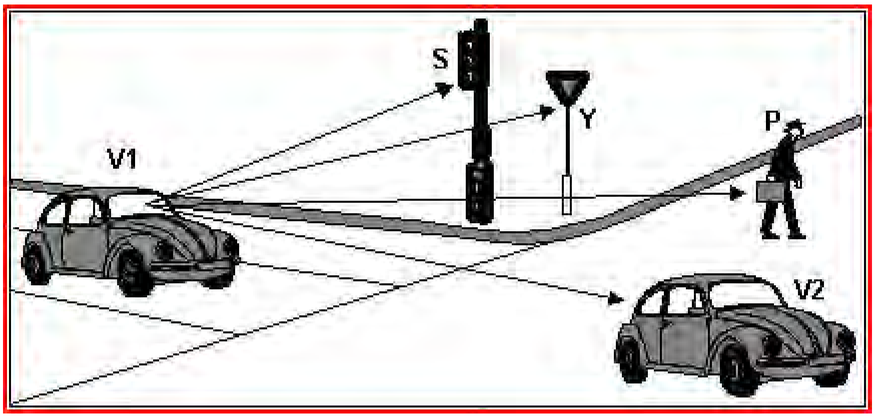


Optimal route in a road traffic network

BMI Paper

Author:
Valentina Maccatrozzo

VU University Amsterdam
De Boelelaan 1081
1081 HV Amsterdam
The Netherlands

Optimal route in a road traffic network

BMI Paper

Author:
Valentina Maccatrozzo

Supervisor:
Dr. Sandjai Bhulai

March 3, 2011

Abstract

We develop a road traffic Cellular Automata model suitable to an urban environment. North, east, south and west car displacements are possible and road crossings are naturally implemented as rotary junctions. We consider the traffic in a small part of Amsterdam, a Manhattan city-like area. Besides the simulation, we apply a Reinforcement Learning technique to derive optimal routes. The Reinforcement learning approach implies to define a value function V on the state space X, and to learn the values for each state $x \in X$. To make a complete description of the system, the state space becomes too large. Therefore we propose the use of a Temporal-Difference Q-Learning algorithm, that avoids having to learn the complete transition model because the Q-value of a state can be related directly to those of its neighbours.

Preface

This paper is part of acquiring the Masters degree in Business Mathematics and Informatics. Business Mathematics and Informatics is a multidisciplinary program, aimed at business process optimization by applying a combination of methods based upon mathematics, computational intelligence and business management. These three disciplines will also play a central role throughout this paper.

The subject of this study is on how to determine the optimal route in a road traffic network. To model the behavior of vehicles in a network system, we will use Cellular Automata (CA) and Reinforcement Learning (RL) techniques. The purpose is to find an appropriate model to describe the traffic network system, where multiple streets are connected to each other. We will use RL techniques, which are techniques to learn optimal decisions based on the simulation type of algorithms. We will investigate how to apply the Temporal-Difference (TD) Q-Learning algorithm to derive optimal routes.

We will provide an overview of existing approaches to simulate a road traffic network and to determine optimal routing. Then we will give a basic introduction to the mathematical background of these techniques. After implementing a simulation program based on CA to model the traffic dynamics, we will define the TD Q-Learning algorithm to derive routing policies. Eventually we will validate the model by applying it.

Finally, I would like to thank my first supervisor, Dennis Roubos, and my actual one, Sandjai Bhulai, for their help and support in writing this paper.

Valentina Maccatrozzo
Amsterdam, 2011

Contents

Introduction

Road transportation, *e.g.*, the efficient movement of people and goods through physical road and street networks, is a fascinating problem. Traffic systems are characterised by a number of features that make them hard to analyze, control and optimise. The systems often cover wide physical areas, the number of active participants is high, the goals and objectives of the participants are not necessarily parallel with each other or with those of the system operator (system optimum vs. user optimum), and there are many system inputs that are outside the control of the operator and the participants (the weather conditions, the number of users, etc.).

In addition, road and street transportation systems are inherently dynamic in nature, that is, the number of units in the system varies with time, and with a considerable amount of randomness. The great number of active participants at present at the same time in the system means a great number of simultaneous interactions.

In this work we propose a Cellular Automata (CA) model to describe the behavior of the vehicles. By means of simulation and the implementation of the CA model in the simulation, the behavior of the vehicles through a network can be modeled. At each intersection, a vehicle can make a decision to either turn left, turn right or drive forward. The optimal decision is learned by means of Reinforcement Learning (RL) techniques. The purpose is to define a value function on the state space, and to learn the values for each state. A complete description of the system would consist of the actual position for every vehicle in the network, therefore the state space becomes very large. The solution we propose is to use a Temporal-Difference Q-Learning algorithm. This is a particular RL algorithm which avoids having to learn the complete transition model because the Q-value of a state can be derived directly from those of its neighbours. One can identify a direct relation with the act of driving a car, during which we are actually concerned with what happens around us.

Chapter 1

Road traffic network models

In this chapter we provide an overview of existing methods to find optimal routing in road traffic networks and a presentation of our choices.

1.1 Different points of view

1.1.1 Simulation models

Simulation of road traffic networks can be divided into two categories: microscopic and macroscopic. Road traffic micro-simulation models are computer models where the movements of individual vehicles travelling around road networks are determined by using simple car following, lane changing and gap acceptance rules. They are becoming increasingly popular for the evaluation and development of road traffic management and control systems.

Macroscopic models provide an aggregated representation of traffic, typically expressed in terms of total flows per hour. In such models, all vehicles of a particular group obey the same rules of behaviour. A key limitation of macroscopic simulation models is their aggregate nature. Because they treat traffic flow as a continuous process, they are incapable of capturing the discrete dynamics which arise from the interactions of individual vehicles. In addition, because they are deterministic, these models can provide only average traffic flow metrics. Higher moments of throughput, travel time, and speed are impossible to characterize. The usefulness of most of these models is limited to characterizing the long run behavior of traffic flow and cannot be used for real time traffic analysis and control. [5]

By contrast, micro-simulation models provide a better, and ‘pure’, representation of actual driver behaviour and network performance. They are the only modelling tools available with the capability to examine certain complex traffic problems, *e.g.* intelligent transportation systems, complex junctions, shockwaves, effects of incidents. In this paper, we will concentrate on microscopic simulation models.

One way to model microscopic simulation is represented by car-following and lane changing models. Car-following models focus on describing the detailed manner in which one vehicle follows another. Lane change models try to imitate the behaviour of a vehicle that does a lane changing, behaviour that is hard to simulate. There are many simulators that use such models. For instance PARAMICS [7], MITSIM [25], AIMSUN2 [4] and HUTSIM [11]. In particular, such simulators implement both models in a complementary way.

In [15] a microscopic model of interurban traffic is developed, which uses fuzzy modelling techniques to describe behavioural processes. Fuzzy logic [26] allows the introduction of a quantifiable degree of uncertainty into the modelled process in order to reflect 'natural' or subjective perception of real variables and these can include measures of degrees of 'desire' and 'confidence' in each information source. This approach allows the formation of a traditional (modular) rule base, but is far easier to define and redefine as new data becomes available. Behavioural rules are developed to describe car-following and lane-changing models.

Another interesting apporach is proposed in [17]. In order to be able to have a simulator tool that can work for different models, they adopt a traffic model based on a road database built according to the physical road conditions. As a result, our vehicle model can run only along running lines in a running line network and is able to omit operations of the steering wheel like railways. So, a vehicle runs reasonably on one-dimensional spaces but not on two-dimensional spaces. Therefore the simulation needs not to consume considerable execution time because of the running line network.

1.1.2 Optimization models

The growth of road traffic and the increasing inconvenience and environmental damage caused by road congestion require a significantly more efficient use of the infrastructure for physical transport. Traffic load is highly dependent on parameters such as time, day, season, weather and unpredictable situations such as accidents, special events or construction activities. If these parameters are not taken into account, the traffic control system will create bottlenecks and delays. So the key is to develop such models in order to consider every, or at least the more important features of the whole system. In [18] the authors propose a 'system-optimum approach, but honor the individual needs by imposing additional constraints to ensure that drivers are assigned to acceptable paths only'. Actually they combine the traffic authority's point of view with the users' point of view. The result is the introduction of the concept of the *normal length* of a path, and this could be either its traversal time in the uncongested network, its traversal time in user equilibrium, its geographic distance, or any other appropriate measure. The only condition imposed on the normal length of a path is that it may not depend on the actual flow on the path. Equipped with this definition, they achieve the goal of finding solutions that are fair and efficient at the same time. Another interesting approach is proposed in [10]. Here the authors underline the importance of considering uncertainty in order to find an optimal path, instead of suboptimal ones. They claim that deterministic solutions ignore the inherent stochasticity of traffic as well as changing traffic conditions. The idea is to use utility or cost functions in order to express the trade-off between speediness and reliability: in particular they consider a variety of stochastic route planning problems, with an emphasis on cost functions that value timeliness without time-wasting.

In [3] a hierarchical routing system is developed, *i.e.*, traffic networks are splitted into several smaller and less complex networks by introducing a hierarchy. The system therefore consists of several distributed routing systems where each one is responsible for one network of the hierarchical network. The route optimization is done with an adapted version of the AntNet-algorithm, a decentralized routing algorithm, which uses intelligent agents that explore the network and find the shortest routes.

1.2 Our approach

In this work we decided to use Cellular Automata (CA) as simulation model and Reinforcement Learning (RL) as optimization model. Our objective is to find optimal routes to drive from one location to a destination. So CA will be used to model the network and the optimal decision has to be learned by means of RL techniques. First we give a general overview of the methods, and in the following chapter we will give a more mathematical description.

1.2.1 Cellular Automata

> *Take a board, and divide it up into squares, like a chess-board or checker-board. These are the cells. Each cell has one of a finite number of distinct colors — red and black, say, or (to be patriotic) red, white and blue. (We do not allow continuous shading, and every cell has just one color.) Now we come to the 'automaton' part. Sitting somewhere at one side of the board is a clock, and every time the clock ticks the colors of the cells change. Each cell looks at the colors of the nearby cells, and its own color, and then applies a definite rule, the transition rule, specified in advance, to decide its color in the next clock-tick; and all the cells change at the same time. (The rule can say 'Stay the same'.) Each cell is a sort of very stupid computer — in the jargon, a finite-state automaton — and so the whole board is called a cellular automaton, or CA. To run it, you color the cells in your favorite pattern, start the clock, and stand back*[1].

Cellular automata models are a very efficient way to implement car motion, as it has been demonstrated by many authors (see for instance [14]). A pioneering work is that of Nagel and Schreckenberg who showed that a CA, *i.e.*, fully discrete dynamics of simple idealized vehicles, can capture several essential features of the real traffic flow [6, 16, 20]. In this approach, cars are represented as points moving on a discretized road with only a small set of possible velocities and accelerations.

In a vehicle-based model, cars represent the 'microscopic' constituents of the system. However, car traffic is a macroscopic phenomena whose typical time and spatial scale extends much beyond that of the single vehicle. The fact that over-simplified microscopic modeling provides an accurate description of the macroscopic level is a common observation for many complex systems. As a matter of fact, the CA approach has been a very successful tool to model several processes in physics and related domains [9].

Single lane car traffic can be modelled as follows. The road is represented as a

[1] C.R. Shalizi, `http://www.cscs.umich.edu/~crshalizi/notabene/cellular-automata.html`

line of cells, each of them being occupied or not by a vehicle. All cars travel in the same direction (say to the right). Their positions are updated synchronously, in successive iterations (discrete time steps). During the motion, each car can be at rest or jump to the nearest neighbor site, along the direction of motion. The rule is simply that a car moves only if its destination cell is empty. In this model, the drivers do not know whether the car in front will move or is blocked by another car. Therefore, the state of each cell $s_i \in \{0, 1\}$ is entirely determined by the occupancy of the cell itself and its two nearest neighbors s_{i-1} and s_{i+1}. This dynamics can be summarized by the relation:

$$s_i(t+1) = s_{i-1}(t)(1 - s_i(t)) + s_i(t)s_{i+1}(t), \tag{1.1}$$

where t denotes the iteration step.
A richer version of the above CA model has been developed in [14, 16, 24]. The cars may have several possible velocities $u = 0, 1, 2, ..., u_{max}$. Let u_i be the velocity of car i and d_i the distance, along the road, separating cars i and *i+1*. The updating rule is:

- The cars accelerate when possible: $u_i \rightarrow u'_i = u_i + 1$, if $u_i < u_{max}$.
- The cars slow down when required: $u'_i \rightarrow u''_i = d_i - 1$, if $u'_i \geq d_i$.
- The cars have a random behavior: $u''_i \rightarrow u'''_i = u''_i - 1$, with probability p_i if $u''_i > 0$.
- Finally the cars move u'''_1 sites ahead.

This rule captures some important behaviors of real traffic on a highway: velocity fluctuations due to a non-deterministic behavior of the drivers, and âĂIJstop-and-goâĂİ waves observed in a high density traffic regime (*i.e.*, some cars get stopped for no specific reasons).
In [8] a two-dimensional road network model is presented. They assume that horizontal roads consists of two lanes, one for eastward motion and the other for westward motion. Vertical streets are composed of northbound and southbound lanes. The road junction is defined as a rotary: a central point around which the traffic moves always in the same direction. A vehicle in a rotary has priority over any entering car.
Figure 1.1 illustrates how the implementation works. The four middle cells constitute the rotary. A vehicle in the rotary (like *b* or *d*) can either rotate counterclockwise or exit. A local flag *f* is used to decide the motion of a car in a rotary. If *f*=0, the vehicle (like *d*) exits in the direction allowed by the color of its lane (see figure caption). If *f*=1, the vehicle moves counterclockwise, like *b*. The value of the local turn flag *f* can be updated according to the modeling needs: it can be constant for some amount of time to impose a particular motion at a given junction, completely random, random with some bias to favor a direction of motion, or may change deterministically according to any user-specified rule. As in the one-dimensional rule, a vehicle moves only when its destination cell is empty. Far from a rotary, the state of the destination cell is determined by the occupation of the down-motion cell. This is also the case for a vehicle turning at the rotary. On the other hand, a car wanting to enter the rotary has to check two cells because it has not the priority. This check is made by looking at the turn flag *f* of the neighboring cells having the priority.

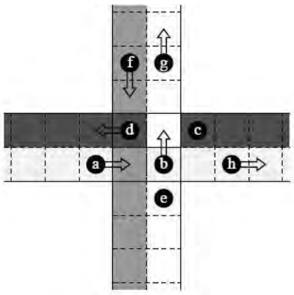


Figure 1.1: *Example of a traffic configuration near a junction. The four central cells represent a rotary which is traveled counterclockwise. The grey levels indicate the different traffic lanes: white is a northbound lane, light gray an eastbound lane, grey a southbound lane and, finally, dark grey is a westbound lane. The dots labeled a, b, c, d, e, f, g and h are cars which will move to the destination cell indicated by the arrows, as determined by the cell turn flag f. Cars without an arrow are forbidden to move.*

For instance, car *c* cannot enter the rotary because b is going to move to the white cell. Car *e* cannot move either because it sees *b* (and cannot know whether or not *b* will actually move). Car *a*, on the other hand can enter because it sees that *d* is leaving the rotary and that the grey cell ahead is free.
Similarly, the incoming vehicle to a given cell is computed differently inside and outside of the rotary. The light grey cell occupied by car *b* has two possible inputs: with priority, it is the vehicle from the grey cell at west; if this cell is empty, the input will be the incoming lane, namely the car labeled *e*.

1.2.2 Reinforcement Learning

Reinforcement learning is a method used to find the best solution to a real problem facing a learning agent interacting with its environment to achieve a goal. The most important feature distinguishing reinforcement learning from other types of learning is that it uses training information that evaluates the actions taken rather than instructs by giving correct actions. As a result the challenge of making a trade-off between exploration and exploitation arises. To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions that is has not selected before. The agent has to exploit what it already knows in order to obtain rewards, but it also has to explore to make better action selections in the future. Another key factor of reinforcement learning is that it explicitly considers the whole problem, which is in contrast with many approaches that consider subproblems without addressing how they might fit into a larger picture.
Beyond the agent and the environment, one can identify four main subelements of a reinforcement learning system:

- The *policy* is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine behavior.

- A *reward function* defines the goal in a reinforcement learning problem. Roughly speaking, it maps each perceived state (or state-action pair) of the environment to a single number, a reward, indicating the intrinsic desirability of that state.
- A *value function* specifies what is good in the long run. Roughly speaking, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.
- A *model* mimics the behaviour of the environment. For example, given a state and action, the model might predict the resulting next state and next reward.

Markov Decision Process

A stochastic process has the Markov property if the conditional probability distribution of future states of the process, given the present state and a number of past states, depends only upon the present state and not on the given states in the past, *i.e.*, it is conditionally independent of these older states. In other words, the evolution of a Markov process from some point in time t_n does not depend on the history but only on the current state $\mathcal{S}_n$. It can be seen as a memoryless property. In mathematical formulas this looks like:

$$\begin{aligned} &Pr\{\mathcal{S}_{t_n} = s_n | \mathcal{S}_{t_1} = s_1, \ldots, \mathcal{S}_{t_{n-1}} = s_{n-1}\} \\ &\qquad = Pr\{\mathcal{S}_{t_n} = s_n | \mathcal{S}_{t_{n-1}} = s_{n-1}\}, \end{aligned} \tag{1.2}$$

for all $t_1 < \cdots < t_n$.
In the case of real-time travel-time predictions, we would like to develop a Reinforcement Learning task in the framework of Markov Decision Processes (MDPs). Therefore we need state and action spaces, denoted by $\mathcal{S}$ and $\mathcal{A}(s)$, respectively, which we assume are finite. A particular finite MDP is defined by its state and action sets and by the dynamics of the environment. The policy, π, is a mapping from each state to an action, *i.e.* $\pi(s) = a$ means that in state s action a is taken. Given any state and action, s and a, the probabilities of the next state, *s'*, is

$$\mathcal{P}^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\},$$

These quantities are called *transition probabilities.* Similarly, given any current state and action, s and a, together with any next state, *s'*, the expected value of the *reward* is

$$\mathcal{R}^a_{ss'} = E\{r_{t+1} | a_t = a, s_t = s, s_{t+1} = s'\},$$

These quantities, $\mathcal{P}^a_{ss'}$ and $\mathcal{R}^a_{ss'}$, completely specify the most important apsects of the dynamics of a finite MDP. Before we move on, we will make the following three assumptions. Relaxing any of these is possible, but usually leads to additional constraints or complications. Moreover, in practical situations all these constraints are satisfied, but we have to check them before we can apply this method on the real-time travel time problem. Before formulating the assumptions, it is convenient to define the notion of a path in a Markov chain.

Definition A sequence of states $z_0, z_1, \ldots, z_{k-1}, z_k \in \mathcal{S}$ with the property that $p(z_0, z_1), \ldots, p(z_{k-1}, z_k) > 0$ is called a *path* of length k.

Assumption 1 $|\mathcal{S}| < \infty$ and $|\mathcal{A}(s)| < \infty$ for all $s \in \mathcal{S}$.

Assumption 2 For every policy π, there is at least one state $s \in \mathcal{S}$ (that may depend on π), such that there is a path from any state to s. If this is the case we call the chain *unichain*, and state s is called recurrent.

Assumption 3 For every policy π, the greatest common divisor of all paths from s to s is 1, for some recurrent state s. If this is the case we call the chain *aperiodic.*

Value functions

Almost all reinforcement learning algorithms are based on estimating value functions - functions of states (or of state-action pairs) that estimate how good it is to perform a given action in a given state. Informally, the value of a state s under a policy π, denoted $V^\pi(s)$, is the expected return when starting in s and following π thereafter. For a MDP two value functions can be defined:

- the *state-value function for policy* $\pi, V^\pi(s)$, defined as the expected return when starting in s and following π thereafter.

  $$V^\pi(s) = E_\pi \sum_{t=0}^{T-1} r(s_t) = E_\pi \sum_{t=0}^{T-1} \sum_{s' \in \mathcal{S}} p^t(s, s') r(s'), \tag{1.3}$$

  where $E_\pi\{\}$ denotes the expected value given that the agent follows policy π.

- the *action-value function policy* π, $\mathcal{Q}^\pi(s, a)$, defined as the expected return starting from s, taking the action a, and thereafter following policy π.

  $$\mathcal{Q}^\pi(s, a) = r(s_0, a) + E_\pi \sum_{t=1}^{T-1} r(s_t) = r(s_0, a) + E_\pi \sum_{t=1}^{T-1} \sum_{s' \in \mathcal{S}} p^t(s, s') r(s'). \tag{1.4}$$

A fundamental property of value functions used throughout reinforcement learning is that they satisfy particular recursive relationships.
Define $V(s) = \min_\pi V^\pi(s)$, then for any policy π we have

$$V(s) + g = r(s, \pi(s)) + \sum_{s' \in \mathcal{S}} p(s, \pi(s), s') V(s'). \tag{1.5}$$

This equation is called the Poisson equation. This equation averages over all the possibilities, weighing each by its probability of occuring. It states that the value of the start state must equal the value of the expected next state, plus the reward expected along the way. Note that Equation (1.5) does not have a unique solution. There are two possible solutions to this problem: either take $V(0) = 0$ for some reference state 0, or add the additional condition $\sum_{s \in \mathcal{S}} \pi_*(s) V(s) = 0$. Only under the latter condition, V has the interpretation as the total expected difference in reward between starting in a state and starting in stationarity.

Optimal value functions

Solving a reinforcement task means, roughly, finding a policy that achieves a lot of reward over the long run. For finite MDPs, we define an optimal policy in the following way. Value functions define a partial ordering over policies. A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states. In other words, $\pi \geq \pi'$ if and only if $V^{\pi}(s) \geq V^{\pi'}(s)$ for all $s \in \mathcal{S}$. There is always at least one policy that is better than or equal to all other policies. This is an *optimal policy.* Although there may be more than one, we denote all the optimal policies by π^*. They share the same state-value function, called the *optimal state-value function*, denoted by V^*, and defined as

$$V^*(s) = \max_{\pi} V^{\pi}(s), \tag{1.6}$$

for all $s \in \mathcal{S}$. Optimal policies also share the same *optimal action-value function*, denoted by $\mathcal{Q}^*$, and defined as

$$\mathcal{Q}^*(s,a) = \max_{\pi} Q^{\pi}(s,a), \tag{1.7}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$. For the state-action pair (s,a) this function gives the expected return for taking action a in state s and thereafter following an optimal policy. Thus, we can write $\mathcal{Q}^*$ in terms of V^* as follows:

$$\mathcal{Q}^*(s,a) = \max_{a \in \mathcal{A}(s)} \left\{ r(s,a) + \sum_{s' \in \mathcal{S}} p(s,a,s')V^*(s') \right\}. \tag{1.8}$$

Because V^* is the value function for a policy, it must satisfy the self-consistency condition given by the Bellman equation for state values (1.5). Because it is the optimal value function, however, V^*'s consistency condition can by written in a special form without reference to any specific policy. This is the Bellman equation for V^*, or the *Bellman optimality equation.* Intuitively, the Bellman optimality equation expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state:

$$r(s,\pi^*(s)) + \sum_{s' \in \mathcal{S}} p(s,\pi^*(s),s')V^{\pi^*}(s') = \max_{a \in \mathcal{A}(s)} \left\{ r(s,a) + \sum_{s' \in \mathcal{S}} p(s,a,s')V^{\pi}(s') \right\}.$$

At the same time, by the Poisson equation:

$$V^{\pi^*}(s) + g^{\pi^*} = r(s,\pi^*(s)) + \sum_{s' \in \mathcal{S}} p(s,\pi^*(s),s')V^{\pi^*}(s').$$

Combining these two gives the Bellman optimality equation for V^*:

$$V^{\pi^*}(s) + g^{\pi^*} = \max_{a \in \mathcal{A}(s)} \left\{ r(s,a) + \sum_{s' \in \mathcal{S}} p(s,a,s')V^{\pi^*}(s') \right\}. \tag{1.9}$$

The Bellman optimality equation for $\mathcal{Q}^*$ is:

$$\mathcal{Q}^{\pi^*}(s,a) + g^{\pi^*} = \max_{a' \in \mathcal{A}(s')} \left\{ r(s,a') + \sum_{s' \in \mathcal{S}} p(s,a',s')\mathcal{Q}^*(s',a') \right\}. \tag{1.10}$$

For finite MDP's, the Bellman optimality equation (1.9) has a unique solution independent of the policy. The Bellman optimality equation is actually a system of equations, one for each state, so if there are N states, then there are N equation in N unknowns. If the dynamics of the environment are known ($\mathcal{R}^a_{ss'}$ and $\mathcal{P}^a_{ss'}$), then in principle one can solve this system of equations for V^* using any one of a variety of methods for solving systems of nonlinear equations. This can be done by using Dynamic Programming. One can solve a related set of equations for $\mathcal{Q}^*$.

Once one has solved V^*, it is relatively easy to determine an optimal policy. For each state s, there will be one or more actions at which the maximum is attained in the Bellman optimality equation. Any policy that assigns nonzero probability only to these actions is an optimal policy. You can think of this as a one-step search. If you have the optimal value function, V^*, then the actions that appear best after a one-step search will be optimal actions. Another way of saying this is: any policy that is *greedy* with respect to the optimal expected long-term return is turned into a quantity that is locally and immediately available for each state. Hence, a one-step-ahead search yields the long-term optimal actions.

Having $\mathcal{Q}^*$ makes choosing optimal actions still easier. With $\mathcal{Q}^*$, the agent does not even have to do a one-step-ahead search: for any state s, it can simply find any action that maximizes $\mathcal{Q}^*(s, a)$. The action-value function effectively caches the results of all one-step-ahead searches. It provides the optimal expected long-term return as a value that is locally and immediately available for each state-action pair. Hence, at the cost of representing a function of state-action pairs, instead of just of states, the optimal action-value function, when $\mathcal{Q}^*$ is known, allows optimal actions to be selected without having to know anything about possible successor states and their values, that is, without having to know anything about the environment's dynamics.

Explicitly solving the Bellman optimality equation provides one route to finding an optimal policy, and thus to solve the reinforcement learning problem. However, this solution is not always directly useful. It is akin to an exhaustive search, looking ahead at all possibilities, computing their probabilities of occurrence and their desirabilities in terms of expected rewards. This solution relies on at least three assumptions that have to be checked before we can use this method:

1. we accurately know the dynamics of the environment;
2. we have enough computational resources to complete the computation of the solution;
3. meet the Markov Property.

If the solution cannot be implemented exactly, there are many different decision-making methods which can be viewed as ways of approximately solving the Bellman optimality equation [2].

Chapter 2

Mathematical Models and Methods

In this chapter we give a detailed mathematical explanation of the methods we used.

2.1 Cellular automata

In this work, we are interested in simulating the traffic of the city of Amsterdam. The model presented in the previous chapter can be adapted and extended according to the situation to be simulated. In particular we will concentrate on a small part of the city.

2.1.1 A model for urban traffic

Each road segment is discretized into cells of constant length (7,5 meters long) to form a one-dimensional CA. In any given cell, there is at most one vehicle. Each vehicle will move if the next cell is free. We consider one possible velocity, since we consider only cars driving in the city.

The behavior of cars at crossings is modelled as a rotary on which entering and exiting lanes are connected. The junctions are the vertices of the graph representing the city and the connecting road segments are the edges.

A rotary is also a 1D CA in which the first and last cells are connected. Clearly, the maximum capacity of such a junction is limited by the capacity of a 1D CA, which is 1 vehicle every 2 steps. The advantage of representing a junction as a rotary is twofold:

1. the rule of motion for road segments or crossings can be implemented in the same way;

2. vehicles in rotaries always have priority over the other cars; this gives a natural and simple way to deal with concurrency problems in a situation where all car move synchronously.

Just before entering a rotary, the vehicle has to give the right of way if there is a vehicle already in the rotary, otherwise it enters the rotary. When in a rotary cell, a vehicle has a probability 1/2 to exit the rotary. We can distinguish three types of rotaries:

1. rotaries composed by one cell: the crossing of two one-way streets;

2. rotaries composed by two cells: the crossing of a one-way street and a two-way street;

3. rotaries composed by three cells: the crossing of two two-way streets.

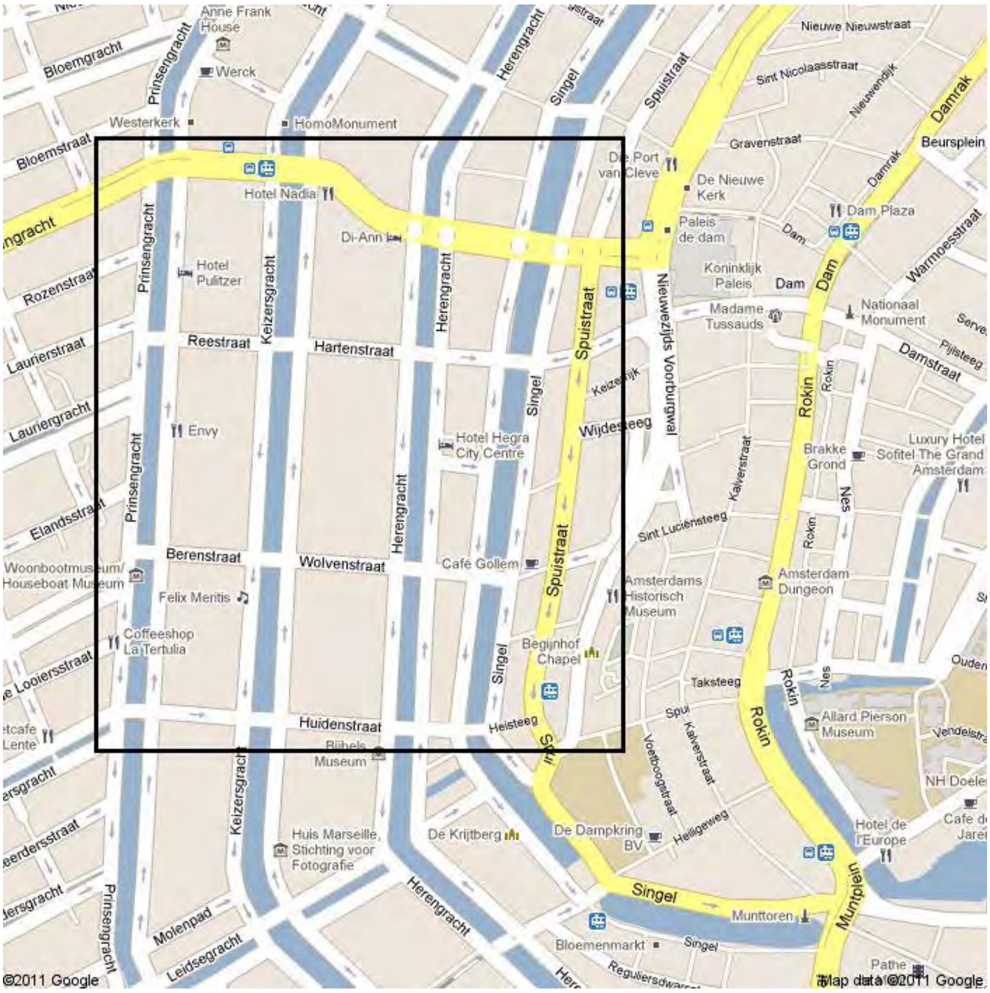


Figure 2.1: The area considered is from the corner made by Prinsengracht and Westermarkt and the corner made by Spuistraat and Reestraat, as indicated by the rectangle.

2.1.2 Routing of vehicles

We consider a very small part of the city of Amsterdam. Figure 2.1 shows the part of the city we considered.

The full network comprises 9 road segments and 20 junctions for about 47 km. After discretization, there are 6308 road cells, 45 entrance and exit cells which connect the junctions to the roads.

In a typical traffic problem, every vehicle follows a given path, which is prescribed by the so-called Origin-Destination (OD) matrix. In our case, the OD matrix is built manually.

2.2 Reinforcement Learning

Reinforcement learning is a computational approach to understand and automate goal-directed learning and decision-making. It is distinguished from other computational approaches by its emphasis on learning by the individual from direct interaction with its environment, without relying on exemplary supervision or complete models of the environment.

Reinforcement learning uses a formal framework defining the interaction between a learning agent and its environment in terms of states, actions, and rewards. This framework is intended to be a simple way of representing essential features of the artificial intelligence problem. These features include a sense of cause and effect, a sense of uncertainty and nondeterminism, and the existence of explicit goals.

The concepts of value and value functions are the key features of the reinforcement learning methods. Value functions are essential for efficient search in the space of policies. The use of value functions distinguishes reinforcement learning methods from evolutionary methods that search directly in the policy space guided by scalar evaluations of entire policies.

We can distinguish between passive and active learning. With passive learning the agent's policy is fixed and the task is to learn the utilities of the states (or state-action pairs); also the rewards are given. With active learning the agent must also learn what to do. In this case the main issue is exploration: an agent must experience as much as possible of its environment in order to learn how to behave in it. In our case we can speak about partial active learning, since our agent knows the basic driving rules.

2.2.1 Active Reinforcement Learning

As we already underlined, an active agent must decide what actions to take, so, if it learns a value function V, it will need to learn a model in order to be able to choose an action based on V via one-step look-ahead as explained by Figure 2.2.

Temporal-difference Q-Learning

Temporal-difference learning is one of the possible ways to approach a reinforcement learning task. This approach uses the observed transitions to adjust the utilities of the observed states so that they agree with the constraint equations.

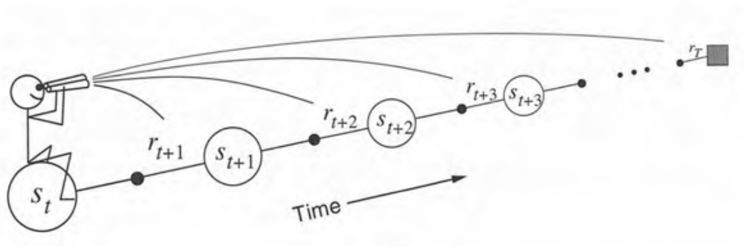


Figure 2.2: We decide how to update each state by looking forward to future rewards and states.

When a transition occurs from state s to state s', we apply the following update to $U^\pi(s)$:

$$U^\pi(s) = U^\pi(s) + \alpha\left(R(s) + \gamma U^\pi(s') - U^\pi(s)\right), \tag{2.1}$$

where α is the learning rate paramater, $R(s)$ is the reward of state s and γ is the discount factor. Because this update rule uses the difference in utilities between successive states, it is often called the temporal-difference (TD) equation.
There is an alternative TD method, called Q-learning, which learns an action-utility function representation instead of learning utilities. We will use the notation $Q(s, a)$ to denote the value of doing action a in state s. Q-values are directly related to utility values as follows:

$$U(s) = \max_a Q(s, a). \tag{2.2}$$

Q-functions may seem like just another way of sorting utility information, but they have a very important property: a TD agent that learns a Q-function does not need a model of the form $P(s'|s, a)$, either for learning or for action selection. For this reason, Q-learning is called a $model - free$ method. As with utilities, wa can write a constraint equation that must hold at equilibirum when the Q-values are correct:

$$Q(s, a) = R(s) + \gamma \sum_{s'} P(s'|s, a) \max_{a'} Q(s', a'). \tag{2.3}$$

We can use this equation directly as an update equation for an iteration process that calculates exact Q-values, given an estimated model. This does, however, require that a model also must be learned, because the equation uses $P(s'|s, a)$. The temporal-difference approach, on the other hand, requires no model of transitions, all it needs are the Q-values. The update equation for a TD Q-learning is:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\Big(R(s) + \gamma \max_a Q(s', a') - Q(s, a)\Big), \tag{2.4}$$

which is calculated whenever action a is executed in state s leading to state s' [19].

The algorithm

The problem model consists of an agent, states S and a set of actions per state A. By performing an action $a \in A$, the agent can move from state to state, *i.e.*, move from cell to cell. Each state provides the agent a reward (a real or natural number). The goal of the agent is to maximize its total reward. It does this by learning which action is optimal for each state. The algorithm therefore has a function which calculates the Quality of a state-action combination. Before the learning phases has started, Q returns a fixed value, chosen by the designer. Then, each time the agent is given a reward (the state has changed) new values are calculated for each combination of a state s from S, and action a from A. The core of the algorithm is a simple value iteration update. It assumes the old value and makes a correction based on the new information.

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha(s_t, a_t)}_{\text{learning rate}} \left[\overbrace{\underbrace{R(s_{t+1})}_{\text{reward}} + \underbrace{\gamma}_{\text{discountfactor}} \underbrace{\max_a Q(s_{t+1}, a)}_{\text{max future value}}}^{\text{expected discounted reward}} - \overbrace{Q(s_t, a_t)}^{\text{old value}} \right],$$

where $R(s_{t+1})$ is the reward observed from s_{t+1}, $\alpha(A, a)(0 < \alpha \leq 1)$ is the learning rate. The learning rate determines to what extent the newly acquired information will override the old information. A factor of 0 will make the agent not learn anything, while a factor of 1 would make the agent consider only the most recent information. In our case, it is a function that decreases as the number of times a state has been visited increases. The discount factor γ is such that $0 \leq \gamma < 1$. the discount factor describes the preference of an agent for current rewards over future rewards. When γ is close to 0, rewards in the distant future are viewed as insignificant, while a factor approaching 1 will make it strive for a long-term high reward. In our case it is 0.7.

Chapter 3

Deriving optimal routes

In this chapter, we are going to describe in details the application and the results obtained with our model. The results of this work can be divided into two sections: the first section regards the application of the Cellular Automata simulation technique to a road traffic network; the second regards the application of a Reinforcement Learning technique to learn in a simulated road traffic network optimal routes.

3.1 Simulation

Since we are simulating a very central area of Amsterdam, we miss information about the real evolution of the traffic state in the city. Information about the main highways can be found, but not about the center. Despite this, we reached some important results that can validate our model.

3.1.1 Scheduling of vehicles

Any vehicle in the network is put in the grid from one of the entrances. The entrance is selected by the generation of a random number. Once in the network, the car will follow a predefined path, as described in the OD matrix. In our model, each car is characterized by a path number which indicates the sequence of crossroads the vehicle must travel to reach its destination. Each junction is labeled. A centralized table gives, for each path number and current junction label, what is the next junction to reach. When the vehicles reach a crossing, they enter the rotary and select the appropriate exit, as specified by the routing direction.

3.2 Reinforcement Learning

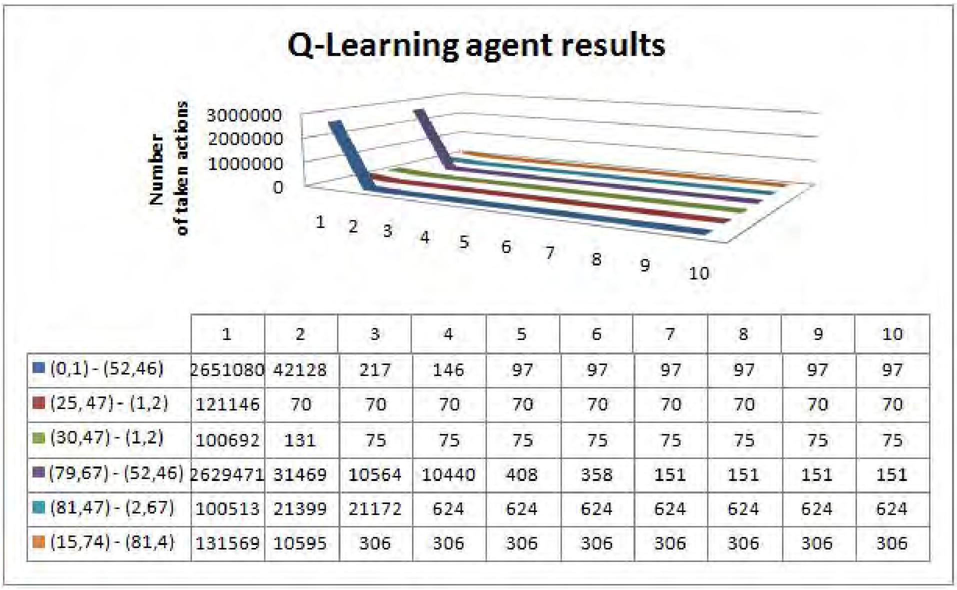
Active reinforcement learning needs to get familiar with the model, in order to learn to behave in the right way. This is the part of the algorithm which is called exploration. The objective of this part is to maximize the long-term well being and to try to get to unexplored states. The part that actually uses the results of the exploration is called exploitation. The objective of this part is to maximize the agent's reward and it depends upon the current utility estimation.

Pure exploitation risks getting stuck in a rut, while pure exploration to improve one's knowledge is of no use if one never puts that knowledge into practice. The agent should make a trade off between the two. We adopted a simple combined approach:

- choose 10000 random actions in the beginning;
- then follow the greedy policy.

3.3 The shortest path

The area of Amsterdam we considered is quite a Manhattan city-like area, but we had to 'shorten' or 'strech' some streets, in order to create a real Manhattan map. Due to this fact, we decided to compare our results with Google Maps[1] looking at the shortest path, and not the fastest, since our results would be not reliable.



	1	2	3	4	5	6	7	8	9	10
■ (0,1) - (52,46)	2651080	42128	217	146	97	97	97	97	97	97
■ (25,47) - (1,2)	121146	70	70	70	70	70	70	70	70	70
■ (30,47) - (1,2)	100692	131	75	75	75	75	75	75	75	75
■ (79,67) - (52,46)	2629471	31469	10564	10440	408	358	151	151	151	151
■ (81,47) - (2,67)	100513	21399	21172	624	624	624	624	624	624	624
■ (15,74) - (81,4)	131569	10595	306	306	306	306	306	306	306	306

Figure 3.1: The results of the Q-Learning agent.

In the graph, the x-axis represents the number of times the agent repeats the attempt to reach its destination, after exploration, while on the y-axis is shown the number of actions taken (visited cells) by the agent to arrive at the destination cell. What we can observe from this data is that the agent learns quite fast, and most of the time after a few trials, it arrives at the optimal policy. Comparing the results with those of Google Maps, the paths are the same. However, the behavior of the agent is not always optimal, as it is shown in Figure 3.2. This figure shows the path number 4 ((79, 67)-(52,46)).

[1] `http://maps.google.com/`

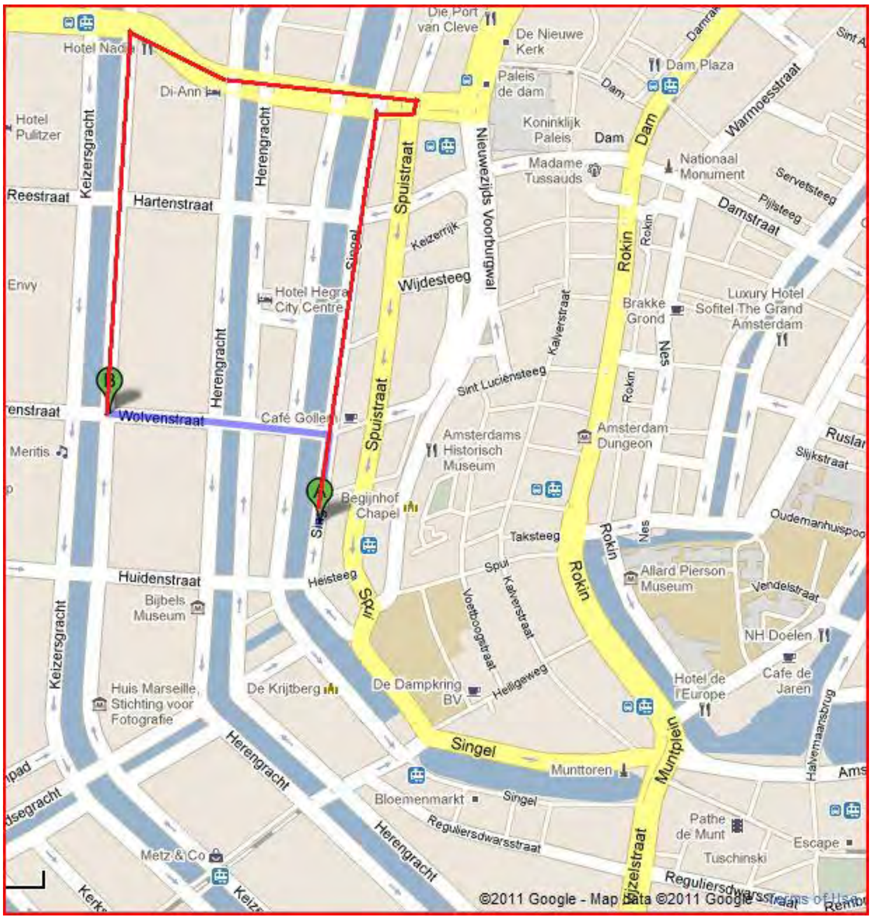


Figure 3.2: Agent vs GoogleMaps

3.4 Conclusions

The model we built has given reasonable results, even if there is still room for improvements.

We can state that it is a good model to represent a traffic road networks and to keep a little state-space description. Actually we think that it is a good model for a driver simulator since a driver actually needs to keep under control only the nearby area.

As it is shown in Figure 3.2, sometimes the agent is stuck on a path. This is probably due to the fact that the exploration part is made by random decisions. So it could happen that some cells are never visited by the agent during exploration, and, therefore, during exploitation, these cells have a very low probability to be visited. Besides, also during exploration, when the agent is in a rotary and the possible cells have the same Q-value, the decision is random.

One possible improvement could be to manage the exploration part with a not completely random strategy. Maybe a different update rule or reduce the randomness in order to visit all the cells. Another possibility could be that of mixing the exploration and the exploitation part: after an exploration period, an exploitation period starts, and when (and if) the agent gets stuck on a (not optimal) path, another exploration period starts. This approach could allow the agent to explore cells it has never visited before. As in Figure 3.2, an exploration period once the agent got stuck on the wrong path, could have helped it to turn left at the right cross.

Bibliography

[1] Dupuis A. and Chopard B. Cellular automata simulations of traffic: A model for the city of Geneva. *Networks and Spatial Economics*, 3, January 2003.

[2] Sutton R.S.and Barto A.G. *Reinforcement Learning: An Introduction.* MIT Press, 2000.

[3] Tatomir B., Dibowsky H., and L.J.M. Rothkrantz. Hierarchical routing in traffic networks. In *Proceedings of the Belgium-Netherlands Artificial Intelligence Conference (BNAIC 2004), Groningen, The Netherlands*, pages 75–82, October 2004.

[4] J.L. Ferrer D. Garcias Barceló J., J. Casas. Modelling advanced transport telematic applications with microscopic simulators: The case of aimsun2. In Heidelberg: Springer, editor, *Traffic and Mobility: Simulation-Economics-Environment.*, pages 205–221, 1999.

[5] Saifallah Benjaafar, Kevin Dooley, and Wibowo Setyawan. Cellular automata for traffic flow modeling, 1997.

[6] W. Brilon and N. Wu. *Evaluation of Cellular Automaton for Traffic Flow Simulation on Freeway and Urban Streets.* Berlin: Springer Verlag, 1999.

[7] Gordon Cameron, Brian J. N. Wylie, and David McArthur. Paramics–moving vehicles on the connection machine. In *Supercomputing '94: Proceedings of the 1994 conference on Supercomputing*, pages 291–300, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[8] B. Chopard, P. O. Luthi, and P.-A. Queloz. Cellular automata model of car traffic in a two-dimensional street network, 1996.

[9] Bastien Chopard and Michel Droz. *Cellular Automata Modeling of Physical Systems.* Cambridge University Press, 1998.

[10] Nikolova E., Brand M., and Karger D.R. Optimal route planning under uncertainty. In *ICAPS*, pages 131–141, 2006.

[11] Kosonen I. *HUTSIM- A Simulation Tool for Traffic Signal Control Planning.* PhD thesis, Helsinki University of Technology.

[12] Tania Jiménez, Philippe Mussi, and Günther Siegel. A road traffic simulator: Car-following and lane-changing. In *Proceedings of the 14th European Simulation Multiconference on Simulation and Modelling*, pages 241–245. SCS Europe, 2000.

[13] Pursula M. Simulation of Traffic Systems - An Overview. *Journal of Geographic Information and Decision Analysis*, (3(1)):1–8, 1999.

[14] Schreckenberg M. and D.E.Wolf. *Proceedings of the conference Traffic and Granular Flow 97.* Singapore: Springer-Verlag, 1998.

[15] J. Wu M. Brackstone McDonald, M. Development of a fuzzy logic based microscopic motorway simulation model. In *Proceedings of ITSC97*, November 1997.

[16] K. Nagel and M. Schreckenberg. A cellular automaton model for freeway traffic. *J. Phys.*, 2:2221–2229, 1992.

[17] Y. Ueda Namekawa, M. and A. Satoh. A road traffic simulation system with a microscopic model using a running line. In *Proceedings of the 18th World IMACS/MODSIM Congress*, 2009.

[18] Jahn O.and Möhring R.H., Schulz A.S., and Stier Moses N.E. System-optimal routing of traffic flows with user constraints in networks with congestion. *Operations Research*, 53:4394–2, 2002.

[19] Russell S. and Norvig P. *Artificial Intelligence: a modern approach.* Pearson, third edition, 2010.

[20] A Schadschneider and M Schreckenberg. Cellular automaton models and traffic flow. *Journal of Physics A: Mathematical and General*, 26(15), 1993.

[21] W.R. Gilks S.Richardson and D.J. Spiegelhalter. *Markov Chain Monte Carlo in Practice.* Chapman & HallCRC, 1995.

[22] Philippe L. Toint. Transportation modelling methods and advanced transport telematics (ATT), 1992.

[23] Pottmeier A. Wahle J., Chrobok R. and Schreckenberg M. A microscopic simulator for freeway traffic. *Networks and Spatial Economics*, 2:371–386(16), December 2002.

[24] D. E. Wolf, M. Schreckenberg, and A. Bachem, editors. *Traffic and Granular Flow*, Singapore, 1996. World Scientific.

[25] QI Yang and Haris N. Koutsopoulos. A microscopic traffic simulator for evaluation of dynamic traffic management systems. *Transportation Research Part C: Emerging Technologies*, 4(3):113–129, 1996.

[26] L. A. Zadeh. Fuzzy sets. *Information and control*, 8(3):338–353, 1965.