Capstone Project

Machine Learning Nanodegree

1. Domain Background

The project domain will be the analysis and optimization of traffic light control using machine learning. Efficient ways of organizing traffic light control in an urban environment are important for numerous reasons like lowering fuel consumption, grid lock prevention and overall faster transportation times. Additional challenges include prioritizing traffic dynamically for emergency or police cars or rerouting traffic in case of accidents. Nowadays traffic lights are controlled by static algorithms using time slots. A machine learning approach could potentially optimize traffic. Because traffic lights today are autonomous, unmanaged systems a simulation environment to test a machine learning strategy is necessary.

2. Problem Statement

The goal is to develop a reinforcement learning strategy to control traffic lights directly in a simulated environment using SUMO[1]. The following tasks need to be accomplished:

1. Preprocess and analyze the simulation scenarios for the cities of Luxembourg[2] and Cologne[3]
2. Use one or more clustering algorithms to find meaningful clusters for the traffic light controlled intersections
3. Choose and implement a reinforcement learner for each of the clusters which controls the traffic lights directly using SUMO's Python interface TraCI[4]
4. Compare the first implementation to random behavior and the scenarios default traffic light algorithms in terms of the aggregates simulation statistics SUMO provides
5. Improve the reward function and re-run the simulations

The reinforcement learner will hopefully improve the overall traffic statistics in terms of total waiting times and fuel consumption.

---

1 www.sumo.dlr.de
2 https://github.com/lcodeca/LuSTScenario
3 http://sumo.dlr.de/wiki/Data/Scenarios/TAPASCologne
4 http://sumo.dlr.de/wiki/TraCI

For a detailed manual of reproducing my results see the Technical How-to in the project root.

3. Evaluation Metrics

The following metrics will be used to evaluate the solution model:

- Duration: average trip duration
- Waiting Time: average time spent standing (involuntarily)
- TimeLoss: average time lost due to driving slower than desired
- DepartDelay: average time vehicle departures were delayed due to lack of road space
- Emergency Stops: The number of emergency stops card had to make

They are provided by the simulation and are directly related to the overall efficiency of the traffic system.

4. Clustering Analysis

a. Introduction

The full clustering analysis can be seen in the Jupyter Notebook clustering_code/clustering.ipynb file. I decided to run the clustering analyses on three datasets:

1. The TAPAS Cologne (cgn) dataset
2. The LuST (lust) Datasets
3. A combination of the two

The idea behind this approach was to see if the different scenarios would have similar clustering results and to get a deeper understanding of SUMO scenarios. Additionally, it was interesting to see if a combined dataset would benefit from including more data. In the following chapters, the cgn-dataset will be discussed most of the time for the sake of brevity although every step of every analysis was executed for each of the three datasets.

b. Importing the Data

Before clustering could be implemented, the dataset needed to be converted into a pandas dataframe. The following values are collected:

- junction_id: the id of the intersection

- junction_type: is always traffic_light[5]
- x,y,z: the physical coordinates of the junction in the simulations
- isRoundabout: Whether the intersection is part of a roundabout
- trafficlight_count: an array containing the number of traffic lights the intersection controlls, the trafficlight's ID and a list of all the connections to the intersection
- avg_lane_speed,avg_lane_length& standard deviations: Because every intersection has a different amount of lanes, the mean was used to aggregate their speed limits and lengths. To mitigate the information loss the standard deviation was also computed.
- edge_types: the different types of edges[6] connected to this intersection
- edge_priorities: The average of the edge's priorities. Because these values are very close together, the standard deviation was not included
- number of lanes: the total amount of lanes in the junction

The script that generates csv files for scenario files can be found in the "clustering_code/dataset-import.py" file[7].

c. Preprocessing the Datasets

A few columns had to be dropped or reformatted. For example, initially I included the x,y and z coordinates in the clustering process. The idea was that maybe intersections that are nearby each other in a geographical sense, could be similar. Due to later analyses I decided to exclude them.

The isRoundabout was converted from {true|false} to {0|1}. One more challenging tasks was to convert the different edge types. I used sklearn's MultiLabelBinarizer(MLB) to accomplish this task. Also it was interesting to run the clustering analyses with and without the MLB to see how much of an impact the types would have[8]. Including the MLB-generated values resulted in different results for the datasets. For the cgn dataset including the types resulted in more or the same amount of clusters depending on the algorithm. For the lust dataset it resulted in more or the same clusters with one exception where it reduced the amount of clusters. Finally, I decided to include the MLB because the amount of traffic lights controlled where better distributed among the clusters. Without

---

5 Initially it was planned to incorporate different junction types in the clustering analysis and there was more than one type in the dataset

6 An example from TAPAS Cologne dataset: ['highway.primary', 'highway.primary', 'highway.residential']

7 A few steps need to be executed before running the importer, they are documented in the comments

8 See MLB-NoMLB-comparison.docx for more details

the MLB sometimes one cluster would include above 90% of all intersections with n-traffic lights and the other m-clusters would include only one intersection.

d. Feature Scaling

Because the different features, like length of lanes and maximum speed are on different scales and vary greatly in range, feature scaling had to be implemented before running PCA. Using just the natural logarithm obviously is not an option because "0" is a common value in my datasets. Two different scaling algorithms were implemented and evaluated throughout the whole clustering analysis. The first one being the MaxAbsScaler which "scales and translates each feature individually such that the maximal absolute value of each feature in the training set will be 1.0. It does not shift/center the data, and thus does not destroy any sparsity." [9] I chose this because especially the features created by the MLB are naturally very sparse and it should stay that way after scaling.

The second algorithm was the RobustScaler which "removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile)." [10] Because of the use of the IQR for each feature independently it is robust to outliers. This should provide some interesting results different from the MaxAbsScaler but it does not necessarily keep the sparsity of the data intact.

e. Dimensionality Reduction using PCA

The next step was to apply dimensionality reduction for both scalers. As can be seen in Table 1 - Explained Variance for cgn-dataset MaxAbs scaler the explained variance for 3 Dimensions is about 62%. I was unsure whether this would prove to be enough. Therefore I subsequently used two datasets for each scaler and dataset. One with 3 dimensions and the other one with the fourth dimension.

Table 1 - Explained Variance for cgn-dataset MaxAbs scaler

Dimension 1	0.2426
Dimension 2	0.4504
Dimension 3	0.6273
Dimension 4	0.7379

In Figure 1- The first 4 Principal Components for cgn- MaxAbs the principal components can be seen. The features created by the MLB clearly influence the components and assist in separating large from smaller intersections together with the number of traffic lights. The robust scaler, as expected, found quite different principal components. The explained variances were a bit higher for the cgn-dataset.

---

[9] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MaxAbsScaler.html

[10] http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html#sklearn.preprocessing.RobustScaler

In addition, a different PCA algorithm Kernel PCA was evaluated. The results turned out to be about the same than using conventional PCA. This is why, it is not discussed in this paper.

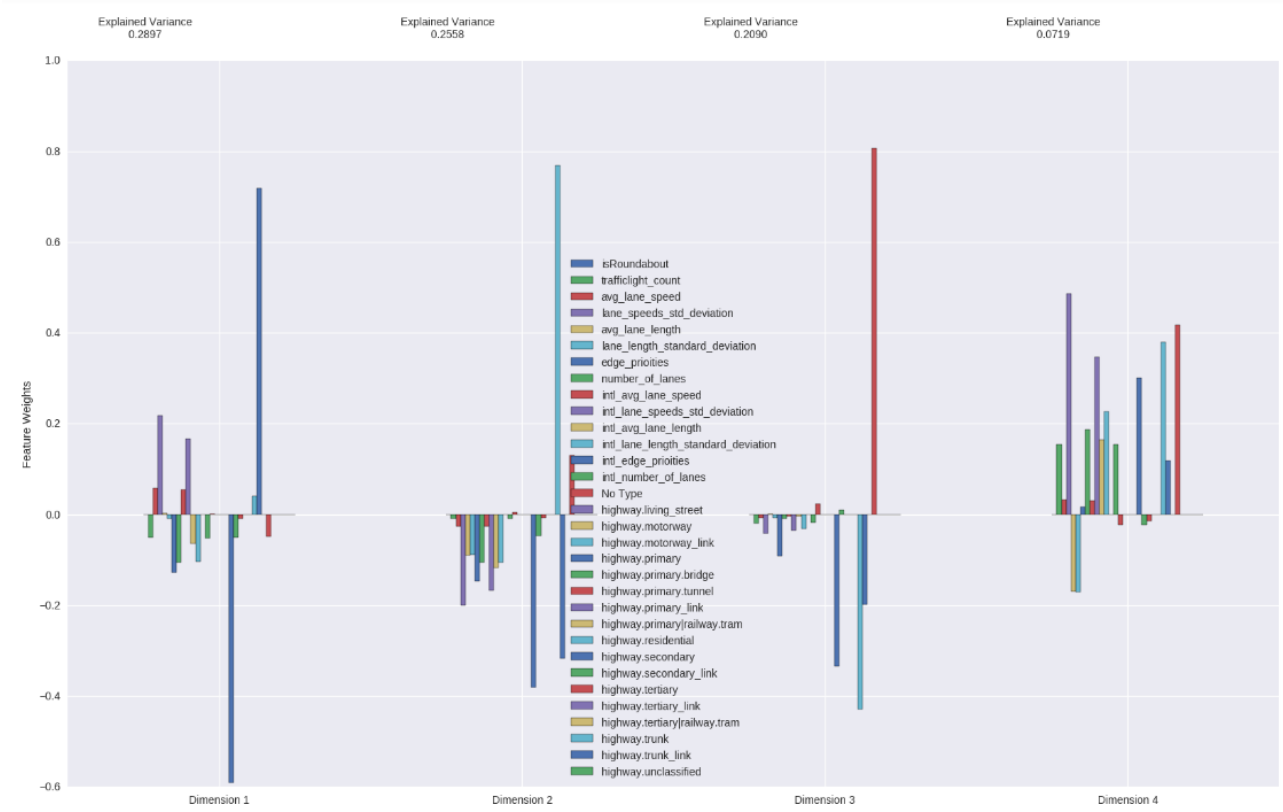

Figure 1- The first 4 Principal Components for cgn- MaxAbs

f. Clustering using the Silhouette Score

For clustering a GaussianMixtureModel[11] was used. The clustering algorithm was executed for every dataset, every scaler and in 3 and 4 dimensions with 2 to 40 cluster centers. The best result according to the silhouette score[12] was chosen for each dataset.

g. Analysis of Clustering Results

For the analysis of the results three questions were important:

1. How do the clusters "look"?
2. How similar are the clusters found by differently configured datasets?
3. How well are the different traffic light counts distributed among clusters?

To answer the first question, I created a number of 3D plots. Figure 2 - 3D Plot for cgn MaxAbs shows some clearly defined and well separated clusters. Figure 3 - 3D Plot cgn-dataset RobustScaler shows less clusters that seem to have a lower inter cluster distance

[11] http://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html#sklearn.mixture.GaussianMixture

12 http://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

and are not as well separated from one and another.

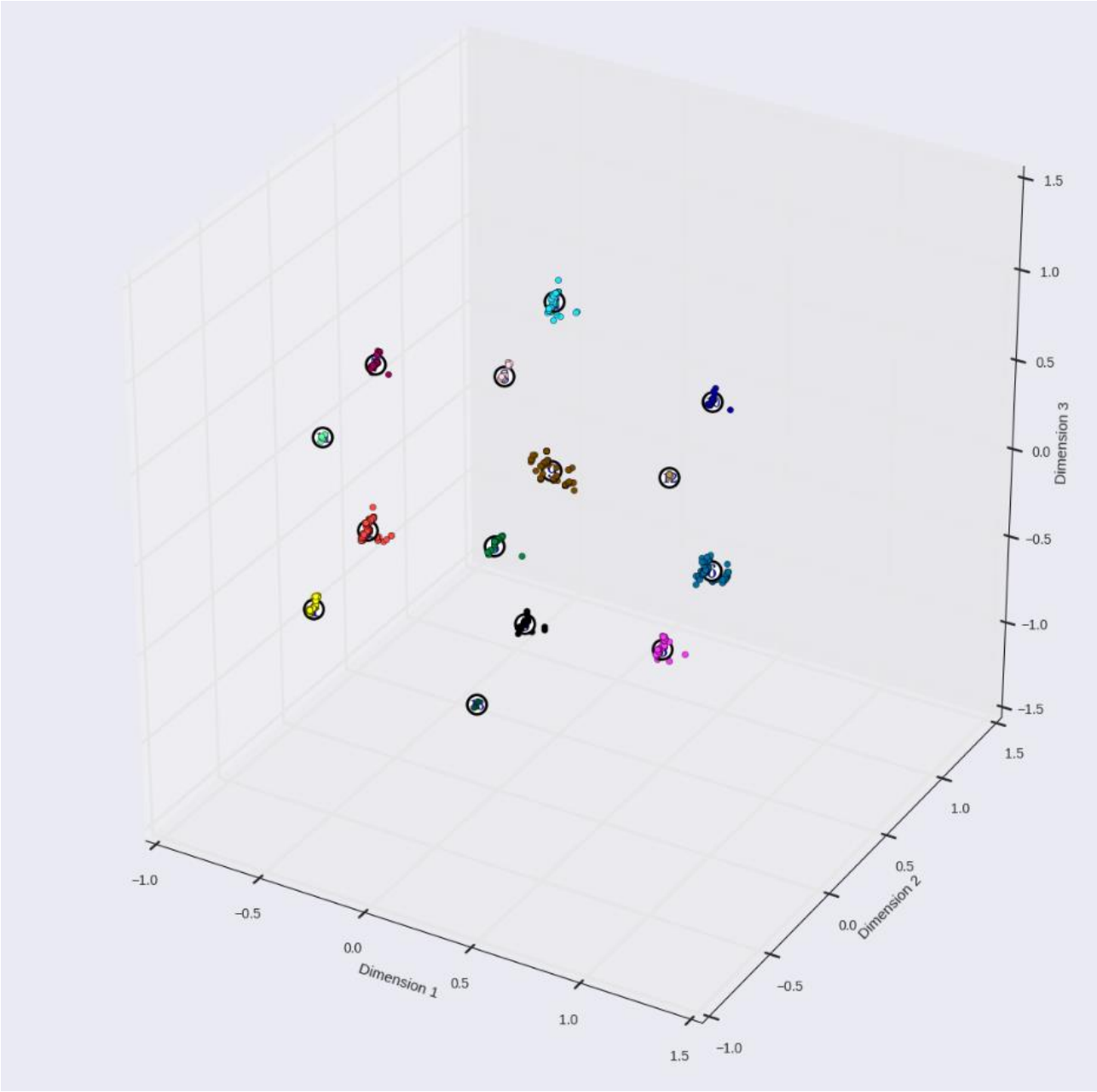

Figure 2 - 3D Plot for cgn MaxAbs

To answer the second question two approaches were chosen. The first is depicted in Figure 4 - Cluster centers for cgn (green), lust(red) and combined(blue) where the different cluster centers can be compared. The rationale behind this 3D plot was that if the different datasets would have a lot in common, the cluster centers produced by the same algorithm should be at least close by one and another. The plot indicates that this is, in fact, not the case. Although the centers are on the same scale[13] they are not really close to each other

[13] Is not always the case, for example with KPCA cgn's centers were on a larger scale than lust and combined centers

or arranged in a similar pattern. On the contrary they do not seem to be similar. For

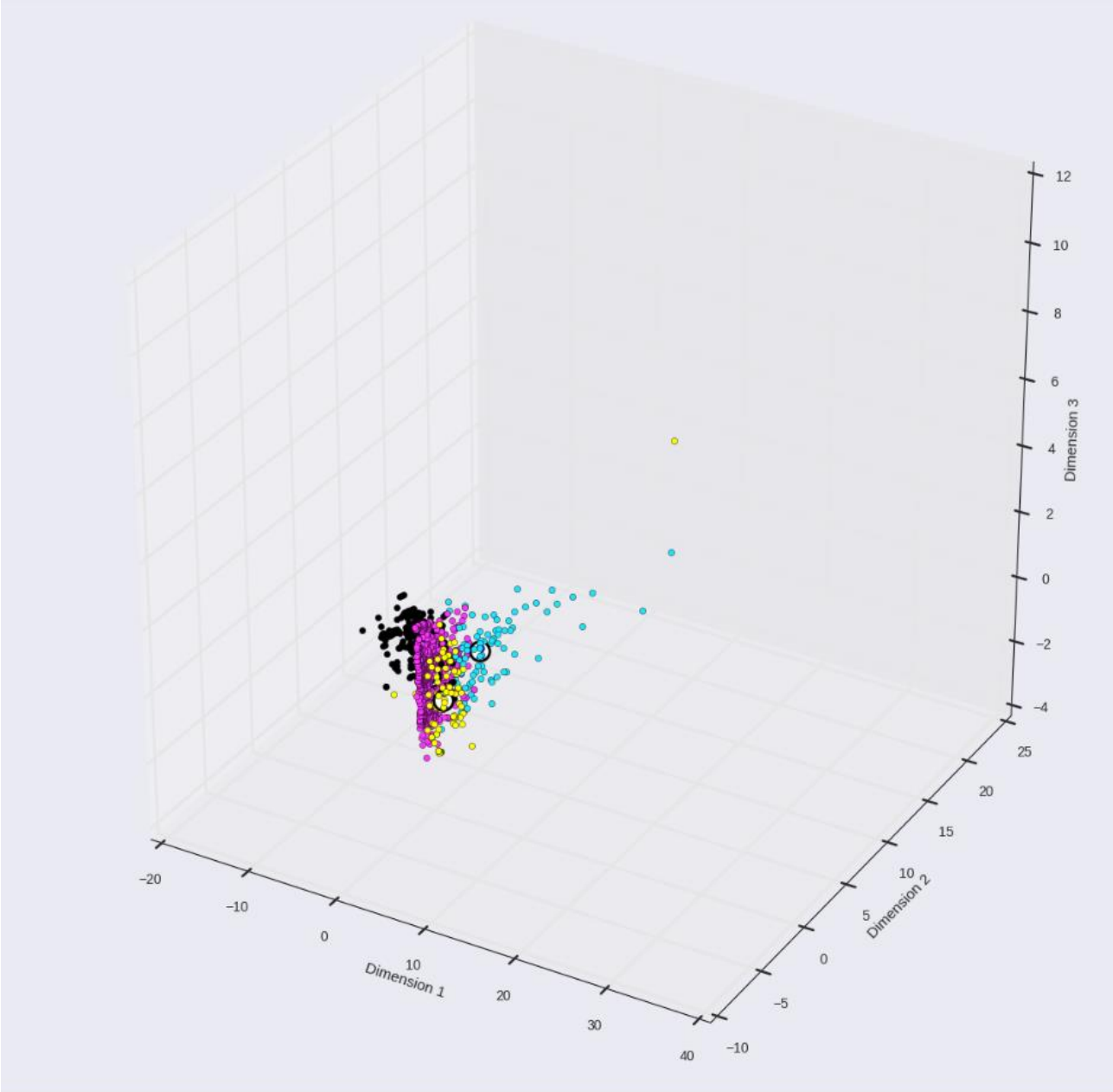

Figure 3 - 3D Plot cgn-dataset RobustScaler

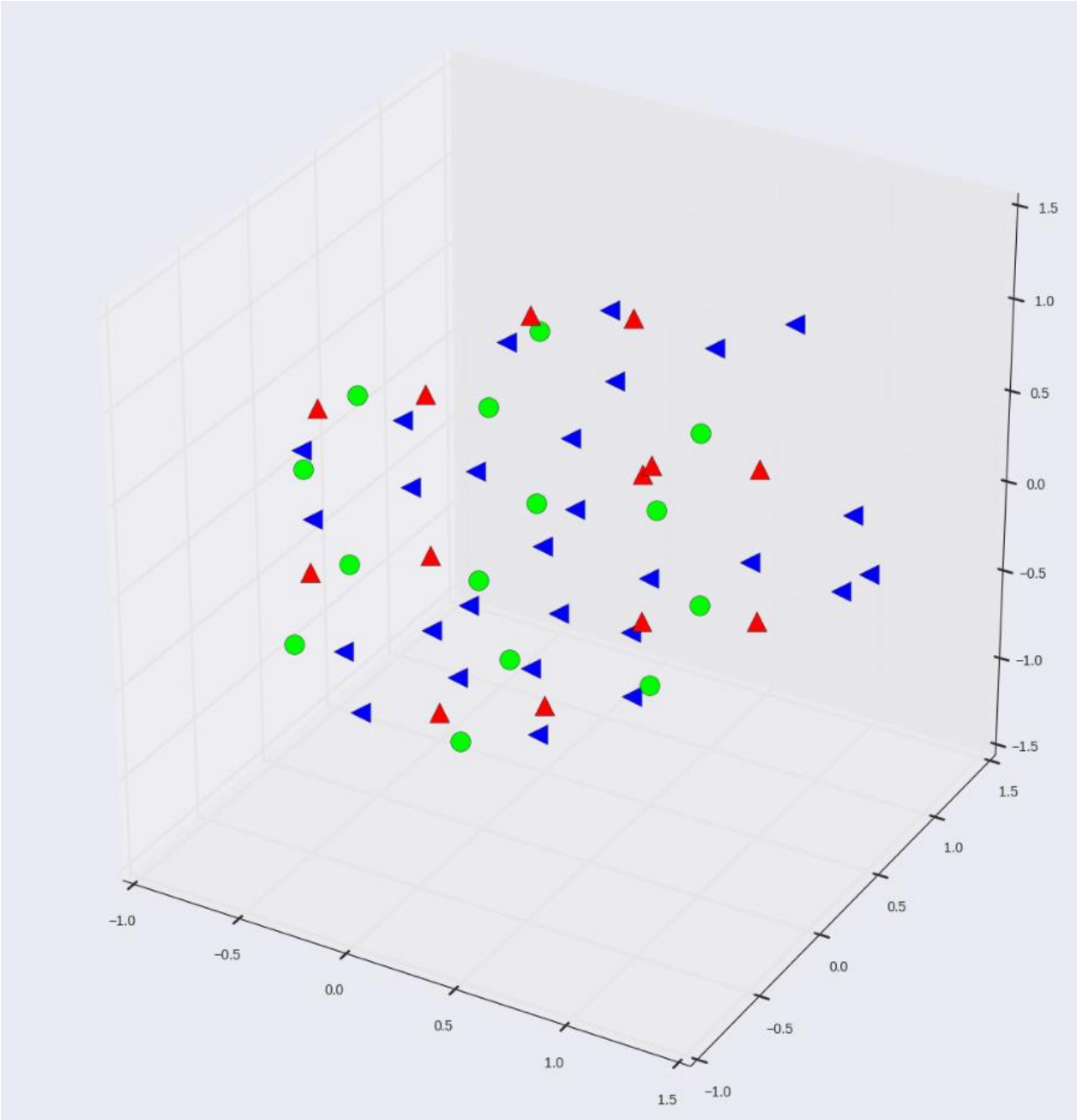

Figure 4 - Cluster centers for cgn (green), lust(red) and combined(blue)

different datasets the question was answered but it remains to be seen how the clusters distinguish between different configurations. To analyze this, it was compared in how many clusters of dataset B, the points of a cluster of dataset A are distributed. For example, if all data points of cluster A of dataset A are in cluster A of dataset B, then the similarity is 100%.

Using this analysis it was determined that adding a fourth dimension in PCA earlier, did not significantly improve clustering results because the clusters were almost all the same. The same is true for KPCA.

The analysis did not provide with a good metric for the different scaling algorithms.

Finally the last question must be answered to understand how different traffic lights are distributed among different clusters. This will be important for the reinforcement learner. Obviously, a good distribution is preferable. Considering the following example table:

	Junctions with 4 traffic lights	Junctions with 5 traffic lights
Cluster A	99	40
Cluster B	1	60

The results for junctions with 5 traffic lights are considered better because the traffic lights are better distributed among the two clusters. As for junctions with 4 traffic lights, almost all of them are in cluster A and it is questionable if including cluster b even makes sense.

To analyze this a heatmap was created and it can be seen in Figure 5 The horizontal axes show

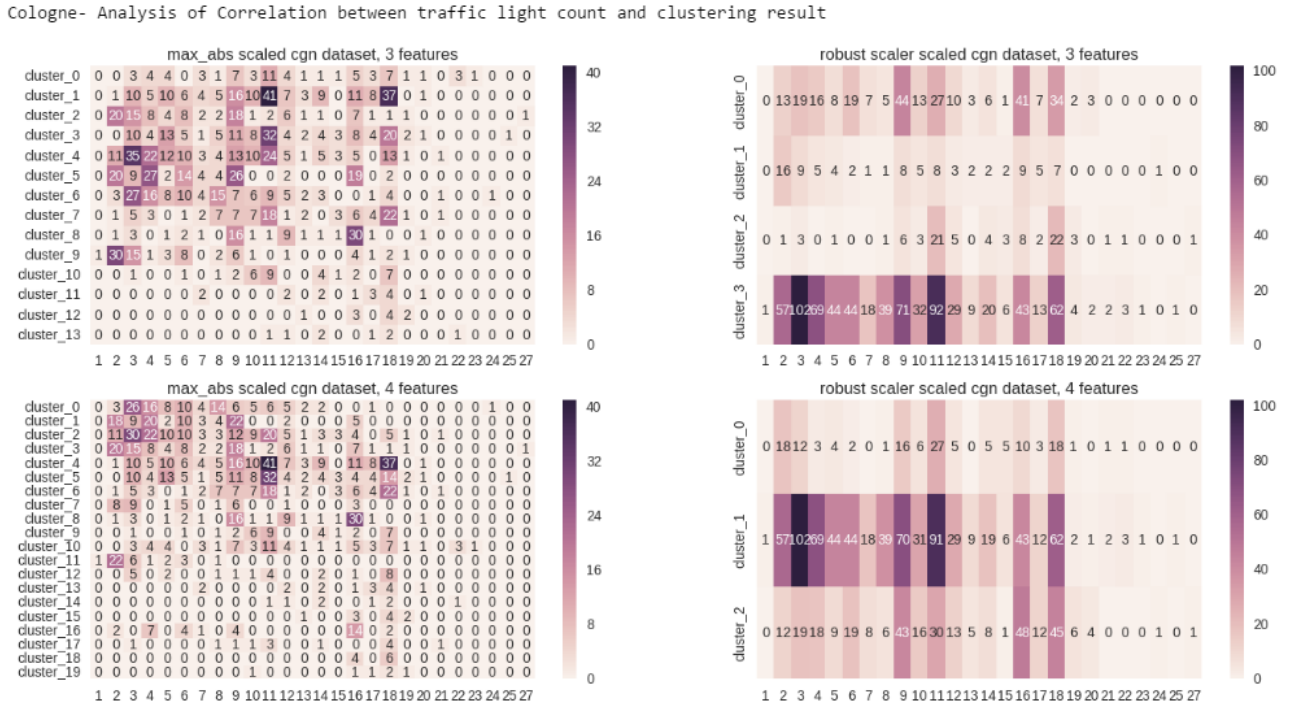

Figure 5 Heatmaps for cgn-dataset, horizontal axes are the traffic light counts and vertical axes the cluster_ids

The different counts of traffic lights, the vertical axes represent the cluster ids and the numbers on the heatmap represent how many instances of junctions there are for the corresponding cluster. For example, in the upper left heatmap, there is 1 junction with 1 traffic light in cluster 9. As a side note, heatmaps were computed for clustering that included the different edge types created by the MLB and for clustering without edge types[14]. In conclusion, the heatmaps show that the MaxAbs scaled clusters have a better distribution of traffic lights among them.

[14] https://github.com/danielpaulus/udacity/blob/master/project%205/MLB-NoMLB-comparison.pdf

a. Conclusion of Cluster Analysis

The most promising clusters seem to be the MaxAbs scaled, 3 dimension PCA clusters.

They will be primarily used for the reinforcement learner. Nevertheless, the other clustering results will be added to the dataset anyway for experimentation purposes.

b. Data Export

As a last step the cluster ids for the different algorithms had to be appended to the original dataset. The notebook creates the necessary csv files for the next phase of the project.

5. Implementing the Reinforcement Learner

The clustering algorithm had an unexpected benefit in terms of performance

a. Traffic Light Control

b. Algorithms and techniques

Initially I wanted to use the PyBrain[15] library to solve the problem. The official documentation was not very helpful but I found a good blog post[16]. The problem eventually ruled out using PyBrain was that the PyBrain RL learner could only handle discrete state spaces. One solution to deal with that was to use intervals for the relevant simulation statistics to discretize them. For example, for the waiting time statistic I would have mapped 0 to "very good", 1-5 to "ok" and everything above 5 to "bad". I could have based the intervals on statistical analyses of a full simulation run. Another problem was that PyBrain apparently has no GPU support.

As a consequence of further research[17] [18] [19], the final solution implements a deep reinforcement learning algorithm because it allows continuous state spaces.

DQN – Deep Reinforcement Learning

I decided to use this implementation of DQN[20] as a foundation for my code.

---

15 http://pybrain.org/

16 http://simontechblog.blogspot.de/2010/08/pybrain-reinforcement-learning-tutorial_21.html

17 https://oshearesearch.com/index.php/2016/06/14/kerlym-a-deep-reinforcement-learning-toolbox-in-keras/

18 https://github.com/matthiasplappert/keras-rl

19 https://homes.cs.washington.edu/~todorov/courses/amath579/reading/Continuous.pdf

20 https://jaromiru.com/2016/10/03/lets-make-a-dqn-implementation/

The basic principle is to extend traditional Q-Learning to continuous state spaces. In Markov Decision Process style problems, states must be discrete to be stored in the Q Matrix/Function. For many applications it is either not feasible or creates a great loss of information to discretize states. This is why it was suggested to approximate the Q-Function using a deep learning neural net.

Usually Q Learning uses online learning, as in every step of the simulation a reward is received and instantly used to improve the learner. This approach is not ideal for neural networks. Because of that the idea of 'Experience Replay' is introduced. Past experiences are stored in memory and in each step, a random batch is retrieved to train the neural network to prevent overfitting.

In addition an exploration factor is used to take random actions based on a decaying probability.

Benchmark

For benchmarking my implementations I compared them to the aggregated statistics of random behavior to see if my learner had actually learned something. In addition I ran the simulations with their own traffic light control algorithms to see if my model could improve the statistics.

Implementation

Preparing Steps

I adapted the code so that my clustering algorithm's results could be used to create one learner for each cluster for each traffic light size. For example, all junctions controlled by 2 traffic lights in the cgn scenario, would be controlled by only eight agents (see Figure 5).

Main Loop

Figure 6 depicts the main program loop. Every agent chooses an action and sets it in the simulation environment. Afterwards, one simulation step is executed. Finally every agent can gather the rewards and make observations for every junction she controls. The benefits of this approach are that every agent can learn many times for every simulation step and less memory is needed as opposed to an "one agent per junction"- approach.

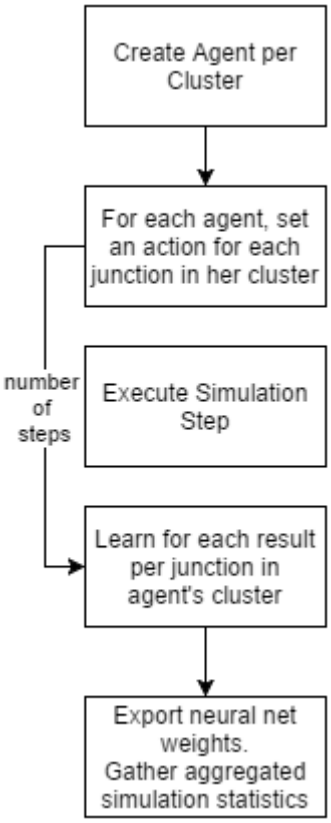

Figure 6 - Reinforcement Learner Main Loop

State Space

The first idea for the state space is to use the following statistics:

- vehicles started to teleport because there is no collision model in SUMO, usually vehicles are teleported if traffic jams or many accidents happen. This is definitely important for the learner

To keep the state space small, an average over all lanes of a junction of the following values was used:

- Waitingtime / LastStepVehicleNumber because this should be low
- co2 emission / LastStepVehicleNumber because this should be optimized
- Fuelconsumption/ LastStepVehicleNumber because this should be optimized
- LastStepMeanSpeed because speeds on the lane directly relate to traffic light efficiency
- LastStepVehicleNumber because the number of vehicles influence the need to switch the traffic light

Action Space

The action space for this particular problem gets very large very quickly due to the exponential growth. One traffic light can have the following states[21]:

Red, green, yellow coming from green, yellow coming from red, priority green.

[21] http://sumo.dlr.de/wiki/Simulation/Traffic_Lights#Signal_state_definitions

The offline state is omitted.

Therefore, the action space size is 5^n with n being the number of traffic lights.

For the cgn dataset 7 traffic lights were the limit. 8 traffic lights exceeded the available GPU RAM. I am well aware that the action space could be reduced by introducing bias to the algorithm. Basically, the actions could be reduced to “go green” and “go red” but the goal of this project is to have the reinforcement learner, learn these rules by itself.

Basic Reward Function

The idea for the reward function was to encourage situations with an equal amount of green and red lights. Usually the number of green and red lights on regular intersections should be about the same. Additionally the average waiting time should be minimized. Thus results the first naïve reward function as:

$$\mathrm{R}(s,a) = 0.2 * \sum greenlights \quad - 0.2 * \sum redlights \; + W(s)$$

Where

$$W(s) = \begin{cases} 0\,, wt = 0 \\ -0.5\frac{wt}{10} < 0.2 \\ -1\frac{wt}{10} > 0.5 \end{cases}$$

And wt as the average waiting time over all lanes.

Observation

For making observations, the SUMO GUI binary was used. A video was made and uploaded to YouTube[22]. The following observations were made:

- traffic lights flicker too frequently
- traffic lights can stay in any phase for as long as the learner wants
- traffic lights can converge to a stable state, f.ex. permanent red phase
- most lights go green eventually, probably because waiting time accumulates
- the traffic light change from any phase to any other phase
- there are long timespans where there are no cars nearby
- clusters of junctions are too big to be controlled, but could be split into smaller parts
- The number of cars leaving the lane does not influence the rewards
- Junctions of the same clusters vary in the number of cars passing through → how many cars frequent the intersection/junction should be included in the clustering analysis

[22] https://youtu.be/1AXYRfx1jkA

Benchmarking

Only traffic lights with sizes from 1-4 were included in these simulations therefore only slight changes in the metrics compared to the simulation's original program can be expected. In Figure 7 Charts comparing the learner to random and simulation native performance regarding the simulation metrics it can be seen that the reinforcement learner performed slightly worse than random actions for WaitingTime, TimeLoss, Duration and DepartDelay metrics.

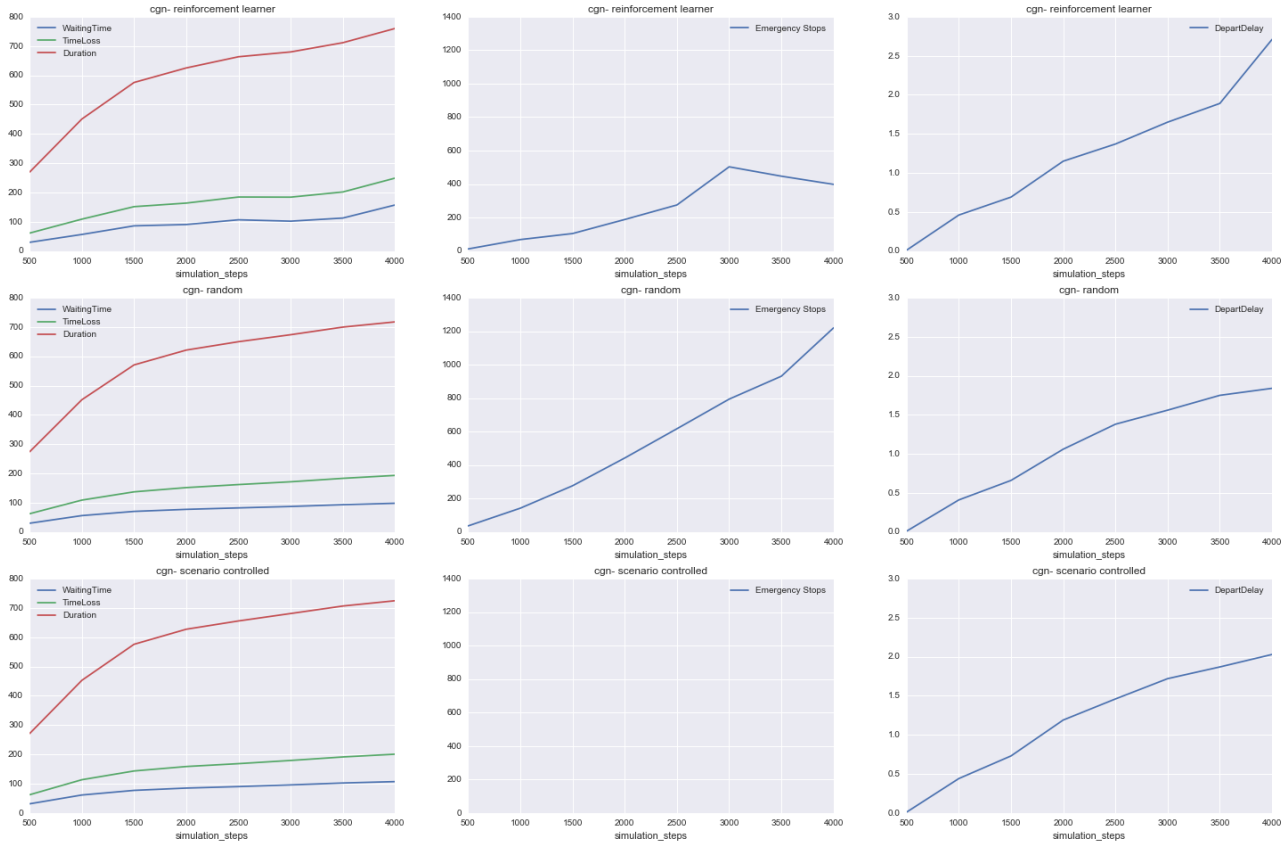

Figure 7 Charts comparing the learner to random and simulation native performance regarding the simulation metrics

The emergency stops are the result of traffic lights switching from green to red instantly, therefore causing vehicles to decelerate very rapidly. They are naturally very high for random actions. The reinforcement learner caused about half as many emergency stops, which is better than random but definitely something that needs to be dealt with.

Conclusion

The reward function and the state did not enable the learner to control junctions efficiently. Calculating rewards by the amount of green or red lights seems like a bad idea. The traffic statistics should have a larger impact on rewards. Also, a slight penalty for changing phases should be introduced to reduce flickering.

Emergency stops increase with the simulation time. The reason are instant changes from green to red lights that make vehicles break very rapidly. This needs to change.

Refinement

Tuning Rewards and State

The second reward function penalizes quick action fluctuation by calculating the hamming distance between to states[23] as a negative reward. In addition, emergency stops are calculated and penalized as a negative reward. Occupancy/haltingnumber+0.1

Thus, results the second reward function:

$$R(s,a,a') = -0.5 * hamming(a,a') - 2 * emergency_{stops} + \frac{occupancy}{haltingnumber + 0.1}$$

The last term is supposed to reward the learner for high traffic flows. The higher the occupancy[24] gets, the more reward is collected when the halting number is low. If there is no occupancy, which means no vehicles are on the lane, no reward can be collected.

The new state is an attempt to remove redundant values like CO2 or fuel consumption and replace them with more meaningful information. The following variables have been chosen:

- LastStepOccupancy
- LastStepVehicleNumber
- LastStepHaltingNumber
- VehiclesStartedToTeleport
- EmergencyStops

Observation

For the second observation, another video was made and uploaded to YouTube[25]. The following observations were made:

- Fluctuation of phases is reduced
- positive rewards for going green when waiting vehicles accumulate seems to be too small
- the traffic lights still switch from green to red, yellow to yellow..
- still the learner can converge to non-periodic, stable behavior. This is bad for traffic actuation, can be seen in the video for 'GS_61794247' with a permanent red phase
- deceleration and distance from vehicles to traffic lights is not in the state space, it is probably needed to learn about yellow phases

Benchmarking

Figure 8 Performance of the tuned reward function and state shows the performance of the runed rewards function. The learner performs reasonable in the first 2000 simulation

[23] SUMO TL states are encoded in string like „rrr" and „rgr"
[24] Occupancy is defined as a double value 0.0-1.0, that describes a lanes occupancy in percent
[25] https://youtu.be/lML3b64DYZw

steps but keeps getting worse. After 10000 steps its metrics are worse than random. This is probably due to the fact, that the learner converges to permanent red phases in some traffic lights. It follows that waiting time, timeloss and trip duration accumulate over time.

The emergency stops however, have been greatly reduced.

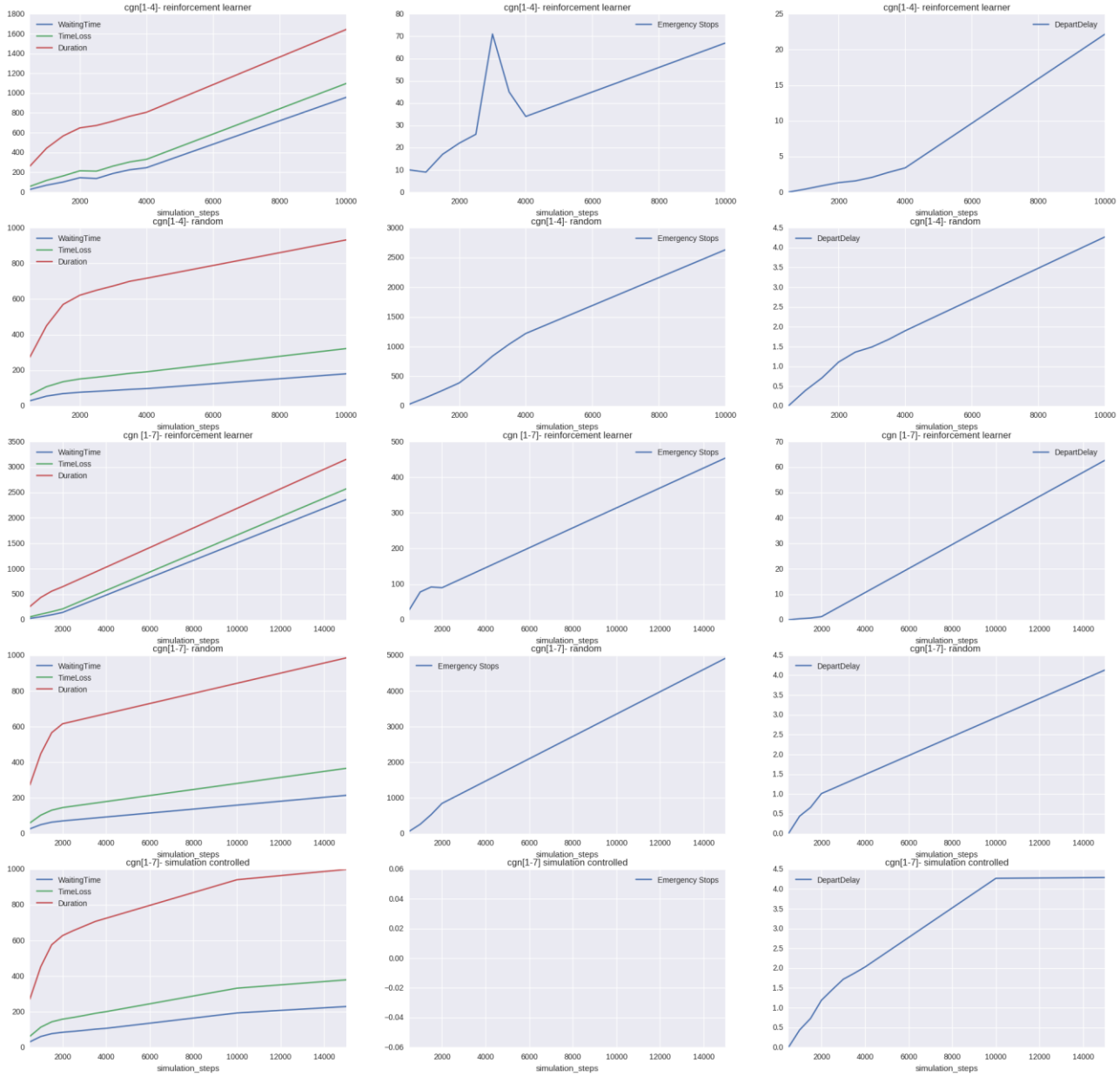

Figure 8 Performance of the tuned reward function and state

Conclusion

Although the new reward function performed better, there is still room for improvement. One crucial bit of information that is not included in reward nor state is the actual distance of the vehicles on the lanes to the traffic lights. Currently, every vehicle on every lane is considered equally important. This can work out on very short lanes but is problematic on very long lanes for obvious reasons. Including this information could enable the learner to learn the correct use of yellow phases. Consequently, emergency stops would probably decrease even more and permanent red phases could be prevented.

Reflection

The most difficult parts of the project were:

1. Choosing the right RL algorithm and implementation has proven to be quite difficult but really interesting.
2. Initially I did not suspect the large action spaces. Due to the larger intersections with over 20 traffic lights, and each light having at least 5 different reasonable states, action spaces quickly became too large to handle.
3. Building a good reward function is probably the hardest part in building a reinforcement learner. To further evaluate and tune different reward functions, I would need to create artificial micro-scenarios.
4. Simulation runs took a very long time because of the computational intensive nature of the project. I bought a new GPU for using CUDA and ran simulations mostly overnight.

   Near the end of my project I found out about other papers that tackled my exact problem[26] [27] [28]. Although they used a different approach, by using micro-scenarios and action space simplification, they provide some interesting insights into the problem domain.

   I only scratched the surface of the problem with this paper. I have gained a lot of insight into this very interesting problem domain and have many ideas for improvement. I think my approach of clustering the junctions beforehand worked out pretty good. I am well aware that action space reduction might have been a better idea and using isolated small scale scenarios like in [paper] could have resulted in better results but I think it is more interesting to build a model that learns these rules by itself like in the udacity self-driving car project.

Improvement

The next steps would be to return to the drawing board and figure out a better solution. First the clustering algorithm would have to be improved using statistical data of prior simulation runs. The first attempt at clustering only incorporated static structural information and neglected the actual flow of vehicles. With the newly computed clusters, small artificial scenarios would have to be created that represent the found clusters. This way faster training cycles and simulation runs could be used to improve the reward function.

---

26 https://arxiv.org/pdf/1611.01142.pdf
27 https://esc.fnwi.uva.nl/thesis/centraal/files/f632158773.pdf
28 http://cs229.stanford.edu/proj2016spr/report/047.pdf

Although the option was intentionally discarded for this project, reducing the action space to rule out invalid traffic light configurations[29] would make things really easy. Still, finding an algorithm that can learn these rules by itself is more interesting.

In addition, here are some more ideas to improve the project:

- There is still a great deal to be learned about reward design[30]
- The problem of a large action space needs to be addressed. I was already thinking about moving the learner to a continuous action space- implementation comparable to the one proposed in this paper[31] and map the actions into a continuous domain. Maybe there are other interesting techniques for large discrete action spaces.
- Traci uses TCP/IP for communication, maybe this could be changed to using an internal C++ API
- Because a lot of computational power is needed, cloud computing (Amazon S3..) would be helpful
- Some meaningful visualization[32] of the high dimensional deep learning neural net to better understand the Q-function could be found
- Using different state/reward function configurations for different clusters instead of only one global configuration

29 F.ex. Some Traffic lights never go from green to red as they always include a yellow phase

30 Like in this great dissertation: https://deepblue.lib.umich.edu/bitstream/handle/2027.42/89705/jdsorg_1.pdf?sequence=1

31 https://arxiv.org/abs/1509.02971

32 http://heatmapping.org/