# Traffic Control and Infrastructure Organization Using Reinforcement Learning

Daniel Kuknyo

2022.09.26.

PhD / MsC outline

## Part I
# Introduction

In recent years the traffic of cities became a rising topic with more and more city governments realizing that a motor-focused city design is unsustainable. Cities all around western Europe have started designing their cities around humans and public transit, and not around cars. E.g. Paris is planning to be a 15-minute city, Barcelona is incorporating the superblock design, Finland is creating non-intersecting paths between the common intersections and the Dutch are using intelligent traffic light systems to manipulate traffic flow in order to optimize it for both cars and pedestrians.

If one takes a look at how the Dutch infrastructure is designed, they will see that despite having less car lanes and overall less space for cars, the traffic flows more smoothly. This is thanks to the intelligent design of intersections, traffic lights and infrastructure. The methodology of this has been known ever since the 1970s, but the auto industry has been fighting against it ever since in order to gain more market. In Northern America one can observe what happens if a city is designed with cars in mind, requiring everyone to own a vehicle in order to participate in society. This results in worse accessibility to the city for the disabled, incapable, elderly and young people as well. The methodology of how to create walkable, human-centered and intelligent infrastructure that is optimal for both pedestrians and cars is laid out in detail by books from authors such as Strong Towns, an urban planning organization.

The aim for this research is to able to model a system of roads or city, and being able to pinpoint mistakes made by development engineers, with the goal in mind to make the city more humanly livable, liquidate urban highways and make traffic infrastructure more efficient.

# Part II
# Goals and Outline

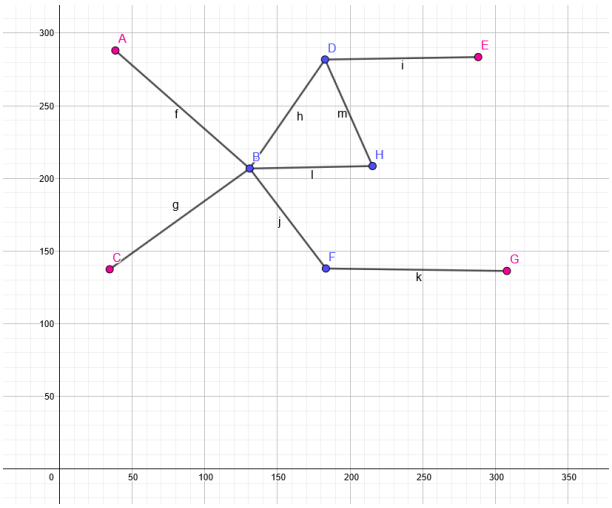
The project will focus on building an interface that models traffic flow in a graph-based structure, then train a reinforcement learning algorithm to find the optimal configuration of the roads in order to transport the most cars in the most effective way possible. Here's when the urban design principles come in: one can easily observe that the most effective way to transport as many cars as possible is if all roads are 8-lane highways. However it's also easy to see that it's miserable to live in a city where there are no quiet, auto-low streets and only 8-lane highways. This might be the best configuration for cars, but it would make the life of people living in the city absolutely horrible. The rewarding system of the reinforcement learning environment will be designed in order to reflect these principles: building cost, traffic light/roundabout tradeoffs, how humans would feel living next to the road. The agent will have to decide where to build, destruct, or make roads 1-way to make the city's transportation flow dynamic but also make it livable for humans. The rewarding scheme will reflect the principles laid down by Strong Towns and other urban planning organizations significant in the field like Happy Cities: Transforming Our Lives through urban design.

# Part III
# Methodology

## 1 Constructing a city

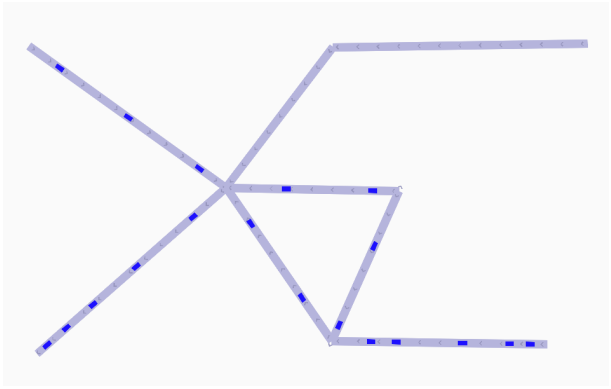
The developed software will provide an interface where the user can make a graph, describing the intersections (nodes) and roads (edges) of the city in question. This for example can be done in GeoGebra and exported into a construction protocol in order to work as an input for the model. Below is an example simple city constructed:

Figure 1:




The interface will read a construction protocol, construct a graph and all the possible pathways from it. And create a starting configuration with 2-way roads between all the nodes of the graph. The vehicle rate and distribution can be controlled before starting the simulation. A constructed "starting" city according to the previously shown graph will look like as follows:

Figure 2:



So far this is a simple configuration for demonstration. The vehicles are passing from one entry point to another, without necessarily choosing the shortest path, or being evenly distributed among all the roads, just as one would find in real life. The drivers' model will incorporate an intelligent behavior, like slowing

down after the car in front is slowing down or gradually speeding up after a light has turned green with a comfortable acceleration parameter.

The road configuration will be examined with multiple metrics like how many steps does it take for the roads to transport 100 cars or how much the road infrastructure would cost. If the agent is handed a road configuration it will be able to find the optimal one, with the least cost, least unnecessary roads and fastest transportation for a given amount of cars.

As a distant goal it would also be reasonable for the application to accept Osmosis data and construct a graph based on that. This would require a more sophisticated preprocessor as the Osmosis data would have to be stripped of metadata and guarantees of format conversions between world-coordinates and graph-node coordinates would have to be implemented. However it is possible to convert into the inner representation format of the road simulation module.

# 2 Deep learning configuration

## 2.1 Action space

A starting configuration will provide the agent with a 2-way road between each graph node. From here on the goal is to add / destroy roads, lanes and intersections in order to optimize the throughput and cost of the road. Each action of the agent will take two graph nodes as a parameter and the roads will be configured accordingly. Any action for node $A$ and $B$ will assume a single directed edge $A \rightarrow B$ starting for $A$ and ending in $B$. There are cases where semantically it would make more sense to have more or less than 2 parameters but these cases can be generalized to an action with two parameters and hence be channeled into a neural network output of the same shape and size as all other cases.

The list of actions for the discrete action space:

1. *add_lane*$(A, B)$*:* Adds a single one way lane going from $A \rightarrow B$.
2. *remove_lane*$(A, B)$*:* Removes a single lane going from $A \rightarrow B$.
3. *add_road*$(A, B)$*:* Adds two lanes between the nodes $A$ and $B$ going $A \rightarrow B$ and $B \rightarrow A$. Only valid in case of nodes that don't have edges connecting them.
4. *remove_road*$(A, B)$*:* Removes two lanes between nodes $A$ and $B$ going from $A \rightarrow B$ and $B \rightarrow A$. Only valid between nodes that have edges connecting them.
5. *add_trafficlight*$(A, B)$*:* Creates a traffic light system to all roads entering the intersection of graph node $B$.
6. *add_roundabout*$(A, B)$*:* Adds a roundabout to all roads entering the intersection of graph node $B$.

7. *reset_intersection(A, B):* Removes current traffic light and roundabout infrastructure to create a right-hand priority intersection in graph node $B$.

## 2.2 State space

### Representation

First, the plan will focus on how to represent a certain type of road between two nodes. The state between graph node $A$ and $B$ will have to be represented by a single number in every case. The bases that this scalar value will have to cover the number of lanes between $A \rightarrow B$ and the type of intersection in the node $B$.

For a state-vector element corresponding to the one-way connection between nodes $A$ and $B$ the possible values are as follows:

1. One-lane road between $A \rightarrow B \Longrightarrow 1$.
2. Two-lane road betwen $A \rightarrow B \Longrightarrow 2$.
3. Two-way road, one lane in each direction: $A \rightarrow B \Longrightarrow 1$; $B \rightarrow A \Longrightarrow 1$.
4. Two way road, two lanes in each direction: $A \rightarrow B \Longrightarrow 2$; $B \rightarrow A \Longrightarrow 2$.
5. One-lane road ending in roundabout between $A \rightarrow B \Longrightarrow 11$.
6. Two-lane road ending in a traffic light junction between $A \rightarrow B \Longrightarrow 22$.

The state vector keeps track of the connections in a directed fashion, storing the number of lanes from $A \rightarrow B$ and $B \rightarrow A$ in separate values. A single scalar of $k$ means that it's a $k$-laned road ending in a right-hand priority intersection without any dedicated infrastructure. A scalar of value $10 < k < 20$ means that the road ends in a roundabout. A scalar of value $20 < k < 30$ means that the road ends in a traffic light based junction. The last digit always translates to the number of lanes going from $A \rightarrow B$.

### Naive implementation

The state space requires precise designing and execution as there's a possibility of exponential explosion if it's represented using an all-to-all fashion. If the state space keeps track of all the incoming connections from all the nodes to all other nodes, for a graph $G$ of $k$ nodes the state space will require a vector of length $k^2 - k$. There's no need to record the recusive connections.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $N_1$ | $N_1$ | $N_1$ | $N_1$ | ... | $N_k$ | $N_k$ | $N_k$ | $N_k$ |
| $N_2$ | $N_3$ | ... | $N_k$ | ... | $N_1$ | $N_2$ | ... | $N_{k-1}$ |

This gives reason to explore the possibilities of being able to shorten the state space vector by eliminating connectioins that are for sure not going to participate in the learning.

**Radius-based approach**

It's easy to notice that it isn't needed to have roads connecting all intersections with all other intersections. This would make the city a convoluted mess. A valid approach to reduce the size of the state vector is to define a radius around the nodes and only store the number of lanes between them. For this a radius parameter will have to be input in order to define the possible connections before running the simulation. On the image below an example of such a circle is shown for graph node $E$ with the radius of 150.

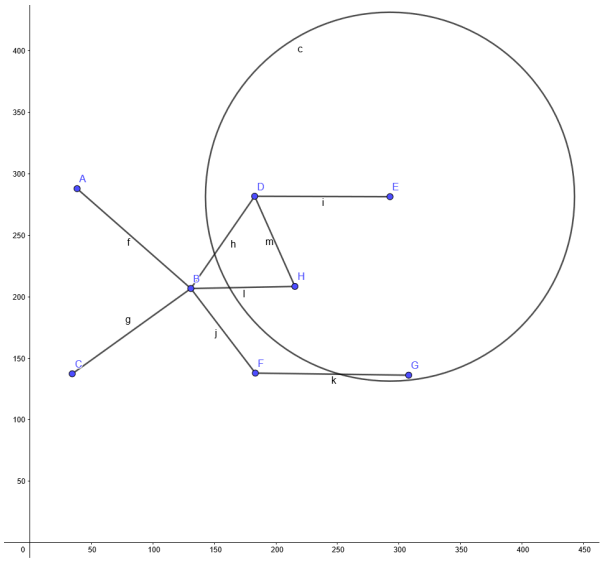



Figure 3:

It is fairly straightforward to see that this would result in a configuration where no roads' length exceed the radius of the neighborhood. This can be considered as a good design approach because it would eliminate the need to build long and straight roads through the city which are shown to be more dangerous regarding traffic accidents. The reason for this is that motorists can go with higher speeds as the road is wide and forgiving like a highway. It is empirically shown that the best way to reduce speeds is to force the drivers to obey speed limits by creating the geometry of the road in a way that enforces it. One won't go with 70-90 km/h on a street that is say 150 meters long.

The disadvantage of this method is that the state vector would be unevenly distributed between the nodes of the graph. Some of the nodes would have more elements in the vector than others resulting an unnecessary high representation. It's also possible that an outlier node gets cut off from the rest, and becomes an 'island' that is unreachable from anywhere. This happening is considered a design fail.

**NN-based method**

The number of tracked connections in the state vector can also be reduced if only the $m$ nearest neighbors to a node is allowed. This would allow for a more evenly distributed state representation as for each node there's only a need to register $m$ connections instead of $k-1$. This approach would however also limit the number of incoming connections into a junction. E.g. if the parameter is set to $m = 4$ the previously shown graph would not be possible to construct and some node may be separated from all other ones, isolating them from the rest of the infrastructure. This is considered a bad resolution of the problem.

A combination of the previously mentioned methods can also be taken into consideration in order to reduce the state vector size, e.g. allowing connections from a specific radius but in cases where there's not a sufficient number of possible neighboring nodes the system will have to find the $m$ closest nodes and keep track of roads coming and going from them.

## 2.3 Rewarding mechanism

// Rewardin scheme comes here

## 2.4 Network architecture

// Network architecture comes here