Decentralised Car Traffic Control using Message Propagation Optimized with a Genetic Algorithm

Martin Kelly, Giovanna Di Marzo Serugendo
School of Computer Science and Information Systems
Birkbeck College, London
jkell01@dcs.bbk.ac.uk, dimarzo@dcs.bbk.ac.uk

***Abstract*—This paper describes a decentralised car traffic control simulation with re-routing and propagation of messages among traffic nodes (roads intersections and traffic lights). The values of the parameters governing the simulations are identified through the use of a genetic algorithm. This paper reports as well on results obtained regarding the convergence of the genetic algorithm towards fittest solutions.**

I. Introduction

Control of road traffic flows is an important concern for any urban area. Traffic managements, already in place, assist traffic officers to monitor and control traffic on cities or motorways. They allow visualisation or analysis of real-time traffic information. Decisions are usually taken from control centres and propagated to users by the means of road signs (traffic lights, motorway screens). However, the density of traffic and the complexity of road networks call for more self-adaptive solutions where traffic control dynamically and seamlessly adapts to traffic conditions.

This paper reports on an on-going work aiming at deriving different models, and their accompanying simulations, of decentralised car traffic control based on message propagation among road elements such as roads intersection, traffic lights, or cars. The main goal is to maximise traffic throughput and minimise travel time of vehicles. Simulations are controlled by diverse parameters whose values are separately established by the help of a genetic algorithm. Our previous model considered a fixed set of journeys involving different types of vehicle (regular and emergency cars). Speed-up and slow-down messages were propagated among road intersections in order to regulate traffic flow [5]. This paper presents our second model, where vehicles can be re-routed in case of congestion, and stop and go messages are propagated among traffic lights nodes instead of vehicle speed in order to optimise traffic under congested conditions. The specificity of our models lies both in the decentralised approach for traffic control, and in the large number of cars used to simulate traffic and high congestion rates.

Section II describes the city and the simulation model. Section III describes the genetic algorithm used for optimising the parameters involved in the model. Section IV discusses preliminary experiments and results, and Section V mentions some related works.

II. Control Model and Simulation

A. City and Model elements

The city is modelled as a square grid of 20 *nodes* by 20, representing a 2km*2km city space. The distance between nodes is set at 100 metres. Nodes are road intersections and each node maintains a traffic light controller. A *lane* is a portion of road between two nodes and has a direction (8 lanes are connected to each node).

A whole simulation comprises 15'200 vehicles travelling permanently in the city streets, and 1'520 different interconnected lanes.

Each simulation starts with 10 vehicles distributed at random across each lane. Each vehicle then randomly targets a destination on the opposite side of the city and generates its optimum route. This provides some routes which are more congested than others from the beginning. Vehicles will undertake journeys to completion and begin again for the entire evaluation period. The fixed number of vehicles roaming the city ensures a constant rate of congestion of 50%. While the number of vehicles is fixed, the number of journeys varies from simulation to simulation: after completing their journeys, vehicles start again a new journey. The fittest solution is then the one that allows the maximum throughput, i.e. the maximum number of journeys completed in a fixed amount of time.

At any point during a journey a vehicle may decide to re-evaluate the remainder of the journey. The pathfinding algorithm, directed by parameters established through the genetic algorithm, determines the most efficient route from that point. In addition to re-routing of vehicles, messages are sent among nodes to ask for modifications of traffic light signalling.

The actual speed through the city has a maximum of 30 miles per hour (50km/h). Vehicles try to move at this speed

1-4244-1340-0/07/$25.00 ©2007 IEEE

unless there is congestion.

The simulation stops after 1'800 seconds of virtual time, i.e. one half-hour of virtual activity.

B. Control and Message propagation

Depending on traffic conditions, messages are sent forward or backward from lanes to nodes to request a change in traffic light signalling. Lanes may send messages forward to request a change in the current signal, e.g. where its lead vehicle is stationary. Messages are also promoted backward to switch the signal away from a lane that is highly congested. Nodes cycle the traffic light (pass a token) between their inbound lanes, in order to grant them access to all outbound lanes.

1) Parameters

The parameters used to control message propagations and re-routing of journeys have each 7 possible values (see Table 1), and are as follows:

EvaluationTrigger: tendency for *vehicles* to *re-evaluate a route* which is taking longer than initially planned to resolve. The parameter value represents the % of chances of re-evaluating a journey which is perceived as running late. EvaluationTrigger is between 3% and 21% (3% increment). This evaluation is made as each step is completed. For instance, if the EvaluationTrigger parameter is set to 21%, a car blocked in a lane will re-evaluate its route in 21% of such cases.

ResponseThreshold: propensity of *nodes* to *respond positively to requests* for changing the traffic signals. The parameter value represents a propensity between 20% and 80% (10% increment) that the target node will actually satisfy the request issued from a lane. The node is evaluated every second cycle (every 2nd virtual second).

RequestThreshold: propensity of *lanes* to *raise a request* to change traffic signals. It is a probability curve, with higher congestion leading to greater likelihood of request transmission. This parameter has the same values as the ResponseThreshold parameter.

RequestLimit: sets the number of requests that a node may ignore before a change becomes mandatory. It is an absolute values ranging from 2 to 14 (increment of 2). It represents the maximum number of cycles during which a lane's message may be ignored. It is a scalar limit (not a propensity) corresponding to the number of evaluations that may pass before a response to a request is given.

Phase: influences the likelihood of a *lane* to *cycle the token* when no cars are within range of the intersection. The Phase parameter determines the minimum distance from the intersection the foremost vehicle must be before the lane will ask to cycle the token. It is an absolute value and represents a distance between 0 and 30 metres (increment of 5). If the next vehicle is beyond this distance from the target node (intersection) the lane will raise a request to change the signal. If the light is green, it will turn it to red, because the car will not make it in time to reach the intersection; if the light is red, the request to cycle is an attempt to make the light green by the time the car arrives at the intersection.

	1	2	3	4	5	6	7
EvaluationTrigger	3%	6%	9%	12%	15%	18%	21%
ResponseThreshold	20%	30%	40%	50%	60%	70%	80%
RequestThreshold	20%	30%	40%	50%	60%	70%	80%
RequestLimit	2	4	6	8	10	12	14
Phase	0m	5m	10m	15m	20m	25m	30m

Table 1: Parameter's Values

The message propagation has an indirect impact on routing and re-evaluations. As traffic builds up, lanes tend to restrict access to themselves, and request access to others. This will have a detrimental affect on journey times, leading to individual vehicles getting closer to the environment's EvaluationTrigger threshold and therefore a good likelihood of route re-evaluation and redirection through less congested lanes.

2) Message Propagation

Messages propagate from lanes to forward and backward nodes. Tokens are employed by nodes to permit an inbound lane access to its outbound lanes. The token (i.e. the green light) is cycled through all inbound lanes one at a time. Lanes employ the use of messages in the simulation in an attempt to influence the node's control of the token. There are three types of message: 1. **forward** messages to **cycle the token** onwards (from green to red or from red to green); 2. **backward** messages to **cycle to token** onwards; and 3. **ignore** messages. Messages are raised in the five following situations:

A lane sends messages **forward** when it detects that vehicles are stationary waiting for the green light, asking the forward node to **cycle its token onwards**, i.e. to rotate the green light towards the lane in order to allow the cars to leave the lane.

A lane may also send messages **backward** when it detects that it is congested, asking the backward node to **cycle its token onwards**, i.e. to rotate the green light towards another lane in order to stop the flow of cars coming in.

When no vehicle are within range of the intersection, the lane may send a **forward** message to the forward node asking it to **cycle the token**, i.e. to change from green light to red because no vehicle is ready to cross the intersection, or to ask for the red light in anticipation of the arrival of a car further down the lane.

When the vehicle at the head of a lane has a green light but is blocked because the lanes in front are congested, the lane sends a **forward** message to the forward node asking it to **cycle the token**, so that the first car will be prevented to block the intersection.

Finally, a lane may also over-write a previous message at any time according to the latest prevailing conditions: **ignore** messages. This includes cancellation messages to null out any previous change requests.

The ResponseThreshold and RequestLimit parameters influence the behaviour of nodes regarding lane's requests. While each vehicle and lane is cycled once per virtual second, nodes are only visited once in every two cycles. This permits a high enough resolution to be effective and to permit nodes to deal only with the latest prevailing conditions as only a lane's latest message request is of significance. The ResponseThreshold parameter determines the likelihood that a node will respond to a message, the RequestLimit parameter sets the number of messages which may be ignored from a node before the message must be responded to. For example a very busy thoroughfare may choose to reject requests for access from a side road with only two waiting vehicles. The RequestLimit attempts to prevent the side-road from constantly ignoring requests for the token based merely on weight of numbers.

Raising a request to either yield the current token, or to rotate the inbound lane's token, will be determined along a curve. Requests are not immediately raised by lanes; this is governed by the RequestThreshold attribute.

The messages propagated from lanes to forward and backward nodes help to ameliorate local conditions. Despite the heavily congested nature of the city during rush-hour we facilitate local variations in congestion through this mechanism.

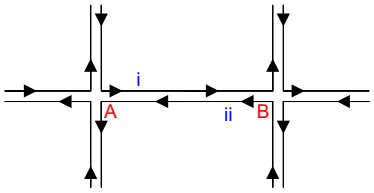

Figure 1: Messages Flows

Figure 1 shows an example of messages flows between lanes and nodes. If lane (i) is congested it will raise a backward message towards A, to rotate the signal in order to stop feeding vehicles towards it. When lane (ii) is blocked due to a vehicle at its head being waiting for a green light, it will raise a forward message to A, to rotate the signal.

C. Routes determination

Each of the 15'200 vehicles evaluates its entire route prior to commencing its journey using a variation of the A*-pathfinding algorithm [1].

Whenever the EvaluationTrigger threshold is reached, the journey is likely to be re-evaluated from its current point to the route's endpoint. The EvaluationTrigger threshold directs the propensity or tendency to re-evaluate, i.e. once this trigger value is reached the likelihood of re-evaluation increases along a curve.

Each step in a route has an anticipated individual cost and an anticipated cumulative cost represented as the number of seconds it should take to get to that point in the journey. If the cost is higher than anticipated each vehicle will check to see if the extra cost is large enough to warrant re-evaluation, if so, it then searches out a new optimum route and updates the anticipated costs from that point onwards.

The A*-pathfinding algorithm works as follows (see Figure 2). Given the current position of the car, the A*-pathfinding algorithm needs to find a new route from that position to the destination point. The best route is the one that minimises the cost of going from the current point to the destination. According to the A*-pathfinding algorithm a route is scored according to the equation $F = G + H$, where G is the cost to go from the current position of the car to a certain node along the considered route, and H is the evaluated cost to go from that node to the destination.

More precisely, from the current point, we build an "open" list of lanes (the list of possible lanes to consider for re-routing). The lane the car came from (current lane) is not part of this list, so it is in the "closed" list (the list of lanes we do not check for the moment). For each lane to consider, the cost to reach the next node is computed (G), it is given by the time needed to reach the next node given the current speed of the cars on that lane (at the time of the re-evaluation of the route). Then, the cost to reach the destination point is calculated (H). Since at this point in the algorithm we do not know yet the exact route, the route is estimated using the Manhattan distance: moving horizontally and vertically directly towards the target destination, without taking into account any possible congestion in between. H is the cumulative cost to reach the target destination from the node being considered. The total cost F is given by $F = G + H$. The lane with the lowest F is chosen in a first instance. The algorithm is then applied recursively to this lane until the target destination is reached, or no route is identified (e.g. cost is higher than current route). The final route is then found by going backwards from the target destination to the initial point. Different routes (with different costs) may pass at common points (nodes or lanes). If during the evaluation of the cost of the different possible routes, the algorithm comes across an already visited point for which the previously calculated cost is lower than the one calculated for the current route, the algorithm drops the current route, switches to the previous route and continues from there.

A*- pathfinding algorithm

```
BEGIN
 Add the starting lane to the open list.
 Loop

  Select the lowest cost lane on the open list.
  Remove it from the open list and add it to the closed list.

//Evaluate the cost of adjacent lanes
   For each of the adjacent lanes:
      If adjacent lanes are on the closed list or are
         invalid candidates ignore them
      Else // if they are valid
         If it is not on the open list
           Add it to the open list
           Set the lane's parent lane to the current lane
           Evaluate the lane's F, G, and H costs
         Else // it is already on the open list
           If previous instance has a lower F cost
                 // (the previous route to that lane is a better
                 candidate)
                  Leave it there and discard this instance,
                  Set the previous instance's parent lane to
                    the current lane.
                  Recaculate F, G for that lane
           End if
         End ifelse
       End ifelse
   End For
  Exit loop if target lane is current lane or if open list is empty

 End Loop

 END
```

Figure 2: A*-Pathfinding Algorithm

Cars take decisions regarding re-routing on the basis of global information having only a short time of validity. Indeed, the value F of the best route is indicative only: it is accurate at the moment of the evaluation, but the actual cost will be known only at the end of the journey. The final result will depend on the individual decisions taken by the different cars.

Simulations are not deterministic. For example the EvaluationTrigger parameter controls a propensity to re-evaluate a slower than expected journey. Therefore two simulations of the 30 minute period would be unlikely to yield identical results, however they would be in the same category of results as each simulation would have the same propensities.

This was chosen to better reflect people's behaviour on the roads – when they start to suspect they will run late different people have different thresholds for re-evaluation.

III. Genetic Algorithm and Fitness Function

15'200 vehicles permanently populate the streets and constantly undertake journeys during the evaluation period. Fitter sets of genes values will permit more journeys to be undertaken during the 1800 seconds of simulation time. Therefore the solution with the most journeys completed is the fittest. To differentiate between competing solutions when evaluation is complete, we also look at the set of current but incomplete journeys. All vehicles have the potential to be in mid-journey when the simulation terminates. For these incomplete journeys we examine how far they had travelled and in what time, i.e. the average speed of the journeys.

A. Fitness Function

The fitness function is comprised of a primary element: the total number of complete journeys; and a secondary element: the average speed for incomplete journeys.

The average speed of incomplete journeys is only relevant when distinguishing genomes that execute the same number of complete journeys.

$$\text{Fitness Function} = \text{journeys}*100 + \text{avgspeed}*10 .$$

Value "journeys" represents the number of completed journeys. The value "avgspeed" represents the average of the speed of all the incomplete journeys.

B. Genetic Algorithm

We evaluate an initial set of 49 individuals. Once that set is evaluated we select candidate genes for reproduction, applying crossover and mutation to yield two child genes. These genes are then run through the simulation. At the end of this evaluation the next generation is selected, again through roulette-wheel selection. All genomes tested must be unique, i.e. we will not evaluate the same gene sequence more than once. If genes prove fitter than the worst case currently maintained, it is added to the set of fittest candidates and the lowest performer is dropped from the gene-pool. Our tests are concerned with two priorities: 1. to identify the fittest candidate from the simulations undertaken, and 2. to demonstrate convergence towards fitter solutions across the entire gene-pool.

1) GA Genes.

The five genes: EvaluationTrigger, RequestThreshold, ResponseThreshold, RequestLimit, and Phase have seven possible values, yielding a search space of 16'807 individual solutions.

2) GA Initialisation

The fittest 49 candidates are maintained in the genepool. Initially this is set by individuals chosen with middling gene values. The 49 generation zero individuals' gene values are set to between 3 and 5 inclusive at random, they are evaluated for fitness. When evaluation is complete an attempt is made to add them to the genepool, because the genepool at this stage has fewer than 49 candidates, the addition is successful. When no more candidates remain from the set awaiting evaluation we breed a new pair for

evaluation.

A new genome may be added to the genepool only if their fitness is higher than the current worst-case. If, after evaluating both individuals, no insertions are made to the genepool the mutation rate is increased (see below) and two new individuals are created from the same generation. The generation count only increases when the genepool changes, i.e. when a successful addition is made. Therefore our genepool is comprised of the 49 fittest individuals across all generations.

3) Crossover and Mutation

During reproduction we select two candidate genomes from the genepool using roulette-wheel selection weighted by relative fitness. A random point from the second to fourth gene is selected and two sub-strands extracted from each parent creating two individual siblings. We then subject the siblings to potential mutation.

Mutation is set at 10% initially. That is, the likelihood that a single gene may be affected. When an attempt to add an individual fails, we increase the mutation rate by one to help to promote wider selection. For generations 0 to 99, the potential bump is set to a maximum limit of 20%. After 100 generations are evaluated, mutation may rise as high as 50%. After each successful addition to the genepool, mutation is reset to 10% - this is to ensure that unnecessarily high mutation rates do not apply to the first candidate offspring of a generation.

4) Remark

Since the pathfinding uses a variation of the A*-algorithm, and initial routing and subsequent rerouting for 15'200 vehicles through a 2km*2km city was an extremely expensive operation in terms of time taken, cells were used to describe areas of common congestion. Penalty and reward had been introduced for moving to a cell of higher congestion or to a cell of lower congestion respectively. These two notions would tend to derive routes of consistent or improved congestion levels without the need to evaluate every lane in that area. Two additional genes (TransitHigh, TransitLow) controlling the penalty and reward were then initially incorporated to facilitate routing and pathfinding throughout the city. However improvements made to the pathfinding algorithm rendered this rather unnecessary. The algorithm was improved through type-specific high-speed custom collections for the open and closed lists, and also by maintaining two in-memory indices which facilitated look-up by lane id, and lane cost. Pathfinding using the new algorithm was approximately 100 times faster than the previous implementation. The TransitHigh and TransitLow genes were therefore discarded as experiments showed that these genes had value and interest only to a less-efficient pathfinding mechanism. With an efficient pathfinding algorithm, the fittest function was not affected by these two values.

IV. Experiments and Results

The results presented here relate to 158 generations obtained through the genetic algorithm. We performed 3315 simulations where mutation was increased automatically in order to find fitter solutions worthy of addition to the genepool. The optimum has been found at generation 133.

1) Global Result

Presently there is about a 13% variation between worst and best case in terms of journeys made. Given that each of the 1520 lanes in the 2km*2km block begins filled to 50% capacity, and there are 15'200 vehicles vying for scarce resources, 13% enhancement can be considered a good result (see Table 2).

	Fitness	Journeys
Minimum	2'454'721	24'546
Maximum	2'774'920	27'748
Average	2'595'877	25'958
Absolute Difference	320'199	3'202
Percentage Difference	13.04%	13.04%

Table 2: Global Results

The city starts off with each of its 1'520 lanes at 50% congestion, i.e. with 10 vehicles on it. Choosing such a high rate of initial congestion is particularly important for our purposes, i.e. to demonstrate that local messaging and auto-rerouting may contribute to improving throughput in a congested urban environment. The difference between best case and worst case was over 13%, i.e. 3'202 extra journeys were catered for.

2) Number of individuals per generation

Significantly at higher generations, many more simulations had to be undertaken at increasing rates of mutation in order to find candidates fit enough to be added to the genepool. The results at central gene values were favourable to the simulations, and improvements, while incremental, demonstrate a tendency to increase fitness across generations. Figure 3 shows the number of new individuals created at each generation. For instance, at generation 122, 51 new individuals (and consequently 51 simulations) have been created. Such peaks correspond to high mutation rate, and correspond to periods where the system had difficulties in finding a solution to insert into the genepool. We can observe that the number required for finding a candidate fitter than the prevailing worst-case in the genepool generally increases steeply as generations increase. Mutation rates were permitted to rise to 50% (that a single element would vary) once generations rose above 100. This was done in anticipation of the difficulty of finding solutions eligible to

enter into the genepool. The simulations appear stuck in local minima around generation 120-130. The number of simulations required to advance peaking around this time. The data shows this effect ameliorated in subsequent generations, once an optimum has been found at generation 133, and lower numbers of simulations are necessary with each advancing generation. This probably shows the effectiveness of the genetic algorithm, and the variable mutation rate.

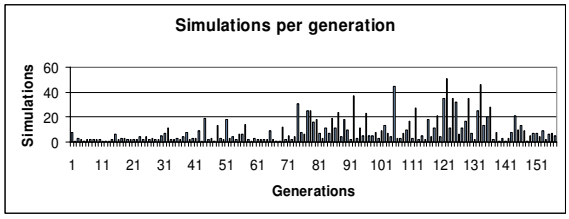

Figure 3: Number of simulations per generation

On Figure 3, we can see in the data the effect that mutation has on the simulations, where new gene values emerge through increasing mutation rates as generations become stifled in local minima. This yields an eligible candidate and the generation proceeds forwards. If we consider as before, the generation 122, with high mutation rate (and high number of individuals created), the system finds a local maximum (maximum value of the fitness function is 2703321 for generation 122). At each peak in Figure 3 corresponds a local maximum in Figure 4. This figure also clearly shows the convergence (a regular linear progression trend) of the genetic algorithm towards fittest generations.

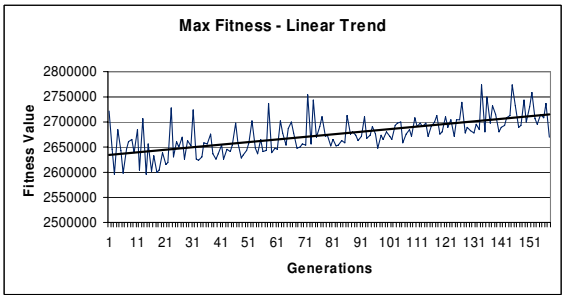

Figure 4: Max fitness function value per generation

3) Convergence of genetic algorithm

Figure 5 shows the progression of the average fitness function values at each generation (over the different individuals created at each generation). This figure shows that the average fitness value remains rather stable. This is due to the fact that there are large differences between minimum and maximum fitness values inside a given generation.

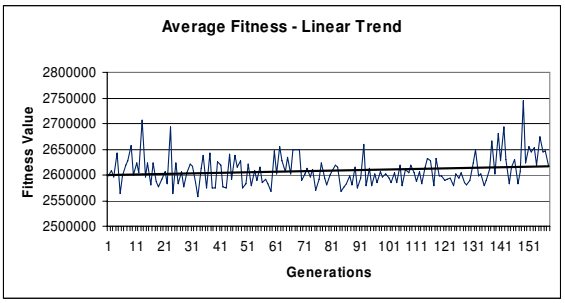

Figure 5: Average fitness function value per generation

4) Genes and Corresponding Parameters values

Avge

Genes	EvalTrigger	ReqThresh	RespThresh	ReqLimit	Phase	Fitness
First Gen	12 %	60 %	60 %	8	15m	2609508
Last Gen	9 %	60 %	50%	8	15m	2617060

Fittest

Genes	EvalTrigger	ReqThresh	RespThresh	ReqLimit	Phase	Fitness
First Gen	18%	70%	50%	8	25m	2646720
Last Gen	3%	40%	50%	6	10m	2668921

Overall

Genes	EvalTrigger	ReqThresh	RespThresh	ReqLimit	Phase	Fitness
Least Fit	9 %	70%	70%	10	30m	2454721
Most Fit	3%	70%	70%	10	25m	2774920

Table 3: Parameters value (best/least fit genes)

The first rows (Average) in Table 3 show the average genes values for the first and last generations (generation 0 and generation 157). The second set of row (Fittest) shows the exact genes values for the fittest individual in the first and last generations. Finally, the third set of rows (Overall) shows the exact genes values for the fittest (gen. 133) and least fit individuals (gen. 143) overall (across all generations). Genes have 7 values representing different cases in the simulation model (see Table 1).

Generally a lower tendency to re-evaluate journeys tends to improve fitness (low values for the EvalTrigger gene). This is likely due to the local cooperation which emerges between lanes and nodes to ameliorate congestion through message propagation. It appears initially that local effort to reduce congestion for all journeys outperforms the individual's tendency to re-route.

It is different for the average values because each generation must find a fitter than current worst-case individual in order to move to the next generation.

The fittest genome in the final generation tends towards a lower propensity to re-evaluate journeys, and a generally mid to low set of remaining characteristics.

As said before, the fittest individual was found in generation 133, and from that point to final the genepool has

been constantly refined with increasing average fitness despite not finding a new topmost candidate.

V. Related Works

Swarm-based traffic control usually employs ant metaphor for inducing a decentralised traffic control. We can cite Ando et al. [2] who apply the pheromone metaphor to provide a decentralised traffic congestion prediction system: cars deposit pheromone along their route which is later retrieved by forthcoming cars. The amount of pheromone deposited depends on the speed of the car, and represents the density of traffic: low speed produces high concentrations of pheromone, while high speed produces low concentrations of pheromone. The amount of pheromone later retrieved by other cars provides an indication about traffic congestion and thus serves for short-time traffic predictions.

Similarly, Hoar et al. [4] use the ant metaphor to communicate among cars and to provide a simulation of traffic dynamics in different scenarios. In this case, an additional evolutionary algorithm, including a swarm voting system for preferred traffic light timing, is introduced in order to minimise the average waiting time of vehicles.

An ant-based system for decentralised re-routing is provided by Tatomir et al. [7]. It follows the AntNet algorithm. Artificial ants roam the network of streets and update routing tables at each node (road intersection) which serve for guiding cars. Data provided by cars themselves is also used to enrich the routing.

Other decentralised solution, without message propagation, can be worth mentioning as well. De Oliveira et al. [3] use a reinforcement learning algorithm for updating the parameters of traffic light controllers at run-time in non-stationary environments, specifically studying individual drivers' behaviours. Rochner et al. [6] use a specific three-layer architecture, where the first layer acts as a "reflex" layer and sits at the level of the traffic light controllers: fixed durations, or variable phases based on traffic detectors information. The second layer is based on monitoring, experience and learning and acts on the parameters of the first layer: identified traffic situations are mapped to parameters of the first layer. The third layer is based on some planning concerns, and uses an internal simulation to help take decisions for unknown situations, by optimising the values of the parameters of the lower layers. The third layer works "off-line" contrarily to the first layer which is for decisions that have to be taken on the fly as congestion arises.

VI. Conclusion

The linear trend shown by Figure 4 lets presuppose that, by further running the genetic algorithm, we could potentially find a better optimum. Thus, in a first instance, we will continue pursuing experiments with the current model in order to identify additional optimums; then we will extend the current experiments by comparing with other techniques and evaluating the current model under different congestion situations. Second, another model is planned which will tackle re-routing of emergency vehicles only. We are also planning to combine these two models together where both regular and emergency vehicles are re-routed and traffic globally optimised. Third, as most simulations of car traffic control, we are using a square grid of routes for modelling the city. Once, the simulation proves to be worthy in this "simplistic" case, it will be necessary to translate it into an actual car traffic schema. Finally, the models proposed consider fixed parameter values that allow the system to adapt to changing conditions within a fixed period of time (e.g. from 8am to 10am). In order to enhance the adaptability to unexpected traffic conditions, it is necessary to integrate into the model the possibility to change these parameters on the fly (e.g. combining instance message propagation with reinforcement learning).

References

[1] http://www.policyalmanac.org/games/aStarTutorial.htm.

[2] Y. Ando et al. "Pheromone Model: Application to Traffic Congestion Prediction". In *Engineering Self-Organising Systems*, volume 3910 of LNAI, pages 182--196. Springer-Verlag, 2005.

[3] D. de Oliveira et al. "Reinforcement Learning-based Control of Traffic Lights in Non-Stationary Environments: A Case Study in a Microscopic Simulator". In *Fourth European Workshop on Multi-Agent Systems (EUMAS'06)*, 2006.

[4] R. Hoar, J. Penner, and C. Jacob. "Evolutionary Swarm Traffic: If Ant Roads Had Traffic Lights". In *Congress on Evolutionary Computation (CEC'02)*, pages 1910--1915. IEEE, 2002.

[5] M. Kelly and G. Di Marzo Serugendo. "A Decentralised Car Traffic Control System Simulation Using Local Message Propagation Optimised with a Genetic Algorithm". *In Engineering Self-Organising Systems*, volume 4335 of LNAI, pages 192--210. Springer-Verlag, 2007.

[6] F. Rochner et al. "An Organic Architecture for Traffic Light Controllers". In *Informatik 2006 - Informatik fur Menschen*, volume P-93 of Lecture Notes in Informatics, pages 120--127. Kollen Verlag, 2006.

[7] B. Tatomir, L. Rothkrantz. "Dynamic Traffic Routing Using Ant Based Control". *In IEEE International Conference on Systems, Man and Cybernetics*, pp. 3970-3975, vol. 4, 2004.