CPLEX, have been unable to produce optimal solutions for MINTB for a small network with 24 nodes and 76 arcs. Thus, solutions to MINTB resort to heuristic methods. Bai [4] solves MINTB using a dynamic slope scaling procedure (DSSP), which is shown to be effective on larger networks, such as Sioux Falls, Hull, and Stockholm. She also creates

Date: August 1, 2005.

Key words and phrases. Genetic Algorithm; Minimum Tollbooth Problem; Toll Pricing.

This research was partially sponsored by the National Science Foundation, grant #033870.

a DSSP-based neighborhood function and uses a simulated annealing method for solving MINTB [4]. However, these local search methods may not always reach a global optimal solution.

Genetic algorithms are well known for their ability to sort through a variety of local optimal solutions until they converge on a global optimal solution [12]. As one of several evolutionary optimization methods, the genetic algorithm (GA) uses ideas from natural selection to evaluate, breed, and filter through random solutions to arrive at an optimal solution after many generations. In the literature, genetic algorithms have been used to solve a variety of toll-pricing problems. Zhang uses a GA to determine the optimal toll locations for second-best link-based congestion pricing, where a simulated annealing method was used to solve for the optimal toll levels [16]. Sumalee uses a genetic algorithm in conjunction with a branch-tree framework to create a toll set in a cordon-pricing scheme [12].

Thus far, a genetic algorithm has not been used to solve MINTB. That is the main goal of this paper. The genetic algorithm for MINTB (GAMINTB) randomly generates binary vectors which represent sets of possible toll locations for a given network. After using a linear programming solver to determine feasibility and calculate opitmal toll levels, the algorithm divides the toll sets into feasible and infeasible groups, and ranks them according to their objective values. The algorithm then uses a weighted selection process to randomly determine which toll vectors should become 'parents,' and it uses the alteration process to introduce new toll vectors. The new population is then evaluated for feasibility and the process continues, until the algorithm reaches the specified number of generations. Then, the solution with the minimal number of tollbooths is given.

This paper is organized as follows: In section 2, a clear mathematical model for solving the MINTB is developed. The genetic algorithm approach is discussed in section 3. Section 4 describes the initial computational results of the study including the verification of the GA's solution with a nonlinear program (NLP) solver. Section 5 discusses two implementation methods for future research. Section 6 concludes the findings of the current research.

2. Problem Description

This section formulates the minimum tollbooth problem using network notation. The transportation network is represented by sets of arcs and nodes corresponding to a set of roads and a set of intersections, respectively. An origin-destination (O-D) pair (also known as a commodity) is a pair of nodes on which a user (traveler) in the system must begin and end. There is usually more than one path available for each user to travel along for a given commodity. To understand this problem mathematically, a brief review of the notation is provided. Let $G(N, \mathcal{A})$ denote a network with the set of nodes N and set of arcs $\mathcal{A}$. The set of origin destination pairs is denoted as K where $o(k)$ and $d(k)$ are the origin and destination nodes for the O-D pair k. Let D_k denote the demand for O-D pair k. The demand for an O-D pair k is simply the number of travelers going from the origin to the destination. Let $x^k \in R^{|\mathcal{A}|}$ represent the flow vector for commodity k. The vector x^k is feasible if it is nonnegative and satisfies $Ax^k = b_k$ where A is the node-arc incidence matrix for $G(N, \mathcal{A})$ and

$b_k \in R^{|N|}$ is the demand vector defined by

$$(b_k)_i = \begin{cases} D_k & \text{if } i = o(k) \\ -D_k & \text{if } i = d(k) \\ 0 & \text{otherwise.} \end{cases}$$

The sum of all flows x^k over all O-D pairs that pass over a given arc a is denoted $v_a = \sum_k x_a^k$. It follows, then, that an aggregate flow vector v is feasible only if $v = \sum_k x^k, Ax^k = b_k$, and $x^k \geq 0, \forall k \in K$. Finally, the cost of each path can be measured by the amount of time it takes to travel from the origin to the destination for each user. Let $s(v)$ be the travel cost vector for a given flow v in the network.

2.1. **User Equilibrium.** The *User Equilibrium* (UE) model is used to describe the behavior of users on a given traffic network when every user chooses the shortest path available (eg. see [8]). Or equivalently, a network is at UE only if the cost of every utilized path for each O-D pair is less than or equal to the cost of every non-utilized path for the respective O-D pair. In mathmeatical terms, a network $G(N, \mathcal{A})$ is at UE with $\tilde{v}$ being the equilibrium flow if it satisfies the following condition:

$$s(\tilde{v})^T(v - \tilde{v}) \geq 0, \, \forall v \in V,$$

where V is the set of all feasible aggregate flow vectors for $G(N, \mathcal{A})$.

$$V = \{v | v = \textstyle\sum_k x^k, Ax^k = b_k, \, x^k \geq 0, \forall k \in K\}.$$

Under UE, each user chooses the path for their own interests. Collectively, UE may not be in the best interest for the system as a whole. If too many travelers select the same path, congestion developes. Typically, the per capita cost of traveling along an arc increases as flow increases, slowing down the network as a whole, and hence, the original, low cost path becomes high cost. An example network is offered in section 2.6 to illustrate the user equilibrium model.

2.2. **System Optimal.** The system model describes the network when it is working as efficiently as possible. If the total travel cost of a given network is minimal, then the network flow, $\bar{v}$, is at *System Optimal* (SO). Mathematically, a network $G(N, \mathcal{A})$ is at SO with $\bar{v}$ being the system optimal flow if $\bar{v} \in V$ satisfies

$$s(\bar{v})^T \bar{v} \leq s(v)^T v, \, \forall v \in V.$$

In the SO model, the network, as a whole, is operating as efficiently as possible. A network at SO is rarely at UE as well. This is because some travelers may pay higher costs than they would in UE, as SO minimizes the average cost per user. For an illustration of SO, refer to section 2.6.

2.3. **Tolled User Equilibrium.** Imposing tolls on a network adds a monetary component to the cost function in terms of time. The new cost for each arc, composed of time and money, is called the *generalized* cost. Let β_a denote the toll on arc a, then the *generalized* cost for arc a is expressed as $(s_a(v_a) + \beta_a)$. If a traveler has the choice between two paths where Path 1 takes less time than Path 2,

but Path 1 also costs money, the traveler may be more inept to take Path 2 depending on how much the toll on Path 1 is. By adding a toll vector β to a network, the user equilibrium, $\tilde{v}$ goes from

$$s(\tilde{v})^T(v - \tilde{v}) \geq 0, \forall v \in V$$

to

$$s(\tilde{v}_\beta + \beta)^T(v - \tilde{v}_\beta) \geq 0, \forall v \in V,$$

where the new equilibrium $\tilde{v}_\beta$ is called the *Tolled User Equilibrium* (TUE).

The TUE model describes the network behavior where tolls can be added such that $\tilde{v}_\beta = \bar{v}$. Tolls allow the network to operate so that when every user does what is best for themself, they are also doing what is best for the system.

2.4. **Toll Pricing Problems.** To find the set of valid toll vectors that will cause TUE to equal SO, refer to Hearn and Ramana [10]. It states that a toll vector β is valid if there exists a ρ vector for each O-D pair k satisfying the following:

$$\begin{aligned} s(\bar{v}) + \beta &\geq A^T\rho^k, \forall k \in K \\ (s(\bar{v}) + \beta)^T\bar{v} &= \textstyle\sum_k b_k^T\rho^k \ . \end{aligned}$$

These are essentially the Karush-Kuhn Tucker (KKT) conditions that ensure $\tilde{v}_\beta = \bar{v}$. Moreover, from this set of valid toll vectors, a traffic planner can select an optimal toll for any specific objective such as minimizing total revenue or in the case of this paper, minimizing the total number of tollbooths.

2.5. **MINTB.** As stated previously, the MINTB objective is to place the fewest number of tollbooths on a network so that when the system is at TUE, it is also at SO. Mathematically, let y be a binary vector which corresponds to β so that

$$y_a = \begin{cases} 0 & \text{if } \ \beta_a = 0 \\ 1 & \text{if } \ \beta_a > 0. \end{cases}$$

Then, the objective of the MINTB problem is to minimize $\sum y_a$ subject to:

$$\begin{aligned} s(\bar{v}) + \beta &\geq A^T\rho^k, \forall k \in K \\ (s(\bar{v}) + \beta)^T\bar{v} &= \textstyle\sum_k b_k^T\rho^k \\ 0 \leq \beta_a &\leq My_a, \forall a \in \mathcal{A} \\ y_a &\in \{0, 1\}, \forall a \in \mathcal{A}. \end{aligned}$$

The first two constraints guarantee β is a feasible (valid) toll vector. In the last two constraints, M is merely an arbitrarily large constant ensuring that y and β correspond to one another.

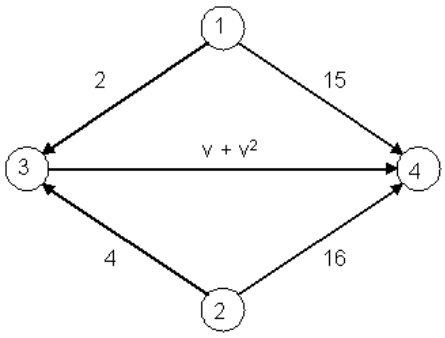

FIGURE 1. An Example Network

2.6. **An Illustrated Example.** Figure 1 is a 4-node, 5-arc network with the given costs. Note that the cost on Arc 5 is dependent on the flow. The cost vector for this network is

$$s(v) =< 2, 4, 16, 15, v_5 + v_5^2 > .$$

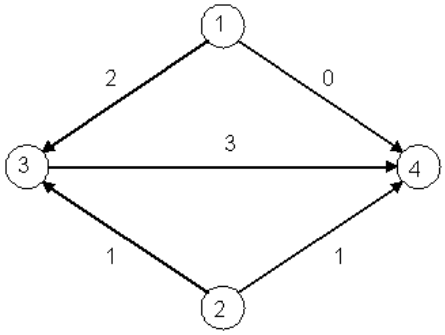

FIGURE 2. User Equilibrium Flows

There are two O-D pairs: 1 to 4 and 2 to 4. The demand on each is 2 cars. The network is at UE when $v =< 2, 1, 1, 0, 3 >$. This can be verified by the new cost vector

$$s(v) =< 2, 4, 16, 15, 12 > .$$

because, for the first O-D pair, the cost for path 1-4 is 15 and the cost for path 1-3-4 is 14, thus path 1-4 is not utilized. For the other commodity, the cost for both paths is 16, so they are both utilized. Note that the total travel cost for the system is 60 units of time.

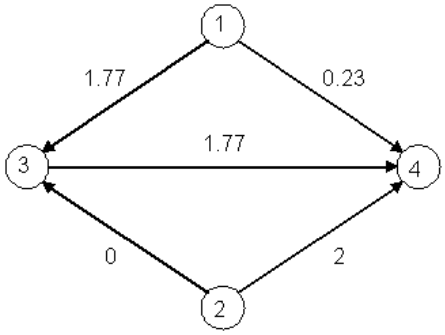

FIGURE 3. System Optimal Flows

In Figure 3 the network is at SO. The total travel cost for this flow on the network is approximately 47.67 units of time. This is the minimum travel time for the system as a whole.

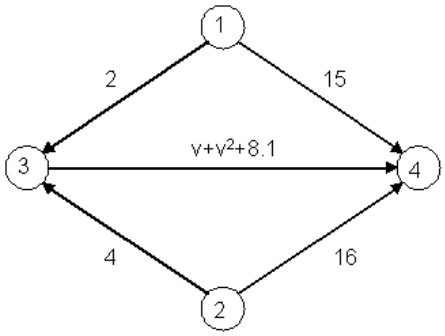

FIGURE 4. Tolled User Equilibrium Flows

Finally, in Figure 4, a toll of 8.1 has been added to Arc 5. This toll raises the *generalized* costs for paths 1-3-4 and 2-3-4 such that when the network is at SO, the travel cost for path 1-4 is equal to the cost on path 1-3-4, so this O-D pair is at UE. Furthermore, the travel cost on 2-3-4 becomes larger than the travel cost on path 2-4 so this O-D pair is at UE as well. Since both O-D pairs satisfy UE, the new TUE is the same as the SO.

3. Genetic Algorithm

3.1. **Overview.** In the past few decades, genetic algorithms have become an increasingly popular heuristic method for problem-solving. The idea of a genetic algorithm is simple from a theoretical standpoint; it uses the principles of evolution to narrow the solution set of a problem down until it gets as close to optimality as possible. Genetic algorithms use the variables of a particular problem to generate strings of numerical alleles which act like chromosomes. The variables in a genetic algorithm can be encoded into the strands in a variety of ways, but most algorithms represent them in binary form. Although the user of a genetic algorithm has some freedom to modify certain parameters or processes, most genetic algorithms follow the same basic steps (eg. see [15]).

(1) *Initialization*

The algorithm begins by randomly generating a starting population. Each strand is designed based on certain parameters which can be modified by the user, including the population size and the length of each strand.

(2) *Evaluation*

This portion of the algorithm rates each chromosome by assigning it a fitness score. This score is based on how well the decoded strand satisfies the algorithm's objective function. The strands are then ranked based on their respective fitness scores. This section also determines whether or not each solution is feasible. The user must decide whether it would be best to exclude infeasible solutions altogether, penalize them by lowering their respective rankings, or merely accept their presence without making any changes.

(3) *Selection*

This process determines which strands from a population will be allowed to reproduce and create the next generation. There are a variety of ways to determine which strands move on, but most involve assigning each strand a probability. This selection probability can be uniform, but it is generally weighted in order to increase the likelihood that the fittest solutions will reproduce.

(4) *Alteration*

The most popular methods of alteration are crossover and mutation. The crossover process is designed to take the traits from a set of parents, and pass them on to a new set of (ideally stronger) offspring. Although there are many ways to implement it, crossover typically involves taking a set of alleles from one parent, and switching them with the alleles of the other parent strand. Some forms of crossover produce two offspring, while others elect to produce one child, and allow one of the parent strands to continue on to the next generation. The mutation process in a genetic algorithm is designed to ensure diversity among the population, thereby decreasing the likelihood of convergence on a locally optimal solution. Mutating a strand usually involves randomly changing some of the numerical values in the offspring. Some genetic algorithms elect to use the principle of elitism, which automatically sends the highest ranking parents to the next generation. Others use immigration [1], which will only breed a percentage of the next generation, and fill the remaining slots with randomly generated strings. The use of elitism generally helps the algorithm to converge faster, while immigration slows the convergence rate and increases diversity among the strands.

(5) *Termination*

After reproducing, most genetic algorithms are designed to cycle through the evaluation, selection, and alteration using the new and altered strands as the new parent population. Genetic algorithms stop after a specified number of generations.

3.2. **The Genetic Algorithm for MINTB (GAMINTB).** The genetic algorithm for MINTB begins by generating a population of size N chromosomes which is used to represent the presence of a set of toll locations. The strands are of length $|\mathcal{A}|$, where $|\mathcal{A}|$ is equal to the number of arcs, and they are indexed so that the ith gene corresponds to the ith arc on the network. Each strand, y, is binary, where a 0 represents an arc that does not have a tollbooth, and a 1 represents an arc that has a tollbooth. For example,

$$y = (0, 1, 0, 0, 1)$$

would represent a 5-arc network with tolls on arcs 2 and 5. The number of chromosomes can be changed in our program by modifying the value of N. The algorithm uses an initialization method comparable to Zhang's [16] in order to prevent bias among the strands. A random number generator creates an $|\mathcal{A}|$ length strand of random numbers between 0 and 1. These numbers are then rounded to the nearest integer (0 or 1). The process is repeated until the number of strands is equal to the specified population size.

After the initial population of toll locations is produced, it must be evaluated in some way. The y vectors are ranked based on two criteria: feasiblility and the number of tollbooths. A y vector is feasible if, for its corresponding β, there exists a ρ so that the KKT conditions are satisfied. For evaluation, the y's are divided into feasible and infeasible groups and then each group is ranked according to the fewest number of tolls. For the final rankings, they are regrouped so that feasible solutions are ranked higher than all of the infeasible solutions. Initially, for deciding on feasibility, the GAMINTB kept track of β's and ρ's for each y. However, since the GAMINTB randomly generates starting values for all of the populations, it becomes very difficult to find the correct β's and ρ's making it very difficult for the GAMINTB to distinguish between feasible and infeasible y solutions. Thus, in order to distinguish between feasible and infeasible y's, the GAMINTB relies on the manual input of infeasible solutions, with hopes that a high power LP solver can be added at a later time. Two other methods for handling feasibility will be discussed in Section 5. It is important to consider infeasible solutions because some of them may be very similar to the optimal solution, and might only require a slight alteration before they can also be feasible. Because of the ellaborate alteration process in the GAMINTB, the good qualities of an infeasible solution may be passed down; however, due to the ranking methods, it is very difficult for infeasible solutions to continue on from one generation to the next.

After the chromosomes are ranked, a set of parents is chosen to produce the next generation. In nature, 'survival of the fittest' typically applies to reproduction. Although the strongest individuals

have the best chance of survival, they are not the only ones that reproduce. Rather than settling on a uniform selection probability, GAMINTB bases a strand's selection probability on its ranking (fitness score). It uses a weighted selection probability comparable to the one used by Sumalee [12]. The weighted probability of selection for a strand of rank i is

$$P(i) = \frac{2((N+1)-i)}{N(N+1)}$$

where N is the population size. Thus, the likelihood of choosing the strongest strand ($i = 1$) for the breeding process is $\frac{2}{(N+1)}$, while the probability of choosing the weakest individual ($i = N$) is $\frac{2}{N(N+1)}$. After the selection probability has been determined for each strand, the genetic algorithm uses a 'roulette wheel' selection process to determine the parents [12]. Each slot on the roulette wheel is assigned an interval between 0 and 1, and the upper and lower bounds are assigned by cumulatively stacking the selection probabilities. A random number generator is then used to select the strands that will produce the subsequent generation. If the random number is between the bounds of a specified slot, that slot's respective chromosome will be chosen as a parent. The algorithm picks the nearest even integer to $R \cdot N$ parents, where R is the reproduction rate defined by GAMINTB.

Once the parents are selected, they are randomly paired to create offspring. Since the GA relies on the passing down of good traits from generation to generation, each pair of parents should pass on each shared allele to one child. Because each y chromosome is a binary strand, each child can only receive a 0 or 1 for each unshared parent allele. The GAMINTB objective is to minimize the sum across the strand, so giving each child a 0 for each unshared parent allele seems logical. However, when implementing this crossover method, the MINTB GA converges to a strand of 0's, which is most likely infeasible, and decreases the variety of solutions. Giving the child a 1 in place of the 0 results in equally bad solutions. Since a critical feature of the GAMINTB is randomness, each child is given a random binary number wherever the parents disagree.

Research done by Ahuja et. al. [1] shows that immigration is a useful tool for adding diversity into a GA. The GAMINTB replaces those strands not selected for reproduction with entirely new strands, following the method used in the initialization phase.

The GAMINTB selects the mearest integer to $(M \cdot N)$ chromosomes for mutation randomly from the entire population pool, where M is the rate of mutation. Being that each y chromosome is a binary strand, mutation is merely changing one randomly selected allele from a 0 to a 1 or 1 to a 0 for a chromosome that is selected.

4. Computational Results

4.1. **Overview.** This section presents the results of the genetic algorithm tested for performance and implementational comparisons over six small example networks. The six examples are built upon two basic network structures, the 4-node diamond structure and the 5-node split structure shown in Figures 5 and 6, respectively.

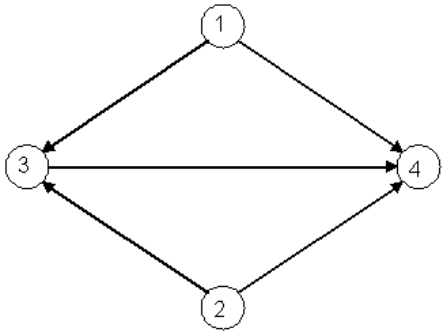

FIGURE 5. A 4-node 5-arc network

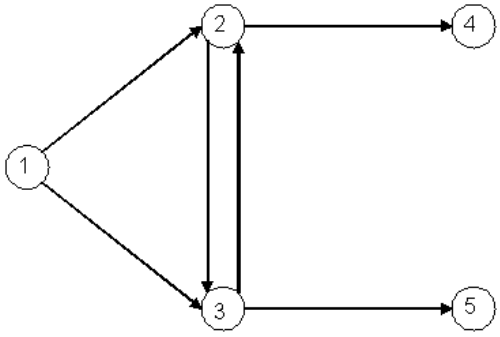

FIGURE 6. A 5-node 6-arc network

The first three example networks were derivations of the 4-node diamond and the last three were derivations of the 5-node split. The demand for each network was randomly selected, while the cost function vector for each was decided by the Bureau of Public Roads (BPR) cost function.

$$S_a(v_a) = T_a(1 + (\frac{v_a}{C_a})^{P_a}),$$

where $S_a(v_a)$ is the cost function, v_a is the variable traffic flow on arc a, T_a is the free travel time constant, C_a is the capacity, and P_a is the delay time factor for arc a. The constants were randomly selected for each example network as follows:

$$T_a \in [1, 10]$$
$$C_a \in [10, 25]$$
$$P_a \in \{0, 1, 2, 3, 4\}.$$

To ensure the GAMINTB converged on the optimal solution, LINGO 7, an NLP solver was used to solve MINTB. For each example network, the GAMINTB converged on the solution produced by LINGO. Among other tasks, LINGO was also used as the exterior LP solver for the GAMINTB by determining the set of infeasible solutions for each network. The GAMINTB was programmed and solved using Fortran. The tests were run on a computer platform with a 2.6 GHz Pentium 4 processor and 512 Mb of RAM.

The *quality* of the GAMINTB is defined as the percentage of the final population that is the optimal solution. It can be used as a measure of both the convergence rate of the GAMINTB and the diversity

created within the GAMINTB. The algortihm performed well if the quality fell between 40% and 60%. A solution quality in this range indicates that a clear winner was discovered, yet a fare amount of diversity was introduced throughout the algorithm.

4.2. **Implementational Comparison Test.** The implementational comparison test compared the quality of varying levels of immigration and mutation, and quality differences between implementing uniform and weighted selection. This test demonstrated the effectiveness of each implementational change according to the GA's quality and aided in the selection of the rate of reproduction, which determined the immigration rate, the percentage of chromosomes to mutate, and whether to implement uniform or weighted probabilities in the selection of parents.

The test was run as follows: for each example network, the GAMINTB was run with mutation alone, immigration alone, and with both; the tested rates of alteration for each scheme were 20%, 30%, and 40%. The initial population size was 100 and the generation number was 20. The selection probability was weighted. After the GAMINTB was run, the qualities were recorded. For the effects of mutation and immigration on the convergence rate, the genetic algorithm yielded the following results.

TABLE 1. Comparing the Effects of Mutation and Immigration

The Percentage of Optimal Solutions in the Final Generation										
Network	Mutation Rate			Immigration Rate			Rate of Both			Neither
	20%	30%	40%	20%	30%	40%	20%	30%	40%	0%
1	82%	70%	62%	61%	30%	31%	56%	11%	3%	100%
2	79%	74%	73%	75%	50%	32%	59%	37%	15%	100%
3	82%	74%	64%	66%	43%	27%	62%	30%	13%	100%
4	80%	65%	64%	61%	37%	10%	43%	22%	8%	100%
5	82%	76%	67%	70%	38%	35%	45%	12%	14%	100%
6	82%	69%	63%	64%	41%	8%	42%	16%	10%	100%

Table 1 makes it clear that immigration is much more effective in slowing the convergence rate of the GAMINTB. However, immigration also allowed enough optimal solutions into the final population to minimize any worries that one might have about the algorithm not converging. The results looked very promising with the standard immigration rate of 30% that was used by the GAMINTB. Mutation on the other hand, was not nearly as effective. It did slow the convergence rate, but only after the mutation rate was raised to an absurdly high level. (A typical mutation rate usually hovers between 2% and 5%)[15]. In most cases, the effect of mutation on the genetic algorithm is minimal. Utilizing crossover alone lead to an extremely high quality for the GAMINTB, which implied that the solution may have converged prematurely. Immigration allowed for more variety among the strands and minimized our

fears of premature convergence. Mutation had a much less drastic impact on the convergence rate, and it was not essential to the success of our genetic algorithm.

Strong arguments have been made in favor of determining the parents based on a uniform probability or on a weighted probability. Ahuja, Orlin, and Tiwari argue that biasing selection in favor of the fitter individuals leads to a faster convergence, and that the use of uniform probability provided them with better results [1]. Conversely, Sumaleee used a weighted probability with a 'selection bias', which could focus on choosing the better strands [12]. The GAMINTB is flexible in that it can switch between a uniform probability or a weighted probability, by merely modifying one of the parameters. For this test, the population size and generation number remained at 100 and 20, respectively. After testing both schemes, the algorithm generated the following results.

TABLE 2. Comparing Uniform and Weighted Probability

Network	Uniform	Weighted
1	7%	30%
2	3%	50%
3	11%	43%
4	1%	37%
5	7%	38%
6	5%	41%

Table 2 supports each of the previous arguments. Both probability methods were able to generate the optimal solution for MINTB, although the algorithm that used the weighted probability converged at a much higher rate. However, the convergence rate for the weighted scheme is not high enough to generate much alarm, while some of the uniform cases were dangerously close to not converging at all. Thus, it can be concluded that GAMINTB is much more effective when using a weighted selection probability. The benefits of uniform probability may have been limited by the fact that the algorithm was only tested on smaller networks. The reliability of weighted probability is certainly higher in these cases, because the odds of converging on a locally optimal solution are much lower due to the presence of a smaller solution space.

4.3. **Performance Test.** The performance test demonstrated the effect of population size and generation number on run-time and quality. A heuristic approach was used in order to create an efficient algorithm for solving MINTB and it was important to confirm that the algorithm was, in fact, efficient. To test how the GAMINTB ran with different population sizes and generation numbers, an experiment was conducted as follows: the mutation rate (percentage of population to be mutated with each generation) and immigration rate (percentage of population to be filled by immigration with each generation) were set at zero and thirty percent, respectively and the selection probability was weighted;

the first part of the experiment fixed the number of generations at 100 and ran the GAMINTB with population sizes of 500, 1000, 1500, and 2000; the next part of the experiment fixed the population size at 100 and ran the GAMINTB with generation numbers of 500, 1000, 1500, 2000. For each trial, the CPU time and quality of the GAMINTB were recorded. The following tables display the results.

TABLE 3. CPU Time for Variable Population Size

Example	Population Size			
	500	1000	1500	2000
1	1.25	4.16	8.83	15.04
2	1.34	4.37	8.97	15.33
3	1.38	4.41	8.90	15.05
4	1.04	3.82	8.06	14.06
5	1.47	4.71	9.59	16.27
6	1.45	4.67	9.4	16.06

Table 3 shows the CPU times with varying population sizes. It is very clear that the CPU time almost doubled with every linear step. This trend was apparent in all six networks, so it is fair to say that the CPU time grows exponentially when increasing the population size.

TABLE 4. CPU Time for Variable Generation Size

Example	Number of Generations			
	500	1000	1500	2000
1	0.56	1.16	1.68	2.26
2	0.64	1.29	1.92	2.55
3	0.72	1.43	2.15	2.84
4	0.35	0.70	1.05	1.38
5	0.74	1.47	2.21	2.92
6	0.71	1.42	2.14	2.84

Table 4 shows the CPU times with varying generation numbers. The table revealed a linear correlation for all six examples. Thus, it may be better to increase the number of generations before increasing the population size.

On the other hand, it is important to look at the quality gains from increasing each. This concept is illustrated in Tables 5 and 6.

TABLE 5. Quality for Variable Generation Size

Example	Number of Generations			
	500	1000	1500	2000
1	24%	38%	38%	37%
2	35%	34%	35%	50%
3	18%	30%	46%	43%
4	35%	35%	40%	37%
5	14%	10%	19%	15%
6	39%	28%	36%	39%

TABLE 6. Quality for Variable Population Size

Example	Population Size			
	500	1000	1500	2000
1	38%	43%	38%	39%
2	41%	43%	41%	43%
3	38%	46%	44%	41%
4	34%	41%	38%	39%
5	21%	24%	15%	18%
6	36%	38%	36%	32%

Notice that in Table 6 after raising the population size above 1000, the quality began to decrease, while in Table 5, increasing the number of generations the quality displayed an upward trend for the six example networks. Because of these trends, it is better to increase the number of generations than it is to increase the population size as the numbers get very large.

Overall, as illustrated in Figures 7, 8, 9 and 10, increasing the number of generations increases the total CPU time less rapidly and still produces similar quality results when compared to increasing the population size. Each of these figures demonstrate the following performance data calculated on the six networks. Networks 1 through 6 are marked by $\diamond, \square, \triangle, \times, *$, and $\circ$, respectively.

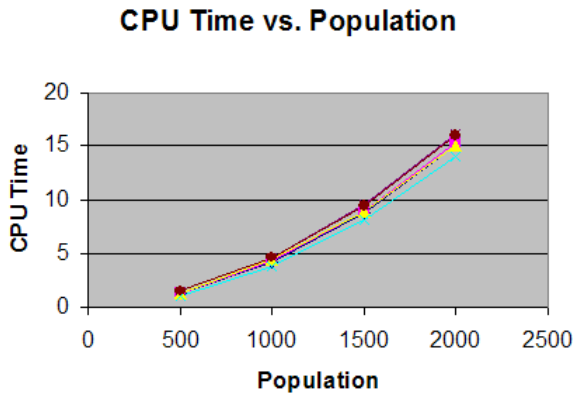

FIGURE 7. CPU Time vs. Population Size

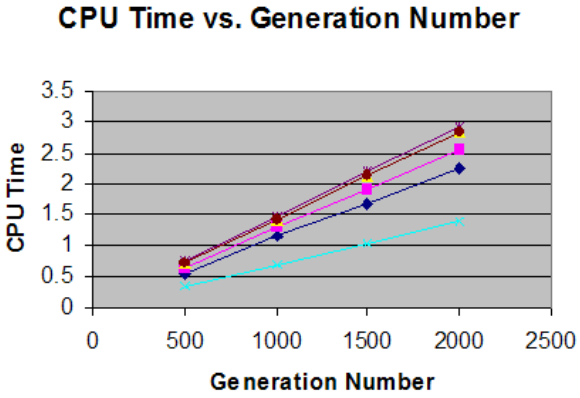

FIGURE 8. CPU Time vs. Generation Number

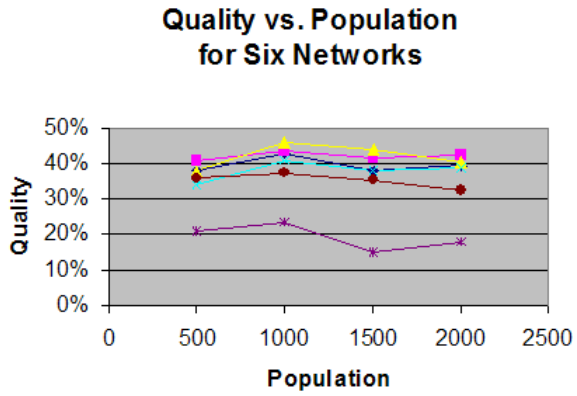

FIGURE 9. Quality vs. Population Size

5. FUTURE RESEARCH

5.1. **Penalty Method.** An alternative method for running the GA without the use of an exterior LP solver is to relax the KKT conditions and add them into the objective function as a penalty. This change will resolve the issue of dividing the y's into feasible and infeasible groups. Instead, they will all be rated by the number of tolls, but they will also be penalized for not satisfying the constraints.

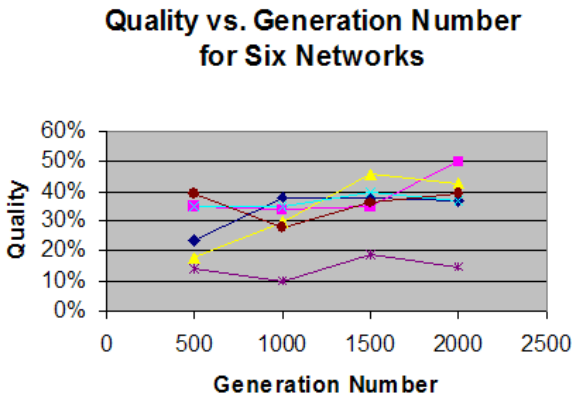

FIGURE 10. Quality vs. Generation Number

To implement this method, the objective function will be composed of three parts. The first of which is the total number of toll booths, which is our main priority to minimize. The second is adding on a scalar of

$$\sum_k (A^T \rho^k - (s(\bar{v}) + \beta))^2$$

if $A^T \rho^k > (s(\bar{v}) + \beta)$. This is the penalty for not satisfying the first constraint. The last component of the objective function is adding on a scalar of

$$((s(\bar{v}) + \beta)^T \bar{v} - \sum_k b_k^T \rho^k)^2,$$

which penalizes y for not satisfying the second constraint. Penalties are weighted because for some of the infeasible solutions, they can grow to very large quantities, which makes the number of tolls trivial. By scaling the penalties down, the number of tolls for each y remains important, but the infeasible solutions are still penalized.

The problem with this method, is that it still depends on accurate β's and ρ's, which is difficult for the GAMINTB. Initial attempts for implementing the penalty method for the GAMINTB yield promising results; however, a more precise algorithm for obtaining β's and ρ's is necessary. When the β's or ρ's are not entirely perfect, the penalty for a feasible solution could be greater than that of an infeasible solution. These occurrences cause discrepencies and, often, a failure in producing the optimal solution. Many issues with the penalty method need adequate exploration. It is left for future research.

5.2. **Nested Method.** A second alternative to the original genetic algorithm is to nest GA's for the toll locations (y), the toll levels (β), and the Lagrangian multipliers (ρ). This means that for each y chromosome, a GA is used over a population of β chromosomes to find the best β for that y. Similarly, for each β chromosome, a GA is used over a population of ρ chromosomes to find the best ρ for that particular y, β pair.

As the y chromosomes are evaluated, selected, and bred, so too are the β and ρ chromosomes, with some changes. First, for evaluation, each β and ρ cannot be evaluated by the original objective,

$\sum_a y_a$. They must either be rated as feasible or infeasible by the KKT conditions, or evaluated by the objective of the penalty method previously discussed.

By internalizing the entire algorithm, the nested method eliminates the need for the exterior LP solver. Theoretically, this internalization would seem preferable, yet many practical problems still remain to be sorted out in further research. First, the main problem with the nested method is that the GA over β is evaluated upon an approximate ρ chromosome and thus the GA over y is evaluated upon an approximate β, ρ chromosome pair. There is no guarantee that approximations built upon approximations will converge on the optimal y solution at all. Second, this tri-level nesting dramatically increases the run time. The original GA ran through g generation loops, where g is the number of generations, the nested method runs through g^3 generation loops. The number of loops due to the size of the population increases similarly.

6. Conclusions

This paper presents a genetic algorithm as a heuristic method for solving the minimum tollbooth problem. The algorithm effectively solves MINTB for six small networks, matching the optimal solutions generated by LINGO 7. The numerical results show that when searching for a better solution, the number of generations should be increased before the population size, due to a lower CPU time, and a higher projected convergence rate. The use of immigration is much more effective in promoting diversity and lowering the algorithm's convergence rate than the more traditional mutation process. This paper concludes that the use of weighed probability in the selection process is better for solving MINTB, although it is doubtful that uniform probability would be so ineffective on networks with larger solution spaces. However, due to the lack of an advanced linear programming solver, calculations cannot be made on larger networks. That problem is open to further research. Other ideas for future research include penalizing the objective value of solutions that fail to satisfy the optimality conditions, rather eliminating them entirely. It is also possible that nested genetic algorithms could be used to solve for an optimal β and ρ, effectively eliminating the need for an LP-solver.

References

1. Ravindra K. Ahuja, James B. Orlin, and Ashish Tiwari, *A greedy genetic algorithm for the quadratic assignment problem*, Computers and Operations Research **27** (2000), 917–934.
2. R. Arnott and K. Small, *The economics of traffic congestion*, American Scientist **82**.
3. Lihui Bai, *Computational methods for toll pricing models*, PhD dissertation, University of Florida, Center for Applied Optimization, Industrial and Systems Engineering Department, 2004.
4. Lihui Bai, Donald W. Hearn, and Siriphong Lawphongpanich, *A heuristic method for the minimum toll booth problem*, 2003.
5. Lihui Bai, Donald W. Hearn, and Siriphong Lawphongpanich, *Decomposition techniques for the minimum toll revenue problem*, Networks **44** (2004), no. 2, 142–150.
6. Halim Ceylan and Michael G. H. Bell, *Genetic algorithm solution for the stochastic equilibrium transportation networks under congestion*, Transportation Research Part B **39** (2005), 169–185.

7. Paolo Ferrari, *Road network toll pricing and social welfare*, Transportation Research Part B **36** (2002), 471–483.
8. M. Florian and D.W. Hearn, *Network equilibrium models and algorithms*, Handbooks in Operations Research and Management Science **8** (1995).
9. M. Florian and D.W. Hearn, *Network equilibrium and pricing*, Handbook of Transportation Science, 2nd Edition (2003), 373–412.
10. D.W. Hearn and M. Ramana, *Solving congestion toll pricing models*, Equilibrium and Advanced Transportation Modeling (1998), 109–124.
11. Simon Shepherd and Agachai Sumalee, *A genetic algorithm based approach to optimal toll level and location problems*, Networks and Spatial Economics **4** (2004), 161–179.
12. Agachai Sumalee, *Optimal road user charging cordon design: a heuristic optimization approach*, Computer-Aided Civil and Infrastructure Engineering **19** (2004), 377–392.
13. Dirk Thierens, *Scalability problems of simple genetic algorithms*, Evolutionary Computation **7** (1999), no. 4, 331–352.
14. G. Wang, Z. Wan, X. Wang, and Z. An, *Genetic algorithm for solving convex quadratic bilevel programming problems*, (2003).
15. Wayne L. Winston and Munirpallam Venkataramanan, *Introduction to mathematical programming*, 4th ed., Brooks/Cole-Thomson Learning, Pacific Grove, CA, 2003.
16. Xiaoning Zhang, *Optimal road pricing in transportation networks*, Ph.d. diss., Department of Civil Engineering, Hong Kong University of Science and Technology, Hong Kong, 2003.