Comparative analysis of urban traffic control algorithms

Robert, Varga - Technical University of Cluj Napoca,
supervisor Mihai, Hulea

***Abstract*-The current paper proposes to analyze different control algorithms for optimizing urban traffic control. The aim of the applied algorithms is to minimize queue length. Firstly fixed phase times are allocated, secondly phase times are dynamically changed, thirdly a dynamic algorithm with "interrupts" is used and lastly a centralized fixed time strategy using genetic algorithms is implemented. These are compared using test results from simulation.**

I. INTRODUCTION

Automatic traffic control is employed in almost all big cities in these days. The first systems of this type were implemented in the '60s. In 1987 the United States and Canada had 300 functional systems which controlled 20000 traffic lights [1].

The goal of using such systems is to optimize traffic flow in big cities. Using specific strategies overall traffic delays can be reduced by up to 10-40%. Cases of traffic congestions (jams) can also be reduced. These, if not handled, produce the degradation of the traffic system infrastructure, cause significant delays and are a source of pollution.

Using automated control systems that command traffic lights is a solution to these problems that is also efficient from the perspective of financial investment. Statistics show that installation and maintenance costs are returned in the first year of functioning.[2]

II. PROBLEM STATEMENT

Given a system of 4 intersections placed in the manner presented in figure 1 it is required to minimize the queue length for any given input for the system. Input is given by specifying the flow in vehicles/minute for every 8 input bands. Input flow is randomly chosen from the interval:

$$X_i \in \left[X_{given_i}(1-d), X_{given_i}(1+d) \right], i \in 1,8 \quad (1)$$

where d represents the deviance (in percents) and X is the input vector containing entries for input flows of system inputs in counter-clockwise order.

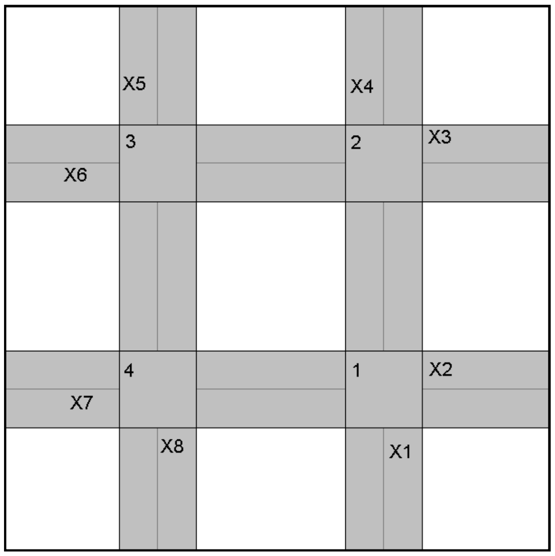


Figure 1. System of controlled intersections

The algorithms used for solving the optimization problem have knowledge of the queue length at any given moment for any band at any intersection.

An auto vehicle if it is granted permission to pass may choose randomly with a certain probability (by default 30%) to turn right or to head straight ahead (remaining 70%). This simplification enables phase time allocation algorithms that used two phases. Phases allow traffic flow from opposing bands simultaneously. Every street has a single band.

III. STATE OF THE ART

One of the most often utilized strategies today is TRANSYT, developed by Robertson. It models the traffic system using nodes that represent intersection and arcs that correspond to roads. Based on input data the algorithm does a heuristic "hill-climbing" based on a performance indicator to find the (local) minim/maxim.[3]

A coordinated dynamic strategy that uses a mathematical model that is similar to the one treated in this paper is called the "Store-and-Forward" method. The system is decomposed in traffic cells which are described in a linear state-space:

$$x(k+1) = x(k) + B\Delta g(k) + D\Delta d(k) \quad (2)$$

in which x maintains the number of vehicles in the cell, B and D are matrices with constant elements, g – green time vector, d – perturbation vector [1].

The use of genetic algorithms for solving time allocation problems has been treated in other papers. Reference [4] deals with dynamic intersection signal control optimization (DISCO) for a system of intersection from Hong Kong. Three strategies are tested: fixed cycle time and fixed phases; fixed cycle time, variable phases; no cycle, variable phase lengths. Superior results are obtained in comparison with the system in use even in case of the weakest algorithm that uses fixed phase times.

IV. Algorithms used

The mathematical model used for traffic cells can be described by the following equation in discrete time steps k:

$$x_i(k+1) = x_i(k) + in_i(k) + \sum out_j(k-\tau) - out_i(k) \qquad (3)$$

where x is a traffic cell whose state represents the number of vehicles in the cell; *in(k)* is the input flow function that has the value 1 with probability $X_i(k)\cdot \frac{1}{4\cdot 60}$. This produces on average X_i vehicle arrivals over 240 time steps equivalent to one minute; out(k) is the output function that has the value 1 if traffic flow is enabled and 0 otherwise. The summation represents vehicles arriving from other intersection after τ time steps.

The probabilistic input function enables the simulation of random arrival events.

1. Static (offline)

Using the input flows supplied for the system the controllers placed at intersections allocate static phase times calculated using the following relations:

$$S_{i0} = mask(2i-1) * X$$
$$S_{i1} = mask(2i) * X$$
$$S_i = S_{i0} + S_{i1}$$
$$t_i = \frac{S_{i0}}{S_i}C; t'_i = \frac{S_{i1}}{S_i}C; i \in 1,4 \qquad (4)$$

Phase *t1* signifies traffic flow enabled on horizontal streets at intersection 1, *t'1* represents vertical flow enabled, *C* is cycle time.

The operator "*" signifies multiplication of elements at the same position and a summation of the resultant elements. The mask provides the weights of the inputs that contribute to phase time length and is based on the car flux that passes the intersection in the direction opened during the respective phase. For instance car flow in horizontal direction at intersection 1 is given by input flow X2 + (1-p)X7 + pX8 – the latter part is due to those who chose at intersection 4 to travel to intersection 1 (see Table I, row 1).

Table I contains 8 values per row for every input grouped in pairs separated by commas in each cell for the phases indicated in the first column.

TABLE I
MASK VALUES

Phase	values			
t1	0,1	0,0	0,0	1-p,p
t'1	1,0	0,1-p	0,(1-p)p	0,(1-p)p2
t2	0,(1-p)p2	1,0	0,0	0,(1-p)p
t'2	1-p,p	0,1	0,0	0,0
t3	0,0	1-p,p	0,1	0,0
t'3	0,(1-p)p	0,(1-p)p2	1,0	0,1-p
t4	0,1-p	0,(1-p)p	0,(1-p)p2	1,0
t'4	0,0	0,0	1-p,p	0,1

Offset values are fixed to 2 seconds (equal to the travel time between neighboring intersections, to produce a "green wave") for intersections 2 and 4; and set to 4 seconds for intersection 3. Intersection 1 is considered to be reference (offset 0).

2. Dynamic – (online)

Cycle time is fixed and specified from the beginning. Phase times are reallocated at every cycle start in function of the queue length at moment of decision from the current intersection. The decision is made locally without knowledge of neighboring queue length values.

Relations used are as follows:

$$t_i = \frac{H_i}{H_i + V_i}C; t'_i = \frac{V_i}{H_i + V_i}C; \quad ,i \in 1,4 \qquad (5)$$

where Hi – sum of all queue lengths for auto vehicles waiting on horizontal streets at the moment of calculation (new cycle start); Vi – the same for vertical streets; C – fixed cycle time.

This enables phase time allocation that is independent of other intersections. For steady input flows phase time lengths tend to stabilize thus giving an approximate static allocation.

The same offset configuration is used as in the previous algorithm.

3. Dynamic – with interrupts

No fixed cycle time is used. Specific conditions are checked periodically (250 milliseconds). If these are met an "interrupt" occurs, meaning that the current phase ends. There are imposed minimal and maximal phase lengths in order to prevent deadlock and/or starvation. This algorithm uses information from neighboring intersections – is collaborative.

Figure 2 depicts a single intersection to explain the conditions. In this figure, notations and the conditions are dependent on intersection number (*nr*) and so they can be rotated to produce conditions for all 4 intersections.

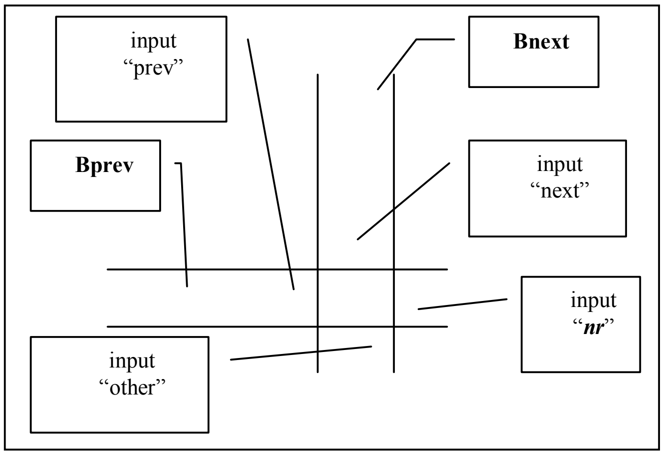


Figure 2. Input bands and auto vehicles waiting at neighboring intersections

The logical conditions checked for this specific configuration are (& - logic and, | - logic or):

- to interrupt vertical flow:

```
Bnext > limit & Bnext > cars[next] |
cars[prev] > limit & cars[prev]>cars[next] |
cars[nr] > limit & cars[nr] > cars[next]
        & cars[next] < limit & Bprev < limit
```

- to interrupt horizontal flow:

```
Bprev > limit & Bprev > cars[prev] |
cars[next] > limit & cars[next]>cars[prev] |
cars[other] > limit & cars[other] > cars[prev]
        & cars[prev] < limit & Bnext < limit
```

where cars["x"] represents the queue length at entrance "x" at the moment of evaluation (dynamically updated).

The algorithm prevents the accumulation of auto vehicles past a fixed number limit (the value for this has to be smaller than the maximum number of auto vehicles permitted to wait in order to permit decisions to be taken in time).

No offset strategy is used because in this case the cycle time is variable.

4. Genetic

Cycle time and phase time is allocated once (fixed cycle and phase time) and are calculated centrally. A population size of 1000 is used and a high mutation rate of 90%. Chromosome structure is presented in the following table (Table II).

TABLE II
CHROMOSOME GENE STRUCTURE

Bits	Value represented
0-6	Cycle time (0-127 in seconds)
7-13	Phase 1 length intersection 1
14-20	Offset for intersection 2
21-27	Phase 1 length intersection 2
28-34	Offset for intersection 3
35-41	Phase 1 length intersection 3
42-48	Offset for intersection 4
49-55	Phase 1 length intersection 4

Offsets are set to values retrieved from chromosomes in modulo of the cycle time. Note: Intersection 1 is considered to be the reference for offsets (has offset 0).

$$offset_i = offset_{chr_i} \mod C \quad ,i \in 1,4 \tag{6}$$

Phase times are recalculated to fall between minimum and maximum values. Phase 2 length for every intersection is simply the difference between the cycle time and phase one.

$$t_i = t_{\min} + \frac{t_{\max} - t_{\min}}{127} t_{chr_i}; \quad t'_i = C - t_i \quad ,i \in 1,4 \tag{7}$$

Fitness function calculation entails a simulation of 1000 seconds length. When manipulating vehicle numbers instead of incrementing/decrementing with a certain probability a rational number is added/subtracted that is proportional to the probability. The maximum queue length is retained (b_{max}) and the fitness becomes:

$$f_i = \frac{300}{b_{\max}} \tag{8}$$

for all chromosomes in the population.

Simulation is interrupted if queue length exceeds a given a limit (100 chosen) to save calculation time.

To enhance the convergence of the algorithm search plane smoothing is applied. New fitness values will be (applied to all the population):

$$f'_i = \begin{cases} \overline{f} + \left(f_i - \overline{f}\right)^{\alpha}, if\ f_i > \overline{f} \\ \overline{f} - \left(\overline{f} - f_i\right)^{\alpha}, if\ f_i < \overline{f} \end{cases} \tag{9}$$

Starting value for α is 5 and it is decremented with 0.5 on every iteration until it becomes 1. [5]

The figure from attachment 2 shows the evolution of maximum fitness values from the whole population using different kinds of genetic algorithms. Classical genetic algorithm uses roulette-wheel type selection and doesn't have search-space smoothing. It can be seen that by using smoothing a better result is obtained. By choosing a tournament type selection - grouping chromosomes in pairs - assures a steady increase of fitness values (see attachment 1) but for harder problems fails to reach near-optimal solution.

V. Application Architecture

Simulation is realized using an application developed using Java language that incorporates both the system of controllers and the simulator with a rudimentary view of the current system state.

The classes implementing the simulator are grouped in a single packet so it can be used with different packets that contain code for the controllers and the algorithms used. Note: Different algorithms imply different controller structure because of different phase time allocation strategies.

Controller class implements all controller code, controllers responsible for different intersections are recognized by their assigned number which is identical to intersection number. Calculations and algorithms are a function of this number enabling high code reusability.

Values, constants, probabilities, input values etc. are stored in the class Data and are accessible from every class. Classes refer to these values so changing system behavior is simple and can be done by changing only the value from the Data class.

Communication is peer-to-peer, it is realized using low level programming on sockets. The queue length at every moment is sent as a packet to the simulator and to neighboring controllers (if it is the case).

An overview of system architecture may be seen in Figure 3:

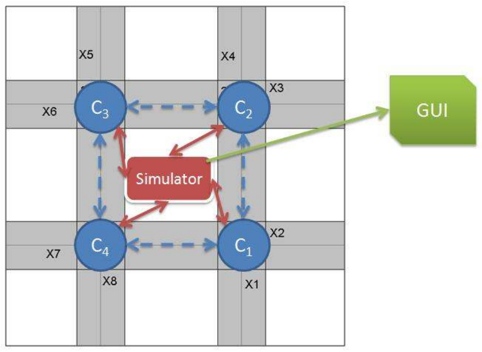


Figure 3. System architecture overview

VI. Test Results

Testing was realized using an application developed based on the system architecture presented before. The application generates text files with data that is submitted to analysis after the simulation is done.

Three input configurations were selected to show differences between algorithms (see Table 3). Simulation time was 10 minutes. For the purpose of shrinking simulation time all time constants are approximately 3-4 times smaller than in actual real life situation. The application permits simulation at a slower clock rate which will produce a realistic pace.

Algorithms are compared using 3 parameters, namely:

- average queue length – average number of cars waiting
- maximum queue length – global maximum for all intersections
- travel time – approximate average time to travel through the network by choosing a random route

Figures 4, 5 and 6 show the results obtained for these three parameters after simulation for algorithms 1, 2, 3 and 4 (x-axis).

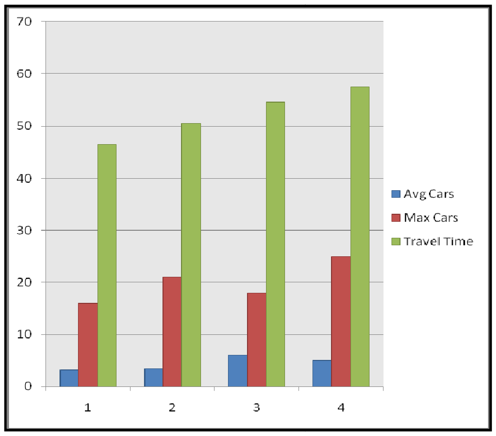


Figure 4. Results for input configuration one

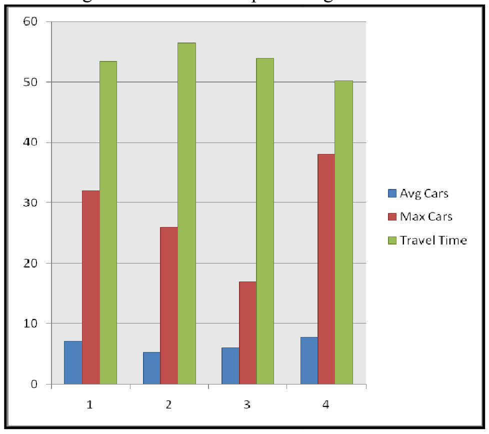


Figure 5. Results for input configuration two

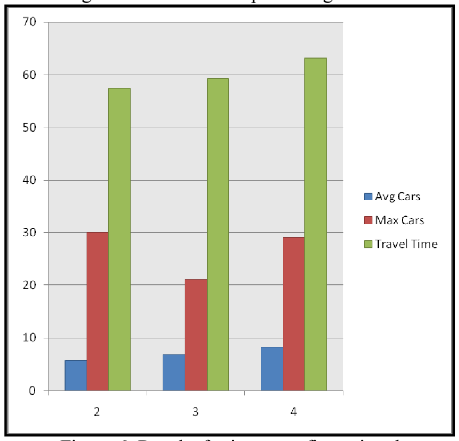


Figure 6. Results for input configuration three

Average travel time is estimated by taking an average over vehicle departure and arrival time moments and then subtracting the obtained values.

In Table III every column contains the input flux for the inputs as presented in Figure 1. The rows represent the three different configurations tested. The total system input increases from 370 to 540 and to 640.

TABLE III
INPUT VALUES IN VEHICLES/MINUTES

	X1	X2	X3	X4	X5	X6	X7	X8
1	70	20	50	50	20	70	70	20
2	70	50	50	70	100	50	100	50
3	100	70	70	70	70	100	70	70

Following tables (Table IV, V and VI) present phase time 1 (T1), phase time 2 (T2) and offset values (O) where appropriate in seconds. Each row corresponds to an algorithm: 1-Static; 2-Dyanmic; 3-With interrupts; 4-Genetic. (Note: real values are 3-4 times larger)

TABLE IV
PHASE TIMES AND OFFSET VALUES FOR INPUT CONFIGURATION NUMBER ONE

	C1		C2			C3			C4		
Alg:	T1	T2	T1	T2	O	T1	T2	O	T1	T2	O
1	7.5	12.5	10	10	2	15	5	4	12.5	7.5	2
2	6.5	13.5	10.5	9.5	2	14.5	5.5	4	12.5	6.5	2
3	13	11	14.5	12.5	-	16	16.5	-	13.5	12.5	-
4	11	19	16	14	23	20	10	15	19	11	10

TABLE V
PHASE TIMES AND OFFSET VALUES FOR INPUT CONFIGURATION NUMBER TWO

	C1		C2			C3			C4		
Alg:	T1	T2	T1	T2	O	T1	T2	O	T1	T2	O
1	10	10	8.5	11.5	2	8.5	11.5	4	10	10	2
2	10	10	8.5	11.5	2	7.5	12.5	4	10	10	2
3	10	9	8	11	-	17	12	-	18	13	-
4	12	18	14	16	23	11	19	19	19	11	4

TABLE VI
PHASE TIMES AND OFFSET VALUES FOR INPUT CONFIGURATION NUMBER THREE

	C1		C2			C3			C4		
Alg:	T1	T2	T1	T2	O	T1	T2	O	T1	T2	O
1	-	-	-	-	-	-	-	-	-	-	-
2	11.5	8.5	10	10	2	10.5	9.5	4	9.5	10.5	2
3	12	16	14	14	-	16	13	-	13	15	-
4	11	19	15	15	1	17	13	2	13	17	2

For algorithms 1 and 2 a fixed cycle of 20 seconds is used. Phase values for algorithms that produce variable phase times (2 and 3) are average values taken over the whole simulation. As mentioned before, algorithm 3 uses no offset because of the variable cycle time.

Test results are different on every simulation. This is so because input flow is allowed to change inside an interval and vehicle arrival is random (vehicles may arrive or not at every 250 milliseconds).

VII. CONCLUSIONS

After studying figures 4-6 it follows that the static algorithm is not efficient when used for input configuration 3 and it is the least efficient in the cases when the system is more loaded.

Dynamic algorithms prove to be the best as they keep maximum queue lengths at a low value. Algorithm 3 results show the minimal queue length values for cases 2 and 3.

Genetic algorithm generates a solution even for the most difficult case. In comparison with the first algorithm it creates better solutions for a more loaded system (input configuration 2 and 3).

Algorithm 1 is the simplest to implement because controllers need to be programmed to fixed phase times. It is the best solution for simple networks with under-saturated traffic.

Algorithm 2 needs sensors for detecting the number of vehicles waiting. Also the controllers need to allocate phase times at every cycle. It works for loaded networks because it uses recent data obtained from the sensors.

Algorithm 3 also needs sensors and in addition controllers need to exchange information between each other so a network connection is also necessary. It is the most secure because it checks neighboring intersections to prevent congestion.

Algorithm 4 can be used when the aim is to use fixed phase times that are optimal for a certain network and if coordinated work is imposed.

VII. FUTURE WORK

System architecture permits adding other modules to it which implement other algorithms. Research may continue on improving genetic algorithm results when tournament type selection is used. This has the advantage of a steady fitness function value growth but in the current form does not converge to near-optimal solutions.

REFERENCES

[1] Markos Papageorgiou, *"Review of Road Traffic Control Strategies"*, 2003

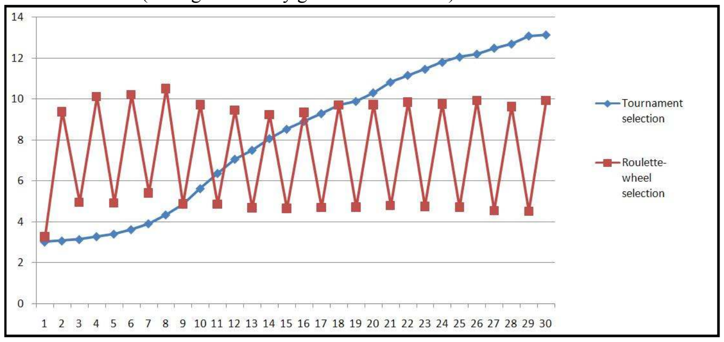
[2] Edward Lieberman, Ajay K. Rathi, *"Traffic Simulation"*

[3] Dennis I. Robertson, R. David Bretherton, *"Optimizing Networks of Traffic Signals in Real Time – The SCOOT Method"*

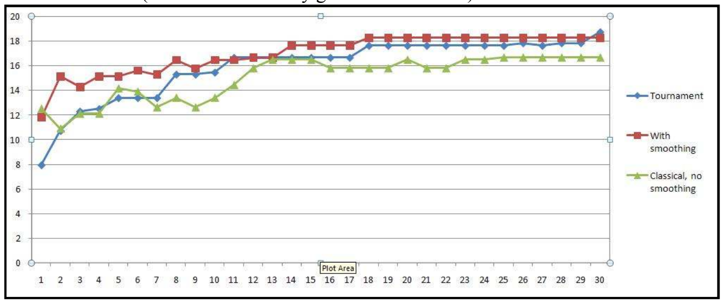
[4] Hong K. Lo, M. ASCE, Andy H. F. Chow, "*Control Strategies for Over-saturated Traffic"*, 2004

[5] C. I. Karr, E. J. Gentry, *"Fuzzy Control of a pH Reactor Using Genetic Algorithms"*, IEEE Trans., 1993

Attachment 1 (average fitness by generation number):



Attachment 2 (maximum fitness by generation number):


Plot Area
Tournament
With smoothing
Classical, no smoothing