# Reinforcement Learning of Traffic Light Controllers Adapting to Accidents

Jiří Iša Julian Kooij Rogier Koppejan Lior Kuijer

*Design and Organisation of Autonomous Systems 2006*
University of Amsterdam

February 2, 2006

**Abstract**

Traffic has become an important part of modern society. Research on efficient traffic light control has become a new and interesting subject within the AI field. Traffic models and different machine learning methods have been developed, but so far, no attention has been given to dealing with accidents. In this project, new reinforcement learning methods are suggested and compared to existing methods, on a novel traffic model that does include accident simulation. Results indicate that careful consideration should be given to the accident model, as well as to the basic traffic model we extended upon.

## 1 Introduction

The increase of population along with the significant increase in the number of vehicles on the road has resulted in some of the main problems of our modern society, namely traffic-jams and traffic accidents. These problems are largely present in and around urban areas and have a major impact on both the economy and the environment. In such areas, traffic lights form an important traffic control mechanism.

One of the goals of traffic light control is to produce fair waiting times for all vehicles at junctions (where traffic lights are placed) and of their overall trip. In most cases, this is done using some fixed protocol, meaning, the light is red for some time interval and green for some subsequent time interval. The intervals usually change during peak hours, but are still static. Such an approach leads to a fixed behavior, that does not reflect dynamic changes of the traffic environment. A solution could be to apply Machine Learning (ML) methods.

Unfortunately the policies such methods learn are not alway optimal. Still, even a learned suboptimal policy might be more suitable and can outperform a fixed one. Several ML methods have already been applied to this problem including fuzzy logic [2], genetic algorithms [3], and reinforcement learning (RL) [5, 6, 7].

Our goal in this paper is to focus on dealing with accidents efficiently using a RL based traffic light control mechanism. Existing and new RL algorithms for traffic light control are evaluated on their ability to prevent congestions in traffic environments that include accidents.

Note that our aim is not to design a system that prevents accidents. Instead, our system should be capable of controlling the traffic flow efficiently, even in such complex situations where accidents do occur.

## 2 Background on Traffic Light Control

The project extends previous research on existing RL techniques used for traffic light control. Traffic simulators, such as the open source project *Green Light District* (GLD)[1] [6], which is a Java based application, enable the comparison of different methods under varying circumstances. Furthermore it allows to easily create extensions for the basic framework.

Within this simulator a RL approach is taken to optimize a *traffic light controller* (TLC). The task of a TLC is to create an optimal traffic flow throughout a given traffic network of lanes and junctions. A flow is optimal in the sense that, on average, a car driving through the network can reach its destination as fast as possible. In other words, the TLC should prevent unnecessary waiting times for traffic lights, and traffic-jams should not emerge.

Reinforcement learning has proved to be a useful ML approach to automatically configure traffic lights [4, 5] . However, until now simulations did not include any model of accidents, nor were traffic light controllers designed with such situations in mind.

The investigation described in this paper continues in the line of research of [4]. Their version of the GLD simulator was taken as a base and extended upon on in several directions to support accident modeling.

## 3 Simulation of Traffic and Accidents

Further development of the simulator, and the addition of new features, was an important part of the project. Therefore, this section will give an overview of the simulator's features and its implementation.

### 3.1 Simulator Framework

The GLD simulator is capable of modeling traffic networks such as the one shown in figure 1. On the left side of figure 1, a 'Manhattan style' traffic network is shown. This network is the one used in all of the reported simulations.

Traffic networks consist of roads connected by *junctions*. Roads consist of several parallel *lanes* over which cars can move. Traffic lights are placed at junctions that

[1] http://stoplicht.sourceforge.net

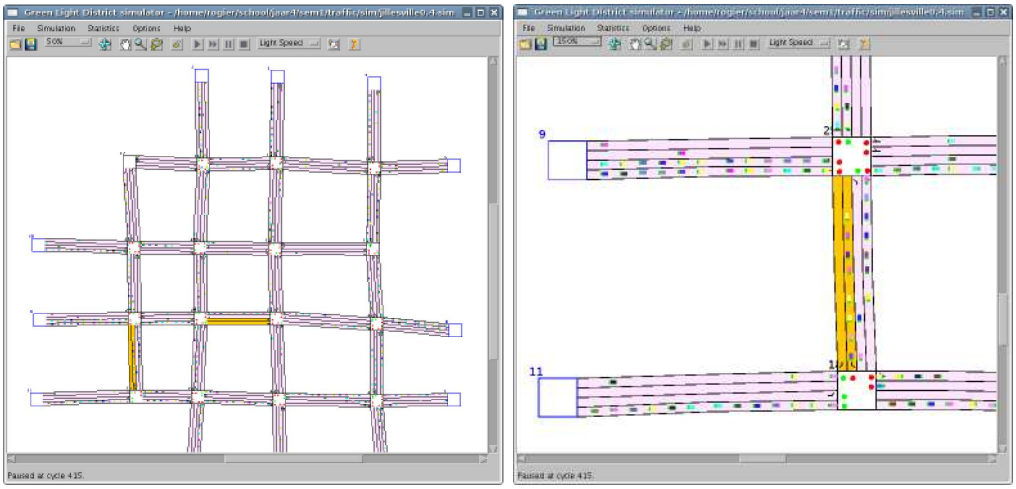

Figure 1: Screenshots of the *Green Light District* simulator. The shown network is the one used for the simulations described in this report. On the left a general overview (a), on the right a close-up (b).

connect more than two roads, such that each lane has one traffic light that can be turned to red or green. Cars enter and exit the network at so called *edge nodes*. When a car enters the network at such an edge node, it is assigned another edge node as its destination.

The shortest route to this destination determines the path the car will take through the traffic network. Since there is only a finite number of edge nodes, shortest paths to each edge node from every junction can be precalculated.

Lanes that are disabled, due to a simulated accident, are colored yellow in figure 1. The figure on the right is an enlarged portion from the bottom-left corner of the figure on the left. Here we get a better view of edge nodes, roads, junctions, cars and accidents. The simulator uses discrete time steps (cycles). Each cycle, the selected TLC is used to determine which lights to turn red or green, and cars are moved simultaneously with one step.

## 3.2 Extending the Model with Accidents

This section describes in more detail how the basic simulator has been extended with accidents. Also, several correlated issues are addressed that needed to be solved in order to make the extended simulator work.

### 3.2.1 Accident Modeling

In every cycle of the simulation an accident can occur with a preset probability. Lanes with more cars are more likely to be selected as an accident spot. Once an accident happens, all traffic going through this street will become impossible. This is done by

rerouting all lanes leading to the accident spot to other directions.

Special care is given to the fact that there must always remain at least one way to get from every junction to every other junction. This is necessary to ensure that the network does not get separated into disjunct parts. All cars on a lane with an accident are allowed to keep driving as if nothing happened. The only effect of the simulated accident is that no other car can enter the accident area.

### 3.2.2 Rerouting Cars around Accident Areas

Once lanes have been disabled due to an accident, the precalculated shortest paths information is not valid anymore. Several cars may have planned their routes through affected lanes. This situation may lead to cars waiting at a junction in order to enter a disabled lane. Of course, these cars will not be able to enter it until the accident has been resolved (which happens at random). As an effect, the roads around an accident area will be filled by cars that are unable to continue on their route, and that portion of the network will get congested.

In order to handle this situation, *rerouting* has been added to the simulation. If rerouting is switched on in the simulator settings, the shortest path information is recomputed when an accident occurs. Cars are assigned a new route if necessary, thus the traffic planned to pass through the accident area will instead be rerouted around it. Note that rerouting reflects the actual behaviour of road users in the real world.

## 3.3 Extending the Model with Stuck Car Removal

Sometimes, a car will get stuck during simulation - it will never be able to move again. This situation can arise when a real traffic-jam occurs: Multiple cars are in a deadlock situation, preventing each other to move.

As a result, affected cars will not reach their final destination. In the original simulator framework, the performance measures were only updated when cars did reach their destinations (since only then a car's trip waiting time will be known). Stuck cars that never exit the network are therefore never taken into account by these measures.

Another problem for an adaptive learner would be, that its weak performance in the beginning might immediately lead to the deadlock situations, the city gets congested and the learner does not, in fact, get a chance to learn. As a solution to this problem *stuck car removal* was added. With this feature turned on, cars that have not moved during $\lambda$ consecutive cycles are removed from the simulation. In each of the $\lambda$ previous cycles, such a car would have received some negative reward. However, an additional penalty of $p$ can be set, which will be applied when a stuck car is actually removed. The controller should learn not to depend on removing stuck cars. In the simulations that made use of stuck car removal, $\lambda$ was set to 20 cycles.

# 4 Reinforcement Learning and Traffic Light Control

The GLD package has several TLC algorithms already implemented. One of the available controllers is *TC-1 Optimized*, an improved version of *TC-1* [5, 6, 7], which uses a RL technique to optimize its policy. The TC-SBC controller, which is derived from the basic TC-1, is included as well and will be described in section 4.2. The RL approach TC-1 takes is a bit different from how RL is used normally in the literature. It will be assumed here that the reader has some basic understanding of RL, if not, [1] provides a good introduction.

## 4.1 The TC-1 Controller

TC-1 uses a multi-agent approach to RL. Every car in the simulation corresponds to an agent. Hence it is called a *car-based* model. This approach is based on the following insight: Modeling all possible traffic configurations around a junction would very quickly lead to a very large state space. Instead the contributions of each individual car at the junction are summed. No intelligence is required from the car, as the *controller* is the one responsible.

The state of a car is modeled as a tuple $s = [n, i, p, d]$, with $n$ the number of the next junction, $i$ the lane of the car towards the junction, $p$ the position of the car in the queue in front of the light, and $d$ the destination of the car. The state space $S$ consists of all values for $s$.

A value function $Q : S \times \{red, green\} \rightarrow \mathbb{R}$ is learned to expres the expected waiting time for a car, specified by $s \in S$, to have traffic light $i$ of the junction turned red or green. The gain of turning the light green for the car is then expressed by $Q(s, green) - Q(s, red)$. However, turning a traffic light to green affects the whole *queue* $L_i$ of cars waiting in front traffic light $i$.

The optimal traffic light configuration at junction $n$, of the set of possible configurations $A_n$, can therefore be computed as

$$A_n^{opt} = \text{argmax}_{A_n} \sum_i \sum_{s \in L_i} (Q(s, green) - Q(s, red)) \tag{1}$$

The $Q$-values are learned by performing an update after each cycle for all occupied states:

$$Q(s, a) \leftarrow \sum_{s'} P(s, a, s')(R(s, a, s') + \gamma V(s')) \tag{2}$$

Here $P(s, a, s')$ is the state transition probability of $s$ to $s'$, when action $a$ is preformed. $R(s, a, s')$ is the reward function for the state transition, $\gamma$ the future discount parameter and $V(s) = \sum_a P(a|s)Q(s, a)$. The model $P(s, a, s')$ is estimated during simulation by dividing the number of transitions, given $a$, from $s$ to $s'$ by the total number of transitions from $s$. The reward function $R$ simply gives only a negative reward in case the car didn't move:

$$R(s, a, s') = \begin{cases} -1 & \text{if } s = s' \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The parameter $\gamma$ was set to 0.9 during all simulations.

### 4.2 The TC-SBC Controller

The TC-SBC algorithm is an extension of TC-1 and has shown to be capable of outperforming it [4]. TC-SBC stands for *Traffic Controller - State Bit for Congestion*.

TC-SBC takes explicit information of the congestion of lanes around a junction into account. The car state representation is extended by one bit $c$, to keep the state space small, representing the *congestion* on the next lane. The bit is set to true if the fraction of the destination lane filled by cars is above a threshold $\theta$:

$$s = [n, i, p, d, c] \qquad \text{where} \qquad c = \begin{cases} 1 & \text{if } (w_d/N_d) > \theta \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where $w_d$ is the current number of cars on destination lane $d$, and $N_d$ the maximum number of cars that fits on the lane. A value for the threshold $\theta$ that has been reported to work well is 0.8 [4]. This project used this value for $\theta$.

### 4.3 New Controller Algorithms

In addition to TC-1 and TC-SBC, two new TLC algorithms are proposed, aimed specifically to handle accident situations: TC-SBA (*Traffic Controller - State Bit for Accidents*) and TC-SBAC (*Traffic Controller - State Bits for Accidents and Congestion*). Both algorithms are straightforward extensions of TC-1.

#### 4.3.1 The TC-SBA Controller

TC-SBA extends the TC-1 car-based state space with an extra *accident bit*, similar to what TC-SBC does for congestion. A state is now given by the tuple $s = [n, i, p, d, a]$, where the $a$ bit is set to true if the destination lane $d$ leads to a junction $j$ which is part of an accident area. This is the case when the junction $j$, that is reached after following the destination lane $d$, has an outgoing lane that is disabled by an accident. This can be expressed by following equation:

$$a = \begin{cases} 1 & \text{if } D \cap O_j \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where $D$ is the set of disabled lanes due to accidents, and $O_j$ the set of all lanes leaving from junction $j$.

#### 4.3.2 The TC-SBAC Controller

The TC-SBAC controller extends TC-1 with both a congestion bit (eq. 4), as TC-SBC does, and with an accident bit (eq. 5), as TC-SBA. This enables TC-SBAC to learn $Q$-values depending on both a possible congestion on the destination lane, and on possibly passing an accident area.

A clear disadvantage is that the two additional bits result in a state space four times the size of that of TC-1. Training the controller during the simulation can suffer from this. In parts of network that are not affected by an accident, $Q$-value are learned for the $[n, i, p, d, 0]$ sub state space. This experience is however disregarded as soon as the occurrence of an accident does affect the area. The TLC will now learn values for the $[n, i, p, d, 1]$ state subspace for which it has had no experience yet.

# 5 Results and Evaluation

## 5.1 Experiment Setup

Tests were performed with varying option settings. The different parameters that were considered are listed in table 1. All tests consisted of 10 runs, each lasting 50,000 cycles. For the best possible comparisons, each situation was tested for each algorithm under the same conditions. For all tests, the traffic network depicted in figure 1 was used.

| Parameter | Options |
|---|---|
| Accidents | Enabled, Disabled |
| Car Rerouting | Enabled, Disabled |
| Stuck Car Removal | Enabled (no penalty)<br>Enabled (with penalty)<br>Disabled |

Table 1: Different parameters and their values that were varied over experiments.

As the tests were performed, several performance measures were recorded for each algorithm. However, some are of particular interest to us as they prove to be good indicators for the congestion inside the network.

**Average Trip Waiting Time:** The most obvious measure, used in previous articles, shows the average number of cycles that a car has to wait on its whole journey through the city. This is the value minimized by the RL algorithms. For the purpose of this article, this measure proved to be insufficient. Once there are cars in a deadlock, they do not arrive to their destination edge node and they cannot be incorporated in the statistics. The average trip waiting time remains low, although there are cars waiting for a very long time. The following two measures do not have this feature and are used in the rest of the paper.

**Average junction waiting time:** The average time a roaduser has to wait at any given traffic light. This measure is a good indication of the congestion level within

the traffic network. The more cars (stuck) in the network, the higher the average junction waiting time.

Note that, if the stuck car removal option is set, cars will not be able to wait in front of a junction for too much time. The average junction waiting time will, in this case, be low.

**Total waiting queue length:** If the lanes leading from an edge node are fully congested, new cars entering the network at that edge node will be put in a waiting queue. This measure is the sum over the waiting queue lengths at all edge nodes.

An increasing total waiting queue length indicates that the network cannot handle the load of new cars trying to enter. If the measure decreases congestions are being resolved, as there finally becomes room available for the queued cars.

## 5.2 Accidents and Car Rerouting

As accidents have been added to the model, it seems obvious that cars should be allowed to re-plan their routes around the accident area. To investigate the validity of this claim, three test cases were designed to evaluate TC-SBC, TC-SBA and TC-SBAC using accidents. Per test case, the performance of one algorithm is measured using accidents without car rerouting, and using accidents with car rerouting.

### 5.2.1 TC-SBC

The TC-SBC algorithm was not designed with accidents in mind. As figure 2a clearly demonstrates, not enabling car rerouting leads to enormous junction waiting times very quickly. But if there is rerouting, this algorithm works well.

### 5.2.2 TC-SBA

Next, the TC-SBA algorithm is tested. Its use of an explicit representation of accidents might make one believe it should outperform TC-SBC on these test cases. However, comparing figure 2b to figure 2a shows that this is clearly not the case. Rerouting does, on the other hand, prove to be very useful for this algorithm as well.

Rerouting cars around an accident area may lead to new traffic bottlenecks, as other the traffic load will be redistributed over the other (so far unaffected) lanes. TC-SBA is not equipped to identify such spontaneous congestions. TC-SBC is, which can explain why it outperforms TC-SBA.

### 5.2.3 TC-SBAC

TC-SBAC can learn $Q$-values using both congestion and accident information. It is expected that it performs better than both TC-SBC and TC-SBA. Figure 2c shows that again that rerouting improves performance. Also, TC-SBAC is indeed outperforming TC-SBC and TC-SBA, according to expectations.

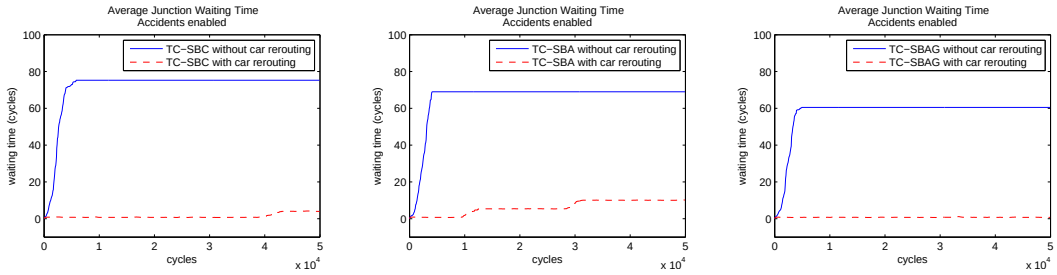


Figure 2: From left to right the average junction waiting time for: TC-SBC (a), TC-SBA (b) and TC-SBAC (c). Tests were performed with accidents enabled, but no stuck car removal and with using rerouting (continuous line) and without (dashed line).

The three test cases have shown that the RL based controllers benefit greatly from rerouting cars in the accident enabled simulations. Moreover, rerouting vehicles around accident areas is an intuitively preferred model. TC-SBA has shown to perform worst of the three algorithm, while TC-SBAC performs best.

## 5.3 Stuck Car Removal

As was previously mentioned, stuck cars can be removed. When a car is removed it is given a penalty $p$. This parameter can be varied and its value determines the impact on the RL algorithm.

Empirical evidence showed that a penalty of zero provided the best results. A higher penalty did not made the RL algorithm perform any better. In fact, the penalty is out of proportion with the learned $Q$-values. As a consequence, performance decreases. The results imply that cars waiting for a long time accumulate sufficient negative reward and there is no need for an extra penalty.

Hence, in the following sections we will describe the difference between not removing and removing cars without extra penalty.

### 5.3.1 TC-SBC

In figure 3 the results are shown for the *Total Waiting Queue Length* with and without removing of stuck cars. As can be seen in figure 3a, when both accidents and rerouting enabled, but even better in figure 3b, when accidents are enabled but rerouting is disabled, TC-SBC needs stuck cars to be removed in order to continue the simulation. Without it, the traffic network gets congested. This happens around cycle 40,000 for at least one simulation in figure 3a and around cycle 5000 in figure 3b. As soon as the traffic network is congested, cars will not arrive on their final destinations and no further learning is possible.

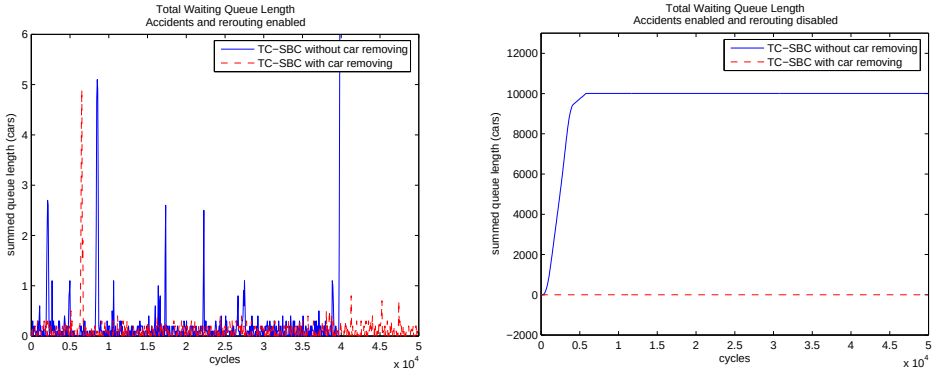


Figure 3: Total waiting queue length results for TC-SBC. On the left (a) both accidents and rerouting are enabled, on the right (b) accidents are enabled but rerouting is disabled.

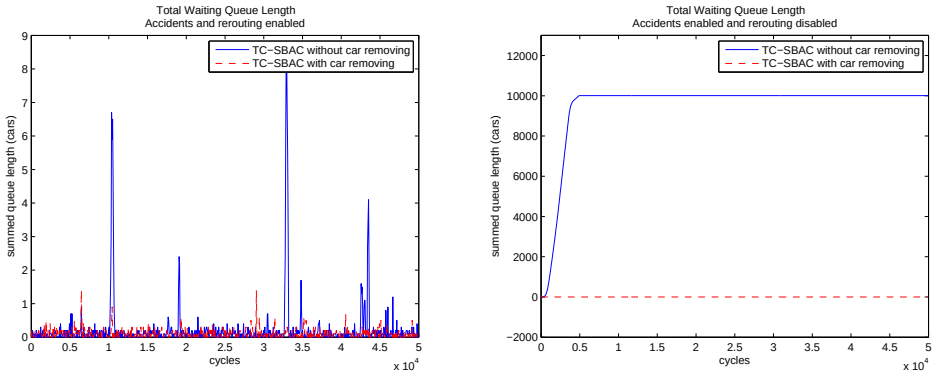


Figure 4: Total waiting queue length results for TC-SBAC. On the left (a) both accidents and rerouting are enabled, on the right (b) accidents are enabled but rerouting is disabled.

### 5.3.2 TC-SBAC

On the contrary, as long as both accidents and car rerouting are enabled, TC-SBAC is able to cope with the traffic even when no cars are removed (Figure 4a). Nevertheless this situation will not hold when car rerouting is disabled, see Figure 4b.

## 5.4 Comparison

The tests over the TC-SBC, TC-SBA, and TC-SBAC algorithms provided several useful comparisons. However, it is also important to compare these traffic controllers to the TC-1 Optimized, and a hand-coded one: *TC-Most Cars*.

TC-Most Cars is a simple TC algorithm that selects per junction the traffic light configuration that allows most cars to move. Obviously, no machine learning technique

is used and behavior is not adaptive. In a preliminary investigation, this controller appeared to work well in several cases.

Tables 2 and 3 present an overview of the five algorithms for respectively total waiting queue length, and average junction waiting time. The numbers are taken by averaging each measure over the last 10% of the cycles, as at that time, the controllers should have had sufficient time to learn.

Three configurations of the traffic model were used. The first one has no accidents (thus no rerouting either) and no stuck car removal, the second does use accidents, including rerouting, and no stuck car removal, while the third configuration has accidents, rerouting and stuck car removal (with no additional penalty). The bold numbers indicate which of the four RL based controllers works best per configuration.

Total Waiting Queue Length

| | *acc: no, rerout: no stuck removal: no* | *acc: yes, rerout: yes stuck removal: no* | *acc: yes, rerout: yes stuck removal: yes* |
|---|---|---|---|
| TC-SBC | **4001.93** (5165.97) | 1000.84 (3164.09) | 0.109091 (0.0659942) |
| TC-SBA | 7005.36 (4834.03) | 2000.96 (4218.23) | 0.118182 (0.057902) |
| TC-SBAC | **4001.93** (5165.97) | 0.136364 (0.0842211) | **0.0844156** (0.0214275) |
| TC-1 Optimized | 8005.13 (4219) | **0.120779** (0.0419936) | 0.119481 (0.0907647) |
| TC-Most Cars | 7372.88 (4355.75) | 6003.53 (5166.52) | 0.288312 (0.475119) |

Table 2: Averaged *total waiting queue length* (including variance over 10 runs) over cycles 45000 - 50000.

Average Junction Waiting Time

| | *acc: no, rerout: no stuck removal: no* | *acc: yes, rerout: yes stuck removal: no* | *acc: yes, rerout: yes stuck removal: yes* |
|---|---|---|---|
| TC-SBC | **8.29086** (11.8978) | 4.11925 (10.3684) | 0.75311 (0.0886949) |
| TC-SBA | 17.5943 (24.5797) | 10.0754 (19.6350) | 0.753426 (0.0622103) |
| TC-SBAC | **8.29086** (11.8978) | 0.757305 (0.0647718) | **0.740379** (0.0573645) |
| TC-1 Optimized | 13.9696 (13.1718) | **0.749236** (0.0613753) | 0.743713 (0.0780901) |
| TC-Most Cars | 7.80976 (2.99729) | 48.7703 (44.2321) | 1.21792 (0.111205) |

Table 3: Averaged *average junction waiting time* (including variance over 10 runs) over cycles 45000 - 50000.

In the case of no accidents TC-SBAC will always access only half of its state space, corresponding to the state space of TC-SBC. It is therefore no surprise that these two algorithms perform exactly the same in the first configuration.

Besides, the other two RL controllers perform much worse. The high total waiting queue lengths indicate that those TC's lead to large congestions throughout the network. This is supported by the average junction waiting time results.

In the second configuration where there are accidents, TC-SBC and TC-SBA are

unable to learn a good policy. TC-1 Optimized outperforms all other algorithms. Still, TC-SBAC does not do that much worse.

For the third configuration, stuck car removal leads to almost equal average junction waiting times for all four RL controllers. This result was expected, as explained in section 5.1, and is part of the motivation to use the total waiting queue length measure. The latter shows that the algorithms perform quite similar as well, but the differences are a bit more pronounced, in favor of TC-SBAC.

Interestingly enough, the average junction waiting time seems to suggest that the hand-coded controller performs extremely well in the first model configuration. But inspection of the total waiting queue length indicates, on the contrary, that the network is not congestion free. Congestions appear mostly at the edge nodes, while other junctions have good traffic flow. Furthermore, handling the dynamics accidents introduce to the simulation is beyond TC-Most Cars' capability.

# 6 Conclusion and Discussion

We have shown RL based traffic controllers can deal with accidents. Using accident knowledge to the advantage of the controller, such as the accident bit, is useful. Still, simply extending TC-1 with such a bit is insufficient. The case of TC-SBA shows, that as soon as the accidents are dealt with, it must also be dealt with congestions (chapter 5.2.2).

Tables 2 and 3 show a significant impact of introducing accidents and related improvements on the increased performance of *all* algorithms. It must be noted, that there are more new changes introduced to the simulation with accidents. The most significant ones seem to be rerouting and redirecting. Even if the rerouting is disabled, the lanes are, due to an accident, *redirected* in order to stop incoming traffic to the accident spot (chapter 3.2.1). All the cars are *forced* to take a new route.

As it was mentioned in chapter 3.2.1, the probability of an accident is proportional to the number of cars present on the lane. Accidents are, like in the real world environment, more likely to appear in congested areas. A consequence of an accident in the congested area and the forced redirection of lanes is, that cars, so far waiting to enter the problematic congested road, are forced to take another path to their destination. This redirection then stops the propagation of the congestion, because these cars are no longer waiting to enter the next road.

This might be intensified even more by the chosen level of accidents and their average duration. For the purposes of the simulation the probability of an accident is set in such a way, that there is on average one accident in the city all the time. However, the average duration of an accident is relatively short in the simulation compared to the real world. We believe, that this rapidly changing environment, where accidents appear and disappear continuously, suits TC-1 Optimized very well. It has a very small state space, and hence it can adapt much faster, than the more advanced algorithms that need more time to stabilize their behavior. Traffic in a real city might be expected to behave somewhere in-between the configuration with accidents and the configuration without

accidents.

The removing stuck cars option, artificially added to help the controllers to learn, does not correspond to what happens in the real world, and also does not bring any performance gain. Thus we suggest not to use it in further research.

# 7 Future Research

Several issues could benefit from future research, or could even be addressed by applying a few basic modifications to the model.

## 7.1 Different Accidents Pattern

As it is mentioned above, the amount and duration of accidents might play a crucial role. The correct setting resembling the real world scenario ought to be found. It might be expected, that congestion-aware algorithms (TC-SBC, TC-SBAC) will benefit from the configuration closer to our no-accidents-configuration.

## 7.2 Time Based Path Planning

As is shown in this paper, rerouting plays a crucial role when dealing with accidents. When accidents appear, rerouting is necessary. Even an artifact of rerouting - redirecting lanes - seems to improve the performance of the traffic control.

As it can be observed in the real world, road users avoid congested areas and find a path, that might be longer in distance, but shorter in travel time. It is straightforward to use a path planning algorithm, which is time instead of distance based. Such algorithm would take the current traffic status into account.

Because all the intelligence in our model is incorporated in the traffic lights, such a change would not require any extra computation from the cars. The traffic information from the TLC could be used to navigate cars. This would bring traffic control beyond the limitations of traffic light control.

Intuitively, such adaptation would reduce the traffic overhead in those junctions that have the most cars passing through them. It could spread the traffic in a more efficient manner, resulting in less states where some junctions are 'jammed' and others that are completely congestion-free.

## 7.3 State space Representation

The extension of the state space for dealing with accidents adds only one bit but results in a state space which is twice as large as the former. This causes a major increase in the memory requirements and learning time. For example, one way of reducing this complexity would be by representing the congestion and accident status in one and the same bit. From a certain point of view, congestions and accidents have the same effect: the road user cannot pass through.

The main difference is how this information is propagated throughout the network, as mentioned in 4.3. Information about accidents is propagated further through the network than that about congestions. In the used model, road users are aware if their path goes through an 'accident area' two junctions ahead. But the similar information on congestions is available only one junction ahead. A research topic could be to explore possible ways of minimizing the state space while maximizing the distance information is propagated through the network.

# References


[1] R.S. Sutton and A.G. Barto. Reinforcement learning: An introduction *MIT Press, Cambridge, MA*, 1998.

[2] K. K. Tan, M. Khalid, and R. Yusof. Intelligent traffic lights control by fuzzy logic. *Malaysian Journal of Computer Science*, 9-2, 1995.

[3] H. Taale, T. Bäck, M. Preuss, A. E. Eiben, J. M. de Graaf, and C. A. Schippers. Optimizing traffic light controllers by means of evolutionary algorithms. In *Proceedings of EUFIT'98*, 1998.

[4] M. Steingröver, R. Schouten, S. Peelen, E. Nijhuis, and B. Bakker. Reinforcement Learning of Traffic Light Controller Adapting to Traffic Congestion. *Intelligent Autonomous Systems group, Informatics Institute, University of Amsterdam*, 2005.

[5] M. Wiering. Multi-agent reinforcement learning for traffic light control. In *Proceedings of the Seventeenth International Conference (ICML'2000)*, pages 1151-1158, 2000.

[6] M. Wiering, J. van Veenen, J. Vreeken, and A. Koopman. Intelligent traffic light control. *Technical report, Dept. of Information and Computing Sciences, Universiteit Utrecht*, 2004.

[7] M. Wiering, J. Vreeken, J. van Veenen, and A. Koopman. Simulation and optimization of traffic in a city. In *IEEE Intelligent Vehicle symposium (IV'04)*, 2004.