The Traffic Light Lane Queue Agent of Team SUMO

Robert Alms
German Aerospace Center, Institute of Transportation Systems
Berlin, Germany
robert.alms@dlr.de

Michael Behrisch
German Aerospace Center, Institute of Transportation Systems
Berlin, Germany
michael.behrisch@dlr.de

Jakob Erdmann
German Aerospace Center, Institute of Transportation Systems
Berlin, Germany
jakob.erdmann@dlr.de

Yun-Pang Flötteröd
German Aerospace Center, Institute of Transportation Systems
Berlin, Germany
yun-pang.floetteroed@dlr.de

Peter Wagner
German Aerospace Center, Institute of Transportation Systems
Berlin, Germany
peter.wagner@dlr.de

ABSTRACT

A simple rule based algorithm is described which optimizes a single intersection and is tested and parameter-optimized against a whole network. The algorithm observes/calculates the queue-lengths upstream of the intersection and also possible congestion downstream. It favors phases where two lanes which currently have demand get green to maximize the outflow.

CCS CONCEPTS

• **Applied computing** → **Engineering**.

KEYWORDS

traffic simulation, traffic lights, optimization

ACM Reference Format:
Robert Alms, Michael Behrisch, Jakob Erdmann, Yun-Pang Flötteröd, and Peter Wagner. 2021. The Traffic Light Lane Queue Agent of Team SUMO. In . ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The City Brain Challenge 2021 provided a platform to test traffic light control algorithms against a real city network. The simulation scenario consists of a street network together with an exemplary traffic demand and the simulation tool itself provided in a docker container. Every 10 seconds the algorithms under test provided speed and location of every vehicle currently in the network. It could switch the traffic light phase at every junction accordingly. While the network was known to the algorithm, neither the testing demand nor the source code of the simulator were available. The input demand in the final phase covered only 20 minutes.

The primary idea of the challenge was to test machine learning algorithms in the traffic light domain, probably by setting up a simple algorithm which can then be parameterized individually for every junction possibly, even over time.

Our team SUMO[1] however took a different approach which was to check how far we can get with a general purpose algorithm. Essentially, the algorithm works the same on all junctions and maybe also estimates the effect of the objective function on the whole system. From a traffic management perspective this idea corresponds to a decentralized approach without deterministic coordination between intersections like [1],[2] (and different from centralized approaches as [3]), though, the algorithm aims to predict traffic states downstream of individual intersections and therefore might carry coordination aspects inherently.

In the following, we give a description of the implemented algorithm together with a very rough ad hoc estimation on the usefulness of the different improvements, ending with some remarks on the general simulation setup and the objective function.

2 DESCRIPTION OF THE ALGORITHM

The code can be divided into a pre-processing step and the run-time algorithm, where the pre-processing is nothing more than an analysis of the test route set for the expected remaining travel time. By evaluating the given `flow_round3_flow0.txt` as well as two additional demand files generated by the provided `traffic_generator.py`, an average value for the remaining free flow travel time for vehicles travelling a certain edge can be calculated. This value is of course only valid for the given demand, but since it depends on the structure of the network and it is to be expected that also the final demand will be generated with a script similar to the `traffic_generator.py`, the data has been used for the final demand as well. The basic structure of the run-time algorithm has four steps:

(1) aggregating vehicle information
(2) evaluating lane queues
(3) choosing the best phase(s)
(4) ensuring a maximum green time and removing long term jams

[1]Remark: The name of the team stems from a simulation suite *Eclipse SUMO* (www.eclipse.org/sumo/) which is being developed mainly at our institute and supported a lot of our analysis.

City Brain Challenge, KDD Cup 2021, April 01–June 05, 2021, Online

https://doi.org/10.1145/1122445.1122456

This is the order in which the steps are performed. For better understanding we will describe the central step (phase choosing) first.

2.1 Choosing the phase

The central part of the algorithm iterates over all traffic light junctions (agents) and takes as an input a weight for each incoming lane (think of it as an elaborated queue length), the computation of which is described in detail below.

Having a queue length for each incoming lane we still can choose between different phases which maximize throughput. The initial assumption of simply giving green to the phase which maximizes the sum of the queue lengths of the lanes, which are given green, did not hold. Instead we choose an approach favouring phases that can service two queues of waiting vehicles over those phases which service a single queue as long as each of the two queues is long enough (PREFER_DUAL_THRESHOLD). As a second criterion for deciding between candidate phases, the currently active phase is preferred as long as it has a sufficiently long dual queue or a minimum of number of single queued vehicles (10 / HEADWAY). As a third criterion, phases with longer queues are preferred.

Furthermore the longest queue needs to extend the current queue by some SWITCH_THRESH which models switching cost (the all red time between phases). After all it may still occur that certain phases do not get switched to and single vehicles starve, so after a MAX_GREEN_SEC the phase always gets switched.

2.2 Vehicle Information

We implemented a self-made observation function which forwards the complete results of `self.eng.get_lane_vehicles()` to the agent that is a dictionary mapping each lane of a traffic light controller (an agent) to the list of vehicles on that lane.

The agent itself extracts the necessary information for the vehicle. Firstly, it tries to rebuild the historic route of the vehicle by finding a shortest path between the previously observed and the current position of the vehicle. In this way, the driven part of the routes of all vehicles currently in the scenario is known. If the vehicle route ended already, a special marker is appended to the stored route.

Furthermore the free flow travel time for the driven route is calculated and stored for the vehicle.

After all initial vehicle info has been evaluated, we can calculate for each edge a distribution of possible future routes such that one gets a (continuously updated) probability distribution for the future route of each vehicle.

2.3 Lane Queues

The second step is to calculate the lane queue lengths for every incoming lane at a junction. The basic idea is that the priority when a lane gets green time should depend on the number of waiting vehicles on that lane. But not all vehicles on the lane are *equal* in the sense that they have different chances to pass the junction in the next interval, either because of their distances to the junction or because of their destination lanes being jammed. Vehicles have also different expected contributions to the objective function (see the Sec. 5). Consequently, the vehicles are assigned different weights.

While calculating the lane queues at a junction, there is repeatedly the need to determine whether the destination lane(s) for an incoming lane can still accept vehicles or if it is jammed. So the algorithm calculates for every outgoing lane whether the lane either has many vehicles and a low average speed or a lot of vehicles in the insertion buffer[2] Those lanes are marked as jammed.

Moreover, the vehicles which contribute to the queue are determined by checking their time distance (i.e. speed over distance) to the stop line against a threshold and respecting (almost) standing vehicles. For those vehicles the probability that their destination lane is jammed is calculated. If all destination lanes are jammed the probability is 1, if all lanes are free it is 0. For all the remaining cases the possible future routes are evaluated from the probability distribution determined earlier.

Summing over all possible future routes, it is possible to determine a probability that the target lane of the vehicle is jammed. In case of an unknown future lane, a constant is used that is subject to calibration. This gives for each vehicle the probability p_j to have a jammed lane ahead. The second important factor is the (expected) free flow travel time t_f of the vehicle (see Sec. 5 for a detailed discussion). Since the expected remaining travel time has been determined by the pre-processing, the free flow time up to the current edge is known precisely.

The weight of a vehicle can be determined using the following equation:

$$w = \frac{T}{t_f}(1 - p_j) \tag{1}$$

where T is the average expected free flow travel time over all vehicles (again a constant which can be calibrated). Please notice that the value may range from 0 to ∞, but a *regular* vehicle should have a weight of about 1.

The weight of a lane queue is then simply the sum over all weights of the vehicles considered relevant. In order to cover the case of short edges leading to a junction, where upstream jams may extend to different edges, this evaluation is also extended to all predecessor lanes on upstream roads, but only if there is no traffic light junction in between.

2.4 Long term jams

For every lane which has more than a defined number of vehicles on it and operates below a defined speed threshold, a jam bonus is collected over time leading to eventually resolved long term jams. This jam bonus is added to every lane right at the end of the queue length calculation. Unlike the queue length above which is recalculated on every iteration, this bonus adds up over time until a switch to a different traffic light phase occurs.

3 IMPROVEMENTS

The basic algorithm as described involves several thresholds and constants which need to be adapted. In total, the algorithm has 22 parameters, which were optimized individually (see table 1). In order to do so, first the possible ranges for all parameters are determined. Then a rastering of the value range has been done by using equidistant samples (usually ten) in the range of values. In

[2]The insertion buffer is a region at the beginning of the lane where vehicles are placed which want to enter the lane but have not found a suitable position yet.

this way, almost optimal values with respect to the test data set could be achieved.

In the process, several automated multidimensional optimization schemes like `optimparallel`, `scikit-optimize` or `scipy.optimize.fmin_cobyla` have been tested as well, but due to the very fragile environment (high random fluctuation of the results) they were considered too unstable.

4 EVALUATION USING SUMO

In order to evaluate the provided scenario, we used the microscopic traffic simulator SUMO [4].

To visualize the traffic state at certain time steps, we utilized the *sumo-gui* by loading the given network and current demand based on the *info_step*-outputs from the CBengine. Fig. 1 shows a screenshot of such a traffic state, illustrating the demand on each edge. The color of each vehicle depends on their current speed level.

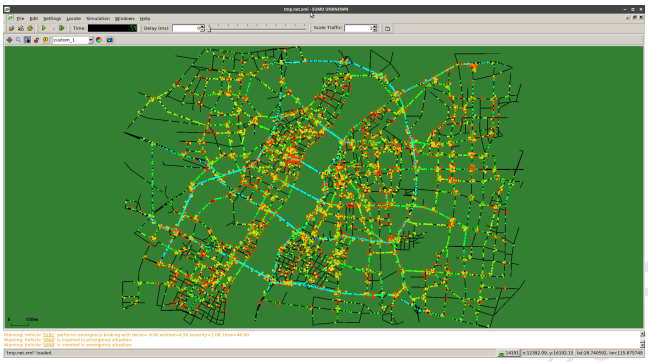

Figure 1: Snapshot of SUMO simulation for the given network with vehicles colored by speed level

Considering the previously mentioned insertion buffer, we knew that the illustration in Fig. 1 only was a rough approximation of the actual traffic state. Therefore we further analyzed the demand, by plotting the missing data, which was vehicles waiting to be inserted into the scenario, with the help of the *sumo-gui*. Fig. 2 presents the given network with coloured edges depending on their status of vehicles waiting to be inserted.

This data gave us the insights in regions, where the algorithm still did not perform well enough. Furthermore the SUMO interface enabled the analysis of the routes of each individual vehicle as well as an easy comparison of different states of the traffic network at different time steps. Last but not least we could spot some interesting anomalies of the simulation engine such as vehicles with speed 0 at the very beginning of an edge which were not part of the insertion buffer as they did not start their route on that edge. Unfortunately not all of these issues could be resolved until the end of the competition.

5 OBJECTIVE FUNCTION

In the context of traffic light optimization the primary goal of a control algorithm is usually to maximize the traffic flow at each controlled intersection when looking at the a single intersection or minimizing the sum of the time loss for all vehicles when looking

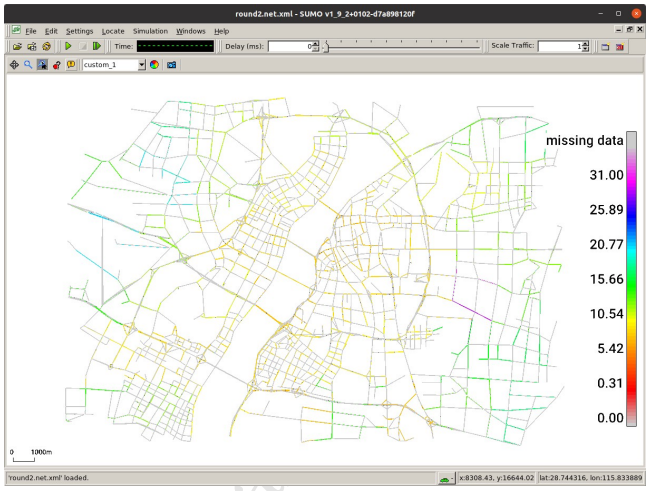

Figure 2: Snapshot of SUMO with given network; edges coloured by vehicles waiting to depart

at the network level. Evaluating this metric is mostly done for a whole day or at least the peak hour(s).

During the contest the assumed objective was to *serve* the maximum number of vehicles, where served really meant to get past the intended departure time of a vehicle without hitting the threshold of the average travel time loss.

There are several problems with this approach which also influence the algorithm quite heavily. The most important one is the calculation of the time loss function for an individual vehicle which is essentially a quotient of the real and the free flow travel time. By calculating the time loss this way, it is beneficial to favor vehicles with short routes (and therefore short free flow travel times), which shows clearly in the strong weight given to those vehicles.

Furthermore, as showed up during the competition, the total evaluation did not really measure the number of vehicles, which made it through the network (or even onto the streets), but only the time it took until the arbitrarily chosen threshold of 1.4 in average delay has been reached. In this way, junctions, which were already crowded, could not benefit from getting one more vehicle on the street (because it made no difference if the vehicle is in the buffer or on the street) and also vehicles, which could make it to their destination, did only a small contribution to the overall result.

Last but not least, the traffic, generated in the final round, only covered a period of about 20 minutes and usually the algorithms did not even reach that time before hitting the threshold. While this is a very short period for an evaluation in itself (it basically only covers the time to fill the network to lockdown), it also limits the amount of data, which can be collected for running vehicles and also on origin destination relation, since only very few vehicles make it to their destinations.

Collecting all these difficulties our team noticed that there are really different views on the optimization problems, which occur in the context of traffic lights, and we could not make too much progress with the algorithms already at hand.

Table 1: Parameters of the model

Name	Final value	Description
DUAL_SWITCH_THRESH	0.1	difference in queue lengths which triggers switch optimized
SWITCH_THRESH	0.5	difference in queue lengths which triggers switch optimized
ROUTE_LENGTH_WEIGHT	689.0	calibration term for factoring estimated free flow travel time into vehicle weight
MIN_CHECK_LENGTH	29.0	look upstream to find more queued vehicles if a lane is shorter than this
JAM_THRESH	0	at which relative occupancy a lane is considered jammed
SPEED_THRESH	0.025	at which relative speed a lane is considered jammed
JAM_BONUS	0.1	bonus vehicles to add to a jammed lane per act call (every 10s) until it gets green
HEADWAY	2.05	Time gap between vehicles to estimated minimum queue length that justifies switching
SLOW_THRESH	0	at which relative speed a vehicle is considered slow, to be optimized
MAX_GREEN_SEC	164.0	maximum green duration for a single phase
PREFER_DUAL_THRESHOLD	4.0	How many vehicles could pass the intersection in 5s all red (if we have more vehicles in a dual queue switching, beats single queue discharge)
STOP_LINE_HEADWAY	10.8	seconds to the stopline to be included in the queue
BUFFER_THRESH	1.0	number of vehicles in lane insertion buffer to consider lane jammed
FUTURE_JAM_LOOKAHEAD	3	number of future junctions to check for traffic saturation
SATURATED_THRESHOLD	44	initial threshold for classifying junction as saturated
SATURATION_INC	5	per-junction discount of saturation threshold
MIN_ROUTE_PROB	0.9	calibration factor for route length estimation in preprocessing
MIN_ROUTE_COUNT	7.0	calibration factor for route length estimation in preprocessing
DELAY_WEIGHT	35.0	calibration term for factoring estimated free flow travel time into vehicle weight
BUFFER_JAM_THRESH	1.8	threshold for accepted number of jammed vehicles on the outgoing edge
BUFFER_SPEED_THRESH	0	threshold for speed on outgoing edge to detect jam
UNKNOWN_LANE_JAM_PROB	1.2	calibration term for estimating downstream jam if the target lane is not known

6 CONCLUSION

The algorithm developed performs quite well on the control task given by the KDD challenge, proving to be robust for various demand levels. Still, given the dense traffic state and continuously high demand throughout the scenario, often above the local capacity of intersections and edges, the algorithm has its limitation for further optimization of the entire network. As mentioned in the beginning, the algorithm corresponds to a decentralized traffic management approach, by optimizing junctions individually. In large networks with traffic conditions as given in the challenge, this approach can not maximize the throughout over multiple consecutive intersections, that affect each other, due to large backlogs and inflated demand patterns. We suspect, a scheme to coordinate multiple intersections within such a large network in order to solve regional congestion, caused by specific capacity limits in that area, would improve the overall network performance. We assume that this is where machine-trained models, capable to generalize better based on their training, might be superior to solve these specific problems for such traffic conditions.

7 OUTLOOK

All the resulting code will be made publicly available on GitHub once all publications in connection with the algorithm are done.

We plan to reevaluate our algorithm with a SUMO simulation on an available traffic scenario which we scale up to a total grid lock as in the contest in order to learn more about the spill back effects and how much looking into downstream junctions can benefit a traffic light algorithm. In this way, we can add another evaluation facet to our existing traffic light algorithms and extend them with more functionality tailored to those extreme situations.

ACKNOWLEDGMENTS

Thank you to the organizers of the competition for investing the time and effort to host it. We certainly learned a lot about traffic simulation engines as well as about evaluation scenarios.

REFERENCES

[1] N.H. Gartner. 1983. OPAC: A demand-responsive strategy for traffic signal control. *Transp. Res. Rec. J. Transp. Res. Board* 906 (1983), 75–81.

[2] Carlos Gershenson. 2005. Self-organizing Traffic Lights. *Complex Systems* 16 (2005). Issue 1. https://www.complex-systems.com/abstracts/v16_i01_a02/

[3] P.B. Hunt, D.I. Robertson, R.D. Bretherton, and R.I. Winton. 1981. *SCOOT-a Traffic Responsive Method of Coordinating Signals.* Technical Report Technicalreport; No. LR 1014 Monograph; Transport and Road Research Laboratory.

[4] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. 2018. Microscopic Traffic Simulation using SUMO, In The 21st IEEE International Conference on Intelligent Transportation Systems. *IEEE Intelligent Transportation Systems Conference (ITSC).* https://elib.dlr.de/124092/