

Optimizing Trafic Light algorithms in Urban Street Grids

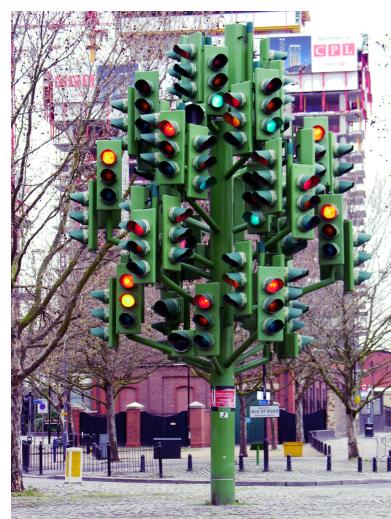
Introduction to Computational Science

Jan Kramer
Klaas Kliffen

January 29, 2016

Abstract

Abstract



Contents

1	Introduction	2
2	Related work	2
3	Concept	2
3.1	Grid of intersections	3
3.2	Algorithms	3
4	Implementation	5
5	Results	5
6	Evaluation	6
7	Conclusion	6
7.1	Future work	6
A	Result tables	6

1 Introduction

Waiting is never fun, especially if you have to get somewhere by car and you are encountering a red traffic light. Sometimes traffic lights work well, but sometimes it yields a green light for an empty lane, while another lane is packed with cars. This may not only lead to irritation, but also to dangerous situations if someone decides to ignore the traffic light out of frustration with the traffic light switching algorithm. In addition bad traffic light switching algorithms also affect the flow of traffic on a larger scale than a single traffic light.

Some effort has been put into researching algorithms and into simulating and using machine learning techniques for determining the best possible algorithms. Instead of using a complex algorithm, this report evaluates the performance of regular time-based approaches and simple loop detection.

Outline of this report

Section 2 discusses the use of reinforcement learning for traffic light algorithms. Section 3 describes the concept of our simulation. Section 4 goes into implementation details of the concept. Section 5 describes the testing set-up used and shows the experimental results. Section 6 discusses the results and Section 7 concludes. In the appendix Section A contains tables with statistics for simulations with various parameters.

2 Related work

The paper by Wiering [1] describes a global algorithm for traffic light switching. This algorithm is based on multi-agent reinforcement learning to minimize the overall waiting time of cars in the city. Which basically means that the traffic lights strive as a group to minimize the waiting time by finding a good switching algorithm. Wiering found that in his simulations that a reinforcement learning algorithm can outperform traditional non-adaptable systems.

3 Concept

Instead of complex adaptive learning algorithms, simple algorithms are used in this report. To simplify the calculations and to keep the scope small our simulation will abstract from the reality. Therefore we place the following restrictions on the simulation.

Firstly the simulation will only consider cars, there is no other traffic and thus no biking lanes or pedestrian lights. The reason is that car traffic by itself can be made quite complex already. Secondly to reduce this complexity

all cars will be considered all equal. So there is no difference between a large truck and small car. We choose this restriction to simplify the implementation. In addition cars do not have complex behavior, they just drive forward and choose randomly which lane to pick when entering an intersection.

Thirdly we simplify the traffic network, since it normally can take a quite organic shape. This organic shape is hard to model and often unique to a city. So the roads will be generalized to a grid, in which each car takes up exactly one grid space. Time will be discretized by allowing cars one action per time step:

- If not in front of light: move forward one space in the current grid if the next grid space is empty, otherwise wait.
- If in front of light: move to the next grid space when the light is green, otherwise wait

3.1 Grid of intersections

The real world is broken down to intersection. Each intersection consists of a number of lanes for each direction. For each lane a destination direction is set. When a car passes the traffic light, it passes on to the next intersection. This is done by keeping track of a grid of intersections. Each direction of an intersection is connected to another intersection. This way a car can navigate from one intersection to another. Complex graph like structures can be created this way, depending on the number of directions.

Each intersection has 2 traffic lights per incoming direction. One for turning left and one for turning right or going straight ahead. We do not allow for states at which collisions between cars can happen. So there are two possible algorithms for the traffic lights.

3.2 Algorithms

For this assignment two different algorithms will be used. There will also be a version of both algorithms with loop detection. A time stamp will be used as an input for each of the algorithms, along with a variable for setting the amount of time-steps for a light to switch.

Simple

The simple algorithm is based on old-fashioned traffic handling by a human in the center of the intersection pointing at the lane which may move. At each time-step, one of the four directions will get the green light and cars can move. After a given amount of time-steps, the next direction will be set to green and the current direction set to red. In pseudo code:

```

1 SET direction TO (time stamp DIV switch time) MOD number of
  directions
2 SET lights of lanes in direction TO green

```

Two Sided

The two sided algorithm uses the fact that two opposing lanes can move at the same time. Cars moving straight forward or turning right can move at the same time as the cars on the opposing lane moving forward and right. The same is true for the lane turning left. In pseudo code:

```

1 SET direction TO (time stamp DIV switch time) MOD number of
  directions
2 SET light of the lane in the direction TO green
3 SET the light in the opposing lane TO green

```

Loop detection

The loop detection can be applied to both of the previous algorithms. For this, each traffic light need to keep track of which lanes currently have green light and at what time stamp the last change happened. It also needs to keep track of the front of each lane and whether a car is present in it. It will only set lights to green if the next lanes according to the algorithm are not empty. If it encounters an empty lane, it will look for the next lane. If the next lane is not empty, set those lights to green. If all lanes are empty, keep all lights red and wait for a car to appear. If the change time has not yet expired, check if there are still cars present. If not, schedule a new change. In pseudo code:

```

1 SET direction TO stored direction
2 IF time stamp - last change time >= switch time THEN
3   SET direction TO direction from algorithm
4   IF front of lanes are filled THEN
5     SET lights of the lanes from the algorithm TO green
6     SET last change time TO time stamp
7     SET stored direction TO direction
8   ELSE
9     FOREACH other direction
10    IF front of other lanes in direction is filled THEN
11      SET lights of that direction TO green
12      SET last change time TO time stamp
13      SET stored direction TO direction
14    END LOOP
15  ELSE
16    IF front of lanes in direction are empty THEN
17      SET last change time TO time stamp - switch time

```

4 Implementation

The grid of intersections consists of 9 intersections. Each intersection is connected to 4 others: top, left bottom and right. For boundary intersections, a periodic boundary condition is in place where cars moving out of the grid will reappear on the opposing side of the grid. The grid layout can be seen in Figure 1.

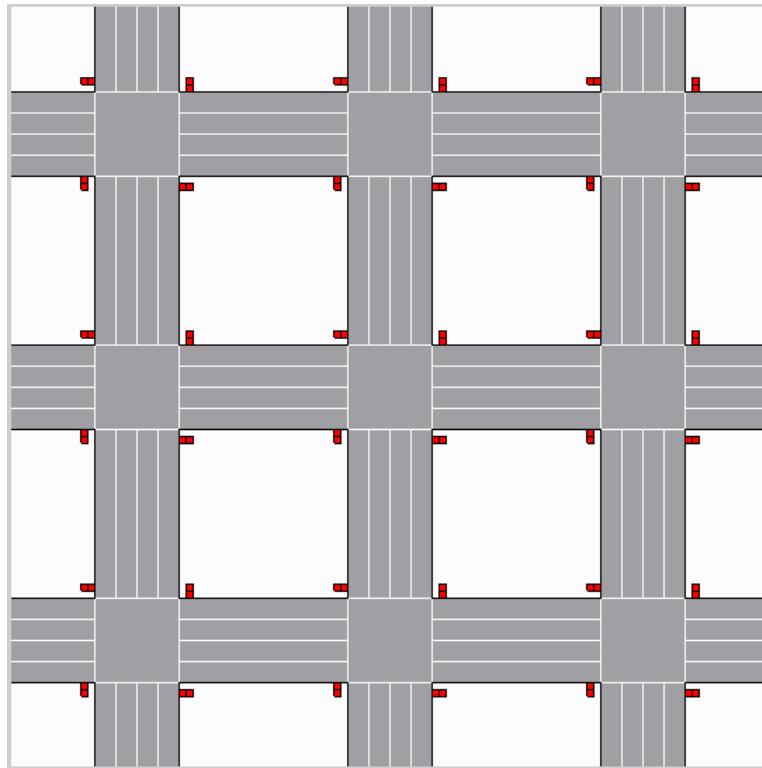


Figure 1: The empty intersection grid with all lights red.

5 Results

The simulations varied by using different number of cars: 50, 100 and 200, resembling different traffic loads. The different switch times used were: 2, 4, 8, 16 and 32. These were chosen since they were either a $\frac{1}{4}$, $\frac{1}{2}$, 1, 2 or 4 times the length of the lanes (8). For graphing purposes, each algorithm was simulated for 500 simulation steps. To get consistent results, the random generators are seeded with the same seed for each run.

6 Evaluation

Patterns

7 Conclusion

7.1 Future work

- Setting chance choosing between the left lane or the other independently. Now all is equal
- Giving cars incentive to drive to a certain destination
- Make the amount of cars change over time (rush hour effect)

References

- [1] WIERING, M. Multi-agent reinforcement learning for traffic light control, 2000.

A Result tables

cars	50	100	200
switch time			
2	0.049751	0.21393	0.39801
4	0.16418	0.26368	0.50249
8	0.19403	0.35323	0.32836
16	0.19403	0.33333	0.55721
32	0.19403	0.33333	0.25871

Table 1: Average minimum queue time for the simple loop algorithm

cars	50	100	200
switch time			
2	1.5822	3.301	6.9473
4	2.0785	4.2088	7.357
8	2.0209	4.6099	8.8599
16	2.0209	4.6005	9.7013
32	2.0209	4.6005	11.136

Table 2: Average queue time for the simple loop algorithm

cars	50	100	200
switch time			
2	3.8408	7.9751	17.866
4	5.0348	9.6169	16.716
8	4.9104	10.209	19.259
16	4.9104	10.358	22.612
32	4.9104	10.358	29.373

Table 3: Average maximum queue time for the simple loop algorithm

cars	50	100	200
switch time			
2	5.4776	9.2736	13.871
4	5.199	8.5522	13.448
8	5.2438	8.2388	12.214
16	5.2438	8.2687	11.592
32	5.2438	8.2687	10.796

Table 4: Average throughput for the simple loop algorithm

cars	50	100	200
switch time			
2	0.089552	0.16915	0.59204
4	0.1592	0.39801	0.46269
8	0.21393	0.37811	0.60199
16	0.21393	0.31841	0.47761
32	0.21393	0.31841	0.47761

Table 5: Average minimum queue time for the two-sided loop algorithm

cars	50	100	200
switch time			
2	1.3433	3.1006	6.7539
4	1.713	4.0883	7.5746
8	1.9167	4.6661	9.0356
16	1.9167	4.789	9.1792
32	1.9167	4.789	9.1792

Table 6: Average queue time for the two-sided loop algorithm

cars	50	100	200
switch time			
2	3.4179	7.8159	17.204
4	4.2786	9.0498	17.418
8	4.6169	10.751	18.97
16	4.6169	10.891	21.095
32	4.6169	10.891	21.095

Table 7: Average maximum queue time for the two-sided loop algorithm

cars	50	100	200
switch time			
2	5.6517	9.4478	14.03
4	5.408	8.6219	13.274
8	5.3134	8.2388	12.035
16	5.3134	8.1343	11.965
32	5.3134	8.1343	11.965

Table 8: Average throughput for the two-sided loop algorithm

cars	50	100	200
switch time			
2	0.54545	0.26368	0.49254
4	2.908	3.0955	2.8557
8	2.88	2.75	2.5721
16	5.2474	4.0588	4.0546
32	19.298	9.5122	8.1783

Table 9: Average minimum queue time for the simple algorithm

cars	50	100	200
switch time			
2	2.8838	3.7831	6.6417
4	6.0868	6.8707	8.4318
8	12.088	12.903	13.527
16	21.503	22.225	23.952
32	52.025	47.191	48.236

Table 10: Average queue time for the simple algorithm

cars	50	100	200
switch time			
2	4.3889	8.5721	17.139
4	6.8712	9.5377	15.498
8	13.51	14	17.876
16	17.423	17.662	19.53
32	35.553	30.878	30.318

Table 11: Average maximum queue time for the simple algorithm

cars	50	100	200
switch time			
2	4.796	8.8557	14.109
4	3.6667	6.9254	12.502
8	2.5423	4.8806	9.5572
16	1.7711	3.4428	6.5174
32	0.801	1.7214	3.4726

Table 12: Average throughput for the simple algorithm

cars	50	100	200
switch time			
2	0.54545	0.26368	0.49254
4	2.908	3.0955	2.8557
8	2.88	2.75	2.5721
16	5.2474	4.0588	4.0546
32	19.298	9.5122	8.1783

Table 13: Average minimum queue time for the two-sided algorithm

cars	50	100	200
switch time			
2	2.8838	3.7831	6.6417
4	6.0868	6.8707	8.4318
8	12.088	12.903	13.527
16	21.503	22.225	23.952
32	52.025	47.191	48.236

Table 14: Average queue time for the two-sided algorithm

cars	50	100	200
switch time			
2	4.3889	8.5721	17.139
4	6.8712	9.5377	15.498
8	13.51	14	17.876
16	17.423	17.662	19.53
32	35.553	30.878	30.318

Table 15: Average maximum queue time for two-sided algorithm

cars	50	100	200
switch time			
2	4.796	8.8557	14.109
4	3.6667	6.9254	12.502
8	2.5423	4.8806	9.5572
16	1.7711	3.4428	6.5174
32	0.801	1.7214	3.4726

Table 16: Average throughput for the two-sided algorithm