Project Report
Modelling and Simulation

Erik Bijl s2581582
Emilio Oldenziel s2509679

September 2017

1 Introduction

Traffic lights are one of the most common controllers of traffic flow nowadays. If we don't use traffic lights the traffic flow could become chaos. In order to handle the traffic flow, traffic lights must monitor cars and decide which cars can pass an intersection and which cars have to wait until the green light. Different methods can be applied to traffic lights resulting in different situations. We introduce a model to schedule and control traffic lights, with different strategies depending on the longest waiting queue or depending on the car that is waiting the longest time. In this project we model traffic lights at a singular intersection, and expand these to a 2D grid of intersections.

1.1 Research question

We are interested in the optimal behaviour of traffic lights under several conditions. To do this we want to simulate a traffic light setup and apply different approaches to lead the cars through. We want to discover which setup is performs best in each of the situations, with as question: Which strategies of traffic lights perform optimally during certain types of traffic conditions?

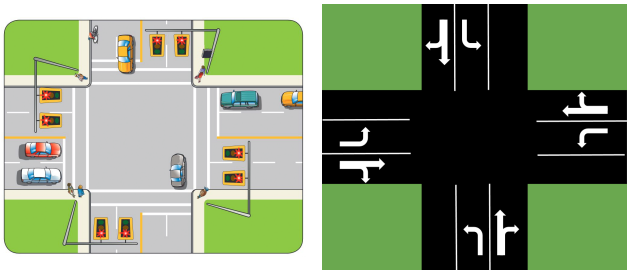
2 Single intersection

In this section we will describe our model, the implementation of this model and the results of a single intersection.

2.1 Description

Our intersection consists of 4 directions which we call: north, east, south and west. Each direction has three lanes for riding cars: one lane for cars going straight and to the right, one for cars going to the left and one lane for incoming

cars from other directions. The basic idea and abstract visualisation is given by figure 1. Each direction has two traffic lights for outgoing lanes.



(a) Visualisation of a traffic intersection

(b) Abstract view of our traffic intersection

Figure 1: Visualisation of intersection

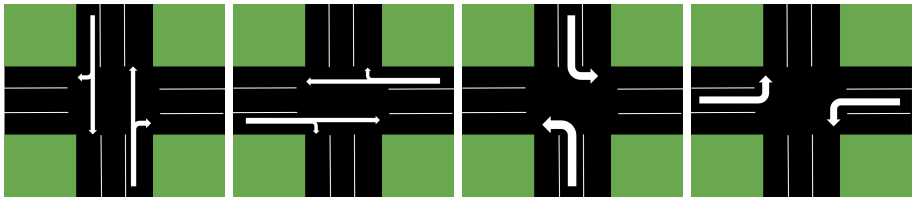
For these four directions the traffic lights can be green or red. We have four different situations where two traffic lights can be green for opposite positions and same directions. The remaining traffic lights are red which gives the following four situations where the mentioned traffic lights are green as illustrated by figure 2:

(a) North: straight and right and South: straight and right

(b) East: straight and right and West: straight and right

(c) North: left and South: left

(d) East: left and West: left



(a) (b) (c) (d)

Figure 2: The four situations of traffic lights that can be on green

As we mentioned earlier, each direction has two lanes of incoming cars. We see these lanes as a queue where arrived cars can be enqueued on and cars that drive away can be dequeued from. As one expects we use a first in first out (FIFO) strategy which lets a car pass when all cars in front have driven away. Enqueueing will be random so a car has an equal chance of starting at every

direction and lane. To put the traffic lights on green different methods can be used. One of these methods is Round Robin which equally divides the turns over the traffic lights. Another method is to measure the longest queue and put this direction on green. A third method could be to keep track of the car that has the longest waiting time and put that direction on green. Of course combinations of these can be made to find the best strategy and in practice we do as shown in the implementation part.
As we see in real life, not all cars take the same amount of time to drive away when a traffic light is green. Therefore, for each car that will drive away we will set a time that it takes for a car to drive away.

2.2 Implementation

Considering a traffic light intersection T with the set of queues Q consisting of $Q_{i,j}$ where $1 \leq i \leq 4 \wedge 1 \leq j \leq 2$ which has length N_i, j. Here i is the light, starting at 1 with north and going clock-wise. j is the left or right queue for the traffic light. We want to find the $Q_{i,j}$ which has the most priority. To do this we take N_i for all queues and the car with the longest waiting time $C_i^{max} \in Q_i$, there are defined as:

$$C_{i,j}^{max} = max(C_k^{waiting} | C_k \in Q_{i,j} \in Q) \tag{1}$$

$$T_{green1} = argmax(N_{i,j} * C_k^{max} | Q_{i,j} \in Q) \tag{2}$$

$$T_{green2} = (mod(T_{green2_i} + 2, 4), T_{green2_j}) \tag{3}$$

To determine the amount of seconds that the situation needs to be on green described by T_{lock}, we take the length N_{green1} of the queue T_{green1} and the length N_{green2} of the queue T_{green2}. Then T_{lock} is described by

$$T_{lock} = max(N_{green1}, N_{green2}) \tag{4}$$

By doing this we put the lock in such a way that it is possible to let all cars $C \in Q_{i,j}$ pass if the C^{lock} for all cars in Q is set at 1 second.

2.2.1 metrics

We use two metrics for measuring the performance of the traffic light intersection. The first one is average waiting time per Queue in every round described as:

$$T_t^{wait} = \frac{\sum_{i=1}^{4} \frac{\sum_{j=1}^{N_i} t - C_j^{t_{in}}}{N_i}}{4} \tag{5}$$

Where y is the current round and t_{in} is the round where the car was added to the queue.

The second one is the maximum waiting time W over all queues:

$$W = max\{C_{i,j}^{max} \text{ for all } i \in \{1, ..., 4\}, j \in \{1, 2\}\} \tag{6}$$

2.2.2 Input types

In our model we have a possibility of three input types: These are the following given by iteration t, amount of iterations T and amount of cars per round n:

1. Constant flow: At each iteration the amount of adding cars will be equal. Given by the following function:

$$n(t) = 1$$

2. Rush hour: At the beginning of the simulation there will be a peak of adding cars what will result in a lot of traffic at the intersection. Given by the following function:

$$n(t) = \begin{cases} 10 \cdot \sin(\frac{t}{T \cdot \pi \cdot 10}), & \text{if } t < T/10 \\ 0, & \text{otherwise} \end{cases}$$

3. Frequent peaks: Cars will be added as peaks on a regular basis. A situation that will occur when cars are passed to other traffic lights. Given by the following function:

$$n(t) = \begin{cases} 2, & \text{if } \mod(x, 5) == 0 \\ 0, & \text{otherwise} \end{cases}$$

2.2.3 Running the simulation

The simulation can be started by the following function call:

```
single_intersection(rounds, input_type, green_mode, lock_mode)
```

The parameters are defined as:

- Rounds, the amount of rounds we want to schedule. In each round cars can be added, drive away and different traffic lights can be changed to green.
- input_type, the method that will schedule the cars at each iteration. The following options can be chosen:
    1. Constant flow
    2. Rush hour
    3. Frequent peaks

- green_mode, the method that is used to schedule which traffic lights are green. The following options can be chosen:
    1. Monitored, a method that monitors the length of the queue and combines this with the car that waits as longest.
    2. Round Robin, a method that equally divides the time over each traffic light.
- lock_mode, the method that determines how long a traffic light will be on green. The following options can be chosen:
    1. Let cars pass till all currently waiting cars are passed.
    2. At each iteration a new evaluation is created to put a traffic light on green.

Our output is given by figure 3 with input:

```
single_intersection(1000, 1, 1, 1)
```

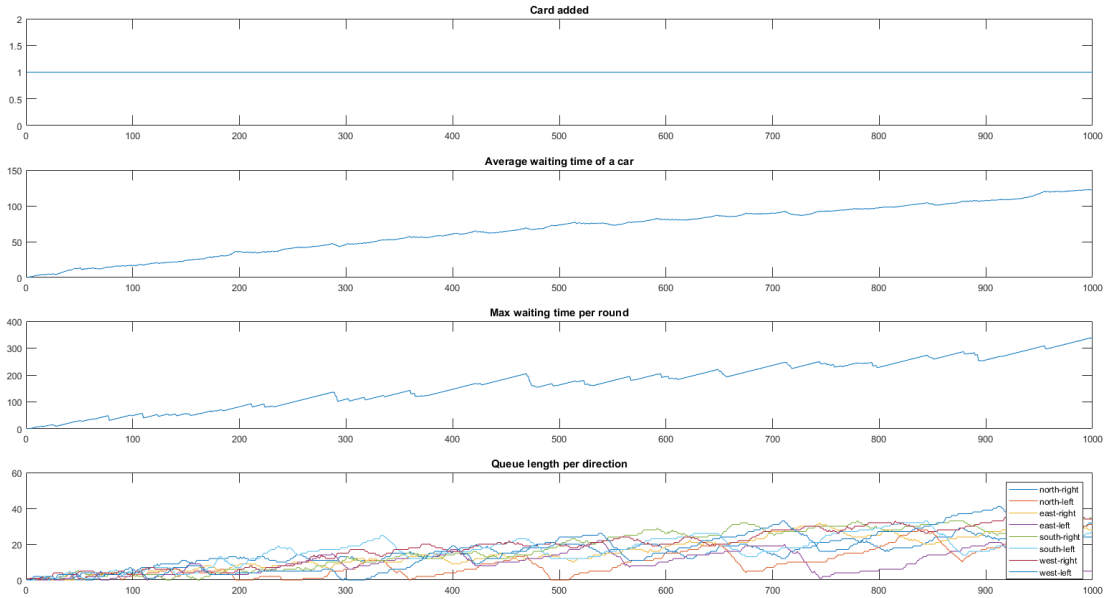


Figure 3: Example output

The following figures are plotted:

- Cards added, shows the amount of cars added per round. We see an constant addition of cars at each iteration.
- Average waiting time of a car, shows the average waiting time for each car in the queues per round.
- Max waiting time per round, shows the longest waiting time of a car waiting for a traffic light of all queues.
- Queue length per direction, shows for each queue the amount of cars that are arrived at this traffic light. When the traffic light is green, we see that this decreases. If the traffic light is red, a increasing trend is shown.

2.3 Experiments

Experimenting with our model will consist of measuring the performance of the four possibilities we can make of our settings. We apply those four kind of strategies to our three different input manners given in section 2.2.3. The four kind of strategies are the following:

- green_mode == 1 and lock_mode == 1. The traffic lights that will be green are chosen by a combination of the car that waits as longest and the length of each lane. If a traffic light is set on green, it will be green till the queue is empty, unless there are more added to the lane.
- green_mode == 1 and lock_mode == 0. The traffic lights that will be green are chosen by a combination of the car that waits as longest and the length of each lane. After each iteration a new evaluation is made to change which traffic light to green.
- green_mode == 0 and lock_mode == 1. A Round Robin scheduling is applied to choose which traffic light will be green. If a traffic light is set to green, it will be green till the queue is empty, unless there are more added to the lane.
- green_mode == 0 and lock_mode == 0. A Round Robin scheduling is applied to choose which traffic light will be green. After each iteration a new evaluation is made to change which traffic light to green.

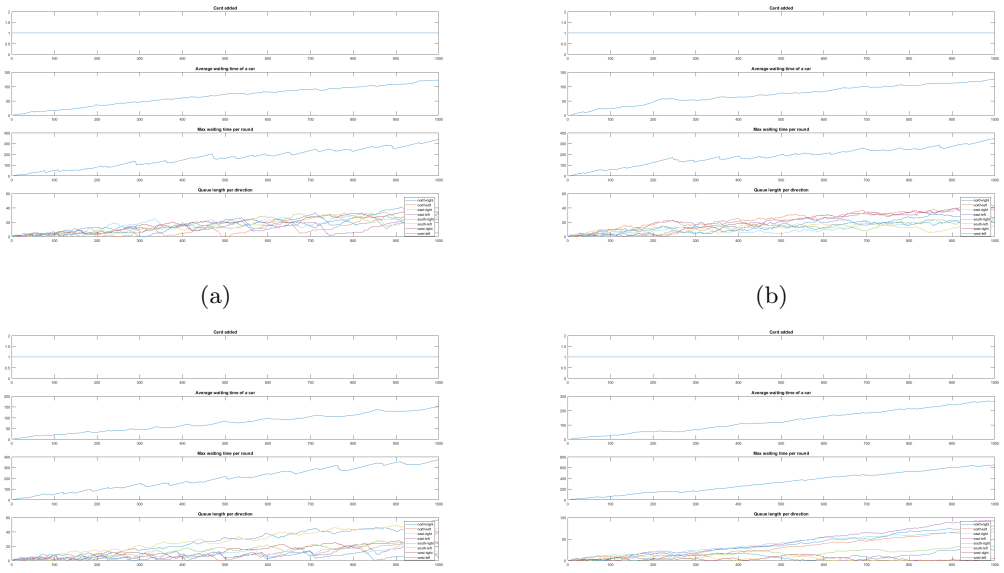
We apply these four kinds of settings to different scheduling strategies we see in real life. The three kind of adding cars to random places at the intersection are given by the input type.

2.3.1 Constant flow

The simplest traffic model is the one where we add a constant amount of cars every round. In our experiment we add one car every round which results in quite a lot of traffic. The car is added to one of the four lights where is is set two one of the two lanes. We run this simulation on 1000 seconds with a

constant traffic flow (input_type == 1). The commands and their results are the following:

```
single_intersection(1000, 1, 1, 1)
single_intersection(1000, 1, 1, 0)
single_intersection(1000, 1, 0, 1)
single_intersection(1000, 1, 0, 0)
```



(a) (b)

(c) (d)

Figure 4: Results for a constant input

	(green_mode, lock_mode)			
	(1,1)	(1,0)	(0,1)	(0,0)
average_waiting	125	130	150	280
max_waiting	320	340	360	640

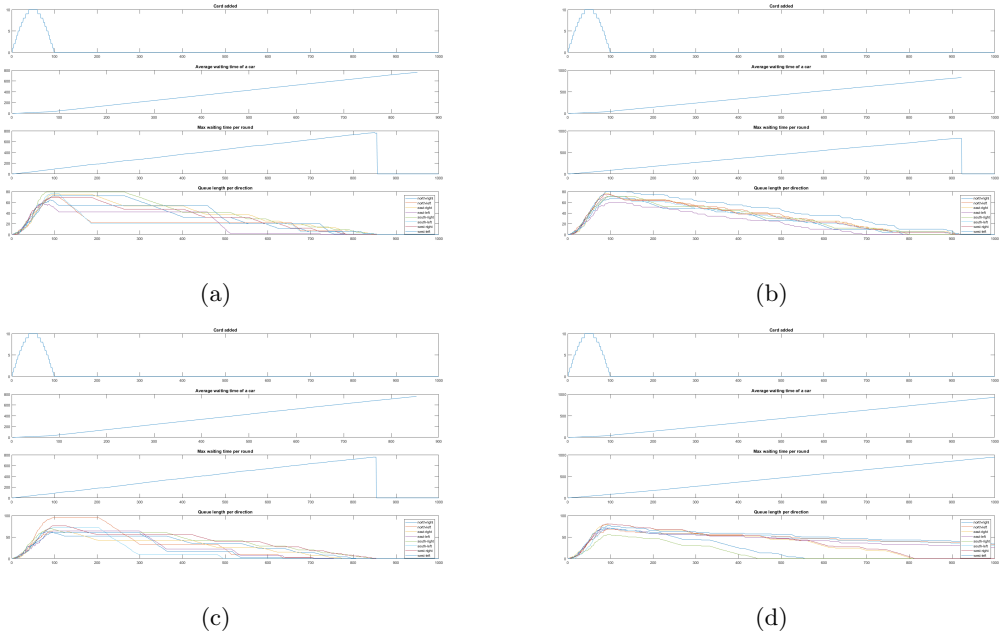
We can clearly see that the Round Robin schedules are not so effective for the average waiting time (green 0), especially in combination with the re-evaluation per round (lock 0). We observe that our chosen optimal scheduling strategy with emptying the queues is the most optimal one.

2.3.2 Rush hour

Rush hours are a daily event in city's like Groningen, all motorists want to be at their work at nine in the morning. Unfortunately this creates long waiting

queues, to let people be in time at their work the traffic light has to empty the queues as quickly as possible. Therefore, we introduced another metric which is the amount of seconds till all queues are empty so the rush hour is completely over. We simulate the rush hour with different settings by the following commands, the results are given below.

```
single_intersection(1000, 2, 1, 1)
single_intersection(1000, 2, 1, 0)
single_intersection(1000, 2, 0, 1)
single_intersection(1000, 2, 0, 0)
```



(a) (b)

(c) (d)

Figure 5: Results for a rush hour input

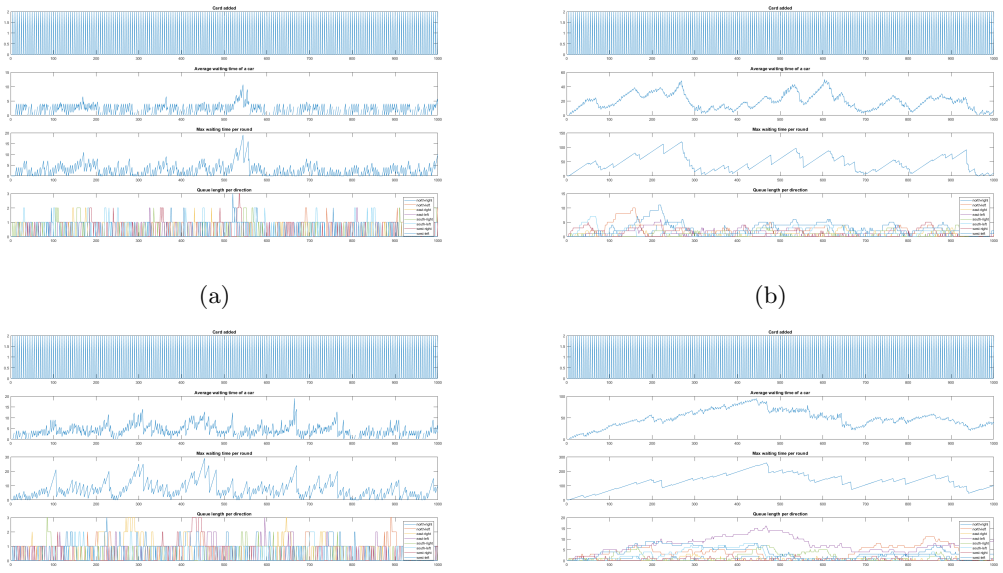
	(green_mode, lock_mode)			
	(1,1)	(1,0)	(0,1)	(0,0)
average_waiting	770	850	770	920
max_waiting	780	820	780	940
no cars waiting	860	920	860	1000+

For this situation both traffic light selection schemes; maximum demand and Round Robin the metrics are the same when used with emptying the waiting queue. Apparently during rush hours it is more important to let cars pass then to determine which queues have the most right to get a turn. This makes sense because during rush hours each traffic light has cars waiting. Round Robin with re-scheduling per round takes significantly more time then the other ones.

2.3.3 Frequent peaks

A more natural constant method of adding cars is to add multiple cars on frequent occasions. This is often the case when cars are passed to other traffic lights. The commands and the results are given below:

```
single_intersection(1000, 3, 1, 1)
single_intersection(1000, 3, 1, 0)
single_intersection(1000, 3, 0, 1)
single_intersection(1000, 3, 0, 0)
```



(a) (b)

(c) (d)

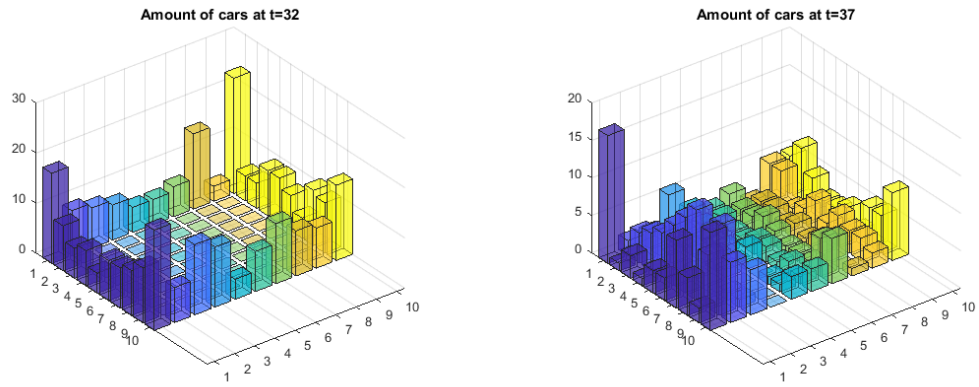
Figure 6: Results for frequent peak input

	(green_mode, lock_mode)			
	(1,1)	(1,0)	(0,1)	(0,0)
average_waiting	11	50	18	95
max_waiting	19	130	29	260

The results of this simulation is totally different then the previous ones. Each of the four settings show totally different behaviour. The (1,1) strategy shows an outcome that is far better than all other ones. We see that emptying the queues shows significantly better performance on the time that the car that waits the longest time. This has an evident reason that the queue with this car is surely queued in the queue emptying strategy.

3 Grid

After we created our single intersection we created a grid of intersections. This is a combination of intersections that let cars drive from one side to another. The cars are added at the edges of the grid as one can see in figure 7a, cars have a route to the other side of the grid. The program that runs the grid creates a bar plot where each bar represents one traffic light. The height of the bar shows the amount of cars currently waiting at the traffic light. Figure 7



(a) Cars are added at the edges

(b) grid visualisation example

Figure 7

One can execute the grid with the following command:

```
grid_intersection(simsec, input_type, grid_size, green_mode, lock_mode)
```

The arguments are similar to the single_interaction we explained in 2.2.3. In addition there is a grid_size which specifies the height and the width of the grid. The figure of 7 has a grid_size of 10. The output of the grid is similar but different than that of the single_intersection. The following three figures are plotted:

1. The amount of cars that are added to the grid.
2. The average waiting time of a car at all traffic lights.
3. The maximum waiting time of all cars over all traffic lights.

3.1 Experiments

We will perform the same experiments on the grid as we did on the single_intersection. We will perform our experiments with 100 iterations and on a grid of grid_size

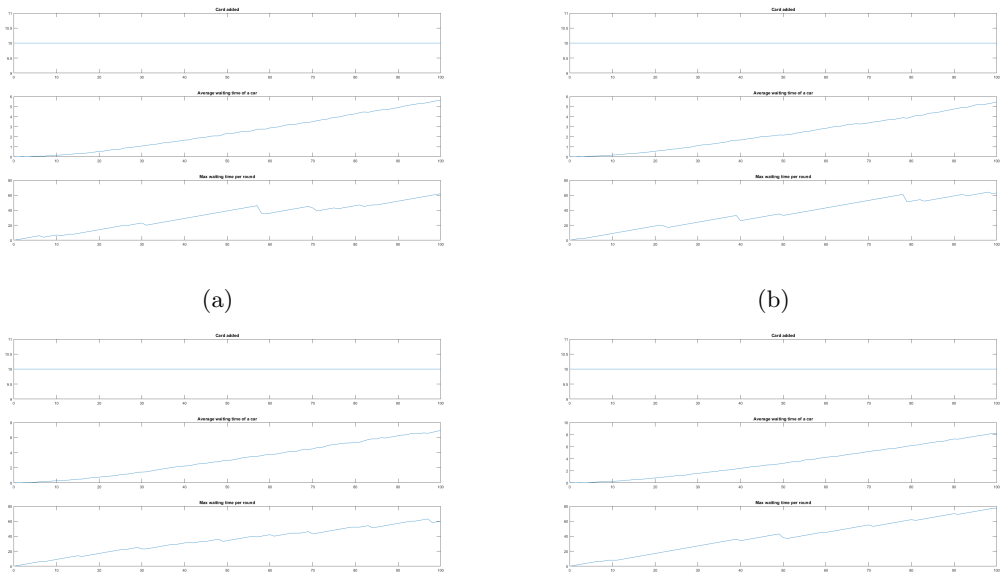
== 10. The average time and maximum waiting time are now summed over each iteration. The input types are similar but have a different formula given iteration t, amount of iterations T and amount of cars per round n:

1. Constant flow:
$$n(t) = 10$$
2. Rush hour:
$$n(t) = \begin{cases} 10 \cdot sin(\frac{t}{T \cdot \pi \cdot 10}), & \text{if } T/10 \\ 0, & \text{otherwise} \end{cases}$$
3. Frequent peaks:
$$n(t) = \begin{cases} 10, & \text{if } \mod(x, 5) == 0 \\ 0, & \text{otherwise} \end{cases}$$

3.1.1 Constant flow

The constant flow has as input a constant amount of cars. The code is executed by the following commands and their results are given below:

```
grid_intersection(100, 10, 1, 1, 1)
grid_intersection(100, 10, 1, 1, 0)
grid_intersection(100, 10, 1, 0, 1)
grid_intersection(100, 10, 1, 0, 0)
```



(a) (b)

(c) (d)

Figure 8: Results for a constant input

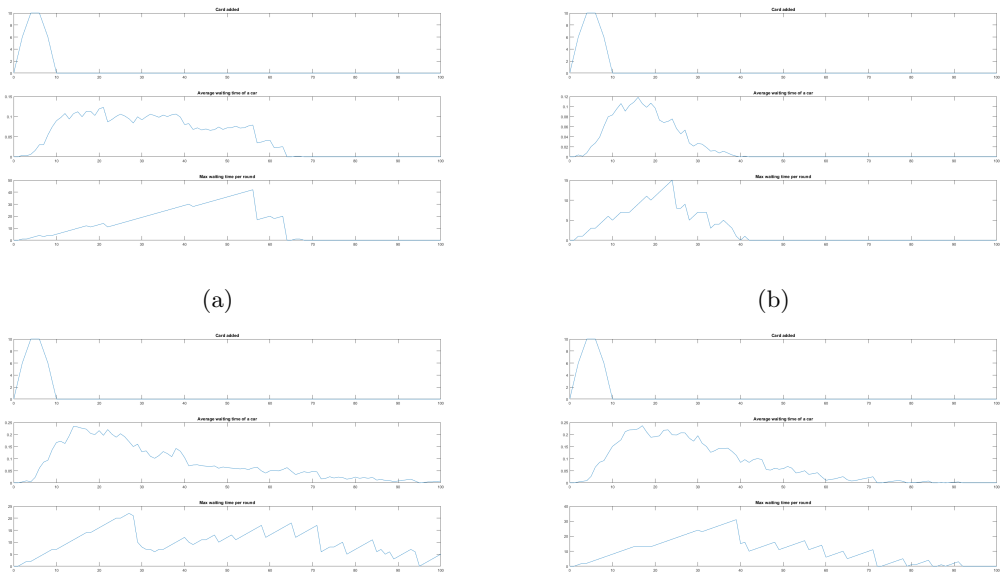
	(green_mode, lock_mode)			
	(1,1)	(1,0)	(0,1)	(0,0)
average_waiting	5,5	5,4	7	8,2
max_waiting	62	62	60	76

From the results we can see that the green_mode is the most important factor in the grid. Intelligently choosing which traffic light should be changed to green is advised here. The results between let all cars pass or only one are not significant.

3.1.2 Rush hour

The rush hour has a peak at the beginning of cars at the beginning and stops then with adding cars. The code and results are given below:

```
grid_intersection(100, 10, 2, 1, 1)
grid_intersection(100, 10, 2, 1, 0)
grid_intersection(100, 10, 2, 0, 1)
grid_intersection(100, 10, 2, 0, 0)
```



(a) (b)

(c) (d)

Figure 9: Results for a rush hour

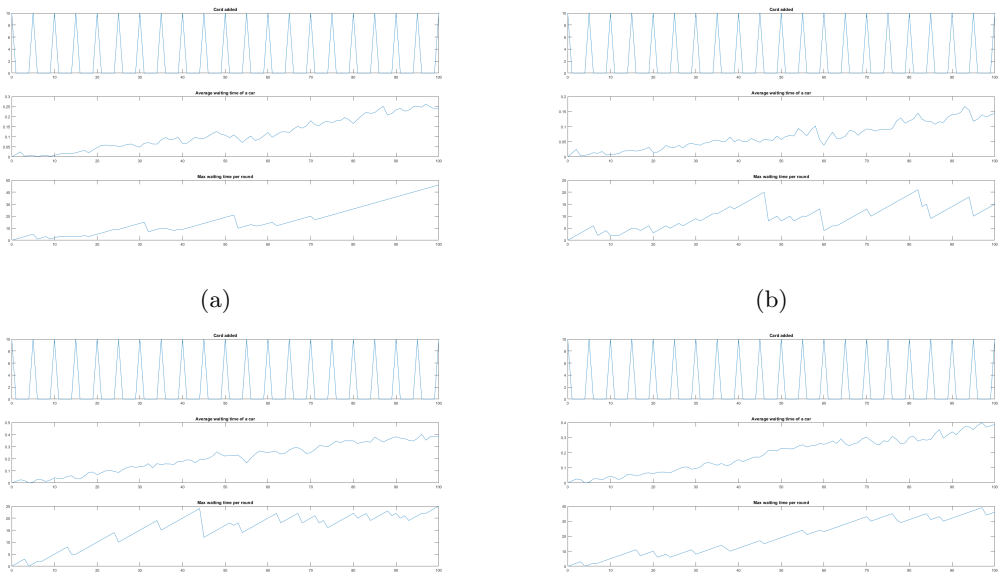
	(green_mode, lock_mode)			
	(1,1)	(1,0)	(0,1)	(0,0)
average_waiting	0,13	0,12	0,23	0,13
max_waiting	42	15	22	45
no cars waiting	64	42	95	65

In these results we see there is clearly one optimal solution. This is the monitored approach to choose which traffic light to change to green in combination with doing this each iteration. Another striking result is that (1,1) and (0,0) show similar effects. something we weren't expecting because these are opposite settings.
Output.mp4 shows a movie of a rush hour grid simulation. This shows how a $30 * 30$ grid over a few hundred rounds.

3.1.3 Frequent peaks

This method adds cars once in an amount of iterations. The commands and results are given below:

```
grid_intersection(100, 10, 3, 1, 1)
grid_intersection(100, 10, 3, 1, 0)
grid_intersection(100, 10, 3, 0, 1)
grid_intersection(100, 10, 3, 0, 0)
```



(a) (b)

(c) (d)

Figure 10: Results for frequent peaks

	(green_mode, lock_mode)			
	(1,1)	(1,0)	(0,1)	(0,0)
average_waiting	0,23	0,15	0,4	0,4
max_waiting	45	20	25	40

Looking at the results, we can clearly indicate a best approach. Both average waiting time and maximum waiting time are reduced the most by the monitored and once each round mode. Another observation is that (0,1) reduces max_waiting more than the intelligent approach. Something which can be described by that Round Robin equally divides turns over the traffic lights.

4 Conclusions

Our main goal of this project was to create a model of traffic lights on a grid and simulate real life situations on this model. We have managed to create a single intersection containing traffic lights and were able to extend this to a grid of intersections. We experimented with this on different input modes as traffic input models which we also see in daily life to benchmark different strategies. We can conclude that the optimal strategies that we created show different results on a grid than on a single intersection. On a single intersection we can conclude that monitored choosing which traffic light to change to green and letting all current cars pass is the best approach for our three situations. On a grid this is slightly different, there we can conclude that monitored scheduling which traffic light to change to green and doing this each iteration is the optimal approach. This is the case here in all three situations.

From these observations we can conclude that monitored scheduling which traffic light to change to green is a good way of scheduling traffic lights. In our experiments we implemented this in only one stategy. It will be interesting to see how we can slightly change this approach to look for further optimisation's.

In addition to the observations on different strategies are described but unfortunately we didn't have the time to mention all details or to fully explain them. We are very happy with our model and results. The model could serve as a basis for a project in coming years to implement even more complex strategies and models.