The Use of Genetic Algorithm for Traffic Light and Pedestrian Crossing Control

Ayad Mashaan Turky, Mohd Sharifuddin Ahmad and Mohd Zaliman Mohd Yusoff,

University Tenaga Nasional, Km 7, Jalan Kajang-Puchong,
43009 Kajang, Selangor, Malaysia.

Summary
The increase in urban traffic has resulted in traffic congestions, long travel times and increase hazards to pedestrians due to inefficient traffic light controls. These scenarios necessitate the use of new methods in the design of traffic light control for vehicles and pedestrian crossings.

In a conventional traffic light controller, the traffic lights change at constant cycle times which are clearly not optimal. The preset cycle time regardless of the dynamic traffic load only adds to the problem. It would be more feasible and sensible to pass more vehicles at the green interval if there are fewer vehicles waiting behind the red lights or vice versa.

We apply the genetic algorithm technology in the traffic control system and pedestrian crossing to provide intelligent green interval responses based on dynamic traffic load inputs, thereby overcoming the inefficiencies of the conventional traffic controllers. We apply such technology to a four-way, two-lane junction based on two sets of parameters: vehicles and pedestrians queues behind a red light and number of vehicles and pedestrians that passes through a green light. The algorithms dynamically optimize the red and green times to control the flow of both the vehicles and the pedestrians.

To represent a typical traffic flow system, we use the Cellular Automata for modeling vehicular motion behind the traffic lights. We developed an algorithm to model the situation of a four-way two-lane junction based on this technology.

We compare the performance between the genetic algorithms controller and a conventional fixed time controller and the results show that the genetic algorithms controller performs better than the fixed-time controller.

Key words:
Traffic light, Genetic algorithm, Pedestrian crossing, Cellular Automata

1. Introduction

The monitoring and control of vehicular traffic and pedestrians pose a major challenge to transport authorities around the world. The escalating number of vehicles in cities not only has a huge environmental impact, but also results in loss of lives on the road. This situation demands a comprehensive approach involving a system in which both the traffic controls for vehicles and pedestrians are coordinated so that road users are safe and traffic is smooth flowing.

Currently, pedestrian crossings pose a significant hazard in many countries, both in developed and developing countries due to the increase in vehicles number. Each year a staggering figure of 500,000 pedestrians are killed all over the world and in China alone from 2000-2004, half a million pedestrians were killed [1].

The European Transport Safety Council (ETSC) claims that 15 to 30 percent of the transportation mode used is walking. According to a telephone survey conducted by the Royal Automobile Club of Spain in the year 2000, walking is highly recommended as part of a healthy lifestyle with no negative side effects. However, it has been the victim of badly controlled traffic, thus increasing the mortality rates of road users. In the large cities of Europe, especially in Spain, people walked to their destinations but this is being seen as dangerous as pedestrians are more vulnerable to road accidents than passengers and drivers of cars [2].

In a conventional traffic light controller, the traffic lights change at constant cycle times which is clearly not the optimal solution. The system calculates the cycle time based on average traffic load and disregards the dynamic nature of the traffic load, which aggravates the problem of congestion. Consequently, we see an urgent need to optimize traffic control algorithms to accommodate the increase in vehicles in urban traffic that experience long travel times due to inefficient traffic light controls and to improve pedestrian's safety.

In this paper, we propose an optimal control of traffic lights using genetic algorithm (GA), in a four-way, two-lane junction with a pedestrian crossing. The innovative design of the pedestrian crossing is also based on such algorithm, which includes pedestrians as one of the parameters. Genetic algorithm is introduced in the traffic control system to provide an intelligent green interval response based on dynamic traffic load inputs, thereby

Manuscript received February 5, 2009
Manuscript revised February 20, 2009

overcoming the inefficiencies of conventional traffic controllers. In this way, the challenges are resolved as the number of vehicles are read from sensors put at every lane in a four-way, two-lane junction and pedestrianss are monitored at the road junction.

The features inherent in genetic algorithm play a critical role in making them the best choice for practical applications, namely optimization, computer aided design, scheduling, economics and game theory. It is also selected because it does not require the presence of supervisor or observer.

However, genetic algorithm, without prior training, continuously allow permanent renewal of decisions in generating solutions. Instead of trying to optimize a single solution, they work with a population of candidate solutions that are encoded as chromosomes. Within these chromosomes are separate genes that represent the independent variables for the problem at hand.

There are a number of specific attributes of genetic algorithms that give them an edge over other traditional optimization techniques. These are:

- A genetic algorithm works from a population, not a single point, and hence it is less likely to be trapped at a local optimum.
- Derivative freeness, i.e. a genetic algorithm does not need the objective function's derivative to do its work.
- Flexibility, i.e., a genetic algorithm can function just fine regardless of how complex the objective function is; the only thing it requires of the function is that it is executable (i.e., its value can be calculated given the values of the decision variables).
- Because of its implicit parallelism, a genetic algorithm can handle combinatorial problems efficiently. It has been shown that as the size of the search space or number of solutions increases exponentially, the time requirements for the GA to reach a solutions only grow linearly. This feature is particularly useful for on-line optimization of transportation problems such as traffic control.
- A genetic algorithm naturally lends itself to parallel implementation. This follows from its functional components structure.
- Genetic algorithm, is, for the most part, based on intuitive notions and concepts.

Our preliminary review of the literature indicates that genetic algorithm has not been tested on pedestrian crossings. We have, therefore, attempted to implement this algorithm and study its effects on this problem.

2. Related Work

The first known attempt to apply fuzzy logic in traffic control was made by Pappis, and Mamdani [5]. They simulated an isolated signalized intersection composed of two one-way streets with two lanes in each direction without turning traffic. The fuzzy controller reduced average vehicle delay compared to an actuated controller.

Tan, Khalid and Yusof [3] describe a fuzzy logic controller for a single junction that mimics human intelligence. They used two sensors for each lane. The first sensor behind each traffic light counts the number of cars passing the traffic lights, and the second sensor behind the first sensor counts the number of cars coming to the intersection at distance from the lights. The fuzzy logic controller determines the time that the traffic light should stay in a certain state, before switching to the next state. The order of states is predetermined, but the controller can skip a state if there is no traffic in a certain direction. The amount of arriving and waiting vehicles are quantized into fuzzy variables, like many, medium and none. The activation of the variables in a certain situation is given by a membership function, so when there are 5 cars in the queue, this might result in an activation of 25% of 'many' and 75% of 'medium.' In their experiments, the fuzzy logic controller showed to be more flexible than fixed controllers and vehicle actuated controllers, allowing traffic to flow more smoothly, and reducing waiting time.

Foy et al. [4], documents the use of genetic algorithm to optimize timing plans. The application object is an octothorpe-shaped traffic network with four intersections. Every intersection can run a two-phase plan. This method uses nine decision variables including the total green time of all phases, phase orders and splits. These nine decision variables are coded with 24 bits. The objective function is the reciprocal of the total waiting time. A simulation model is used to evaluate the optimizing method. Results show that genetic algorithm is indeed a parallel optimizing method compared with traditional search methods.

Zhiyong et al. [6] propose an improved immunity genetic algorithms for an urban area coordinated traffic control system. The system adopts a two-level hierarchical distributed construction, with parameters that are hierarchically optimized at an interval of 5-30 minutes. In each interval, the cycles and offsets are optimized in the central controller while the splits are optimized in intersection controller. For a given performance index, such as minimizing the mean vehicle delay or number of stops etc., an improved immunity GA is used to optimize the cycle, offsets and splits. To ensure that the proposed method is plausible, simulations were conducted with positive results.

In [7] a mutation operation of genetic algorithm has a logistic chaotic mapping applied upon and then a chaotic mutation is implemented, thus building a chaotic genetic algorithm. From this population, 5% of the individuals are selected randomly to search according to chaotic dynamic searching process. The results of this search replace the previous individuals who were part of this population. It shows that the new algorithm converges more quickly and avoids local optimization and premature convergence when simulated in CORSIM (CORridor SIMulation).

3. Model Design for Traffic Light

In our design, we simulate five sensors; each sensor detects the number of vehicles for each lane. The fifth sensor detects the pedestrian queue. The system calculates the green and red light times to be given for vehicles and the vehicles queue length behind the red light plus the time taken for each vehicle to arrive at its target destination in static and dynamic modes, i.e., if vehicle V comes from lane X goes to a destination in lane Y, the system calculates the time that it takes to travel from X to Y.

3.1 Variables

We define the input variables as follows:

1. Vehicles Passing, VP: the Number of Vehicles that pass through a green light.
2. Pedestrians Passing, PP: the Number of Pedestrians that pass through a green light.
3. Vehicles Queue, VQ: the Number of Vehicles behind a red light.
4. Pedestrians Queue, PQ: the Number of Pedestrians behind a red light.

The variables VP and PP are required to calculate the queue length behind a red light, whereas the variables VQ and PQ are used to calculate the green time in the next cycle.

We define the output variables as follows:

1. Queue of Vehicles, QR_V: The Number of Vehicles behind the red light per second in static and dynamic modes.
2. Queue of Pedestrians, QR_P: The Number of Pedestrians behind the red light per second in static and dynamic modes.
3. The Duration, D: The time it takes for a vehicle to travel from a source to a target destination in static and dynamic modes.

The variables QR_V, QR_P and D are required to check our dynamic model's performance by comparing it with the static model.

3.2 Cellular Automata

One way of designing and simulating (simple) driving rules of cars is by using cellular automata (CA). CA use discrete partially connected cells that can be in a specific state. For example, a road-cell can contain a car or is empty. Local transition rules determine the dynamics of the system and even simple rules can lead to chaotic dynamics [8].

We use cellular automata algorithm in this paper because it allows us to represent significant events that occur during congestions such as traffic standstill, resume motion, return to standstill again, and so on. In the model, we identify a vehicle's basic attributes that include medium speed, maximum speed, vehicle location, desired speed, current acceleration, and vehicle unique identification number.

We identify and define the parameters of the model's entities, i.e. the vehicle and lane as follows:

1. Length of lane: Following from the fundamental concept of CA, each lane is divided into a number of cells and each lane has a unique ID number. We define the length of a lane in cell number and set the number of cells for each lane to 15. The size of a cell is 16 pixels and a vehicle occupies two cells (32 pixels). We use an array to represent the cells in each lane and the array is identified with a unique ID number. This enables the program to check if the cell is empty or occupied and to detect the vehicles actual position.

   We set a vehicle's width to 30 pixels and its height to 16 pixels. All vehicles have the same size with four different colors. Each vehicle is identified by unique ID number and has a known fix travel route that is determined by the source and destination points. Figure 1 shows a typical configuration of a lane created with Cellular Automata. In our model, we have eight such lanes.

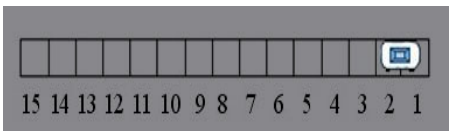

Figure 1: A Typical Lane of the Model

2. Number of Vehicles: This is the number of vehicles in a lane.

3. Vehicles Speed: The number of cells traveled by a vehicle in a given time. The value of this speed ranges between zero and the maximum value. The maximum speed is 5 cells per second.

4. Traffic Signals: As clearly shown in Figure 2, our model has five traffic lights, four for traffic control and one for pedestrian crossing. Each traffic light has an individual ID for identification, and has three signal modes, red, yellow and green. We use an array to store the set times for the three signal modes.

Our four-way, two-lane junction (hence eight lanes), has an entry node and an exit node for each lane and one intersection node (junction). The intersection node consists of four cells and each node has an individual ID. We set an array for each node, the size of which depends on the number of cells in the node. The pedestrian crossing area is set across the lanes B and B1.

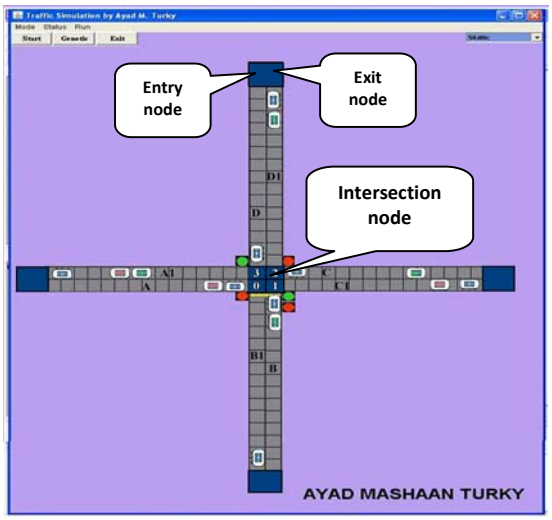

Figure 2: The Traffic Model

As apparent in Figure 2, each vehicle can either move forward, stay at its current position or turn left or right. A typical algorithm for the Cellular Automata implemented for our model is as follows:

1. **Rule1**: check decision-point
2. If decision-point is passed
3. Then go to Rule3
4. Else go to Rule2
5. **Rule2**: check cell type
6. If cell type='decision'
7. Then examine the behavior of the vehicles, which direction the Vehicle take, followed by a turn in that direction; decision point will be passed
8. Else go to Rule3
9. **Rule3**: check cell
10. If cell isn't occupied by a vehicle and cell type = 'empty' or cell type = 'decision'
11. Then Move
12. Else go to Rule4
13. **Rule4**: check adjacent cells
14. If the cell over to the left (right) isn't occupied by a vehicles
15. Then Move to the left (right) cell
16. Else Wait

3.3 The Model's Algorithm

In our algorithms, we establish the following algorithmic steps: initialize population, evaluate population, chromosomes selection and chromosomes recombination.

1. **Initialize population:** Each chromosome contains two genes, the first gene is red time, RT, and the other one is green time, GT. We set the chromosomes population to 100. Chromosomes need to be encoded to represent the problem that genetic algorithm is meant to resolve. In our algorithm, we use binary encoding to encode the chromosomes. In this encoding technique, every chromosome is a string of bits 0 or 1. This gives many possible chromosomes, even with a small number of alleles. See Figure 3 for an example of chromosomes with binary encoding.

Chromosome A	101100101100101011100101
Chromosome B	111111100000110000011111

Fig. 3 Binary Encoding

2. **Evaluate population:** This provides a way to rate how each chromosome (candidate solution) solve the problem at hand. It involves decoding the chromosomes into the variable space of the problem and then checking the result of the problem using these parameters. The fitness is then computed from the result.

   Crossover Fraction: With the crossover fraction=0.8, we used two point crossover operation performed on the parent's generation, the result of which is stored in a mean array. In this array, the parent's generation is merged with the children. These steps are repeated until the total number of the crossover operation is half the size of the initialization. We can then say that the crossover operation is completed.

Mutation Fraction: With the mutation fraction=0.2, we performed this operation on the parent's generation. From the results in the mean array, a random number is generated and the result of comparison between this number and mutation fraction are determined by the occurrence or non-occurrence of mutations. These steps are repeated until the total number of mutation operations is half the size of the initialization. We can then say that the mutation operation is completed.

3. **Chromosome selection:** The chromosomes are selected for propagation to future populations based upon their fitness. Chromosomes which have high fitness value have a good chance to be chosen for future population. For selection of chromosomes, we use the "Roulette-wheel with probability of selection that is proportional to fitness" based upon the fitness of the chromosomes. See Figure 4.

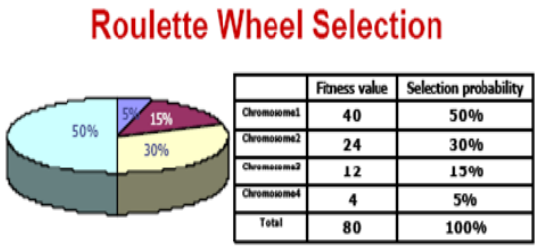

Fig. 4: Roulette Wheel Selection

The fitness function is then computed from the result. The algorithm determines the fitness function to identify the solutions. It computes the fitness function based on many parameters (queue, density, green and red light times). The fitness function consists of two parts:

(i) The algorithm calculates the green times, GT_V for vehicles, and GT_P for pedestrians, due to the queue formed behind a red light,

$$GT_V = (VQ * \text{Time for Passing}) \quad \text{(i)}$$

$$GT_P = (PQ * \text{Time for Passing}) \quad \text{(ii)}$$

where, VQ is the number of vehicles behind the red light, and Time for Passing is the time required for a vehicle and a pedestrian to pass a green light. We set the Time for Passing to 3 seconds for both vehicles and pedestrians. We compare this value with past green times to obtain a good value for the green times, GT_V and GT_P.

(ii) In the same way, the algorithm calculates the length of queue for vehicles, VQ, and pedestrians, PQ, which forms during the red time, i.e.

$$VQ = (VP * RT_V * V_{AVG}) \quad \text{(iii)}$$

$$PQ = (PP * RT_P * P_{AVG}) \quad \text{(iv)}$$

where, VQ and PQ are the number of vehicles and pedestrians respectively, and RT_V and RT_P are the red times for vehicles and pedestrians respectively. V_{AVG} is the average arrival speed of a vehicle to the junction at the red light and P_{AVG} is the rate of arrival of pedestrians at the red light. We set V_{AVG} to 3 cells per sec. and P_{AVG} to 1 pedestrian per sec.

The quality of performance increases whenever this value, VQ or PQ, decreases, i.e. when there are fewer vehicles or pedestrians behind a red light, the traffic control system performance improves. We give greater attention to optimize the green time at the expense of queue length. Therefore, we multiply a ruling parameter and give the priority for vehicles without being unfair to pedestrians. Thus, the fitness function changes as follows:

$$(XP * RT_X * X_{AVG})\text{^}3 - (GT_X - XQ * \text{Time for Passing})\text{^}2$$

where,

XP = VP or PP, the no. of vehicles or pedestrians that passes a green light,
RT_X = RT_V or RT_P, the red times for vehicles or pedestrians,
X_{AVG} = V_{AVG} or P_{AVG}, the average arrival rate to the red light,
GT_X = GT_V or GT_P, the green times for vehicles or pedestrians,
XQ = VQ or PQ, the number of vehicles or pedestrians behind a red light.

4. **Chromosome recombination:** In recombination, pairs of chromosomes are recombined, possibly modified, and then placed back into the population as the next generation. The process continues again at evaluation until the problem represented by the chromosomes is solved, or some other exit criterion is met such as convergence, or the maximum number of generations is reached.

The next step in the operation is evaluating the generation to determine the resulting quality of these individuals compared with the previous generation. This is done by arranging the elements of the array (mean array) in increasing values provided by the fitness function.

Ordering the array elements in this way contributes to better identification of individual generations (parent and child generations). The first set of the elements of this array (mean array) is copied to the parent's array. These elements from 70% of the members of the new generation of parents. The rest (30%) is generated by using a random function. The algorithms read the new inputs after five generations to get good solutions.

4. Comparisons between Static and Dynamic Control Modes

We compare the performance between the static control (fixed cycle time) and the dynamic control (genetic algorithm) modes. To test our model, we use the same input for both modes. In the static mode, we set the green and the red time for the vehicles and pedestrians to 20 seconds for each lane. We set different variable values for vehicle and pedestrian loads (Low, Medium and High) for each mode. However, due to the large number of traffic situations, we only conduct tests for the worst case scenarios as shown in Table 1, i.e. for both the vehicle and pedestrian loads at high value (Vehicle=7, Pedestrian=7).

Table 1: Values of Variables for Different Test Scenarios

Test No.	Mode	Test Scenarios for Simulation
Test I	Static	$GT_V = GT_P = 20$ s; $RT_V = RT_P = 20$ s Vehicle Load = High Pedestrian Load = High
Test II	Dynamic	Vehicle Load = High Pedestrian Load = High

We do not conduct tests for the Low and Medium vehicle and pedestrian loads on the assumption that the test results would be better than the worst case scenarios of Table 1.

For each test, the results show the output values of the following variables for both static and dynamic modes:

- The Duration, D: The time it takes for a vehicle to travel from a source to a target destination,
- Queue of Vehicles, QR_V: the number of vehicles behind the red light per second,
- Queue of Pedestrians, QR_P: the number of pedestrians behind the red light per second.

In the dynamic control mode, the times determined by genetic algorithm depend on four parameters:

- the number of vehicles passing a green light, VP,
- the number of pedestrians passing a green light, PP,
- vehicles queue, VQ,
- pedestrians queue, PQ.

The algorithm processes these parameters resulting in synchronized green and red times for the test scenarios. If there are no vehicle and pedestrian queues in one lane, the green time will be zero.

4.1 The Duration, D

Table 2 and Table 3 show the results of the durations of vehicles traveling from a source to a target destination. We present two scenarios in each mode.

Table 2: Duration of Traveling Time for Static Mode

V-ID	Start Time	Arrival Time	Duration (s)
1	18:44:32	18:44:38	6
3	18:44:32	18:44:38	6
7	18:44:34	18:44:41	7
10	18:44:38	18:44:45	7
11	18:44:39	18:44:46	7
4	18:44:32	18:44:48	16
2	18:44:32	18:44:48	16
6	18:44:34	18:44:49	15
5	18:44:33	18:44:49	16
9	18:44:38	18:44:50	12
8	18:44:38	18:44:51	13
16	18:44:44	18:44:51	7
12	18:44:41	18:44:53	12
14	18:44:42	18:44:54	12
18	18:44:47	18:44:54	7
20	18:44:49	18:44:56	7
22	18:44:50	18:44:56	6
13	**18:44:41**	**18:44:59**	**18**
19	18:44:48	18:45:00	12
21	18:44:50	18:45:00	10
15	**18:44:42**	**18:45:02**	**20**
17	18:44:44	18:45:02	18
23	18:44:50	18:45:03	13
24	18:44:50	18:45:04	14

Result 1: See Table 2
(Static Mode, Vehicle ID 15, Vehicle ID 13):

- Scenario 1: Vehicle ID 15 takes 20 seconds to travel from lane A to lane D1.
- Scenario 2: Vehicle ID 13 takes 18 seconds to travel from lane C to lane D1.

Result 2: See Table 3
(Dynamic Mode, Vehicle ID 15, Vehicle ID 13):

- Scenario 1: Vehicle ID 15 takes 8 seconds to travel from lane A to lane D1.
- Scenario 2: Vehicle ID 13 takes 9 seconds to travel from lane C to lane D1.

The results show that the dynamic (genetic) mode performs better than the fixed time (static) mode.

Table 3: Duration of Traveling Time for Dynamic Mode

V-ID	Start Time	Arrival Time	Duration (s)
1	18:46:42	18:46:47	5
3	18:46:42	18:46:47	5
2	18:46:42	18:46:49	7
4	18:46:42	18:46:49	7
6	18:46:44	18:46:50	6
5	18:46:42	18:46:50	8
7	18:46:44	18:46:51	7
8	18:46:47	18:46:53	6
9	18:46:47	18:46:54	7
10	18:46:47	18:46:55	8
11	18:46:49	18:46:56	7
12	18:46:50	18:46:57	7
14	18:46:51	18:46:58	7
13	**18:46:50**	**18:46:59**	**9**
15	**18:46:52**	**18:47:00**	**8**
17	18:46:53	18:47:01	8
16	18:46:53	18:47:02	9
18	18:46:56	18:47:02	6
23	18:46:59	18:47:05	6
24	18:46:59	18:47:05	6

4.2 The Vehicles Queue, QR_V

Table 4 shows the results of the rate of vehicle queues behind a red light. The results show that the dynamic model performs better than the static model. We present one scenario for the vehicle queues for each mode:

Table 4: Static and Dynamic Queue (Vehicles)

Time (Sec)	No. of Vehicles behind a Red light	
	Static	Dynamic
1	0	0
2	0	0
3	0	0
4	3	3
5	4	0
6	6	1
7	6	1
8	**7**	**1**
9	2	2
10	3	1
11	3	1
12	3	1
13	4	1
14	5	2
15	7	2
16	0	0
17	0	0
18	0	0
19	0	0
20	1	1

Result 3: See Table 4
(Vehicle Queue, Static Mode)

- Scenario 1: The number of vehicles (Vehicles Queue) behind the red light per second at the 8th second is 7 vehicles.

Result 4: See Table 4
(Vehicle Queue, Dynamic Mode)

- Scenario 2: In the same second the number of vehicles (Vehicles Queue) behind the red light is 1 vehicle.

The results show that the dynamic (genetic) mode performs better than the fixed time (static) mode. Figure 5 shows the contrasting results of the Vehicles Queue between the static and dynamic modes in graphical format.

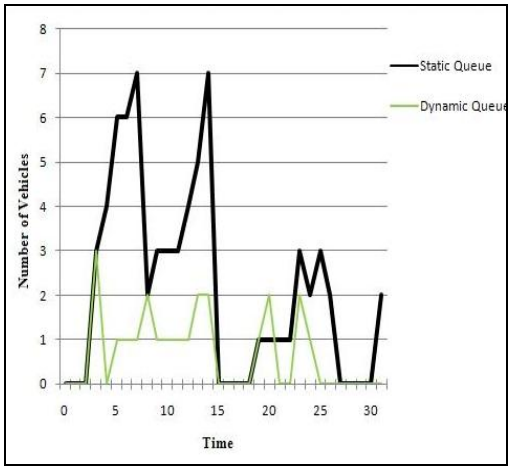

Figure 5: Vehicles Queue behind a Red Light

4.3 The Pedestrians Queue, QRP

Table 5 shows the results of the pedestrian queues behind a red light. The results show that the dynamic (genetic) mode performs better than the fixed time (static) mode. We present one scenario for the pedestrian queues for each mode:

Result 5: See Table 5
(Pedestrian Queue, Static Mode)

- Scenario 1: The number of pedestrians (Pedestrian Queue) behind the red light per second at the 8th second is 9 pedestrians.

Result 6: See Table 5
(Pedestrian Queue, Dynamic Mode)

- Scenario 2:In the same second the number of pedestrians (Pedestrian Queue) behind the red light is 3 pedestrians.

The results show that the dynamic (genetic) mode performs better than the fixed time (static) mode.

Table 5: Static and Dynamic Queue (Pedestrians)

Time (Sec)	No. of Pedestrians behind a Red light	
	Static	Dynamic
1	1	1
2	1	1
3	1	1
4	3	3
5	5	1
6	7	0
7	7	0
8	**9**	**3**
9	0	0
10	0	1
11	0	1
12	0	0
13	0	0
14	0	0
15	0	3
16	1	0
17	0	0
18	0	0
19	0	0
20	0	3

Figure 6 shows the contrasting results of the Pedestrian Queue between the static and dynamic modes in graphical format.

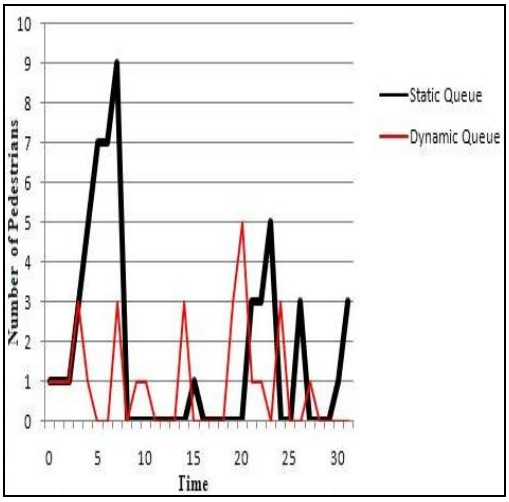

Figure 6: Pedestrian Queue behind a Red Light

5. Conclusions and Further Work

From the results, we can conclude that the dynamic control model performs better than the static control model. Due to its flexibility, the dynamic control model is able to calculate the optimal green time based on the number of vehicles and pedestrians behind a red light and the vehicles and pedestrians queue lengths. The results also show that significant time gain is experienced for a vehicle traveling through the GA-controlled traffic light system. In our further work, we will extend the application of our algorithms to include two or more similar junctions connected to our traffic model.

References

[1] Zhen Liu, Simulation of Pedestrians in Computer Animation, in Proceedings of ICICIC (2) 2006. pp. 229~232.
[2] European Transport Safety Council (ETSC), http://www.etsc.be/stats3.ppt.
[3] Kok Khiang Tan, Marzuki Khalid and Rubiyah Yusof, Intelligent Traffic Lights Control by Fuzzy Logic, Malaysian Journal of Computer Science, Vol. 9 No. 2, December 1996, pp. 29~35.
[4] Foy, M. D. et al., Signal Timing Determination Using Genetic Algorithms. Transportation Research. Record 1365, National Research Council, Washington, D.C., 1992, pp. 108~115.
[5] Pappis, C. P., and E. H. Mamdani, A Fuzzy Logic Controller for a Traffic Junction, IEEE Transactions Systems, Man, and Cybernetics, Vol. SMC-7, No. 10, October 1977, pp. 707~717.
[6] Liu Zhiyong, et al., Immunity genetic algorithms based adaptive control method for urban traffic network signal. Control Theory & Applications, 2006, 23(1): pp. 119~125.
[7] Dong Caojun, et al., Area Traffic Signal Timing Optimization Based on Chaotic and Genetic Algorithm Approach. Computer Engineering and Applications, 2004,40(29): pp. 32-34, 138.
[8] Nagel, K., Schreckenberg, M.: A cellular automaton model for freeway traffic. J. Phys., 1992(I-2):2221-2229.

Ayad M. Turky received his B.Sc. Degree in Computer Science from the College of Computer, University of Al-Anbar, in 2006. He worked as a Technical Support Assistant at the College of Dentistry, University of Al-Anbar, Iraq. Currently, he is enrolled in the Master of Information Technology program at the College of Graduate Studies, Universiti Tenaga Nasional (UNITEN), Malaysia. He has also worked as a Research Assistant at the College of Information Technology, UNITEN in the area of Image Processing. Being a graduate student he has also conducted additional laboratory work for the degree and foundation programs at the College of Information Technology. His research interests include Artificial Intelligence, Evolutionary Computing and Image Processing.

Mohd S. Ahmad received his B.Sc. in Electrical and Electronic Engineering from Brighton Polytechnic, UK in 1980. He started his career as a power plant engineer specialising in Instrumentation and Process Control in 1980. After completing his MSc in Artificial Intelligence from Cranfield University, UK in 1995, he joined UNITEN as a Principal Lecturer and Head of Dept. of Computer Science and Information Technology. He obtained his PhD from Imperial

College, London, UK in 2005. He has been an associate professor at UNITEN since 2006. His research interests includes applying constraints to develop collaborative frameworks in multi-agent systems, collaborative interactions in multi-agent systems and tacit knowledge management using AI techniques.

Mohd Z. M. Yusoff obtained his B.Sc. and MSC in Computer Science from Universiti Kebangsaan Malaysia in 1996 and 1998 respectively. He started his career as a Lecturer at UNITEN in 1998 and has been appointed as a Principle Lecturer at UNITEN since 2008. He has produced and presented more than 40 papers for local and international conferences. His research interest includes modeling and applying emotions in various domains including educational systems and software agents, modeling trust in computer forensic and integrating agent in knowledge discovery system.