Multiagent Reinforcement Learning and Coordination for Urban Traffic Control using Coordination Graphs and Max-plus

Master's Thesis Artificial Intelligence
Specialization Intelligent Systems

Lior Kuyer

Under supervision of:

Dr. Bram Bakker
Dr. Shimon Whiteson

Intelligent Autonomous Systems group
Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Universiteit van Amsterdam

Contents

Acknowledgements

I would like to thank everyone that contributed to the successful completion of this thesis. First, I would like to thank my supervisors, Bram Bakker and Shimon Whiteson, for their guidance, patience and assistance throughout this thesis. Second, I would like to thank my wife Netanela for her love, support, and inspiration. And also for her understanding of the late night sessions. Finally, I would like to thank my parents for all of their support.

Chapter 1

Introduction

The increase of population along with the significant increase in the number of vehicles on the road has resulted in one of the main problems of our modern society, namely traffic jams. These problems are largely present in and around urban areas where traffic lights nowadays form the most important traffic control mechanism. Existing road infrastructure today is often strained nearly to its limits and continuous expansion of this infrastructure is not possible or even desirable due to spatial, economical and environmental reasons. It is therefore highly important to attempt to optimize the behavior of the controllers which will optimize the flow of traffic in such infrastructures.

The optimization problem can be quite complex. In order to find an optimal solution, it would be necessary to know the position, speed and route of every vehicle in the system. Furthermore, all reactions of drivers to changing traffic conditions would need to be known and taken into account. In practice, traffic light controllers need to work with far less information about the current traffic situation. In most cases, the controllers follow some fixed protocol, meaning that the light is red for some time interval and green for some subsequent interval. The intervals usually change during peak hours but are still static. The field of Machine Learning (ML) offers different techniques which can be applied to allow a more flexible and adaptive behavior by the controllers. In fact, the challenge of estimating global optimal behavior of a network of controllers along with the importance of the problem, have led to a great number of research approaches and several ML methods have already been successfully applied to this problem. The latter includes fuzzy logic [3], neural nets [4] and evolutionary algorithms [8]. Most of the methods perform well for small networks but at the same time can only handle networks with a relatively small number of controllers and/or make simplifying assumptions which largely reduce the problem size.

One possible method for finding control strategies in complicated domains is Reinforcement Learning. It has been a popular field in machine learning throughout the last few decades. Its main attraction is that it is a general framework, capable of addressing many real-world problems, and particularly requiring little or no previous knowledge about the solutions to be learned. It can deal with incomplete information and stochastic changes in the environment, which are two complications present in the traffic control problem. In reinforcement learning, the goal for the learner is to learn the best reaction to any situation that it

may find itself in at some point in time and is acquired through continuous interaction with the environment using a process of trial-and-error. The learner, often called an agent in this context, continually chooses actions based on its current perception of the world, and then receives feedback on these actions which modify its concept of optimal behavior in the following situations or states of the environment. Using Reinforcement Learning, the agent can learn in complex and stochastic environments requiring little or no previous knowledge about possible solutions. A general problem of reinforcement learning is the exponential growth in the state-action space. Therefore, problems such as the traffic domain offer a great challenge from the perspective of reinforcement learning due to the large number of possible states in a traffic network.

A possible solution to estimate the optimal behavior of the controllers is to treat a complete network of roads and controllers as a single problem. In this case, the Reinforcement Learning agent (or some other ML method) attempts to estimate the optimal behavior of all the controllers. Clearly, a failure of such method would result in a complete system failure. Moreover, scaling to large networks might be computationally intensive for such methods as all computation is performed by a single agent. Therefore, the lack of scalability of such methods can be a major weakness.

An alternative approach is to view the problem as a Multiagent System (MAS) where each agent controls a subset of the controllers (e.g. one or more controllers). In this approach, the complete network is controlled by a number of agents and the system is no longer centralized but rather distributed such that each agent acts individually and determines its own behavior. Using a MAS, failure of one or more agents does not necessarily result in a system failure like in a centralized control and since computation is distributed over a number of agents it can scale well. In the distributed control, each agent observes only a part of the environment and acts according to the state of its local environment. Thus, each agent needs to learn only an individual behavior to act optimally in its local environment. Unlike the single agent case which has been studied extensively over the past two decades and has clear methodologies and results, the multiagent reinforcement learning problem is still immature. One of the main challenges is to ensure that the behavior learned by the individual agents will also be globally optimal. Intuitively, some coordination between the agents is needed to ensure the latter. Thus, in addition to the challenge of dealing with an extremely large state-space, the traffic domain also offers a major challenge from MAS perspective. I.e. how to ensure that the behavior learned by individual agents controlling subsets of the controllers will be globally optimal.

Wiering et al.[7] developed a method which employs such local agents in a microscopic model in which the agents share the common goal of minimizing the overall waiting time of the traveling vehicles. Although the method scales well for large networks involving many agents it does not allow any explicit coordination among agents. Bakker et al. [11] extended this approach by allowing some implicit coordination (i.e. by extending the state representation). However, this method still excluded explicit coordination among agents. In both cases, each agent computes its own optimal action given its local state.

This thesis extends the work by Wiering et al. [7] by using cooperative learning and explicit coordination among agents. A relaxing assumption is used with respect to the de-

pendence among agents, namely that the action of an agent only depends on the actions of agents which have a direct influence on its environment (i.e. that are directly connected to it). Under this assumption, the global coordination problem may be decomposed into a set of local coordination problems and can be solved within the framework of coordination graphs. Since the system must perform under time constraints, a method is required which scales to problems involving many agents. For this reason we apply max-plus [12], which estimates the global optimal joint action by sending locally optimized messages among connected agents. Furthermore, it allows the agents to report their current best action at any time (even if the action found so far may be sub-optimal).

This thesis makes several contributions. First, max-plus has been used thus far for relatively small problems [12] and has not been applied to such a large-scale problem as traffic light control. Second, although max-plus has been shown to perform well for acyclic graphs, it has only been proven to converge for tree-structured graphs. This work provides more empirical evidence that max-plus performs well on acyclic graphs. Third, this novel approach allows the explicit coordination of traffic light controllers in a manner that scales well for large networks. Fourth, it provides a new understanding of the properties a traffic network must have to require cooperative learning and coordination among agents and shows that if agents do not take into account the behavior of fellow agents in such networks the overall behavior of the system may become inconsistent and unstable. Finally, this thesis empirically demonstrates that cooperative learning and explicit coordination using max-plus outperforms all previous methods in those networks which require coordination among agents.

The thesis is organized as follows: Chapter 2 discusses different approaches to simulate traffic, describes the simulator used in this thesis, introduces some vocabulary used in the traffic literature and gives an overview of existing ML methods which have been applied to the traffic control problem. Chapter 3 describes the single agent reinforcement learning framework as well as some classical algorithms. Chapter 4 extends the single agent case to multiagent systems and discusses the problems that are introduced by allowing many agents to co-exist in the same environment. Chapter 5 shows how different researchers have applied reinforcement learning to the traffic control problem and provides a detailed overview of the vehicle-based approach [7] used in this thesis. Chapters 6 and 7 present our novel approach. Chapter 6 introduces the framework of Coordination Graphs and discusses a number of methods for solving the coordination problem including max-plus. Chapter 7 introduces the solution proposed by this thesis and provides a detailed description of the method. Chapter 8 contains a set of experiments and their results in which the novel method is compared to the previous work. Different scenarios are tested and analyzed to demonstrate the particular strengths of the novel approach. Chapter 9 concludes the work in this thesis and provides a number of directions for future research.

Chapter 2

Traffic Modeling and Control

There are two common approaches for modeling traffic; macroscopic and microscopic models. This chapter will briefly cover both approaches, describe the simulator used in this thesis, introduce some of the vocabulary used in traffic and finally, describe some of the ML methods which have been applied to traffic.

2.1 Macroscopic Models

Macroscopic models simulate traffic flow by considering cumulative traffic stream characteristics (e.g. speed, flow, and density) and their relationships to each other. The simulation in a macroscopic model takes place on a section-by-section basis rather than by tracking individual vehicles. Macroscopic models employ equations on the conservation of flow and on how traffic disturbances broadcast through the system like shockwaves. They were originally developed to model traffic on distinct transportation sub networks such as freeways and rural highways. They can be used to predict the spatial and sequential extent of congestion caused by traffic demand or incidents in a network; however, they cannot model the interactions of vehicles on alternative design configurations.

2.2 Microscopic Models

In contrast to macroscopic models, microscopic traffic models simulate single vehicle-driver units. Thus, the dynamic variables of the models represent microscopic properties like the position and velocity of a single vehicle. Each vehicle is moved through the network according to its physical characteristics (e.g. length, speed, etc.), fundamental rules of motion, and defined rules of driver behavior. In this way traffic models offer a way of simulating various driver behaviors. A microscopic model consists of an infrastructure that is occupied by a set of vehicles. Each vehicle interacts with its environment according to its own rules. Depending on these rules, different kinds of behavior emerge when groups of vehicles interact. One way of designing and simulating (simple) driving rules of vehicles on an infrastructure is by using Cellular Automata (CA). CA use discrete partially connected cells that can be in a specific state. For example, a road-cell can be occupied (e.g. contain a vehicle) or empty. Local transition rules determine the dynamics of the system and even simple rules can lead to a highly dynamic system. In recent years, micro simulation modeling has gained attention in

its ability to visually represent predicted traffic behavior through high quality animation, enabling laymen, such as politicians and the general public, to fully appreciate the impacts of a proposed scheme. A microscopic CA model is also used in this thesis. The next section will describe the model in detail.

2.3 The GLD Simulator

The Green Light District (GLD)[1], a microscopic CA traffic simulator, was used as a test environment. GLD allows one to create and edit infrastructures and write software for the traffic controllers. The following section will describe the simulator.

2.3.1 Infrastructure

An infrastructure consists of roads and nodes. A road connects two nodes, and can have several lanes in each direction. The length of each road is expressed in units (e.g. cells). A node is either an intersection where traffic lights are operational or an edge node. There are two types of agents that occupy an infrastructure, namely, vehicles and traffic lights (or intersections). All agents act autonomously and get updated every time-step. Vehicles enter the network at edge nodes and each edge node has a certain probability of generating a vehicle at each time step. Each vehicle that is generated is assigned a destination, which is one of the other edge nodes and the distribution of destinations for each edge node can be adjusted. There are several types of vehicles, defined by their speed, length, and number of passengers. This thesis uses only one type of vehicle (e.g. a car) which has an equal length as well as equal number of passengers. The state of each vehicle is updated every time step. It either moves with the distance given by its speed and the state around it (e.g. vehicles in front may limit the maximal distance it can travel given its speed) or remains in the same position since the next reachable position is either occupied or the current configuration of lights set by the intersection does not allow this vehicle's lane to move and therefore the vehicle can not cross the intersection. When a vehicle can cross a intersection, it determines which lane it should go next according to its driving policy. Once a lane has been selected, the vehicle is not allowed to switch to a different lane. For each intersection, there are a number of possible configurations of the lights that are safe and are determined a priori. At each time step, the (intersection) controller decides what configuration is best given the current state. Figure 2.1 shows a single intersection. It has four roads that connect it to other nodes (either intersections or edge nodes). Each road consists of four lanes (two in each direction). Vehicles occupy n cells of a lane depending on their length. Traffic on some lane can only travel in the directions allowed on this lane. The latter determines the possible (safe) configurations of the lights offline. For example, the figure shows a lane where traffic is only allowed to travel straight or right.

2.3.2 Vehicle behavior

The simulator allows one to define different behaviors for the vehicles. The behavior of vehicles includes the driving behavior (e.g. changing speeds) and a policy for selecting the path to destination nodes. In the experiments performed in this thesis, the policy of choosing

[1] available at http://sourceforge.net/projects/stoplicht

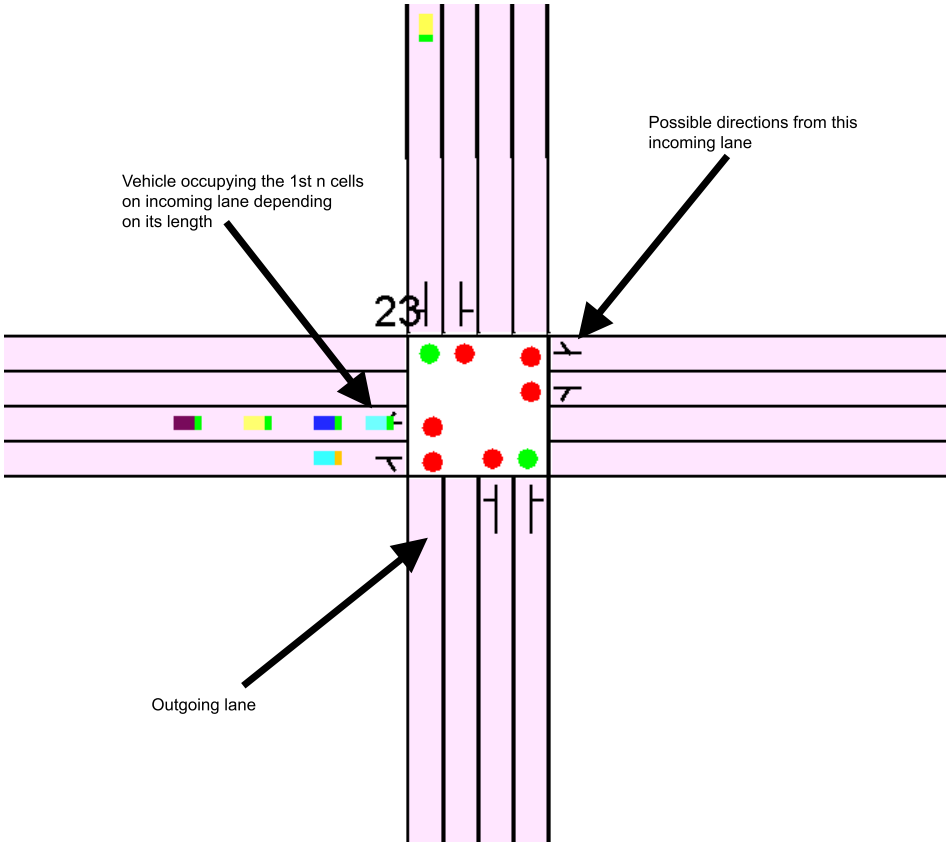

Figure 2.1: A single intersection

a path is simply by always choosing the shortest path to the destination node. We extended the simulation to allow a more dynamic behavior. In previous work, the vehicles traveled at a constant speed and at each phase only a single vehicle crossed an intersection. Here, a Gaussian approach was used for the driving behavior of vehicles which allows vehicles to alter their speed. Using this approach, vehicles are more likely to keep their current speed. The simulation allows vehicles three different speeds of 2, 4 and 6 cells per time step and vehicles enter the network with a speed of 4[2]. Furthermore, we allow a number of vehicles from a single lane to cross an intersection during one time step. The number of vehicles that can actually cross depends on the speed of the vehicles as well as the state at destination lanes. Of course the distance the vehicles travel can in many cases be less (and never more) than their current speed due to the state around them. Overall, our extension results in a more dynamic and realistic simulation.

2.4 Glossary

There are different parameters that can be optimized in the traffic problem including the phase, split, and offset. The following section defines these parameters as well as some other vocabulary used in traffic.

- **Stream**: sequence of vehicles entering an intersection from a certain direction and leaving in a certain direction. If two streams are able to cross an intersection without interfering they are called compatible and otherwise they are antagonistic.
- **Phase**: the time interval where a subset of the lights at an intersection are green, such that a certain set of compatible streams can cross the intersection during that time.
- **Phase scheme**: determines which streams can move simultaneously. I.e. the signal lights are grouped into subsets that will be green at the same time.
- **Cycle**: traditionally, a signal cycle is completed when each phase has been on once. The cycle time denotes the time needed to complete one cycle.
- **Split**: A split determines the distribution of a cycle time into individual phases, thus, determining the length of each phase.
- **Offset**: refers to the time interval between a reference point in time and the start of a new cycle.
- **Saturation**: the saturation of a stream s during a phase p is the ratio of actual (current) traffic volume on the stream and the maximum possible traffic volume given the current length of the phase:

$$saturation(s,p) = \frac{current(s,p)}{max(s,p)} \tag{2.1}$$

  where $current(s,p)$ and $max(s,p)$ refer to the current traffic volume and maximum possible traffic volume respectively. Hence, under-saturated traffic conditions are those where the demand for any lane in the network does not exceed its capacity, and queues

[2]According to the normal distribution, there is a 78% chance the vehicle will keep its current speed when it is 4 and 88% chance if the speed is either 2 or 6.

that build up during red phase can be emptied during green phase while saturated (or over-saturated) traffic conditions denote the opposite.

Traditional methods (e.g. fixed-time controllers) attempt to find optimize the splits, offsets, and cycles and are estimated offline, based on historical data about traffic flow at particular times of day at given intersections. More modern methods (e.g. adaptive controllers) differ from the traditional approaches in that they are usually not concerned with separately optimizing cycle times, splits and offsets. Rather, they attempt to find the optimal lengths of phases given the traffic situation. Furthermore, these methods make use of real-time (sensor) information and adapt the behavior accordingly.

In this paper, the phase scheme is determined offline and an agent can choose a subset of lights that will be set to green at the same time from a finite set of 'safe' subsets (e.g. compatible streams). Furthermore, we do not focus on estimating optimal splits, offsets and cycles like the traditional methods, but rather, at each time step the agents choose their action given the current situation which is either to extend the current phase or begin a new one which can be any of the other subsets.

2.5 Previous Methods

Different Machine Learning approaches have been applied in the traffic domain including fuzzy logic, neural nets, and genetic algorithms. The following section will briefly describe selected research from each of these approaches.

2.5.1 Fuzzy Control

Fuzzy logic can represent fuzzy and qualitative knowledge. The process of reasoning and decision-making can be described as follows: for a lane, if there are many vehicles arriving in, more green time is allocated. Otherwise, less green time is allocated or the phase turns to next one.

In [2], a fuzzy logic controller was designed for an isolated two-phase intersection, which had three inputs and one output. From a fixed number of time steps of green time, decisions were made in a fixed interval to decide whether to extend green time (not exceeding maximum green time) of current phase or change the to the next phase according to traffic volume at all incoming lanes. Simulations had shown that the above method was effective. This is the earliest example applying fuzzy logic to traffic controls.

In [3], Chiu presented fuzzy logic to control multiple intersections in a network of two-way streets with no turning movements. The fuzzy rules were used to adjust parameters such as the cycle time, offsets and splits according to the degree of saturation on each intersection approach. Simulation results showed that fuzzy logic control reduced average delay significantly. This method only takes the flow of the through movement flows into account, however, in practice, turning movement flows should be considered.

Most research is focused on an isolated intersection with fuzzy control method. There are only a few who apply this method to the coordinated control of complex large-scale networks.

In this case, there are many factors and it is difficult to describe the whole system using some qualitative knowledge. In other words, research so far has shown that it is more appropriate to use fuzzy control methods for traffic control of an isolated intersection.

2.5.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) have been widely used in many fields such as signal and information processing and pattern recognition due to their capabilities such as non-linear mapping (or generalization), self-adapting, self-organizing and learning. Although the learning process of neural networks is usually supervised they can also learn in an unsupervised manner by informing the network how well it performed. In unsupervised learning, neural nets can be applied for approximation to strengthen the generalization capabilities and improve precision.

In [4], Spall et al. presented a system-wide traffic adaptive control (S-TRAC) method based on ANN. This neural network controller created the best timing plan according to real-time information of traffic system provided by detectors and some other information such as weather conditions. Simulation at a road with 9 intersections had shown this method was effective.

In [5], Xiupin et al. used a self-learning control method based on neural networks which was applied at an isolated intersection. The control system was composed of two neural networks and a performance evaluation unit. Simulation results showed it could increase traffic capacity of multi-phase intersections and was highly adaptive to changes of traffic flows.

In [6], multi-layer neural networks involving feedback were developed based on Hopfield networks and chaos theory. The inputs were the saturation rate and real-time traffic volume at all lanes. The outputs of the network were the cycle length and green time of every phase. Simulation was implemented at an isolated intersection and did not include turning movements. Results showed that the new method performed better than traditional ones.

The strength of ANNs is mostly in their generalization capabilities. Generally, there are three forms in which ANNs are applied in Traffic Control. The first form is that an ANN is used for modeling, learning and control. In the second form, the generalization capability is used based on other methods. For example, in order to improve the precision of fuzzy control, a neural net can be used to map fuzzy relations or implement fuzzy reasoning. In the third form, a neural net can be used in combination with other methods to improve their generalization capability. Overall, the use of ANNs seems promising whether they are used solely for modeling, learning and control or in combination with other methods.

2.5.3 Genetic Algorithms

Genetic Algorithms (GAs) are randomized optimization heuristics inspired by natural evolution: Starting with a set (population) of randomly generated initial solutions, new solutions are constructed by selecting two relatively good solutions (parents), combining them in some way (crossover) and performing some local modifications (mutation). Resulting new solutions (offspring) are then inserted into the population and may replace some individuals that have

not proven to be successful. Then, the cycle repeats. Since good solutions have a higher probability to survive and generate offspring, the overall quality of solutions is likely to improve over time while the random influence of mutation prevents the system from converging to some local optimum.

Foy et al. [8] were the first to apply GAs to signal timing determination. They optimized phase sequence and green time splits in a network of four intersections for a fixed traffic situation. A simple simulation model was used to evaluate the resulting delays. They concluded that their method can find near-optimal solutions and showed the applicability of GAs to the optimization problem. In the following years, other authors applied GAs to traffic control problems. The use of multi-objective GAs for the signal timing optimization has been investigated by Sun et al. [9]. They focused on the minimization of delay and number of stops as contradicting objectives. They considered only a two-phase isolated intersection, and calculated the performance indices by approximate functions instead of micro-simulations.

The main problem of genetic algorithms is when dealing with large-scale problems, these methods will need relatively much time to converge to their optima. The latter can be said to be disadvantageous for online optimization of large-scale traffic networks. And this might also be the reason why research in evolutionary algorithms for large-scale traffic networks has so far been very limited.

In the remainder of this thesis, we will first introduce the general framework of reinforcement learning (the single agent case followed by multiagent systems), show how reinforcement learning can and has been applied to traffic control and finally present our novel approach.

Chapter 3

Reinforcement Learning

Reinforcement learning (RL) is a general name for techniques in which an agent tries to learn a task by directly interacting with its environment. Throughout our lives, such interactions are a major source of knowledge about our environment and ourselves. Almost every task we are learning, we are aware of how our environment responds to what we do, and we seek to influence what happens through our behavior. Learning from interaction is a fundamental concept underlying many theories of learning and intelligence.

In RL the agent needs to discover what actions are most rewarding in different situations using a process of trial-and-error. It has been studied extensively over the past 20 years and the field of a single-agent RL is nowadays mature, with well-understood theoretical results and many practical techniques. As we shall see in Chapter 4, extending the single agent case to an environment where many agents co-exist is non-trivial as new difficulties arise. The following chapter will formally describe the single agent case which will later be extended to Multiagent Systems (MAS).

3.1 What is an Agent?

The term 'agent' used throughout the RL literature represents the entity which acts in its environment and learns to improve its behavior. A formal definition of an agent is needed in order to have a profound understanding of how such agent will act in its environment.

We can define an agent as any entity that can perceive its environment through sensors and act upon it through actuators [27]. Thus, an agent can be anything from a robot, software program, or biological entity (e.g. human or animal). However, a second restriction in RL is that the agent is assumed to be rational. We can say that an agent is rational if, given what it knows so far, it will always choose an action which optimizes some performance measure. Furthermore, we can say that an agent is autonomous, if it chooses its actions according to what it observes and has learned so far from its own experience rather than relying on a priori knowledge. From this point on, whenever the term agent is used, it is assumed to be a rational autonomous agent.

3.2 The Agent and the Environment

Depending on the nature of the problem at hand, the environment can be either discrete where the number of states is finite or continuous where there would be infinite number of states. Most of the existing methods have been developed for discrete environments. As this paper also deals only with the latter, we will assume a discrete environment.

The agent interacts with its environment at each discrete time step $t = 0, 1, 2, 3, \cdots$ where at each time step, the agent perceives some representation s_t of the current state of the environment where $s_t \in S$ and S is a finite set of all possible states. The agent then selects an appropriate action $a_t \in A$, where A is a finite set of all possible actions that can be performed by the agent. Once the agent performs its action, at the next time step $t + 1$, the agent finds itself in a new state s_{t+1} and receives a (numerical) reward r_{t+1} for this new state according to a *reward function* which is external to the agent. Since the agent is both rational and autonomous then the action it will choose at time t depends on what it has perceived so far (the states it has seen and actions it has taken), but also what it expects to perceive in the future. Formally, if s_t is the perception of the agent at time t, then accordingly in order to choose an optimal action at this time step it will need to make use of the complete history of perceptions and actions up to time t given by:

$$\pi(s_0, a_0, s_1, a_1, s_2, a_2, \cdots, s_t) = a_t \tag{3.1}$$

Thus, the function π, called the *policy* of the agent, would map the complete history of perceptions (and actions) up to time t to an action which is considered optimal at time t. Obviously, a mapping of the complete history would be infeasible in many cases. First of all, the history can include large (or even infinite) number of perceptions and actions that would need to be stored and therefore require large and even infinite storage resources. Second of all, the computational complexity, which would increase over time as more pairs are stored, would be too large to deal with, certainly for real-time applications. Finally, as mentioned earlier, optimal decision making should also take the future into account. In the remaining sections, the theoretical framework for dealing with these problems will be described.

3.3 Markov Decision Processes

The previous section introduced the problem of history and expected future perceptions that are to be used by the agent to compute its optimal policy. Ideally, we would like some compact representation that would summarize over all past perceptions and include all the relevant information. In many cases the current state already provides all (or most) of the relevant information and therefore all past perceptions are not needed. If the state includes all the relevant information from the past then it is said to have the *Markov Property*. Formally, if the current state depends on all the history of events that have led to it then:

$$P(s_{t+1}|s_t, a_t, , s_{t-1}, a_{t-1}, r_{t-1}, \cdots, s_0, a_0) \tag{3.2}$$

In other words, the probability of reaching some state s_{t+1} at time t depends on the complete history of dynamics until time t. However, if the state is said to have the Markov Property then:

$$P(s_{t+1}|s_t, a_t, , s_{t-1}, a_{t-1}, r_{t-1}, \cdots, s_0, a_0) = P(s_{t+1}|s_t, a_t) \tag{3.3}$$

In this case, the state s_t includes all relevant information until time t and therefore the probability of reaching some state s_{t+1} depends only on the dynamics at time t. This allows the agent to predict the next state and also the expected reward it will receive in that state given only the current state and action. If the world (the terms 'world' and 'environment' are used interchangeably in this paper) is Markovian then the agent can make predictions on next states and expected rewards given only the current state and action by iterating over Equation 3.3. Any RL problem in which the state and action spaces are finite and the Markov Property holds is said to be a (finite) Markov Decision Process (MDP).

Even if the world is non-markovian and the current state does not satisfy the Markov property, it can still be used as an approximation. This simplified representation deals with the problems discussed in the previous section by allowing the agent to have a compact representation of the history of perceptions and even more importantly allow it to make predictions about the future.

So far, two elements that characterize RL have been introduced, namely, the policy of the agent and a reward function. Equation 3.3 introduces another important element called a *Transition Model*. The model specifies how the world would change if some action is taken in a given state in terms of probability. If the model is known a priori then the agent can find its optimal action by considering possible future states before these are actually experienced. Otherwise, the agent will need to learn the model from experience. Finally, there are also cases in which the agent might not have access to a transition model at all. Section 3.5 will present methods for both cases.

3.4 Value Functions

The RL agent is not interested in maximizing only immediate rewards but rather in choosing actions that are optimal in the long run. While the reward function indicates the desirability of a state in the immediate sense, the value (or utility) function specifies its desirability over the long run. Hence, the desirability can be defined in terms of expected future rewards. Formally, if an agent follows a policy π and is at state s then the expected accumulative reward by following π thereafter for MDPs is given by:

$$U^{\pi}(s) = E_{\pi}[\sum_{t=0}^{\infty} \gamma^k r_t | s_0 = s] \tag{3.4}$$

where $E_{\pi}[\cdot]$ is the Expectation operator which averages over rewards and stochastic transitions and $\gamma \in [0, 1]$, a discount rate which ensures that the sum is finite. As we are assuming an MDP, $U^{\pi}(s)$ can also be expressed in terms of probabilities of state transitions. Equation 3.4 can then be rewritten in a recursive fashion as:

$$U^{\pi}(s) = r + \gamma \sum_{s'} P(s'|s, a) U^{\pi}(s') \tag{3.5}$$

where r is the reward at s (r_0 in equation 3.4). The last equation is called the Bellman equation and is maybe the most important equation in modern RL [1].

[1] For a complete proof of the above see [1]

Given the above, the optimal utility for an agent in state s is given by:

$$U^*(s) = \max_{\pi} U^{\pi}(s) = r + \gamma \max_{a} \sum_{s'} P(s'|s,a)U^*(s') \tag{3.6}$$

Similarly, we can define an optimal action-value function $Q^*(s,a)$ which indicates the desirability of taking an action a at state s, as the discounted future reward the agent receives for taking action a in state s.

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{3.7}$$

$$Q^{\pi}(s,a) = E_{\pi}[\sum_{t=0}^{\infty} \gamma^k r_t | s_0 = s, a_0 = 0] \tag{3.8}$$

and,

$$Q^*(s,a) = r + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q^*(s',a') \tag{3.9}$$

A policy π which satisfies Equations 3.6 and 3.9 is said to be an optimal policy since the agent will obtain the maximum reward over the long run by following it. There can be a number of optimal policies for a given problem but all policies will share a unique $Q^*(s,a)$ and $U^*(s)$.

3.5 Learning an Optimal Policy

Methods for learning an optimal policy can roughly be classified into two categories, namely, *model-based* methods or Dynamic Programming (DP) methods and *model-free* methods (e.g. Monte Carlo methods, Temporal Difference Learning). The former assume a perfect knowledge of the environment and the agent has access to a transition model. An assumption which is often violated in real-world problems. Furthermore, this family of methods is usually relatively computationally expensive. In model-free methods, the agent has no access to a transition model. The values are estimated only by the experienced perceptions and a reward function. Thus, they attempt to compute an optimal policy without assuming perfect knowledge of the environment and/or with less computation. Two well-known methods, one from each category will be described next.

3.5.1 Value Iteration

Value Iteration is a simple DP method for computing the optimal policy when the transition model is available. We simply begin with random values $U(s)$ and iteratively repeat Equation 3.6 which results in the following assignment operation:

$$U(s) := r(s) + \gamma \max_{a} \sum_{s'} P(s'|s,a)U^*(s') \tag{3.10}$$

In other words, the Bellman Equation is turned into an update rule. The agent can then select the optimal action at each state according to a greedy policy:

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s,a)U^*(s') \tag{3.11}$$

In case the transition model is known a priori, the agent can compute its optimal policy before actually performing the task. Otherwise, the agent must experience the world in order to learn the model.

3.5.2 Q-learning

Q-learning [29] is a model-free method in which the agent has no access to the transition model. In Q-learning, the agent estimates $Q^*(s,a)$ by continuously interacting with the environment in a form of trial-and-error. Like in Value Iteration, the agent begins with random value estimates and after each action receives a tuple $\langle s, a, r, s' \rangle$ where s is the current state, a, the action taken at state s, r, the current reward and s', the resulting state after executing a. For each tuple, the agent can estimate the corresponding action value as:

$$Q(s,a) := (1-\lambda)Q(s,a) + \lambda[r + \gamma \max_{a'} Q(s',a')] \tag{3.12}$$

where $\lambda \in [0,1]$ is the agent's learning rate. If all state-action pairs are visited infinitely often and the learning rate decreases over time, Q-learning has been shown to converge to the optimal $Q^*(s,a)$ with probability 1. The optimal policy is then given by:

$$\pi^*(s) = \max_a Q^*(s,a) \tag{3.13}$$

3.6 Other Issues in RL

Curse Of Dimensionality Like most branches of AI, RL also suffers from this problem. As the agent receives more information that define the state it is in, the size of its (finite) set of possible states increases. One can already note that the growth in the internal representation of the agent is exponential in the number of states and actions. Clearly, there is a trade-off in the state-action representation; on one hand, providing the agent with insufficient information and capabilities could result in decrease and even failure of the agent of performing its task. On the other hand, providing the agent with too much information and capabilities can result in a (too) large set of possible states which is computationally undesired.

> *"In reinforcement learning, as in other kinds of learning, such representational choices are at present more art than science". (R. Sutton and A. Barto.,Reinforcement Learning: an introduction,1998.)*

Exploration vs. Exploitation Another trade-off which arises in RL is that between exploration and exploitation especially in model-free methods as they acquire no knowledge of environment (e.g. transition model). On one hand the agent should prefer actions that it has tried in the past and found to be effective. On the other hand, in order to discover such actions and also to explore the environment in order to reach new states that might be preferable, it has to try actions that it has yet selected. The agent has to exploit what it already knows, but it also has to explore in order to make better action selections in the

future and reach new states it has not encountered thus far. The agent must try a variety of actions and progressively favor those that appear to be best. On a stochastic task, where more than one state can be reached by performing a certain action at a given state, each action must be tried many times to gain a reliable estimate of its expected reward. The exploration-exploitation dilemma has been intensively studied and two well-known exploration approaches are $\epsilon - greedy$ and Boltzmann exploration.

Stationary and Non-stationary Worlds In a stationary environment the dynamics is assumed to always be fixed. This means that $p(s'|s,a)$ is fixed in the sense that s' is either unique for a given action a and state s (e.g. the environment is deterministic) or there is some fixed probability distribution over the possible next states which does not change over time. However, this assumption is often violated in real-world problems where $p(s'|s,a)$ for a given action a and state s does change over time due to change in the dynamics of the environment. In this case, the world can be said to be non-markovian and assuming an MDP would not be valid. In other words, if there is no regularity in the environment, RL approaches might need to relearn everything from scratch since the policy that was computed is no longer valid when the dynamics change. The former will result in a performance drop during this readjustment phase, and also forces the algorithm to relearn policies even for environment dynamics which have been previously experienced.

Observability Another issue is the observability of the environment. If the agent can perfectly sense its environment then it can be said to be fully observable; the agent always knows the current state. However, in many real-world problems, there is usually some noise present in the perception of the agent. For example, if a robot makes use of sensors to detect its current location, the sensors usually provide noisy input which means there is some uncertainty regarding the exact location and therefore the agent can be in one of different possible locations. In this case, the environment can be said to partially observable and therefore stochastic. The latter is an active field of research which is outside the scope of this thesis. Here we assume an environment which is fully observable.

Continuous vs. Episodic Learning A distinction can also be made between two classes of learning tasks, namely Episodic and Continuous learning. In the former, the agent eventually reaches a terminal state and the task is completed. For example, in backgammon a game ends when one of two players wins it and thus the ultimate goal of the agent is to reach the 'win' state. In the latter, the task is continuous and no 'terminal state' exists. The agent needs to continuously choose optimal actions which maximize some performance measure but no 'goal state' is ever reached. The traffic domain can be said to be continuous as no terminal state is ever reached; the agent (traffic controller) needs to continuously act optimally to reduce the waiting time of cars traveling in the network.

Chapter 4

Multiagent Systems

A Multiagent System (MAS) consists of a number of agents that co-exist in the same environment and can potentially interact and/or communicate with one another [14]. In MAS, there is no central control for the agents, the system is distributed in the sense that each agent acts individually. For some systems, such as robotic teams or controller networks (e.g. Traffic Light Controllers), it is a natural way of the problem representation. A MAS even has some advantages over a centralized system [15]. First of all, it may speed up computation due to parallel computing. Second of all, it can be said to be more robust, as the failure of one or more agents does not result in a system failure like in a centralized system. Finally, the modularity of a MAS enables one to add agents if required which makes the system highly scalable. The remainder of this chapter will briefly describe some of the characteristics of a MAS which are central to this thesis.

4.1 Cooperation and Coordination

Agents may be self-interested or (fully) cooperative. In the former, agents may pursue their individual goals and have no need to cooperate in order to achieve a common goal. Here, we focus on fully cooperative agents who need to achieve the common goal of minimizing the waiting time of traveling vehicles in a network. In such system, the payoff or reward function is global (e.g. shared by all agents). Note that the optimal individual action may not be the optimal one for the group. Moreover, choosing an action based on individual payoff might be considered as irrational behavior of the agent since it is not optimal in the global sense. Obviously, the agents need to coordinate their actions in order to find the optimal joint action.

4.2 Homogeneous and Heterogeneous Agents

Agents can be heterogeneous such that they have different types of perceptions of the environment and can perform different types of actions. Otherwise, as also assumed in this paper, the system is homogeneous. In this case, the agents (e.g. the traffic controllers) have the same type of perception and have identical capabilities.

4.3 Knowledge

Knowledge is the information an agent has about the world and the problem to be solved. An agent may receive this information from different sources and uses it to select its actions. This includes what the agent observes about the current world state but may also include information it receives from the other agents about the current state. Sharing knowledge can also reduce the complexity of the problem. For example, agents can share the same value function and thereby they can make use of the experience of other agents. This approach reduces the size of policy search space state space and may also speed up the learning process. The following chapters will discuss in more detail how the agents may share knowledge in the context of traffic control.

4.4 Observability

Except for partial observability discussed in the single agent case, in MAS, another issue is how the agents perceive the world. In some problems, the agents may have or even require the perception of the complete world state. Other problems may be controlled by a MAS in which each agent has limited view of the entire state and the complete state is uniquely identified by the combined observations of all agents. In the context of traffic control the latter is a somewhat natural way of modeling. Here, each agent controls a subset of intersections (e.g. one or more intersections) and the perception of each agent can then be limited to the local environment an agent controls. This of course reduces the overall complexity as the entire state space is decomposed into local states.

4.5 Dynamics

One of the main challenges in MAS is how the group of agents can simultaneously learn an individual behavior that will be optimal for the entire group. A somewhat straightforward solution would be for the agents to treat the other agents as part of the environment which means the agents do not take into account the behavior of other agents in the group. With this approach, the single agent methodologies discussed in the previous chapter can be directly applied. However, when other agents are part of the environment the new state and reward may also depend on these agents. As a consequence the environment may become dynamic from the perspective of a single agent. The other agents are part of its environment and each of these agents changes its behavior as it learns and may affect the agent's state. Therefore, an agent in a MAS should take into account the behavior of other agents as it learns. The latter results in another problem, namely, if all agents take into account the actions of their fellow agents then the growth of state-action space is now also exponential in the number of agents. I.e. iterating over all possible joint actions might be acceptable for (very) small state-action spaces but would not work for problem with a relatively large number of states and agents. A more relaxing approach is to assume that dependence exists only between subsets of agents. In other words, the action of an agent does not depend on the actions of all other agents but rather only a subset of agents who may influence its state. In many real-world problems, the spatial ordering of the agents might be used such that only agents which are relatively close have a direct influence on one another. In some problems, the dependencies can change over time (e.g. cooperative robots moving in the environment) and

the dependencies need to be re-evaluated over time [13]. In other problems the dependencies are known a priori and do not change (e.g. network routing or traffic light control) and the dependencies can be computed offline and plugged in to the agents. For example, in the traffic domain, if each agent controls one intersection then it can be said to assumed to influence only neighboring intersections (i.e. intersections it may directly send or receive vehicles from).

As described in this chapter, extending a single-agent system to multiple agents introduces many complexities. The main problems arise as a result of the exponential growth in both the representation of the action and state space, and the distributed nature of the problem as observations, control, and knowledge are all distributed. In the remainder of this thesis we focus on solution techniques to coordinate and learn the behavior of the agents in a multiagent traffic control system. The focus is on distributed, scalable methods to coordinate and learn the behavior of a large group of agents (i.e. large traffic networks involving many traffic controlling agents). The main idea is to exploit local dependencies in the problem, which makes it possible to solve the global problem as a sum of local problems. The agents are only allowed to operate locally such that they observe local states, receive local rewards based on a decomposed global reward function, and are only able to communicate with a limited number of agents. The next chapter will discuss different approaches of applying RL to traffic, present the approach used in this thesis, followed by our novel approach which extends the latter to allow coordination and cooperative learning of the agents.

Chapter 5

Reinforcement Learning Applied to Traffic Control

Reinforcement learning (RL) for traffic light control has first been studied by Thorpe [24]. The approach was based on a SARSA (e.g. model-free) reinforcement learning algorithm. It used a discrete state-action space to represent the states and actions. The state is characterized by the queue length, positions of vehicles in the queue, and the elapsed time current light lasting. One controller was trained on a single intersection, and the state of that controller was replicated to all intersections of a network after training is completed. This means that every controller optimizes its intersection only locally, and furthermore makes the assumption that every intersection controller should pursue the same policy. The system outperformed both fixed and rule-based controllers in a realistic simulation with varying speed.

Ma et al. [25] applied Q-learning to dynamically control the traffic signals at an isolated intersection. The states include the index of green phase, the lasting time of green light, traffic volume during green phase, mean number of queuing vehicles during red phase, and the prediction of the trend of traffic flow; the action is to change the green phase to red phase, or extend green phase until next decision point. The reward is denoted by traffic volume during green phase between successive decision points divided by the waiting time increased during red phase.

Abdulhai et al. [26] presented a basic framework of applying Q-learning to traffic signal control system. In the case of the isolated intersection, the states include the queue lengths on the four approaches and the elapsed phase time. Actions include extending the current phase and changing to the next one. The reward is defined to be the total delay between successive decision points by vehicles in the queues of the four approaches. And the delay is denoted by a power function of queue length. For a road with multiple intersections, some other states may be added, for example the splits between two intersections. The reward is the weighted summation of rewards of all isolated intersections, where the reward of main road should be weighted more heavily.

All of the methods described above suffered from the same problem, namely, when attempting to scale to larger networks the number of states became extremely large. Even when

some state information was sacrificed (e.g. using only the queue length etc.). Hence, all of the methods were only applied to relatively small networks (e.g. a single intersection) or trained a single intersection and used the same policy for a number of intersections.

A different approach is to exploit the presence of the other type of dynamic entities in traffic, namely, the vehicles. In the vehicle-based representation [7], the global state is approximated by decomposing it into local states based on the vehicles actually traveling in the network. This representation has a number of advantages. First, the number of states grows linearly in the number of lanes and cells and therefore will scale well for large networks. Second, except for the vehicle's location, specific vehicle information, such as the vehicle's speed and destination may be included in the state representation[1]. Third, as the size of the state-space and the convergence of the value function are directly correlated, the value function should converge relatively fast compared to methods where the increase in the size of the state-space is exponential.

At a given moment, the global state can be estimated by considering all of the individual states that are occupied at that time. The update of the state-action space resembles Prioritized Sweeping(PS) [10] where multiple values are updated during one time step. The states whose values should be updated are stored in a priority queue which stores the states priorities that are initially set to the same value. In stead of storing transition probabilities, PS stores the number of times the transition (s, a, s') occured and the total number the state-action pair (s, a) has been reached. In this manner the transition model can be estimated using the maximum likelihood probability given by $\frac{|(s,a,s')|}{|(s,a)|}$. The vehicle-based approach is similar to PS since at each time step we only update those parts of the state-space which are actually visited by vehicles (e.g. a priority queue), and update the state transitions probabilities using maximum likelihood. This approach is described next.

5.1 Modeling

Using the vehicle-based representation the problem can be modeled as follows:

- s: is the global state and is fully observable
- $j \in J$: represents an intersection controller
- A: represents a set of intersection controller decisions (actions) where $A_j \subseteq A$ is the subset of possible actions of intersection j. An action consists of setting the light to green on a subset of the traffic-lights at the intersection. The actions are fixed at intersection level in the sense that the intersection controller has a fixed number of allowed configurations of green lights which is determined a priori according to the phase scheme.
- L: the set of lanes, where $L_i \subseteq L$ is the subset of incoming lanes to intersection i , $l_j \in L_i$ is a single lane.
- The global state s decomposes as: $\langle s \rangle = \langle s_i \rangle = \langle s_{p_{l_i}} \rangle$ where p_{l_i} is a position on lane i.
- $p(s'|s, a)$: represents the global transition model

[1] Although such representation would require the vehicles to be able to communicate

5.2 Learning

Using the above, the action-value function can be decomposed as:

$$Q(s,a) = \sum_i Q_i(s_i, a_i) \tag{5.1}$$

and,

$$Q_i(s_i, a_i) = \sum_{l_i} \sum_{p_{l_i}} Q_{p_{l_i}}(s_{p_{l_i}}, a_i) \tag{5.2}$$

The vehicle-based update rule is then given by:

$$Q_{p_{l_i}}(s_{p_{l_i}}, a_i) := \sum_{s_{p'_{l_i}}} P(s_{p_{l_i}}, a_i, s'_{p_{l_i}})[r(s_{p_{l_i}}, a_i, s'_{p_{l_i}}) + \gamma V(s'_{p_{l_i}})] \tag{5.3}$$

where $s'_{p_{l_i}}$ are all possible states that can be reached from $s_{p_{l_i}}$ given the current traffic situation and the vehicle's properties (e.g. its speed and length). $V(s_{p_{l_i}})$ is a function which estimates the expected waiting time at p_{l_i} by averaging over all possible actions. It therefore represents the expected waiting at a given position independent of the action of the intersection controller. The function is estimated as follows:

$$V(s_{p_{l_i}}) := \sum_{a_i} P(a_i | s_{p_{l_i}}) Q(s_{p_{l_i}}, a_i) \tag{5.4}$$

The transition model can be estimated using a maximum likelihood model by counting state transitions and corresponding actions and the update is given by:

$$P(s'_{p_{l_i}} | s_{p_{l_i}}, a_i) := \frac{C(s_{p_{l_i}}, a_i, s'_{p_{l_{ij}}})}{C(s_{p_{l_i}}, a_i)} \tag{5.5}$$

where $C(*)$ is a function which counts the number of times the event takes place. In order to estimate $V(s_{p_{l_i}})$, we also need to estimate the probability that a certain action will be taken given the state. The latter is estimated using the following update:

$$P(a_i | s_{p_{l_i}}) := \frac{C(s_{p_{l_i}}, a_i)}{C(s_{p_{l_i}})} \tag{5.6}$$

The global reward function decomposes as:

$$r(s, s') = \sum_{L_j \in L} \sum_{l_i \in L_j} \sum_{p_{l_i} \in l_i} r(s_{p_{l_i}}, s'_{p_{l_i}}) \tag{5.7}$$

and

$$r(s_{p_{l_i}}, s'_{p_{l_i}}) = \begin{cases} 0 & s_{p_{l_i}} \neq s'_{p_{l_i}} \\ -1 & \text{otherwise} \end{cases} \tag{5.8}$$

5.3 Optimal Decision Making

Using the decomposition of the global state, the optimal joint action is decomposed as follows:

$$\pi^*(s) = \arg\max_a Q(s,a) = \arg\max_{a_i} Q_i(s_i, a_i) \tag{5.9}$$

and,

$$\arg\max_{a_i} Q_i(s_i, a_i) = \arg\max_{a_i} \sum_{l_i} \sum_{p_{l_i}} O_{p_{l_i}} Q_{p_{l_i}}(s_{p_{l_i}}, a_i) \tag{5.10}$$

where $O_{p_{l_i}}$ is a binary operator which indicates the occupancy state at p_{l_i} given by:

$$O_{p_{l_i}} = \begin{cases} 0 & p_{l_i} \text{ not occupied} \\ 1 & \text{otherwise} \end{cases} \tag{5.11}$$

For exploration, Wiering used the ϵ-greedy method, where at each time step the controllers will choose a random action with some small probability. The update rule of the V and Q values is recursive and is computed iteratively using Dynamic Programming where the number of iterations is fixed. Since allowing many iterations can be very costly, Wiering used a small number of iterations (e.g. 1).

The optimal joint action is decomposed into a vector of the individually optimal actions of the agents. The method does provide a global overview as the values are propagated over the entire network by the traveling vehicles. Thus, agents share information regarding the expected waiting time of vehicles and use values at neighboring intersections to update the values on their own incoming lanes. Furthermore, the method does not require the vehicles to communicate as vehicles movements can be estimated externally (e.g. by using some sensors or cameras). One drawback of the method is that none of the agents takes into account the actions of other agents. As mentioned, the actions of agents can influence the state of other agents and this may lead to unstable behavior of the entire system if agents do not take the latter into account.

5.4 The Congestion-Based Extension

In [11], Bakker et al. extended Wiering's approach by including congestion information in the state representation. The value function $Q_{p_{l_i}}(s_{p_{l_i}}, a_i)$ was extended to $Q_{p_{l_i}}(s_{p_{l_i}}, c_{dest}, a_i)$ where $c_{dest} \in 0, 1$ is a single bit indicating the congestion level at the next lane for the vehicle currently at p_{l_i}. If the congestion at the next lane exceeds some threshold then $c_{dest} = 1$ and otherwise it is set to 0. The argument for such an extension is that agents effectively learn different state transition probabilities and value functions when the outbound lanes of the intersection are congested and when they are not. The state transition probabilities and expected waiting times are likely to differ for the two cases. In other words, it allows the agents to effectively differentiate between the two cases. An example of how this may lead to improved traffic flow is that an agent may learn that if traffic at an outbound road is congested and at some incoming road many vehicles are headed towards that road, the intersection controller is not likely to set the light to green on that inbound road. Therefore, it will give precedence for vehicles heading in some other direction where there is no congestion which also allows the neighboring intersection more time to resolve the congestion. This

approach showed improvement with respect to Wiering's approach on a saturated network.

Including more information can have a positive influence on the optimality of the actions. However, the cost of creating larger state spaces is the increase in complexity and the rate of convergence of the value function. More complex state spaces usually require more learning in order to converge. Moreover, this approach requires the vehicles to communicate as the controllers need to know the destination lanes of vehicles and this information is not directly accessible to the controllers.

The following chapters describe our novel approach using Coordination Graphs and max-plus in which agents also take into account the actions of their fellow agents.

Chapter 6

Coordination Graphs

Chapter 4 discussed the problem of dynamics in MASs which suggest that agents should take the behavior of other agents into account. The latter introduced a new problem, namely, the exponential growth in the state-action space. A relaxing approach was also named which is to assume that dependence exists only between subsets of agents. In other words, the action of an agent does not depend on the actions of all other agents but rather only a subset of agents who may influence its state. Under this assumption, the global state-action space can be decomposed into local functions involving only subsets of agents. The optimal joint action can then be estimated by finding the joint action which maximizes the global payoff now defined in local terms. The framework of Coordination Graphs allows us to do exactly that, namely, decompose the state-action space into such local terms. The following chapter will describe the framework and describe a number of methods that can be used to estimate the optimal joint action when decomposing the global problem into local terms including max-plus.

A Coordination Graph (CG) can be described as an undirected graph $G = (V, E)$ in which each node $i \in V$ represents an agent, and an edge $e(i, j) \in E$ between agents i, j means that a dependency exists between them and therefore they need to coordinate their actions. CGs, or undirected (unnormalized) graphs, have been extensively studied in the context of probabilistic graphs (e.g. Bayesian Networks) where the global function, consisting of many variables, is decomposed into local functions where each depends on only on a subset of the variables. The maximum a posteriori (MAP) configuration is then estimated by sending locally optimized messages between connected nodes in the graph. Although estimating the MAP configuration in a probabilistic graph model and finding the optimal joint action of a group of cooperative agents in a CG are two different tasks, in both cases, the function that is being optimized is decomposed in local terms. Therefore, message-passing algorithms that have been developed for estimating the MAP configuration in probabilistic networks are directly applicable for estimating the optimal joint action of a group of agents in a CG [12].

The global coordination problem is decomposed into a set of coordination problems, each involving a subset of the agents. The global action-value function $Q(s, a)$ is then decomposed into a sum of local functions given by:

$$Q(s,a) = \sum_{i \in V} Q_i(s, a_i) \tag{6.1}$$

where a_i is the action of agent i and of all agents on which it depends. Since any arbitrary graph can be converted to one with only pair-wise dependencies [23], the global action-value function can be decomposed into pair wise payoff functions given by:

$$Q(s,a) = \sum_{i,j \in E} Q_{ij}(s, a_i, a_j) \tag{6.2}$$

where a_i and a_j are the corresponding actions of agents i, j respectively. Figure 6.1 shows a CG of 8 agents where the dependence among agents is decomposed into pair-wise functions. The remainder of this chapter will describe methods which can be used to estimate the optimal joint action in a CG including max-plus.

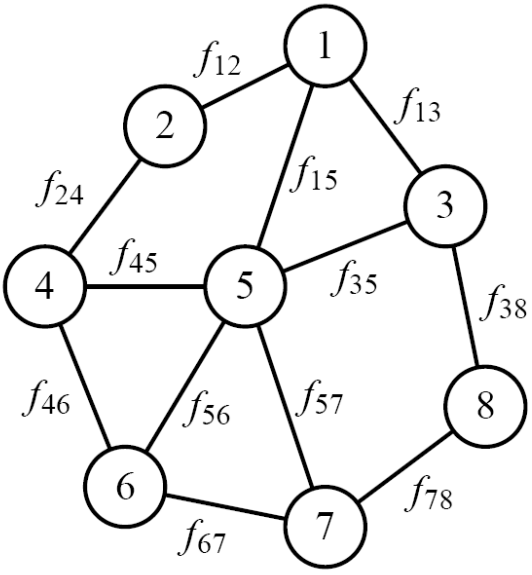

Figure 6.1: A coordination graph involving 8 agents with only pair-wise dependencies

6.1 Variable Elimination

The Variable Elimination (VE) [19] computes the optimal joint action using the decomposition given by Equation 6.1 and is essentially equivalent to the VE algorithm in a Bayesian Network [20]. It performs two steps, namely a forward and a backward step. In the forward step, agents are eliminated according to the dependencies in the coordination graph until there is only one agent left. Each time an agent is eliminated, a new function is created based on the local action-value functions in which this agent is involved which is independent of the eliminated agent. This conditional function computes the maximal value the agent can achieve for each combination of the other agents involved. In this way, the agents are iteratively eliminated until one agent remains. This agent simply returns the action that maximizes the last conditional function (which is independent of all other agents). Then a

backward step is taken, in the reverse order of elimination in which each agent simply selects its maximizing action according to the actions of the agents that already reported their action which are now assumed to be fixed.

Although this algorithm always finds the optimal joint action and also does not depend on the elimination order, it is computationally expensive. The execution time is exponential in the induced width of the graph and therefore does not scale well [14]. Furthermore, the actions are known only when the backward step is complete. The latter can be a problem for systems that must perform under (time) constraints and it is desirable that the agents are able to report their (sub)optimal action at any time. The next section will describe the max-plus algorithm which attempts to overcome these two problems.

6.2 The Max-plus Algorithm

The max-plus algorithm approximates the optimal joint action by iteratively sending locally optimized messages between connected nodes (e.g. agents) in the graph. It is based on belief propagation or the max-product algorithm [21] which estimates the MAP configuration in Belief Networks. Such models reason about uncertain knowledge using probability theory. A problem is described using several random variables which take on values in a specific domain. The combination of all random variables specifies a joint probability distribution. The number of different combinations scales exponentially with the number variables, and therefore the model also specifies stochastic relations between the variables which indicate the dependencies of the system for the given problem. Each random variable is associated with a conditional probability distribution which specifies the probability for its value based on the value of a subset of the other variables. Variables which do not directly depend on each other are said to be conditionally independent. The joint distribution then factors into a product of conditional distributions. Probabilistic graphical models are used to answer inference questions about the random variables. For example, computing the posterior distribution of a random variable given evidence about the value of some other variables, or computing the maximum a posteriori (MAP) configuration of the random variables, that is, the assignment of the variables with the highest probability. The MAP for each node is estimated using only the incoming messages from other nodes. Although it has been shown that the message updates converge to a fixed point after a finite number of iterations for tree structured graphs, it has not been proven to converge for graphs with cycles. Nevertheless, the algorithm has been successfully applied to such graphs [12][22][23].

Using the decomposition given by Equation 6.2, we can define a message from agent i to a connected agent j as:

$$\mu_{ij}(a_j) = \max_{a_i}\{f_{ij}(a_i, a_j) + \sum_{k \in \Gamma(i)\backslash j} \mu_{ki}(a_i)\} + c_{ij} \tag{6.3}$$

where $\Gamma(i)\backslash j$ is the subset of all agents connected to i except for j and c_{ij} is either zero or can be used to normalize the messages. The message approximates the maximum value agent i can achieve for each action of agent j based on the function defined between them and incoming messages to agent i from other connected agents (except for j). At each time step, we can define the value for an action a_i of agent i as:

$$g(a_i) = \sum_{j \in \Gamma(i)} \mu_{ji}(a_i) \tag{6.4}$$

and the optimal action as:

$$a_i^* = \arg\max_{a_i} g(a_i) \tag{6.5}$$

Accordingly, the optimal joint action can be defined as:

$$a^* = \arg\max_{a} \sum_i \sum_{j \in \Gamma(i)} f_{ij}(a_i, a_j) \tag{6.6}$$

Unlike VE, where an agent needs to sum over all action combinations of its neighbors in order to compute its optimal action (or best-response), max-plus allows the agent to sum over the incoming messages from its neighbors which are defined over its own actions. Moreover, the agent is not required to wait until the process is completed. Rather, it can compute its optimal action at any time given the messages it has received thus far. This allows the agents to report their actions under (time) constraints even if the global joint action is still suboptimal. In graphs with cycles, incoming messages will eventually also include the agent's own outgoing messages. As a result, messages can grow extremely large and need to be normalized. This can be done by subtracting the average of all values in $\mu_{ik}(a_k)$ using $C_{ij} = \frac{1}{|A_k|} \sum_k \mu_{ik}(a_k)$ in Equation 6.3. Figure 6.2 shows the different messages sent among neighboring agents in a graph with four agents.

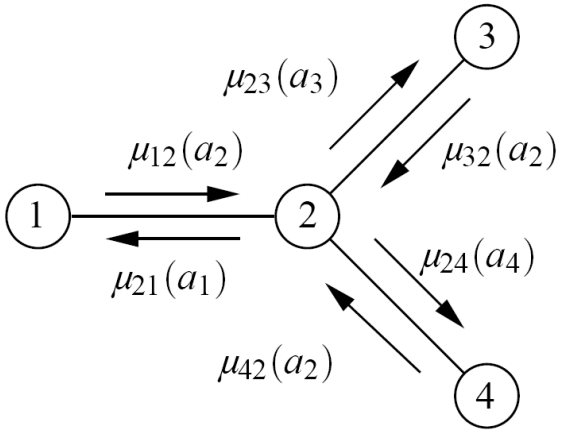

Figure 6.2: Representation of the different messages μ_{ij} sent between neighboring agents

At each iteration, an agent i sends a message μ_{ij} to each neighbor $j \in \Gamma(i)$ given the function f_{ij} and current incoming messages it has received and also computes its current optimal action given all the messages it has received so far. The process continues until some signal is received which means the agents must report their individual actions. As mentioned, there is no guarantee that the algorithm converges in graphs with cycles. The global payoff corresponding to the joint action might even decrease when the values of the messages oscillate.

As a result no assurances can be given about the quality of the corresponding joint action. This necessitates the development of an anytime algorithm [12] in which the joint action is only updated when the corresponding global payoff (as defined in Equation 6.6) improves. Therefore, the max-plus algorithm is extended by, occasionally, computing the global payoff and updating the joint action only when it improves upon the best value found so far. This ensures that every newly stored joint action produces a strictly higher payoff than the previous one. When the joint action has to be reported, the last updated actions are returned. The pseudo-code for the centralized max-plus algorithm is shown in Alg. 1

Algorithm 1 Centralized max-plus algorithm for $CG = (V, E)$

$\forall (i,j) \in E$: $\mu_{ij} = \mu_{ji} = 0$, $\forall i \in V$: $g_i = 0$, and $m = -\infty$
while $fixedpoint = false$ and deadline to send action has not yet arrived **do**
  $fixedpoint = true$
  for every agent i **do**
    for all $j \in \Gamma(i)$ **do**
      send j message $\mu_{ij}(a_j) = \max_{a_i}\{f_{ij}(a_i, a_j) + \sum_{k \in \Gamma(i)\backslash j} \mu_{ki}(a_i)\}$
      if $\mu_{ij}(a_j)$ differs from previous message by a small threshold **then**
        $fixedpoint = false$
      end if
      determine $g(a_i) = \sum_{j \in \Gamma(i)} \mu_{ji}(a_i)$ and $a_i^{'} = \arg\max_{a_i} g_i(a_i)$
    end for
  end for
  if use anytime extension **then**
    if $\mu(a_i^{'}) > m$ **then**
      $(a_i^*) = (a_i^{'})$ and $m = u((a_i^{'}))$
    end if
  else
    $(a_i^*) = (a_i')$
  end if
end while
return (a_i^*)

Chapter 7

Applying Coordination Graphs to Traffic Control

Chapter 5 discussed the vehicle-based approach as proposed by Wiering et al. where the agents do not take the actions of fellow agents into account. As mentioned, when the actions of agents are likely to influence the state of other agents, the overall behavior may become unstable and inconsistent if these are not taken into account. On the other hand, taking the actions into account leads to an exponential growth in the state-action space which makes it hard to estimate the optimal joint action within time constraints. Our approach attempts to overcome these problems by using the framework of Coordination Graphs and the max-plus algorithm discussed in the previous chapter combined with Wiering's vehicle-based approach which already deals with the extremely large state-space. The state-action space is decomposed into pair-wise functions allowing each agent to take into account the behavior of all of its neighbors. For a vehicle-based model, a model-free method would not scale well when max-plus is used. This is due to the fact that max-plus would need to compute the joint action for all vehicles at each time step since the update is vehicle-based. The latter implies that max-plus would have to run many times at each time-step. However, a model-based method would only require an update of the lookup tables based on each vehicle's move and max-plus would need to run once per time step to compute the joint action. It can therefore be said to be computationally cheaper than the model-free method for the vehicle-based approach.

7.1 Modeling

We use the vehicle-based representation as defined in Section 5.1 as follows:

- s: is the global state and is fully observable
- $i, j \in J$: represent two intersection controllers such that $i \in \Gamma(j)$ and $j \in \Gamma(i)$. $\Gamma(i)$ is that set of agents for which a dependence relationship is assumed with agent i.
- A: represents a set of intersection controller decisions (actions) where $A_j \subseteq A$ is the subset of possible actions of intersection j as defined earlier.
- L: the set of lanes, where $L_i \subseteq L$ is the subset of incoming lanes to intersection i , $l_k \in L_i$ is a single lane.

- The global state s decomposes as: $\langle s \rangle = \langle s_i \rangle = \langle s_{p_{l_i}} \rangle$ where p_{l_i} is a position on lane i.
- $p(s'|s,a)$: represents the global transition model

7.2 Learning

The Q function decomposition is given by:

$$Q(s,a) = \sum_i Q_i(s_i, a_i, a_j) \tag{7.1}$$

where $a_i \in A_i$ is the action of agent i and $a_j \in A_j$ is the action of agent j where $j \in \Gamma(i)$ and,

$$Q_i(s_i, a_i, a_j) = \sum_{l_i} \sum_{p_{l_i}} Q_{p_{l_i}}(s_{p_{l_i}}, a_i, a_j) \tag{7.2}$$

The update rule is then given by:

$$Q_{p_{l_i}}(s_{p_{l_i}}, a_i, a_j) := \sum_{p'_{l_i}} P(s_{p_{l_i}}, a_i, a_j, s'_{p_{l_i}})[r(s_{p_{l_i}}, s'_{p_{l_i}}) + \gamma V^{(j)}(s'_{p_{l_i}})] \tag{7.3}$$

Updating the V function:

$$V^{(j)}(s_{p_{l_i}}) := \sum_{a_i, a_j} P(a_i, a_j | s_{p_{l_i}}) Q(s_{p_{l_i}}, a_i, a_j) \tag{7.4}$$

Updating the transition model:

$$P(s'_{p_{l_i}} | s_{p_{l_i}}, a_i, a_j) := \frac{C(s_{p_{l_i}}, a_i, a_j, s'_{p_{l_i}})}{C(s_{p_{l_i}}, a_i, a_j)} \tag{7.5}$$

and,

$$P(a_i, a_j | s_{p_{l_i}}) := \frac{C(s_{p_{l_i}}, a_i, a_j)}{C(s_{p_{l_i}})} \tag{7.6}$$

7.3 Optimal Decision Making

Using the above, we can now define the function between each pair of neighbors as used by max-plus:

$$f_{ij}(a_i, a_j) = \sum_{p_{l_i}} O_{p_{l_i}} Q_{p_{l_i}}(s_{p_{l_i}}, a_i, a_j) + \sum_{p_{l_j}} O_{p_{l_j}} Q_{p_{l_j}}(s_{p_{l_j}}, a_j, a_i) \tag{7.7}$$

which is plugged directly into Equation 6.3. Note that the function is symmetric such that $f_{ij}(a_i, a_j) = f_{ji}(a_j, a_i)$. Thus, using Equation 7.7, the joint action can be estimated directly by the max-plus algorithm.

Like Wiering, we use ϵ-greedy exploration and a 1-step update of value functions. In this approach, each agent can observe the state around each of its neighbors, moreover, it takes

their behavior into account. Learning is cooperative as each pair of neighbors share a value function. Dependence can also be assumed between agents that are not directly connected, but we make the simplifying assumption that dependence exists only between connected agents who may directly influence each other by their actions. Note that the number of value functions is linear in the induced width of the graph.

There are two levels of value propagation among agents. On the lower level, like in Wiering's approach, expected waiting times of vehicles are propagated between neighboring agents and so these values are propagated through the entire network. On a higher level, when making a decision, agents inform each connected agent in the graph the best value they will be able to achieve for each action of this agent given the current state and the values received from other agents about their own actions.

Chapter 8

Experiments & Results

In this chapter the novel approach is compared to the two methods discussed in Chapter 5. This includes Wiering's approach (e.g. vehicle-based) and the extending work of Bakker et al. The methods were called TC-1 (Traffic Controller 1) and TC-SBC (State Bit for Congestion) respectively and we will use the same notation. Wiering compared TC-1 with two baseline strategies. The first is a throughput based strategy, which sets the lights at any time step in such a way that a maximum number of vehicles can pass through the intersection. The second strategy always gives the right of way to the longest queue at the intersection. In under-saturated conditions, the performance of the approach is similar to that of the two baselines. As the traffic volume grows, the estimated travel times of vehicles increasingly differ, depending on congested roads at later stages of their path. TC-1 profits from learning these estimates and outperforms the baselines notably under conditions of heavy traffic. Therefore, we only compare our approach to the two methods (and not to other baselines) under highly saturated conditions.

When can cooperative learning and coordination can be said to be valuable or even critical? Or when is the vehicle-based approach, in which agents do not take into account the behavior of other agents, prone to fail? Here we argue that under highly saturated conditions, cooperative learning and coordination is likely to outperform the previous method(s) when the amount of *local traffic* is relatively small. Local traffic refers to traffic that only involves a single agent (i.e. intersection), meaning that vehicles cross a single intersection and then exit the network. In other words, when the majority of the vehicles traveling in the network pass more than one intersection, cooperative learning and coordination seems to play an important role. In contrary to the previous methods, the novel method shows stability and robustness on such networks.

The reader may develop some intuition by considering two extreme cases. The first case is when all traffic is local such that no vehicle crosses more than one intersection. Clearly, no coordination is needed and problems can be solved locally. Moreover, no agent exists whose state depends on the actions of other agents since there is no traffic flow among agents. The second case is when there is no local traffic such that no vehicle crosses only a single intersection. In this case, the state of an agent may depend on the actions of some other agents as their actions may, and even most likely will, affect the state of this agent.

There are a number of properties a network can have which directly reduce the relative amount of local traffic. First, the straightforward case where networks have no intersections that are connected to more than one edge node. In this case, there is no local traffic as the start and destination nodes are never connected to the same intersection. Second, when traffic is non-uniformly distributed such that for intersections which are connected to more than one edge node the amount of traffic sent between these nodes will be relatively small. In this case, most vehicles coming from these edge nodes travel to the inner part of the network and cross more than one intersection. Third, if some intersections occasionally (but not always) receive a large amount of traffic, then the saturation on lanes connecting agents will increase.

We will first test these properties on small networks and then see if these principles also hold for larger networks involving many agents. All of the experiments are averaged over 10 runs with different random seeds. Furthermore, we assume the environment to be stationary during a simulation. That is, for all edge nodes the probability that a vehicle will enter remains static during a simulation. This also holds for the simulation in which some intersections occasionally receive a large amount of traffic. In this case, some edge nodes generate one vehicle at a given time step with probability p_1, a large number of vehicles with some small probability p_2 and will otherwise generate no vehicles. Throughout this chapter we use the term *spawning rate* to refer to the probability that an edge node will generate traffic at a given time step.

8.1 Statistics

The statistics used in the experiments include the Average Trip Waiting Time (ATWT), Ratio of Stopped vehicles and Total Queue.

ATWT is the actual function that is being minimized. It is the total waiting time of all vehicles that have reached their destination divided by the number of these vehicles.

Ratio of Stopped Vehicles shows the ratio of of all vehicles currently traveling in the network that have not moved per time-step. It is a good indicator of the wellness of the joint action of the intersections as good joint actions should generally allow more vehicles to move.

Total Queue measures the total number of vehicles waiting to enter the network at all edge nodes. It is possible that vehicles generated by an edge node are unable to enter the network since its outbound road is fully occupied. The ATWT will not be updated by these vehicles and it may seem that a method performs well while actually many vehicles with a relatively large waiting time have not yet arrived since they have not yet entered the network.

Together, these three performance measures provide a good overview of the performance of a method.

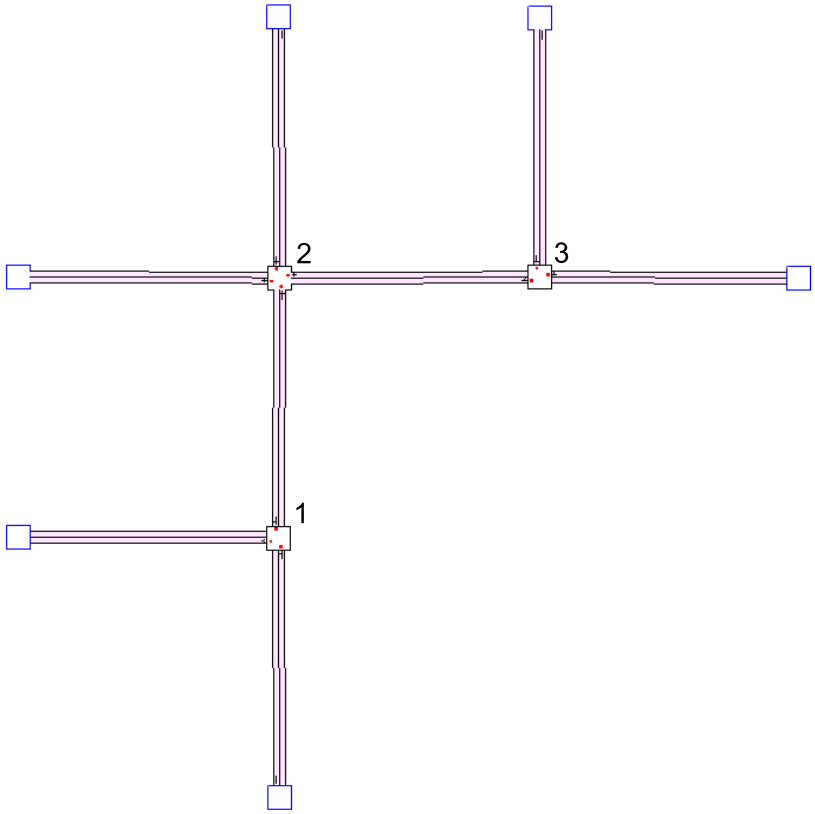

Figure 8.1: A network of three intersections

8.2 Experiment I: Base Case

Figure 8.1 shows a small symmetric network of three intersections. At each phase the intersections can allow traffic to cross from one direction only (e.g. set the light to green only on one of their incoming lanes). All lanes have an equal length and the spawning rates of all edge nodes are equal such that similar amount of incoming traffic is expected from all edge nodes over time. We compare the methods on uniform and non-uniform destinations as well as non-uniform spawning rates. In all experiments, the number of iterations max-plus is allowed to run is limited to 3.

8.2.1 Non-Uniform Destinations

Consider the following scenario for the network depicted in Figure 8.1. All traffic from intersections 1 and 3 is directed to intersection 2 such that the destinations of vehicles traveling from these intersections are any of the edge nodes connected the intersection 2. Traffic from the top edge node connected to intersection 2 is sent only to intersections 1 and traffic from the other edge node is sent only to intersection 3. In this scenario there is no local traffic, all vehicles traveling in the network cross exactly two intersections in order to reach their destination. We also compare the results when traffic is uniformly distributed (e.g. destinations are selected uniformly at random). In both simulations all edge nodes generate a vehicle with a probability of 0.2 per time step. Figures 8.2, 8.3, and 8.4 show the results for the uniform

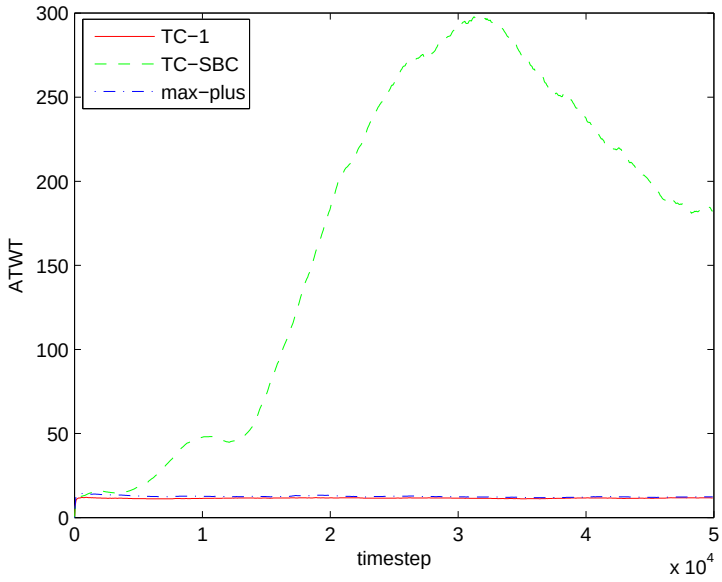

Figure 8.2: ATWT for uniformly distributed traffic on the network with three intersections.

case and figures 8.5, 8.6, and 8.7 show the results for the non-uniform case.

In the uniform case, there is no significant difference in performance between max-plus and TC-1. There are two reason why max-plus will not outperform TC-1 in this scenario. The first reason is the relatively large amount of local traffic which accounts for 20% of all traffic in the network. The second reason is that all intersections are connected to a number of (e.g. 2) edge nodes. Increasing the spawning rates in the uniform case results in the congestion at the outer parts of the network up to the point where all of the methods fail. When the amount of local traffic is reduced but the spawning rates remain the same, the saturation among agents increases. Max-plus is able to deal with this increased saturation but TC-1 fails. TC-SBC performs worse than the other two methods. A reason for this might be the growth of the state-space due to the extra congestion information included in the state representation. As mentioned, the size of the state-space has a direct influence on the convergence rate and the method may therefore need more learning in order to converge to its optimal behavior.

In the non-uniform case where there is no local traffic, it is very probable that the lane from intersection 1 to intersection 2 will become highly saturated as all vehicles traveling from the edge nodes connected to intersection 1 are directed to this lane. If this happens it can be argued that it is important for the two intersections to coordinate their actions since allowing traffic to cross from any of the incoming lanes of intersection 1 connected an edge node is meaningless unless intersection 2 allows traffic on the incoming lane from intersection 1 to cross thereby coordinating a "green wave" between them. Similarly, if the lane from intersection 2 to intersection 1 is fully occupied then allowing traffic to cross from the top edge node connected to intersection 2 is meaningless unless intersection 1 allows traffic on the incoming lane from intersection 2 to cross. As the model is symmetric similar relationship holds between intersections 2 and 3. In such scenarios we can distinct between joint actions

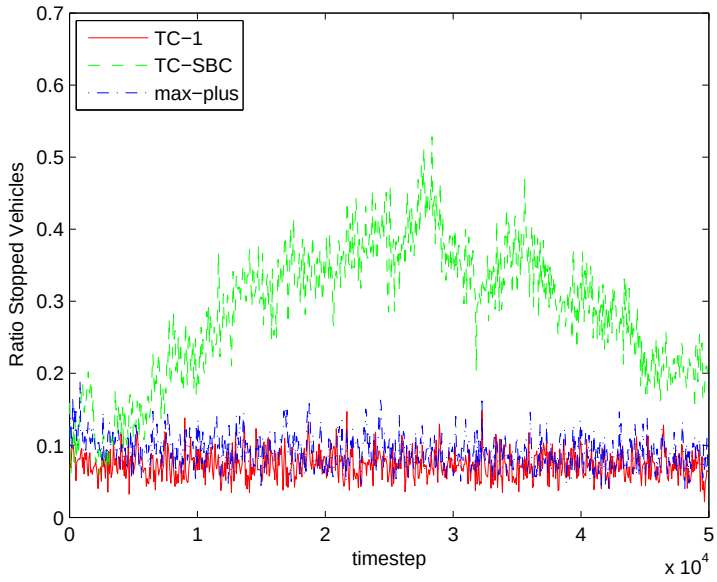

Figure 8.3: Ratio of stopped vehicles per cycle for uniformly distributed traffic on the network with three intersections

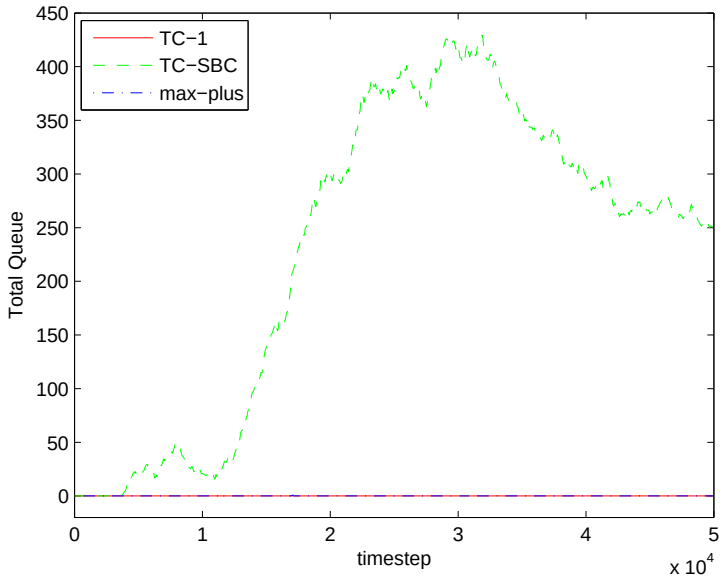

Figure 8.4: Total queue per cycle for uniformly distributed traffic on the network with three intersections

which will deal with the congestion (by coordinating such "green wave") and ones that will not. In the most extreme case, like in the discussed scenario, if the probability that the next vehicle will travel to the congested lane equals 1 then no vehicle will move if such joint action is taken when the destination lane is fully congested. The intersection allows a lane to move but the first vehicle on the lane cannot cross since its destination lane is fully occupied. If such scenarios occur frequently and joint actions are chosen too often for which vehicles can not cross then the system is prone to fail at performing its task. Vehicles will keep entering from the edge nodes and congestion will increase at other lanes as well (in larger networks) up to a point that the system will be unable to resolve the problem. Intuitively, TC-1 is more likely to choose such joint actions. For TC-1, the optimal action of each agent is the one which will minimize the waiting time solely on its incoming lanes and independent of the actions of the other agents. In other words, it is not common knowledge among agents that a certain joint action in a given state is ineffective. However, if the agents learn in a cooperative manner then through experience it becomes common knowledge between them what joint actions are more effective than other in such scenarios and are therefore less likely to be chosen.

In our simulation, vehicles must begin and end their trip at edge nodes. I.e. no traffic is generated from within the network. Since we are only testing highly saturated networks, the spawning rates of edge nodes must be relatively high. Assuming equal spawning rates at all edge nodes, when an intersection is connected to more than one edge node and vehicles travel between these edge nodes quite frequently then max-plus shows no advantage over the previous methods. The problem can then be said to be more local than global. The saturation, or congestion, occurs more at the outer parts of the network (e.g. roads connecting edge nodes with intersections) than at the inner parts of network (e.g. roads connecting intersections). If the spawning rates remain constant but local traffic is reduced, saturation at the inner part of the network increases since more traffic travels through the network. In this case, max-plus is able to deal with higher traffic loads and outperforms the previous methods substantially. Table 8.1 shows the average (and standard deviation) of the 10 runs at the last time step and the difference in performance can clearly be seen on all measures.

Measure	**max-plus**	**TC-1**	**TC-SBC**
ATWT	13.54(2.63)	351.49(365.66)	240.71(315.63)
Ratio stopped vehicles	0.15 (0.02)	0.47(0.24)	0.34(0.21)
Total Queue	0(0)	482.7 (571.7)	302.89(462.18)

Table 8.1: The mean (and standard deviation) of all measures at the last timestep on the network with three intersections when traffic is non-uniformly distributed such that there is no local traffic.

8.2.2 Non-Uniform Spawning Rates

In this experiment, the methods are compared for non-uniform spawning rates. All edge nodes have an equal spawning rate of 0.1. Intersection 1 and 2 have one connected edge node respectively that generates 500 vehicles with probability 10^{-4} and destinations are selected uniformly at random. TC-1 performs slightly better than max-plus. Here, the non-uniform spawning rates have no effect on the distribution of traffic. In other words, 20% of all traffic is still local and congestion is still more likely to occur at the lanes leading into the network

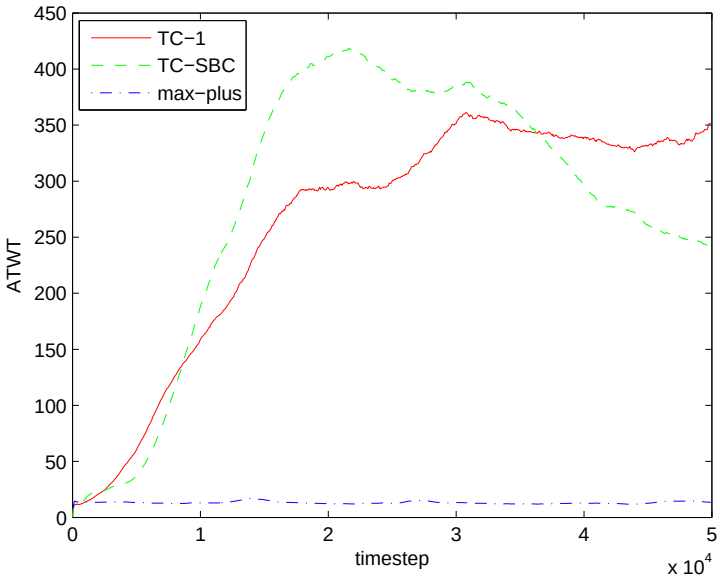

Figure 8.5: ATWT for non-uniformly distributed traffic on the network with three intersections

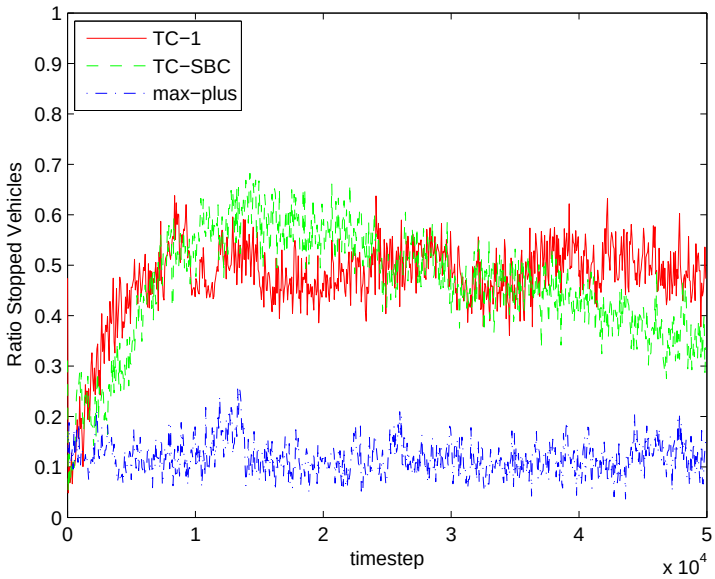

Figure 8.6: Ratio of Stopped Vehicles per timestep for non-uniformly distributed traffic on the network with three intersections

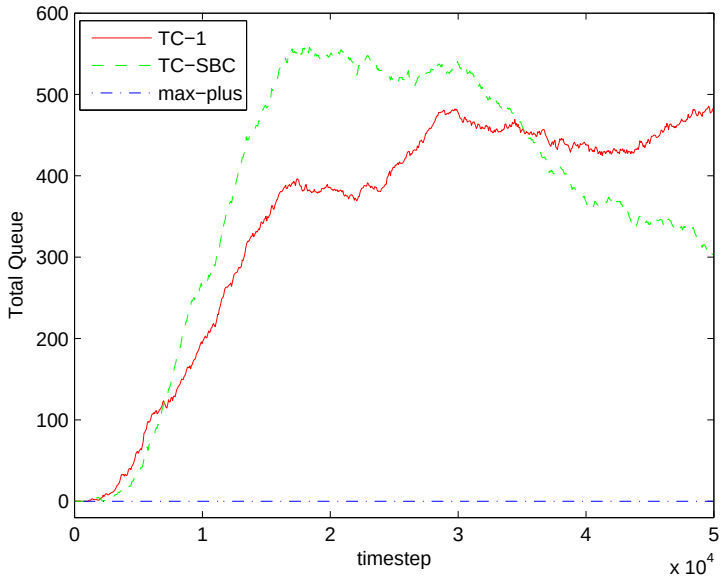

Figure 8.7: Total Queue waiting to enter the network per timestep for non-uniformly distributed traffic on the network with three intersections

than on lanes connecting intersections.

8.2.3 Uniform Destinations without Local Traffic

In this experiment we test the minimal network in which traffic is uniformly distributed but there is no local traffic. Figure 8.11 shows a network of 4 intersections, each connected to a single edge node. In this network, all vehicles need to cross at least two intersections in order to reach their destination and destinations are chosen uniformly at random. The results are shown in Figures 8.12,8.13, and 8.14. Since there is no local traffic, saturation increases on the inner part of the network (e.g. roads connecting intersections). Max-plus is able to deal with higher spawning rates by choosing better joint actions given the global state. Table 8.2 shows the average (and standard deviation) of the 10 runs at the last time step. The average ratio of stopped vehicles for max-plus is 0.09 with a standard deviation of 0.06. On average, max-plus allows more than 90% of the vehicles to move. If we look at TC-1, the average ratio is 0.92 meaning that on average approximately 90% of the vehicles are not moving. The results for TC-SBC are better than TC-1. The ATWT of TC-1 is lower than TC-SBC but this is due to the large number of vehicles in the waiting queue. The ratio of stopped vehicles is lower as well as the total queue. Still, this method performs poorly with respect to max-plus.

Measure	**max-plus**	**TC-1**	**TC-SBC**
ATWT	16.39(0.56)	182.8(351.02)	481.15(583.99)
Ratio stopped vehicles	0.09 (0.06)	0.92(0.17)	0.55(0.32)
Total Queue	0(0)	9259.7(2347.0)	3966.9(4301.5)

Table 8.2: The mean (and standard deviation) of all measures at the last timestep on the network with four intersections where all traffic is non-local

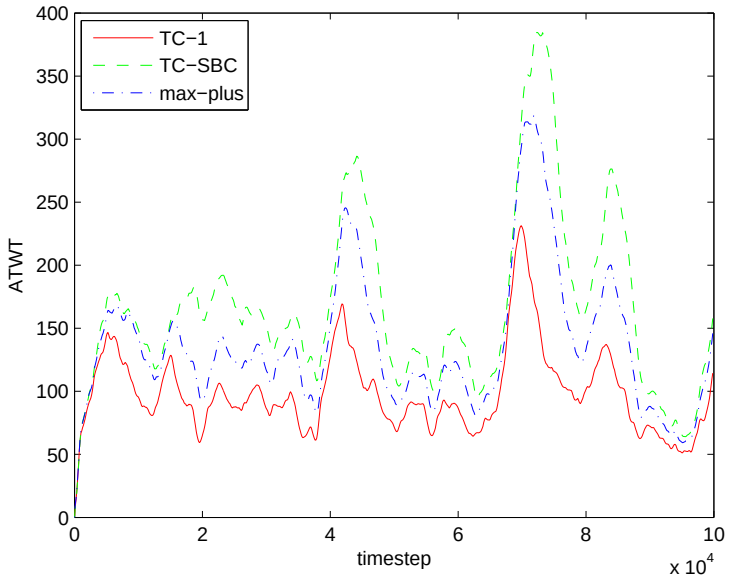

Figure 8.8: ATWT per timestep for non-uniform spawning rates for the network with three intersections

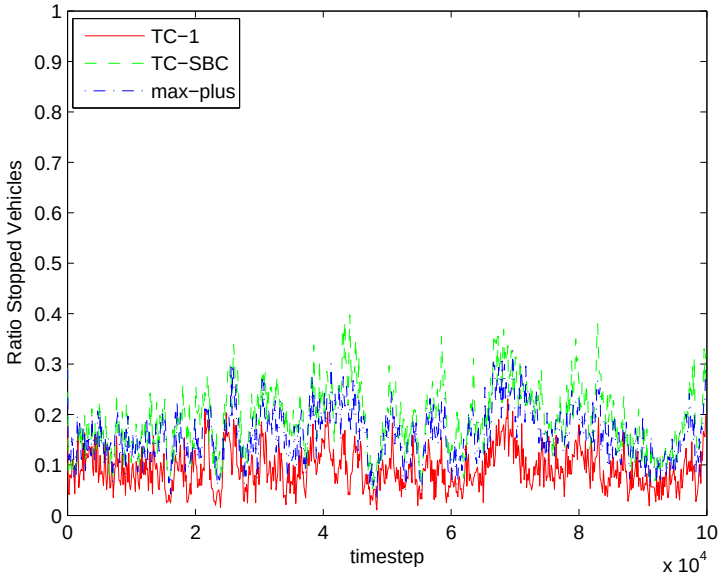

Figure 8.9: Ratio of Stopped Vehicles per timestep for non-uniform spawning rates for the network with three intersections

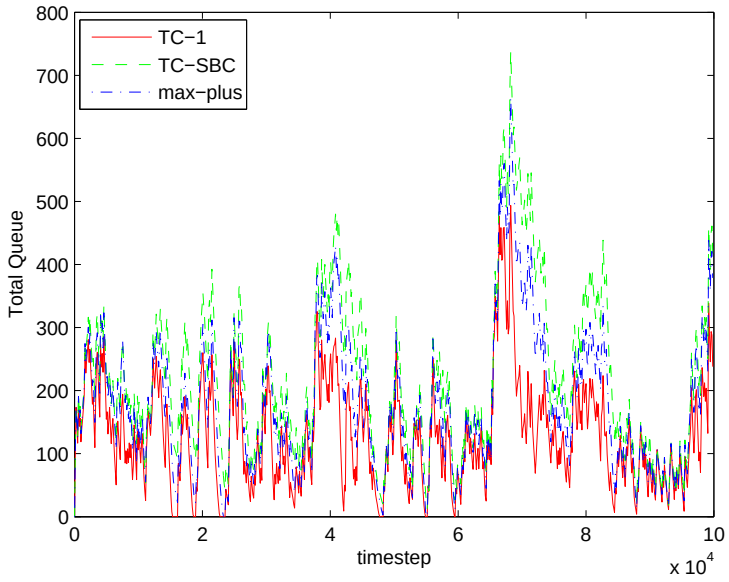

Figure 8.10: Total Queue waiting to enter the network per timestep with non-uniform spawning rates for the network with three intersections

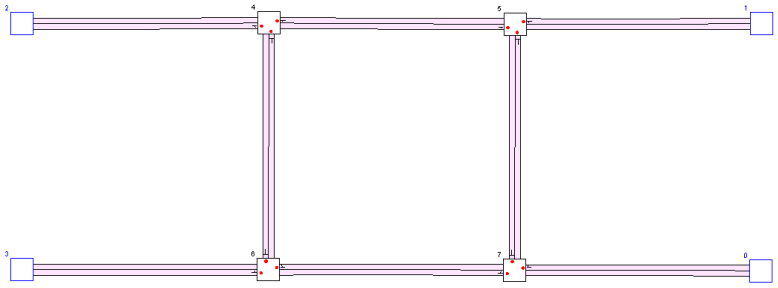

Figure 8.11: Network of with four intersections and no local traffic. All vehicles need to cross at least two intersections in order to reach their destination

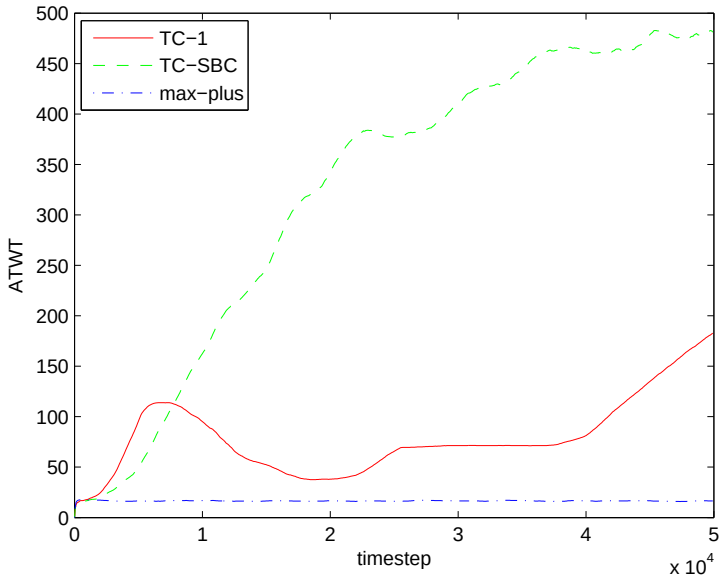

Figure 8.12: ATWT per timestep for the 4-intersection network

8.3 Experiment II: Scaling to a Large Network

The following experiments are used to verify whether the same principles hold for a larger network involving many agents. The network depicted in Figure 8.15 includes 15 agents and roads with four lanes (two in each direction). We test non-uniform vs uniform destinations, non-uniform spawning rates, and uniform destinations without local traffic. Again, in all experiments we limit the number of iterations max-plus is allowed to run to 3.

8.3.1 Non-Uniform Destinations

In the first case, traffic is uniformly distributed over the network. In the second case, traffic from the top edge nodes travel to the bottom edge nodes and vice versa. Similarly, traffic from the left edge nodes travels to the right edge nodes and vice versa. In both cases, spawning rates are equal for all edge nodes. Figures 8.16 and 8.17, and 8.18 show the results for the first case and figures 8.19,8.20, and 8.21 show the results for the second case. In the uniform case, performance of all three methods is almost identical. Increasing the spawning rates will eventually cause congestion at the two intersections connected to two edge nods respectively (see figure 8.15). In this case, all methods fail for approximately the same spawning rate. When we eliminate the local traffic as done in the second case, but keep the same spawning rates then max-plus is the only method that is able to deal with this scenario. Here, all vehicles pass at least four intersection to reach their destination and saturation increases at the inner parts of the network (e.g. between intersections). TC-SBC does perform better than TC-1, but the difference in performance with respect to max-plus is still very large. Table 8.3 shows the average (and standard deviation) of all measures at the last time-step. Max-plus has no waiting queues at all simulations while TC-SBC has an average of 894.8(674.76), the ratio of stopped vehicles is 0.22(0.04) while for TC-SBC it is 0.41(0.09), and the ATWT for max-plus is 13.81(0.61) while for TC-SBC the ATWT at the last time-step is 111.75(66.04).

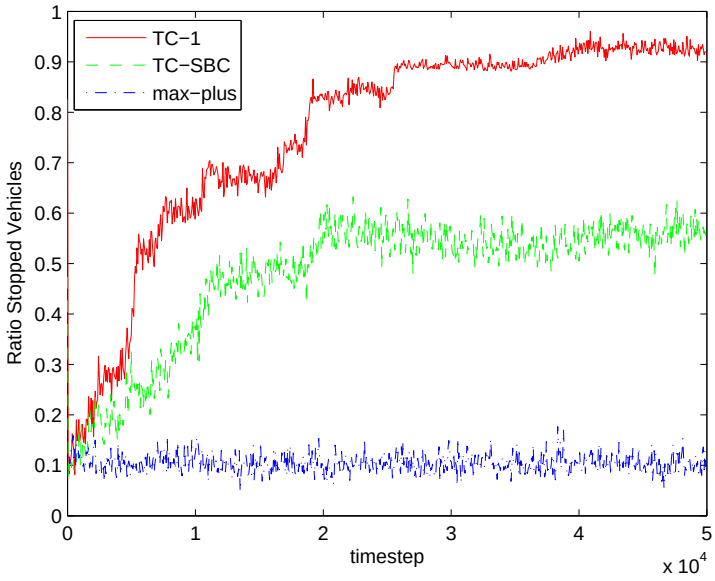

Figure 8.13: Ratio of Stopped Vehicles per timestep for the 4-intersection network

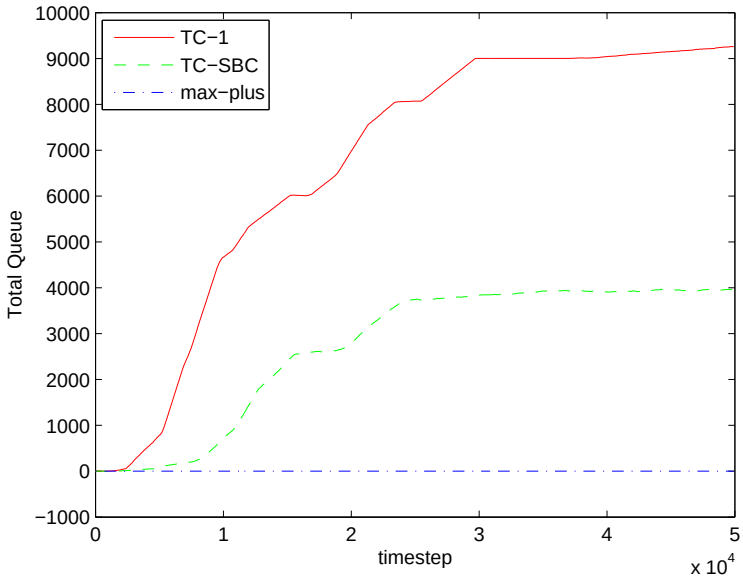

Figure 8.14: Total Queue waiting to enter the network per timestep for the 4-intersection network

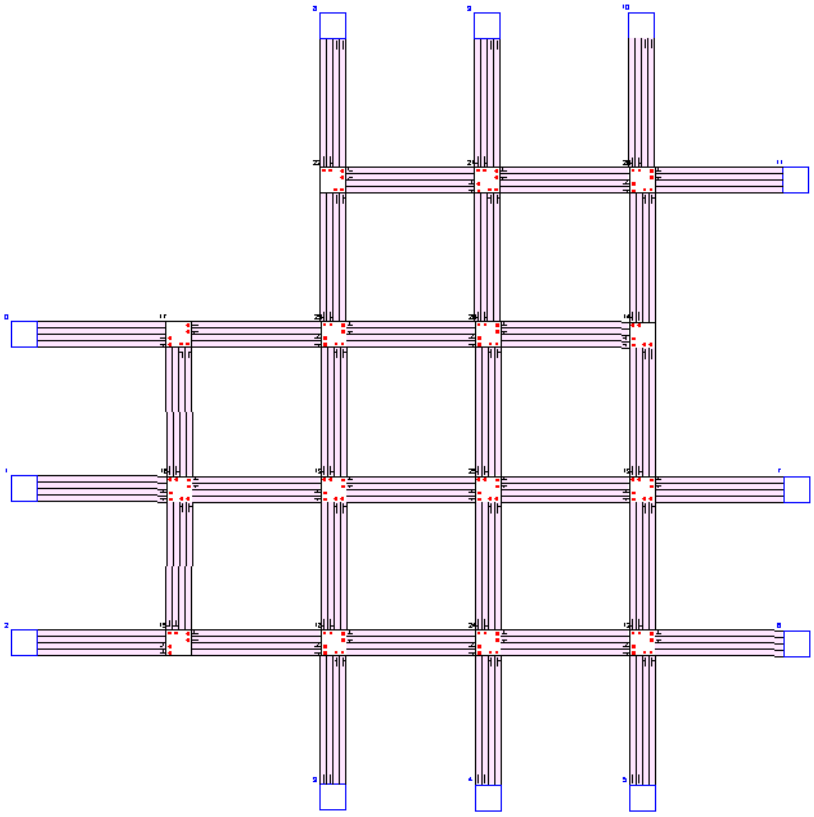

Figure 8.15: A large network with 15 intersections

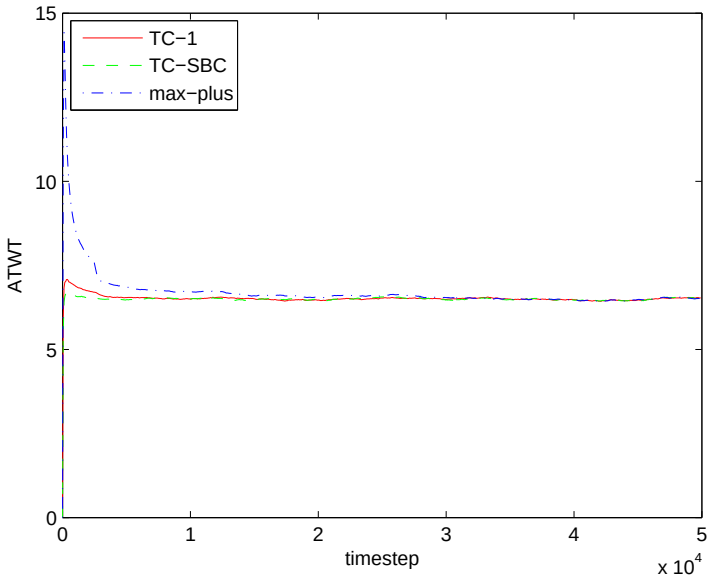

Figure 8.16: ATWT per timestep for uniformly distributed traffic on the large network

Again we can see that while TC-1 and TC-SBC usually choose joint actions which allow approximately 55-60% of the vehicles to move, the joint actions chosen by max-plus allow approximately 75-80% of the vehicles to move throughout the simulations.

Measure	**max-plus**	**TC-1**	**TC-SBC**
Ratio stopped vehicles	0.22(0.037)	0.440(0.096)	0.41(0.09)
Total Queue	0(0)	4096.7(1463)	894.8(674.76)
ATWT	13.81(0.61)	390.93(135.34)	111.75(66.04)

Table 8.3: The mean (and standard deviation) of all measures at the last timestep with non-uniformly distributed traffic on the large network.

8.3.2 Non-Uniform Spawning Rates

In the following experiment we test the non-uniform spawning rates as follows. The center-left, center-bottom and center-top edge nodes generate 2000 vehicles with probability 10^{-4}. The results are shown in Figures 8.22, 8.23 and 8.24. Table 8.4 shows the average (and standard deviation) of all measures at the last time step. The ATWT is somewhat meaningless in this experiment due to the large number of vehicles that have never reached their destination. The main difference can therefore be seen in the Total Queue and Ratio of Stopped Vehicles at the last timestep. The average (and standard deviation) of the total queue at the last time-step for max-plus is 1369.6(1258.9) while that of TC-1 and TC-SBC are 94263(16017) and 56718(48103) respectively. Max-plus in some cases has a waiting queue at the last time-step since some edge node generated 2000 vehicles a short time before but throughout the simulations these queues are dealt with as can be seen in figure 8.24. TC-1 and TC-SBC start failing at a certain point at all simulations and queues begin to build up at edge nodes

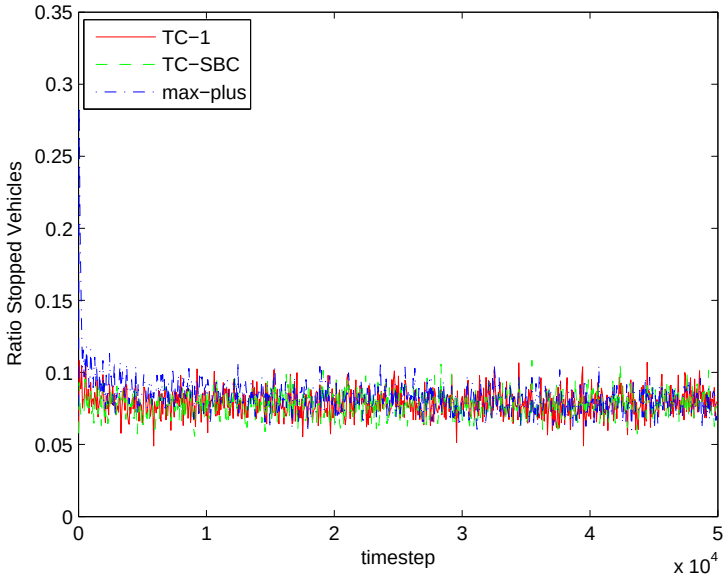

Figure 8.17: Ratio of stopped vehicles vehicles per timestep for uniformly distributed traffic on the large network

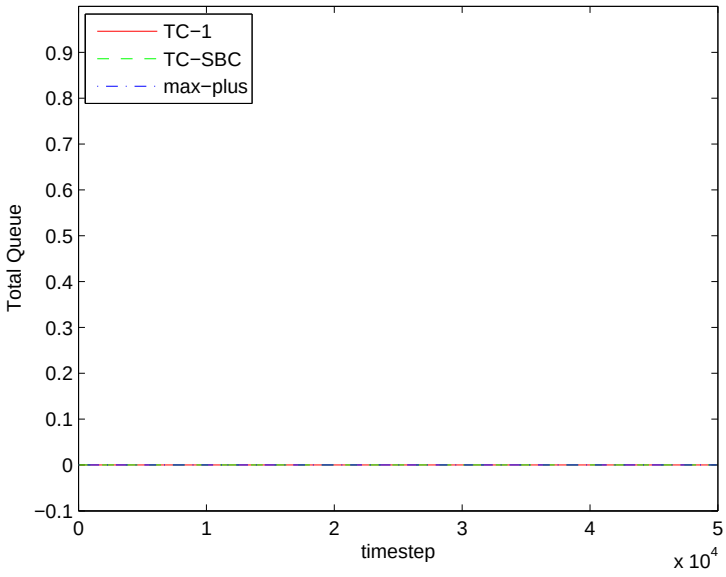

Figure 8.18: Total queue waiting to enter the network per timestep for uniformly distributed traffic on the large network

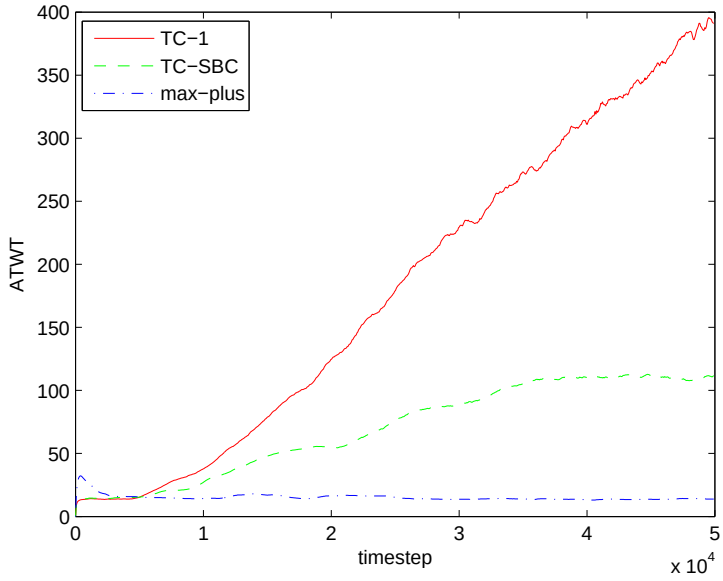

Figure 8.19: ATWT per timestep for non-uniformly distributed traffic on the large network

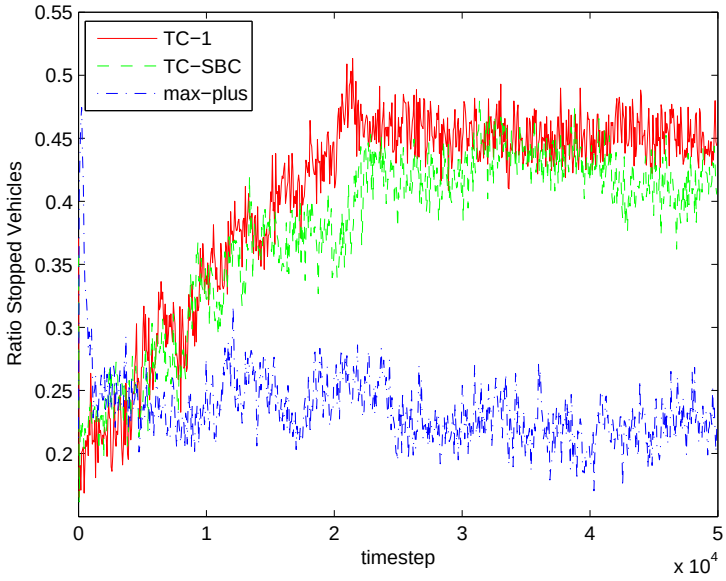

Figure 8.20: Ratio of non-moving vehicles per timestep on a non-uniformly distributed traffic on the large network

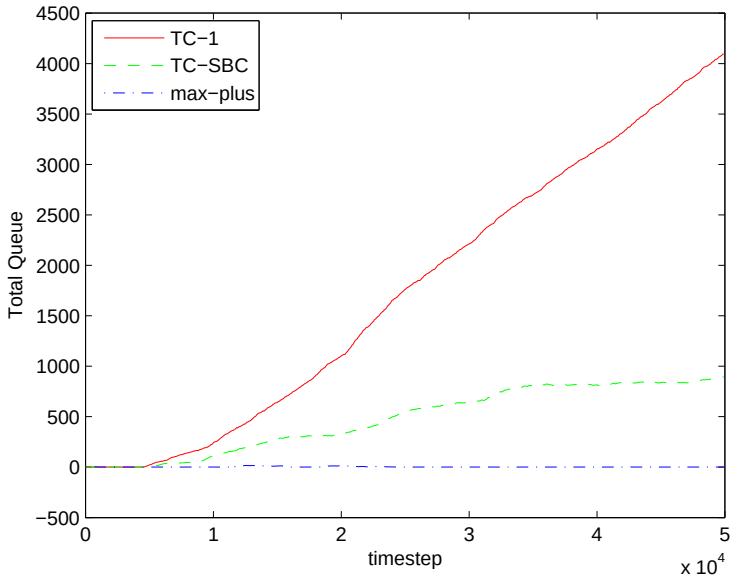

Figure 8.21: Total Queue at all edge nodes per timestep for non-uniformly distributed traffic on the large network

in the network. The latter can clearly be seen in figures 8.23 and 8.24. Again, TC-SBC performs better than TC-1 but the difference with respect to max-plus is also in this case very large. The behavior of TC-1 when the network becomes highly saturated is similar to the one as described in section 8.2.1 but the effect in this experiment is even worse. Consider the following scenario. When the road from some agent i to some other agent j is fully congested, it is meaningless for agent i to allow a stream in that direction to cross unless agent j also allows this road (from agent i) to move. Now, if there is some agent k connected to agent j and most traffic from the inbound road of agent j (from agent i) is headed towards agent k but this last road is also congested, then if and only if all three agents generate a "green wave" then traffic will move. Again, since the agents in TC-1 do not take the actions of their neighbors into account, they never learn that some joint actions are simply ineffective. This can cause (and often does cause) a situation in which almost all the vehicles in the network can not move. The latter can clearly be seen in table 8.4 where the average ratio of stopped vehicles for TC-1 at the last time-step is 0.89. In other words, approx. 90% of the vehicles did not move at the last time-step at all simulations. Since the agents in TC-SBC do take congestion information at outbound roads into account they are able to deal with it better than TC-1 but the difference in performance with respect to max-plus is still very large.

Measure	**max-plus**	**TC-1**	**TC-SBC**
ATWT	155.56(117.87)	381.85(392.78)	199.05(166.76)
Ratio stopped vehicles	0.19(0.06)	0.89(0.02)	0.68(0.34)
Total Queue	1369.6(1258.9)	94263(16017)	56718(48103)

Table 8.4: The mean (and standard deviation) of all measures at the last timestep for the large network with non-uniform spawning rates

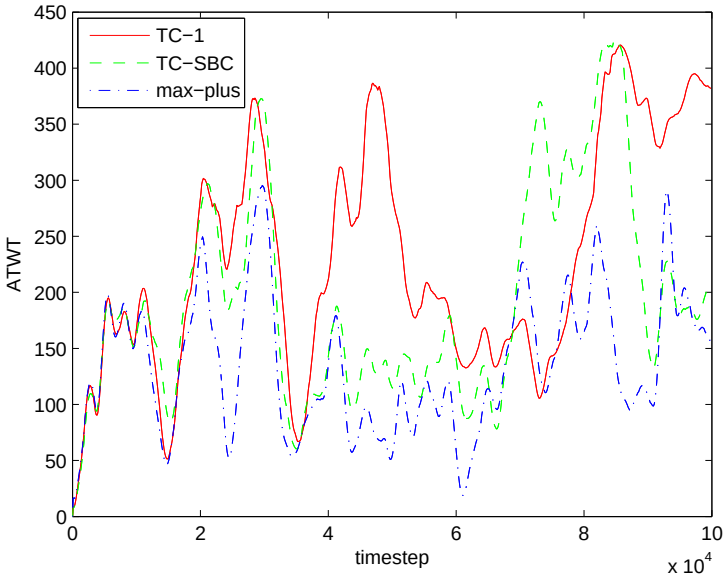

Figure 8.22: ATWT per timestep for the large network with non-uniform spawning rates

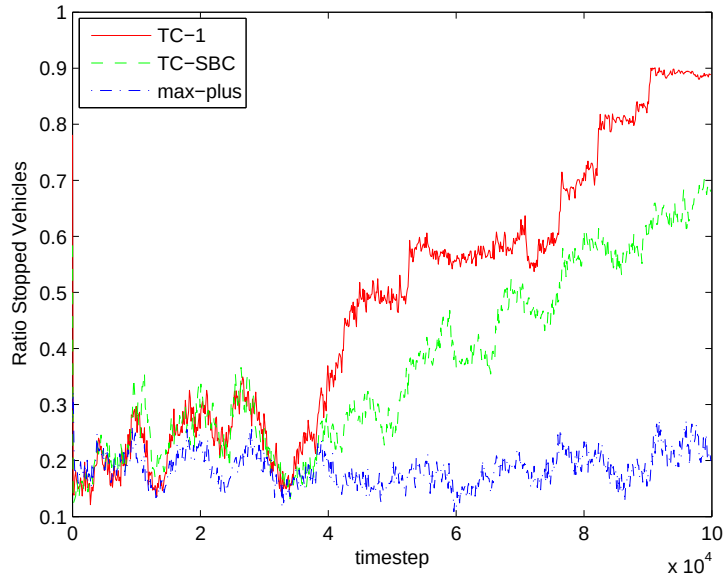

Figure 8.23: Ratio of non-moving vehicles per timestep for the large network with non-uniform spawning rates

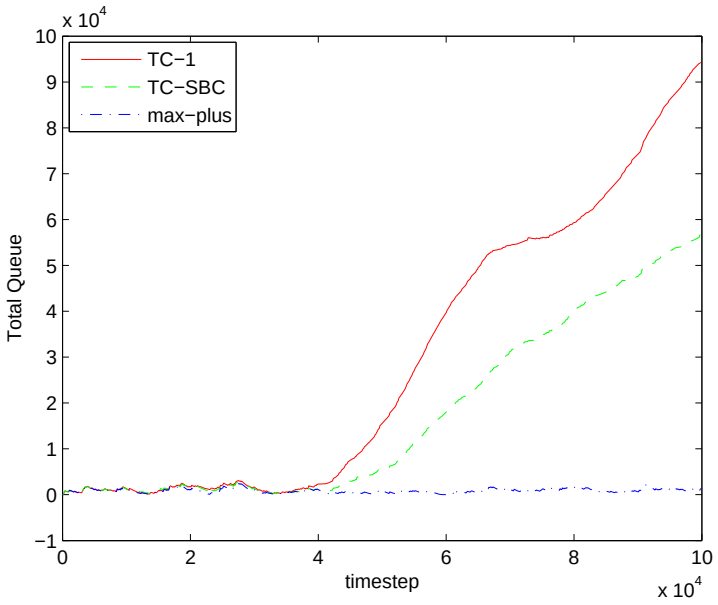

Figure 8.24: Total Queue at all edge nodes per timestep for the large network with non-uniform spawning rates

8.3.3 Uniform Destinations without Local Traffic

In this experiment we eliminate one edge node for the two intersection which are connected to two edge nodes (see figure 8.15). I.e. the eliminated edge nodes do not generate/receive any traffic. Traffic is uniformly distributed over the network as destination nodes are selected uniformly at random and vehicles cross at least two intersections in order to reach their destination. Results for this experiment are shown in figures 8.25, 8.26 and 8.27. Table 8.5 shows the average (and standard deviation) of all measures at the last time step. The difference in performance in this experiment can be seen on all measures. I.e. the difference in all measures is very large between max-plus and TC-1 as well as TC-SBC.

Measure	**max-plus**	**TC-1**	**TC-SBC**
ATWT	14.39(4.18)	183.86(177.99)	218.48(225.54)
Ratio stopped vehicles	0.19(0.04)	0.81(0.23)	0.86(0.08)
Total Queue	0(0)	9007.2(3164.8)	9898.1(345.44)

Table 8.5: The mean (and standard deviation) of all measures at the last timestep for the large network without local traffic.

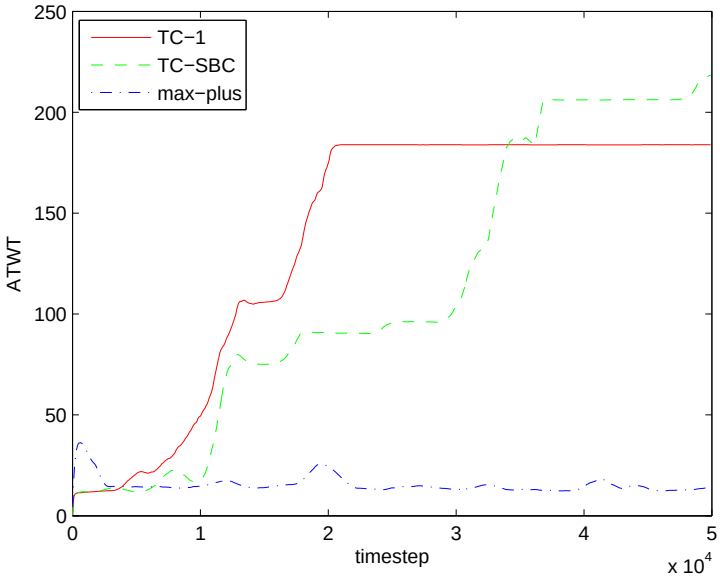

Figure 8.25: ATWT per timestep for the large network without local traffic.

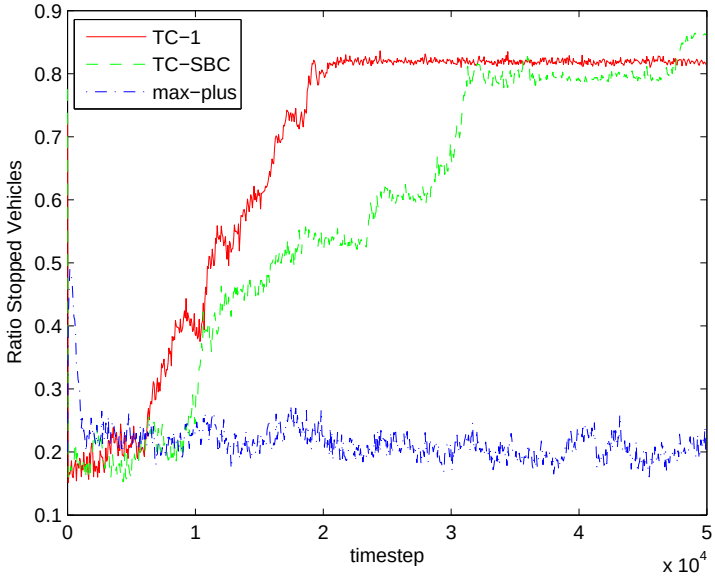

Figure 8.26: Ratio of stopped vehicles per timestep for the large network without local traffic

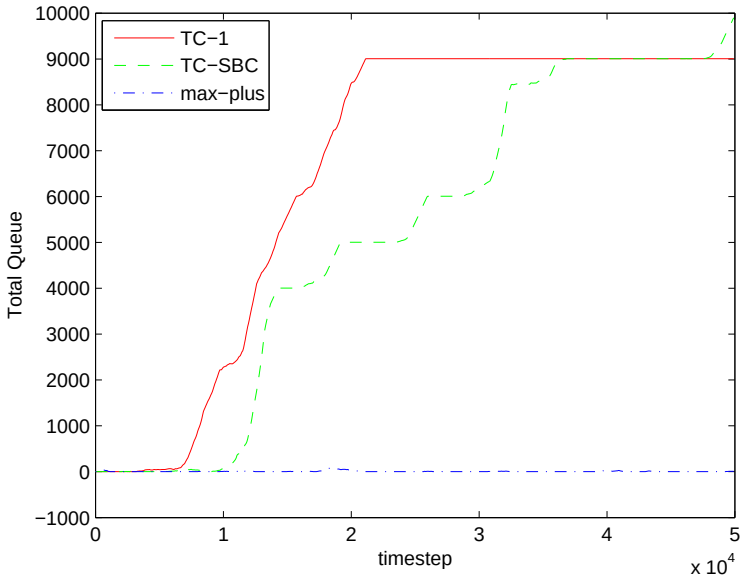

Figure 8.27: Total Queue at all edge nodes per timestep for the large network without local traffic

We began this chapter with a certain claim regarding *Local Traffic* which refers to traffic that only involves a single agent. We claimed that the difference in performance between the novel method and previous ones is highly correlated with this phenomenon. Local Traffic is actually a result of different conditions and properties traffic networks can have that influence the importance of coordination. Some of these are the spawning rates, the distribution of traffic and even the size and structure of the network. An attempt to define the exact conditions (and their relationships) might always be to some extent imprecise due to the large number of possible configurations, conditions and especially the varying dynamics between different scenarios. In other words, the conditions might be valid in certain environments and at the same time invalid in other. What can be said with a high degree of certainty is that an increase in local traffic decreases the value of coordination and when sufficiently high guarantees that coordination is not needed. Furthermore, the lack of local traffic generally implies that coordination is required, although there still might be special cases in which coordination will not be required.

In order to verify this claim we tested traffic networks under different conditions that directly influence local traffic. The results confirmed our claim as the novel method was shown to outperform the previous methods substantially under these different conditions. We also showed that it performs as well as the previous methods in other scenarios (although it generally requires more learning to converge). Thus, our method can be said to be more robust as it generally performs as well as the previous methods but can also deal with additional scenarios in which the previous methods have been shown to fail. We have also shown that under highly saturated conditions, the difference in performance between the novel and the previous methods increases with the size of the network when the amount of non-local traffic increases. In other words, as the number of agents increases the significance of cooperative

learning and coordination among agents also increases. As the system never learns that for some neighboring agents certain joint actions are ineffective in certain states, choosing such non-coordinated joint actions increases the congestion at other roads leading to congested area which in turn causes the congestion to spread throughout the network. Our results indicate that in many cases the majority of vehicles in the network can not move due to the congestion and the inability of the agents to coordinate joint actions which will deal with the problem. Thus, taking the behavior of fellow agents into account enables the agents to distinct between effective and ineffective joint actions and coordinate joint actions which effectively deal with the problem.

Chapter 9

Conclusions & Future Work

This thesis focused on solution-based techniques to learn, and coordinate, the behavior of agents in a multi-agent traffic control system with primary concentration on distributed, scalable methods for large traffic networks involving many agents. The primary objective of this thesis was to exploit local dependencies, making it possible to solve the global problem as a sum of local problems. The agents were restricted to local operation, observing local states and receiving local rewards based on a decomposed global reward function.

We found the vehicle-based approach an efficient and clever way to deal with the extremely large state-space. However, in previous methods (based on the vehicle-based approach), agents did not take into account the behavior of other agents. Hence, the joint action of all agents lacked coordination. We therefore extended this approach to allow agents to take into account the behavior of fellow agents, i.e. permitting *coordination*. In order to deal with the exponential growth of the action space, we assumed that dependence exists only between connected agents. This allowed us to decompose the global problem into a sum of local terms and directly apply methods which were developed for estimating the optimal joint action using such decomposition. Although we described a number of methods used to estimate the optimal joint action (e.g. max-plus and Variable Elimination), we found max-plus the only method that scales well for large problems involving many agents and dependencies.

The results indicate that our approach outperforms the previous methods on highly saturated networks in which local traffic is limited. Local traffic refers to traffic that only involves a single intersection. In such networks, previous methods that do not involve coordination between agents are unable to effectively deal with the congestion. Our approach seems to profit from this knowledge. The extended work of Bakker et al. in which agents are able to distinguish between congested and non-congested outbound roads also seems to improve performance but the improvement shown with our approach remains both novel and significant. We also found that the difference in performance between the novel and previous methods increases with the size of the network when the amount of non-local traffic increases. In other words, as the number of agents increase, the significance of cooperative learning and coordination among agents also increases. This makes our method particularly attractive for solving larger problems involving many agents in complex networks.

Although the vehicle-based approach is able to deal with the large state-space, we found that alternative approaches should also be considered as they may ultimately lead to better policies and more guarantees for convergence. Therefore, in the future we should consider investigating different approaches that allow a compact representation of the state-space. One possible direction that can be explored is to apply statistical analysis on a large set of macroscopic features to determine which would best correlate with the performance of the system. A smaller set of features can then be used to describe the environment thereby substantially reducing the size of the state-space.

In this thesis, we assumed the dependencies among agents to be fixed. A dependence relationship was assumed only between agents that are directly connected in the graph (i.e. connected by a road). It should be noted that for certain problems, different dependencies may lead to better solutions. Therefore, a possible extension of the current implementation would be to find the optimal set of dependencies for different problems. One approach to pursue would be the use of Genetic Algorithms, i.e. beginning with a population of dependencies and through an evolutionary process (as described in Section 2.5.3), creating new (and possibly improved) dependencies for the problem at hand.

Another possible extension would be to implement a distributed version of the max-plus algorithm. The current version, as used in this thesis, is centralized and operates using iterations. In one iteration, an agent computes and sends its messages to the connected agents within the graph in a predefined order. The same functionality can be implemented using distributed implementation where each agent computes and communicates an updated message directly after it has received a new (and different) message from one of its neighbors. This results in a computational advantage over the sequential execution of the centralized version since messages would be sent in parallel, yet necessitates the development of functionalities and protocols presently not supported by the simulator.

Another approach for coordination pursued by some researchers is the online adaptation of vehicle route selection, i.e. coordinating between the controllers and the vehicles based on real-time traffic conditions. This enables vehicles to be distributed more wisely over the network thereby effectively avoiding bottlenecks. There are, however, a number of problems with this approach. First, communication is assumed between the controllers and the vehicles. Second, there may be many cases where the structure of the network may limit this approach, i.e. some destinations may be more popular and may only be reached by a limited number of roads. Third, such an approach would require the cooperation of the drivers which might also be limited. Nonetheless, this approach should be considered with future analysis as previous authors [7] have shown that it effectively deals with bottlenecks and some of their results seem promising.

In this thesis, we make a number of simplifying assumptions that essentially make the problem 'easier' than it would be in practice. First, we assume the environment to be fully observable. In practice, this would hardly be the case as we can expect noisy input from sensors as well as noisy communication among agents. Therefore, the methods need to be extended to deal with partial observation. Second, the environment is assumed to be stationary whereas in practice this assumption is not likely to be satisfied. More research is needed to address this problem. One direction would be to allow agents to learn a number of

policies for different (i.e. "stationary") environments and occasionally choose the policy that is most suitable for the conditions at hand based on given criteria. Third, we assume there are no costs involved for communication among agents whereas in practice these costs would need to be taken into account. Fourth, many variables that would influence performance are excluded. Factors that come into play, for example, are irregular and unpredictable incidents like pedestrians, accidents, illegal parking, and weather. Drivers' differing behavior and characteristics are a further source of variance as they influence the choice of routes and other factors such as speed and distances maintained between vehicles. Overall, our simulation does exhibit many of the characteristics that are present in practice but future work should focus on testing the methods in even more realistic environments and determine the extent to which other factors should be considered.

In conclusion, our novel method outperforms prior methods and introduces a more robust and scalable solution, especially for highly saturated, diverse, and increasingly larger networks. This thesis offers an improved approach to a global issue with promise for widespread application and a demand for further analysis and investigation.

Bibliography

[1] Richard S. Sutton, Andrew G. Barto. Reinforcement Learning: An Introduction, MIT Press, 1998.

[2] C. P. Pappis, and E. H. Mamdani. A Fuzzy Logic Controller for a Traffic Junction. In *IEEE Transactions on Systems, Man, and Cybernetics*,(7: pg. 707-717,1977.

[3] S. Chiu. Adaptive Traffic Signal Control Using Fuzzy Logic. in *Proceedings of the IEEE Intelligent Vehicles Symposium*, pg. 98-107, 1992.

[4] J. C. Spall, and D.C. Chin. Traffic-Responsive Signal Timing for System-wide Traffic Control. in *Transportation Research Part C: Emerging Technologies*, 5(3): pg. 153-163, 1997.

[5] L. Xiupin et al. An Approach of Intersection Traffic Signal Control. In *Information and Control*, 30(1): pg. 76-79, 2001.

[6] D. Chaojun, L. Zhiyong, et al. Urban Traffic Signal Timing Optimization Based on Multilayer Chaos Neural Networks Involving Feedback. In *Proceedings of the First International Conference on Natural Computation (ICNC)*, pg. 340-344, 2005.

[7] M. Wiering, Multi-Agent Reinforcement Learning for Traffic Light Control, in *Proc. 17th International Conf. on Machine Learning*,pg. 1151-1158, 2000.

[8] Foy M. D., R. F. Benekohal, D. E. Goldberg. Signal timing determination using genetic algorithms. In *Transportation Research* Record No. 1365, pp. 108-115.

[9] Sun, D., R. F. Benekohal, S. T. Waller. Multiobjective traffic signal timing optimization using non-dominated sorting genetic algorithm. In *Proceedings IEEE Intelligent Vehicles Symposium*, pp. 198-203, 2003.

[10] A.W. Moore and C.G. Atkenson, Prioritized Sweeping: Reinforcement Learning with less data and less time, in *Machine Learning*, 13:103-130, 1993.

[11] B. Bakker, M. Steingrover, R. Schouten, E. Nijhuis and L. Kester, Cooperative multi-agent reinforcement learning of traffic lights. In *Proceedings of the Workshop on Cooperative Multi-Agent Learning, European Conference on Machine Learning*, ECML'05, 2005.

[12] Jelle R. Kok and N. Vlassis. Collaborative Multiagent Reinforcement Learning by Payoff Propagation, in *J. Mach. Learn. Res.*, volume 7, pg. 1789-1828, 2006.

[13] J. Kok, P. J. 't Hoen, , B. Bakker, and N. Vlassis. Utile coordination: Learning interdependencies among cooperative agents. In *Proceedings IEEE Symposium on Computational Intelligence and Games*, pg 29-36, 2005.

[14] N. Vlassis. A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence. Synthesis Lectures in Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2007.

[15] P. Stone, M. Veloso, Multiagent systems: A survey from a machine learning perspective. In *Autonomous Robots*, Volume 8-3, pg. 345-383, 2000.

[16] C. Boutilier, Planning, learning and coordination in multiagent decision processes. In *Proceedings Sixth Conference on Theoretical Aspects of Rationality and Knowledge*, pg. 195-210, 2006.

[17] C. Boutilier, Sequential optimality and coordination in multiagent systems. In *Pro- ceedings 16th International Joint Conference on Artificial Intelligence*, pg. 478-485, 1999.

[18] G. Chalkiadakis, and C. Boutilier, Coordination in multiagent reinforcement learning: A Bayesian approach. In *Proceedings 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*, pg. 709-716, 2003.

[19] C. Guestrin, M.G. Lagoudakis, and R. Parr,Coordinated reinforcement learning. In *Proceedings Nineteenth International Conference on Machine Learning*, pg 227-234, 2002.

[20] N. L. Zhang and D. Poole. Exploiting causal independence in Bayesian network inference. In *Journal of Artificial Intelligence Research*, 5:301328, 1996.

[21] F. R. Kschischang, B. J. Frey, and H.A. Loeliger. Factor graphs and the sum-product algorithm. In *IEEE Transactions on Information Theory*, 47:498519, 2001.

[22] C. Crick and A. Pfeffer. Loopy belief propagation as a basis for communication in sensor networks. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*, 2003.

[23] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. In *Exploring Artificial Intelligence in the New Millennium*, chapter 8, pg. 239269, 2003.

[24] Thorpe, T. L. and Andersson, C.. Traffic light control using sarsa with three state representations. *Technical report*, IBM corporation,1996.

[25] Ma Shoufeng, et al. Agent-based learning control method for urban traffic signal of single intersection. In *Journal of Systems Engineering*,17(6): pg. 526-530, 2002.

[26] B. Abdulhai, et al. Reinforcement Learning for True Adaptive Traffic Signal Control. In *ASCE Journal of Transportation Engineering*, 129(3): pg. 278-285, 2003.

[27] S. Russell and Norvig, Artificial Intelligence: A Modern Approach. Prentice Hall, 2003.

[28] S. Sen and G. Weiss, Learning in multiagent systems. In *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, chapter 6, pg. 259-298. MIT Press, 1999.

[29] C. J. C. H. Watkins, and P. Dayan. Technical note: Q-learning. Machine Learning, 8: 279-292, 1992.