Signal Timing Determination Using Genetic Algorithms

MARK D. FOY, RAHIM F. BENEKOHAL, AND DAVID E. GOLDBERG

The implementation of a genetic algorithm (GA) (an artificial intelligence technique) to produce optimal or near-optimal intersection traffic signal timing strategies is described. The focus is on examining this application within a simple traffic situation, giving the reader a clear understanding of how the genetic algorithm is used. The problem involves finding a signal timing strategy that produces the smoothest traffic flow with the least average automobile delay. The problem domain has many tentative solutions. Therefore, signal timing design is expected to benefit from the parallel, global, and robust search characteristics of GAs. This gain is realized on a simulated four-intersection traffic network in the current implementation. The GA, by considering how traffic moves among multiple intersections (through simulation), can find a logical, near-optimal timing configuration. When this timing configuration is used in the corresponding real-world traffic situation, minimal total automobile delay is expected.

Many motorists are frustrated with traffic signal timings and believe that they can be greatly improved to allow better traffic flow. This is where computers can be useful in the traffic environment. Computers can improve signal timings and therefore improve travel times, personal attitudes, fuel efficiency, pollution, and safety.

This paper presents ideas relating to the use of computers in an automobile traffic environment, specifically ideas to achieve demand-responsive control. The focus is on the use of a genetic algorithm (GA) to control traffic signals and the benefits that can be attained from its use. The implementation discussed in this paper, which uses a GA to control traffic signals, will be called the Traffic GA.

The goal of the Traffic GA is to find near-optimal traffic signal-timing strategies. To achieve this goal, a simplistic traffic flow simulation model was used. The traffic simulation model is sufficient for the purposes of this study; however, it is not intended to be immediately ready for real-world use (i.e., capacity analysis, comparison of actual versus computed delay, etc.). On the other hand, it is possible to improve the current simulation or insert another, more realistic simulation model into the existing GA and use this system to find near-optimal timing strategies by the techniques described in this paper. By making the simulation model more realistic, it would be possible to compare actual traffic conditions with the Traffic GA's simulation module. A simple simulation model was used because the focus of this research was on the application of a GA to improve traffic flow, not the design of a new, more realistic simulation system. Efforts to make the simulation more realistic are essential in model calibration. However, this paper does not deal with these issues.

University of Illinois at Urbana-Champaign, Urbana, Ill. 61801.

MOTIVATIONS

Traffic control today is in need of an intuitive, robust system to continually optimize traffic signal timings. Intelligent computer traffic control systems are needed to dynamically handle changing traffic conditions.

In pretimed controllers, traffic signal timings are fixed at what is determined to be the most effective timing strategy. Timing determination involves either extensive analysis of traffic data or observations of traffic trends, making it a fairly time-consuming task. Because of the time constraints, timing determinations are done infrequently, making the pretimed control method a static model. Therefore, with this method, signal cycle times and offset times are calculated once, for current conditions, and are then set into the individual traffic signals for an extended period of time (i.e., months). The signal cycle timings do not change with demand. This static characteristic is a clear disadvantage and motivates the development of more dynamic methods.

Demand-responsive controllers offer more flexibility than pretimed controllers because traffic signals can have their timings adjusted on the basis of current demand. In addition, flexibility is gained from the capability of each signal of gathering data continually and automatically, allowing continuous analysis of current situations. In the network of demand responsive intersections, a central computer is necessary to (*a*) read traffic data continuously from the entire network, (*b*) process the network data to produce traffic signal timings for all network intersections, and (*c*) operate the traffic signals in a demand-responsive mode. These applications allow an intersection's traffic signal times to be calculated using data from that intersection as well as from adjacent intersections since all data are aggregated in the central computer.

The commonly known full-actuated and semiactuated traffic controllers, as well as the traffic-adaptive control approaches suggested by Gartner (*1*) and Lin (*2*), are all grouped in the demand-responsive category. In this category of traffic controllers, the signal timings are changed depending on the demand, although the nature of these changes is different. The actuated controllers and the traffic-adaptive approach are based on the traditional programming methods, whereas other approaches, like the one described in this paper, use artificial intelligence (AI) techniques.

One of the advantages of AI techniques is that they can be easily designed to perform demand-responsive control on a network of intersections. This paper concentrates on a genetic optimization search algorithm, called a GA. Other applications of AI to intersection traffic control are given elsewhere (*3*,*4*).

When considering a large number of multiphase traffic signals, the number of possible traffic signal-timing strategies can be very large. For example, for a network of 100 intersections, with a cycle length varying from 30 to 150 sec, the number of phases varying from 2 to 5, and the green time allocations varying at increments of 1 sec, the number of possible signal settings is enormous. If a search for the best timing strategy is repeated every few minutes to update the signal settings and a blind search method were used, the number of computations could easily become prohibitive. On the other hand, an intelligent search and optimization system should be able to avoid nonoptimal regions and learn from its past experiences. This should reduce the number solutions searched and allow the system to converge to a near-optimal solution in much less time. In addition, such a system can be put online to overcome some of the limitations of traditional signal optimization techniques.

The question now is whether GAs can find near-optimal signal-timing strategies that improve traffic flow. An answer to this question will be given for a small test problem consisting of a four-intersection street network, but first a more detailed description of GAs will be given.

DESCRIPTION OF GAs

GAs are algorithms that search by manipulating populations of structures (i.e., binary strings representing data structures that symbolize possible solutions to a problem) into new solution populations using operators patterned after natural genetic operations. These operators may include reproduction, crossover, mutation, and others. The three simple GA operators will be discussed later.

GA components can be split into two parts: application-dependent components and application-independent components (such as the GA operators described later). GAs only require two application-dependent components: a procedure to encode bit strings (chromosomes) into solutions to the problem and an evaluation function that will accept a solution to a problem and evaluate its fitness or rating (this function is often called a black box because the GA does not need to know anything specific about this function). The evaluation function, which is also called the fitness function, is similar to the objective function in traditional search problems. Its purpose is to give the GA a numerical evaluation of a possible solution in the same way that an objective function gives a numerical evaluation of a point in space. A GA uses an evaluation function to locate an optimal solution.

GA Evolutionary Process

GAs begin with a population of randomly generated members. The GA then requests that each individual member in the population have its fitness evaluated. The evaluation is done in the fitness function, and the fitness value is returned to the GA. Once a GA has a completely evaluated population, the GA operates on these members to form a new population. This can be thought of as a generation of parents producing a generation of children. Although the new population contains characteristics of the old population, all the new members are different from the members of the last population, so all of its new members must now be evaluated. As this process continues with fitness evaluation and execution of GA operators, new generations of members are created. The new populations are generally more fit (that is, they have higher fitness values) than earlier populations because evolution favors stronger, more fit individuals. This characteristic can be better understood by examining the three basic GA operators.

The Three Simple GA Operators

The GA used in the project discussed here is a simple genetic algorithm consisting of the three basic GA operators.

First, reproduction is responsible for choosing the members that will be allowed to reproduce during the current generation. These members are selected on the basis of their fitness values. All reproduction operators are biased to choose higher-fitness members over lower-fitness members, so high fitness characteristics are passed on to future generations. After the required number of population members has been selected for reproduction (some duplicates in this selection probably will exist), the next operator, crossover, can proceed.

The crossover operator randomly selects two members (i.e., bit strings) from the new subpopulation. Then a location within these two bit strings is selected at random. The location is used as the swapping point for the two strings, that is, all bits to the right of this location on the first string are exchanged with all bits to the right of this location on the second string. For example, suppose the two following strings were selected for reproduction: String A = 00000000 and String B = 11111111. Then suppose the random bit location was selected as 5, causing the two strings to split after Bit 5. This would result in two new strings, String C = 00000111 and String D = 11111000. After the new population has been filled with crossed-over members, mutation can take place.

The mutation operator is simple: with a small probability, a bit will be selected within a string, and it will be flipped (i.e., a 0 would become a 1 and a 1 would become a 0). Then these final members make up the new population, and all old members, from before reproduction, are thrown out. Because we now have a new population with new members, each member must have its fitness evaluated so this evolutionary process can continue.

These are the three basic GA operators, but many variations on these and other operators exist. A description of other operators and further details about GAs are given elsewhere (*5–8*).

PROBLEM DESCRIPTION

The problem addressed in this study entails finding a near-optimal traffic signal timing configuration at all intersections given the current intersection characteristics. The current characteristics consist of the current number of cars at each lane of each intersection and the external arrival volumes. It is anticipated that after the GA converges, the output will be a near-optimal timing configuration for north/south green phase and east/west green phase for the current conditions for all the intersections in the network. As will be discussed later in this paper, the Traffic GA can be run repeatedly (e.g., every 10 min), where each run takes the newest traffic data and produces new traffic signal timings that are better suited to the current traffic conditions. First, the inputs and outputs for running the Traffic GA will be defined.

Input and Output

The preceding perspective allows the problem of traffic control to be considered a function of two vectors. The first is

$$(\text{input.1}) = \begin{bmatrix} n_{111} \\ \cdot \\ \cdot \\ n_{ijk} \\ \cdot \\ \cdot \\ n_{443} \end{bmatrix} \quad (1)$$

where n_{ijk} is the number of cars on Lane k of Approach j of Intersection i (in this example, $i = 1$ to 4, $j = 1$ to 4, and $k = 1$ to 3. The second is

$$(\text{input.2}) = \begin{bmatrix} v_{11} \\ \cdot \\ \cdot \\ v_{ij} \\ \cdot \\ \cdot \\ v_{44} \end{bmatrix} \quad (2)$$

where v_{ij} is the arrival volume on Approach j of Intersection i. In this example, $i = 1$ to 4 and $j = 1$ to 4.

The result of evaluating these two vectors through the Traffic GA will be one integer value, one binary vector, and one real number vector.

$$(\text{output.1}) = [\text{tgt}] \quad (3)$$

where tgt is total green time given to each intersection for one full cycle. The same total green time is used for all intersections in the current Traffic GA, but there is no reason this cannot be changed.

$$(\text{output.2}) = \begin{bmatrix} d_1 \\ \cdot \\ \cdot \\ d_i \\ \cdot \\ \cdot \\ d_4 \end{bmatrix} \quad (4)$$

where d_i is the direction in which the first green phase will allow traffic to flow at Intersection i, either north and south or east and west. In this example, $i = 1$ to 4.

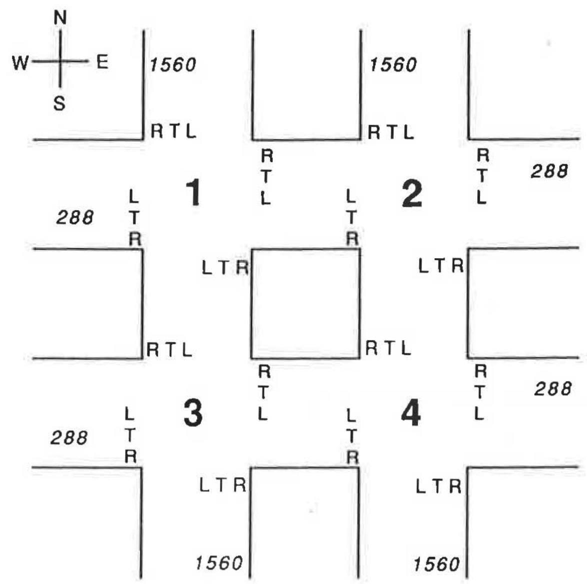

FIGURE 1 Street network configuration (intersections numbered from 1 to 4; L = turning left, R = turning right, T = going straight through). Numbers appearing at external components are arrival volumes in cars per hour.

$$(\text{output.3}) = \begin{bmatrix} \text{nsgt}_1 \\ \cdot \\ \cdot \\ \text{nsgt}_i \\ \cdot \\ \cdot \\ \text{nsgt}_4 \end{bmatrix} \quad (5)$$

where nsgt_i is the proportion of total green time (output.1) that will be allocated for the north/south green phase at Intersection i. In this example, $i = 1$ to 4.

Test Domain

To facilitate simulation and understanding, a typical traffic situation was constructed that was both manageable and comprehensive. The street network has four intersections shaped in a square configuration, each intersection being connected to two other intersections by perpendicular roadways. All GA simulations discussed in this paper were performed using this configuration, shown in Figure 1.

IMPLEMENTATION

The first stage of implementation involved developing a simulation program that could accept both traffic conditions (input.1 and input.2) and a proposed signal-timing strategy (output value output.1 and output vectors output.2 and output.3) and produce an evaluation of this signal-timing strategy under the given traffic conditions. This simulation is needed by the Traffic GA—it is the fitness evaluation black box. This simulation executes cars through the network (see Figure 1).

The simulation is done on a micrograined scale, where all cars are considered separate entities. A car's actions are individually considered at every simulation time step (approximately 3 sec of traffic time), leading to a more accurate real-world representation and increased computational effort. This simulation has limited capabilities and is used only to illustrate the potential that GAs have in locating near-optimal timing strategies. Other simulation models, such as TRAF-NETSIM, are much more complex and can handle a much more diverse set of roadway conditions (*9*).

The simulation module of the Traffic GA at this point should not be compared with other simulation models because the purpose of the Traffic GA is to show how this optimization technique is applied to a traffic situation. The simulation model used here is simplistic at this stage and may not provide simulations more realistic than existing traffic simulation models. Those models have been field tested and validated to replicate real-world traffic conditions, but the Traffic GA simulation has not yet been tested. However, the Traffic GA has a different purpose: to show that a GA can be successfully applied to a traffic timing situation, even with a simplistic simulation model.

The simulator has a number of aspects involving the generation of random events. First, the arrival volumes are specified by the probability of receiving input for any single simulation time step and the bounds on the number of cars coming into the network. The simulator chooses to add input based on the probability and then selects an equally distributed random real number between the given bounds. The integer part of this real number is added as input, and one additional car is added with probability equal to the decimal part of the real number. Alternatively, the simulation could be easily modified to accept single arrival volume values, and the simulator could choose to add input based on the probabilities related to these volumes. Second, the destination of cars is decided at random based on a probability distribution of which lane a car will choose: the left lane (to turn left) (0.15 probability), the middle lane (to go straight through) (0.70), or the right lane (to turn right) (0.15).

Optimization Criteria

The simulation output consists of a value of merit describing how well the cars were able to move through the street network using the given signal-timing strategy under the given traffic conditions. Many different values of merit could have been selected (individually or in combination), including total delay, total number of stops, total linear combination of delay and the number of stops, total cost of losses, total fuel consumption, total person delay, and sum of the squares of the queue lengths (*10*). These criteria options are optimal when they are minimized.

To consider multiple values of merit, an expression that arithmetically combines a number of the individual values of merit could be defined. For example, total delay and total number of stops could be used to define the final value of merit through an expression like

$$m = (k1)(td) + (k2)(ts) \tag{6}$$

where

m = final value of merit,
td = total delay,
ts = total number of stops, and
k1, k2 = specified constants.

In the Traffic GA, total delay, or what we called total average wait time of a car in the street network, was chosen as the preferred evaluation criterion because it is relatively easy to calculate in the Traffic GA's simulation module.

In general, computing automobile delay is a complex process. This process is well documented in the *Highway Capacity Manual* (*11*). The process used to compute delay in the simulation discussed here is simple. However, this procedure is sufficient for the purpose of this study—to examine the application of a GA to traffic signal optimization. The GA can function in the same manner with more complex delay equations. For this study, the delay is computed by counting the total number of cars involved in the simulation and summing the number of cars that were not moving for each simulation time step. The expression for "total average wait time per car" is

$$\frac{\sum_{i=1}^{i=\mathrm{TTS}} w_i}{\mathrm{TC}} \tag{7}$$

where

TTS = total number of time steps executed in a complete simulation,
w_i = number of cars waiting at Time Step i, and
TC = total number of cars in the network.

This expression indicates how long, on the average, a car will be delayed between the time it enters the street network and the time it exits the street network.

This evaluation expression needs to be modified slightly so a GA can use it during reproduction. The GA's only requirement from the simulation module is availability to an objective function (that will produce a fitness value). This function needs to be optimal at maximum values. Therefore, since the evaluation criteria we chose above relates to a minimization problem, it needs to be converted to a maximization problem. This was done by using the inverse of the total average wait time per car. Therefore, since we want to minimize the wait time per car, we'll need to maximize the inverse of this wait time.

Decision Variables

Specifically for the current implementation of the Traffic GA there are nine decision variables: one global variable (total green time) and two local variables for each of the four intersections. The two local variables are (*a*) the directions in which the first green phase will allow traffic to flow (that is the north/south traffic will be allowed to move first = 1 or the east/west traffic will be allowed to move first = 0) and (*b*) the proportion of the total green time allocated to the north/south green phase (a real value between 0.0 and 1.0). This results in a cycle consisting of two phases, a north/south phase and an east/west phase where left-turning cars proceed during traffic gaps (i.e., permitted). The directions of flow for the first phase are determined from the variables above, and the directions of flow for the second phase are assumed to be the directions perpendicular to the first phase's directions (e.g., if the first direction is east/west then the second phase's direction is north/south). Therefore, the GA will not have the opportunity to change the alternating nature of the traffic signals but will be allowed to change which directions get the green phase first.

Note that this choice of decision variables is not fixed. Because of the adaptive nature of GA applications, other decision variables could be easily implemented in the future. For example, if a user wanted to add more than two phases per intersection cycle or wanted to include offsets as decision variables, only the bit string and the simulation would have to be altered. The GA's overall structure would not have to be changed.

Constraints

These decision variables have been established so that almost no external constraints are needed. The only constraints on the variables are the strict limitations on the range of values they may use. First, the direction can only take on binary values because there are only two phases implemented in the current traffic simulation module. Second, the individual green phase times may not be less than 6 sec because times less than this would barely allow any cars to get through an intersection.

Bit String Coding

A direct coding of these nine decision variables was chosen. The global variable, total green time, was coded into a four-bit string mapped between the minimum total green time (24 sec) and the maximum total green time (2 min). The first phase directions are coded directly from a single bit as stated above. The variables that represent the proportion of total green time allocated for north/south green phase are coded into four-bit strings. The four bits are converted to an actual time value by transforming to an integer value between 0 and 15, dividing the number by 15 (to get the number between 0.0 and 1.0), and then mapping it to an integer between minimum green time (6 sec) and total green time (calculated above) − minimum green time (6 sec).

The nine decision variables result in a $4 + (1 + 4) * 4 = 24$ bit string. This string is ordered as follows: the total green time and then the two variables for each intersection are grouped together, and then strung together from Intersections 1 to 4, as shown in Figure 2. This ordering was selected so that intersection characteristics would be adequately near one another, so the GA would have a higher probability of developing tight linkage between the relevant bits (5).

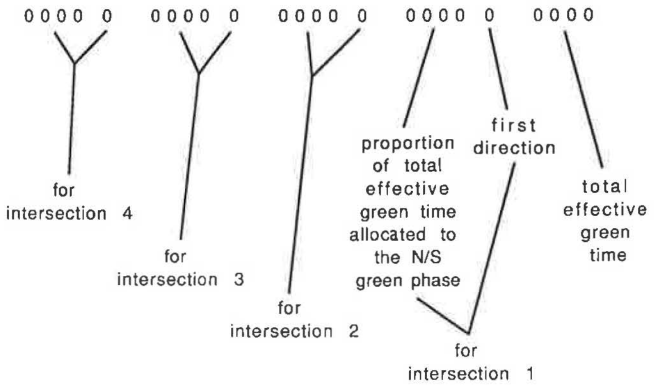

FIGURE 2 Bit string mapping (read right to left).

Traffic GA—Step by Step

As discussed earlier, most simple GAs operate similarly to what is described here, with only the fitness function varying from application to application. Figure 3 is a flow chart of the steps executed by the Traffic GA to find a near-optimal traffic signal-timing configuration for given traffic conditions.

The three main steps involved in the Traffic GA are shown in Figure 3. First, the traffic simulation and the GA are initialized. The initialization of the traffic simulation involves establishing the street configuration and the traffic conditions within the computer program. The simulation need not be reinitialized later in this procedure because all simulations start at these same common conditions. The initialization of the GA involves establishing an initial, completely random population of bit strings. The bit strings symbolize traffic signal timing strategies as described earlier.

The second main step in the Traffic GA is the fitness computation. This involves taking each GA population member and executing a simulation using the timing strategy represented by this member. The fitness evaluation step is executed many times because new population members are continually being generated by the GA. Fitness evaluation is usually continued until the GA has converged; this point is generally defined by the user.

The last main step is the evolution of the GA population. This involves manipulations on the bit strings (i.e., operations on the population members). The three manipulations, or operators, used in the Traffic GA are reproduction, crossover, and mutation, which were described earlier.

The Traffic GA may be run either off-line or on-line. If it is run off-line, the Traffic GA finds a near-optimal signal-timing strategy for any given traffic condition. If it is run on-line, traffic information is continuously gathered from detectors placed on all approaches to all intersections, and the Traffic GA is periodically executed. It is possible to specify very short time intervals between execution, but this would probably not be desirable. To execute, the Traffic GA would be given the most recent traffic data, and then it would be expected to find a near-optimal signal-timing strategy that promoted smooth traffic flow. The new signal-timing strategy would be used in the real traffic signals until the Traffic GA was executed again with new, updated traffic information.

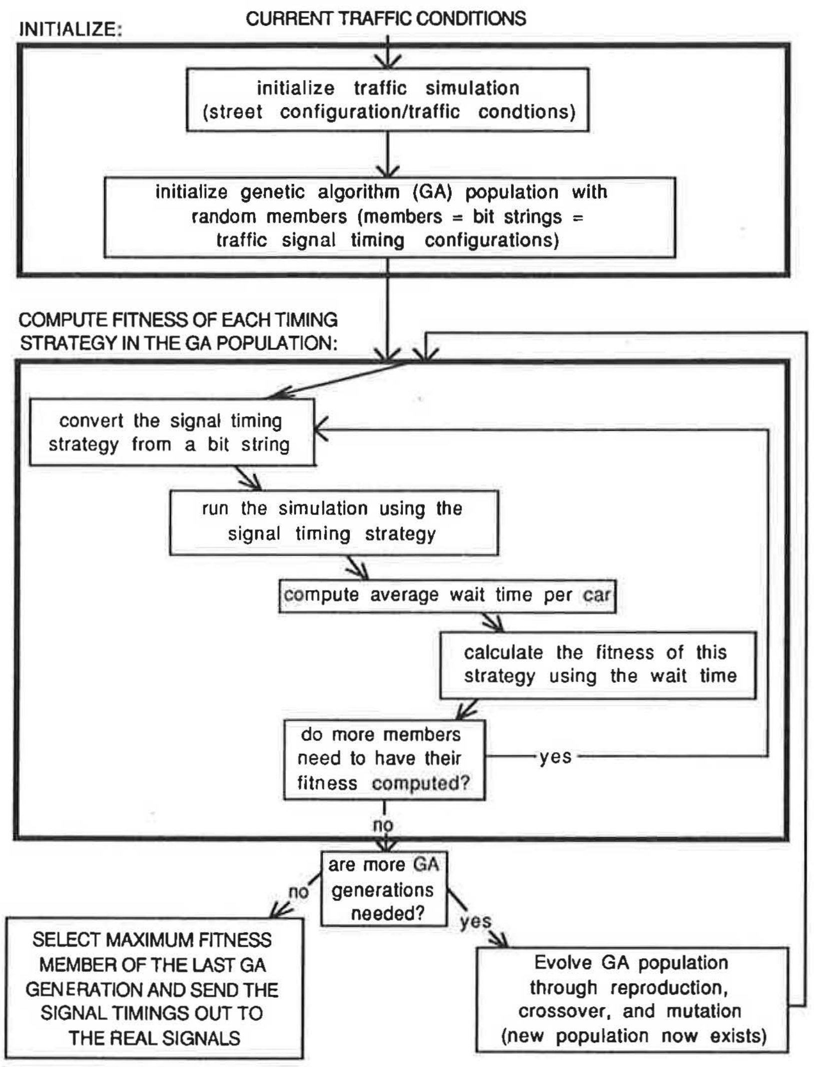
INITIALIZE:
CURRENT TRAFFIC CONDITIONS
initialize traffic simulation (street configuration/traffic condtions)
initialize genetic algorithm (GA) population with random members (members = bit strings = traffic signal timing configurations)
COMPUTE FITNESS OF EACH TIMING STRATEGY IN THE GA POPULATION:
convert the signal timing strategy from a bit string
run the simulation using the signal timing strategy
compute average wait time per car
calculate the fitness of this strategy using the wait time
do more members need to have their fitness computed?
yes
no
are more GA generations needed?
no
yes
SELECT MAXIMUM FITNESS MEMBER OF THE LAST GA GENERATION AND SEND THE SIGNAL TIMINGS OUT TO THE REAL SIGNALS
Evolve GA population through reproduction, crossover, and mutation (new population now exists)

FIGURE 3 Traffic GA procedural flowchart.

COMPUTATIONAL RESULTS

The results of running this GA on typical traffic situations can vary depending on the simulation settings used. All runs performed during the writing of this paper show steady improvement in the average population fitness as the GA population evolves from generation to generation.

Simulator Parameter Settings

In the case examined here, the GA simulations used typical parameter settings: four intersections configured in a square (see Figure 1); yellow time of 3 sec; the probability of a car going straight = 0.70, left = 0.15, and right = 0.15. The average rate at which cars enter an intersection on a green phase was as follows: for cars going straight, one car every 2 sec, right, 1 car every 2 sec, and left, one car every 6 or 12 sec (permitted to enter depending on the arrival volumes of the opposing traffic). The length of time for a car to get from one intersection to another was 24 sec (translates into a distance between all adjacent intersections of 1,000 ft and a constant traveling speed of 28 to 30 mph). The simulation time was 5 min (equal to 100 simulation time steps). The minimum green phase time was 6 sec; the maximum green phase time was 114 sec. The minimum cycle time was 30 sec, and the maximum cycle time was about 126 sec.

Traffic Environment

A common traffic environment was used for all GA runs discussed in this paper. The input.1 vector, the number of cars at all locations, was initialized with typical numbers. This situation was initialized with typical numbers since no particular real-world situation was involved. The task of modifying the code to read in current traffic conditions from detectors, so that real-world problems could be solved, would be very simple. The second input vector, specifying arrival volumes, was set to the values corresponding to the volumes given in Table 1. The north and south approaches were given 5 to 6 times as much traffic as east and west directions. This situation is common in street networks where two opposing directions have considerably higher traffic volumes than the perpendic-

TABLE 1 Arrival Volumes for the Traffic GA Test Run

Approaching	Arrival Volume (cars/hour)
from North to Intersection 1	1560
from West to Intersection 1	288
from North to Intersection 2	1560
from East to Intersection 2	288
from South to Intersection 3	1560
from West to Intersection 3	288
from South to Intersection 4	1560
from East to Intersection 4	288

ular directions because of a busy central business district. For example, if this block of intersections is directly north of a shopping mall, the highest traffic volumes will occur for cars traveling south, into the mall area, and north, out of the mall area.

Traffic GA Results

To obtain a stable, unbiased, average result for this report, five Traffic GA runs with different initial GA populations were executed. Each run was executed for 60 GA generations with a GA population of 50. This means that for each run, the fitness function (the 5-min-of-traffic-time simulation) was executed 3,000 times. At each generation, the average fitness of the generation is calculated and the member in the population with the best fitness value (that is, the shortest wait time) is identified. To produce Figure 4, the average fitnesses of each GA generation (from Generation 0 to 60) of each of the five Traffic GA runs were then averaged together. This produced the "average wait time of population" line. The "minimum wait time of population" was produced in the same manner by averaging the fitnesses of the best-of-generation members for each of the five Traffic GA runs. Figure 4 shows how the GA starts with bad solutions (that is, solutions that produce on-average high wait times) and locates good solutions (that is, solutions that produce on-average short wait times).

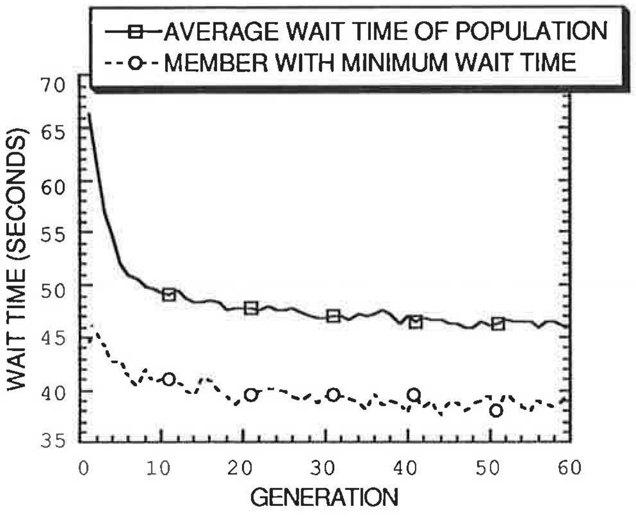

FIGURE 4 Average of five traffic GA runs [best-of-generation (minimum wait time) and generation average (average wait time) results].

The graph shows that the population seems to converge to the optimum or near-optimum member by the 20th or 30th generation. Therefore, it is possible to terminate the GA after 20 generations instead of after 60 generations and still obtain a near-optimal solution. This reduced-generation scenario would reduce the number of simulations from 3,000 to 1,000. The graph also shows that typical minimum wait time values were around 40 sec for these traffic conditions. After the last generation, which in this case was the 60th, the member with the maximum fitness (minimum wait time) can be selected as the best signal-timing configuration and called the solution from the Traffic GA. Then, if this Traffic GA run was performed using real traffic data, the solution could be used to time the real traffic signals to promote smooth traffic flow.

A typical maximal fitness member, actually found by one of the GA runs executed on the traffic environment described in Table 1, is given in Table 2. Note that for all intersections, the green phase time for the north/south (N/S) directions was considerably longer than for the east/west (E/W) directions. Observe that total cycle times are equal because the bit string contains only one field to represent total green time, but again the bit string and simulation could be easily modified to allow different total cycle times. The Traffic GA selected a total cycle time of 60 by itself; this number is not programmed into the GA. The GA found a strategy that used very similar green phase times for the north/south directions and also for the east/west directions. We expect this behavior because similar green phase times often allow the best flow of traffic because cars can move through the network with fewer stops if there is some type of synchronized cycle time (*10*). Finally, the GA could have given green phase times up to 114 sec but only went as high as 45 sec. This is because the GA was searching for a strategy that would minimize wait time, and if it were to allocate more green phase time to north and south directions, the wait times for cars coming from the east and west would increase too dramatically to make this beneficial. This solution provides a 33-sec green band for northbound traffic of Intersections 1 and 3 and another 33-sec green band for southbound traffic of Intersections 2 and 4.

Run Time

One entire GA run, which amounts to a total of 3,000 simulations and 60 generations of a GA, took 2.0358 system CPU sec on a supercomputer (Cray 2 with four processors). This is a reasonably long job time considering that this time would be greater on more readily available processors. On the other

TABLE 2 GA's Maximum Fitness (Minimum Wait Time) Timing Strategy

Intersection Number	First Direction	Green Time (sec)	Second Direction	Green Time (sec)	Total Cycle Time (sec)[a]
1	E/W	12	N/S	42	60
2	N/S	36	E/W	18	60
3	N/S	42	E/W	12	60
4	E/W	9	N/S	45	60

[a] Total Cycle Times have two yellow phases of 3 seconds each added, in addition to the two green times.

hand, if the number of generations were cut by two-thirds to 20 as suggested earlier, the CPU time would be cut by two-thirds because the simulation takes up almost all of the CPU time, whereas the GA operators use very little. Depending on how often a user wants to recalculate a signal-timing strategy and for how many intersections, the required processing time may increase or decrease. It is possible that the required computational effort could be too large, preventing use of the Traffic GA to calculate signal timings in very short time intervals.

Performance

Though this run is only one case, it is expected that the Traffic GA will always converge to a reasonable timing strategy.

Reasonable timing strategies have been found in many different cases not reported in this paper. For example, when arrival volumes were increased to a point of oversaturation, the GA responded by finding signal-timing strategies with longer cycle times, something a traffic engineer also would do if it were possible to have constant human monitoring of traffic signals.

Furthermore, most GA researchers agree that the theory of convergence for simple GAs has become fairly well developed, indicating that the performance reported earlier is typical of GA behavior. The critical components of this theory focus on building blocks (*5,12*), building block growth (*12*), the possibility of being misled by building blocks (*13–15*), and mixing and statistical decision making (*16*).

Therefore, overall Traffic GA results (including the cases not reported here) and the theory of convergence indicate that GAs may be able to solve more difficult problems than traditional control strategies and search methods. GAs seem to be better on both accuracy and convergence time. Finally, the advantages of demand-responsive control over other forms of traffic control include the capacity to constantly examine situations and respond to them with no traffic knowledge and no human attention.

CONCLUSIONS

This paper reported on an application of a genetic algorithm to produce near-optimal traffic signal-timing strategies for a network of intersections. Examples and simulation parameters were included for illustrative purposes and to demonstrate the roll a GA could play in signal-timing determination. The Traffic GA produced reasonable traffic signal-timing plans. The results suggest that this method of searching for an optimal signal-timing strategy has the potential to improve existing traffic control techniques. It is especially encouraging that the GA could find balanced conditions of green phase times and a reasonable cycle length as a function of traffic demand.

The Traffic GA produced logical signal timings using simple GA operators and a simple simulation model. Changing the GA may be warranted if this problem were scaled up to handle many more intersections. Future work on the simulation would be required to make the Traffic GA more realistic and capable of handling more complex intersection flow conditions.

Computer traffic control deserves attention because of the possible benefits from improving traffic flow. An adequate solution to this problem would increase roadway efficiency, reduce travel time, make travel time more predictable, improve safety, cut down on harmful emissions, decrease fuel consumption, and increase driver comfort.

ACKNOWLEDGMENTS

Mark Foy would like to thank Chad Hall for the initial motivation to examine traffic flow and traffic control strategies. Thanks go to the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign for providing the computer time to run the Traffic GA.

David Goldberg acknowledges support by the National Science Foundation.

REFERENCES

1. N. H. Gartner. OPAC: A Demand-Responsive Strategy for Signal Control. In *Transportation Research Record 906*, TRB, National Research Council, Washington, D.C., 1983, pp. 75–81.
2. F.-B. Lin and S. Vijayakumar. Adaptive Signal Control at Isolated Intersections. *Journal of Transportation Engineering*, Vol. 114, No. 5, Sept. 1988, pp. 555–573.
3. J. S. Linkenheld, R. F. Benekohal, and J. H. Garrett, Jr. A Knowledge-Based System for the Design of Signalized Intersections. *ASCE Journal of Transportation Engineering*, Vol. 118, No. 2, March 1992, pp. 241–257.
4. D. P. Mital. An Intelligent Urban Traffic Network Controller and Simulator. *IETE Technical Review*, Vol. 7, No. 1, Jan. 1990, pp. 52–62.
5. D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Mass., 1989.
6. *Proc., International Conference on Genetic Algorithms and Their Applications* (John J. Grefenstette, ed.). Carnegie-Mellon University, 1985.
7. *Proc., Second International Conference on Genetic Algorithms* (John J. Grefenstette, ed.). Massachusetts Institute of Technology, 1987.
8. *Proc., Third International Conference on Genetic Algorithms* (J. David Schaffer, ed.). George Mason University, 1989.
9. *TRAF-NETSIM User's Manual*. Federal Highway Administration, U.S. Department of Transportation, 1989.
10. S. Reljic. TRAFSIG: A Computer Program for Signal Settings at an Isolated, Under- or Oversaturated, Fixed-Time Controlled Intersection. *Traffic Engineering and Control*, Vol. 29, No. 11, Nov. 1988, pp. 562–566.
11. *Special Report 209: Highway Capacity Manual*. TRB, National Research Council, Washington, D.C., 1985.
12. J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.
13. D. E. Goldberg. Simple Genetic Algorithms and the Minimal, Deceptive Problem. In *Genetic Algorithms and Simulated Annealing* (L. Davis, ed.), Morgan Kaufmann, Los Altos, Calif., 1987, pp. 74–88.
14. D. E. Goldberg. Genetic Algorithms and Walsh Functions: Part I. A Gentle Introduction. *Complex Systems*, Vol. 3, No. 2, 1989, pp. 129–152.
15. D. E. Goldberg. Genetic Algorithms and Walsh Functions: Part II. Deception and its Analysis. *Complex Systems*, Vol. 3, No. 2, 1989, pp. 153–171.
16. D. E. Goldberg, K. Deb, and B. Korb. Messy Genetic Algorithms Revisited: Studies in Mixed Size and Scale. *Complex Systems*, Vol. 4, No. 4, 1990, pp. 415–444.

Publication of this paper sponsored by Committee on Traffic Flow Theory and Characteristics.