Learning Coordinated Traffic Light Control

Tong Thanh Pham
Lafayette College
phamt@lafayette.edu

Tim Brys
Vrije Universiteit Brussel
timbrys@vub.ac.be

Matthew E. Taylor[*]
Washington State University
taylorm@eecs.wsu.edu

ABSTRACT

Traffic jams and suboptimal traffic flows are ubiquitous in our modern societies, and they create enormous economic losses each year. Delays at traffic lights alone contribute roughly 10 percent of all delays in US traffic. As most traffic light scheduling systems currently in use are static, set up by human experts rather than being adaptive, the interest in machine learning approaches to this problem has increased in recent years. Reinforcement learning approaches are often used in these studies, as they require little pre-existing knowledge about traffic flows. Some distributed constraint optimization approaches have also been used, but focus on cases where the traffic flows are known. This paper presents a preliminary comparison between these two classes of optimization methods in a complex simulator, with the goal of eventually producing real-time algorithms that could be deployed in real-world situations.

Categories and Subject Descriptors

I.2.6 [**Learning**]: Miscellaneous

General Terms

Algorithms, Performance

Keywords

Reinforcement Learning, DCEE, Traffic Optimization

1. INTRODUCTION

In recent years, multiagent systems have been gaining traction as a credible platform for tackling real-world problems. Distributed problem solving often allows handling of an exponential amount of information and variables that might otherwise cripple a centralized approach. Indeed, the need for this flexibility and robustness is no longer theoretical: the increase in human population in the biggest metropolitan areas has led to tremendous stresses on infrastructure, which must either increase in quantity or improve in quality to simply maintain the current quality of traffic flows. The motivation for further development of multiagent techniques is twofold: to manage the rising complexity in handling electronic infrastructure, and to improve performance by replacing or enhancing existing solutions.

[*]Much of this research was done while the third author was at Lafayette College.

In recent years, interest in applying various computational techniques to the problem of improving traffic signal operation has been on the rise. According to recent surveys [6], delays at traffic signals account for up to 10 percent of all traffic delays in the US. Increasing numbers of traffic lights are being built to handle growing vehicle and pedestrian traffic; this problem domain is rich in terms of the number of potential autonomous agents sharing the same finite resources, that influence both other agents and the efficiency of the human traffic flow.

Recent AI approaches to traffic engineering include applying Distributed Constraint Optimization (DCOP) algorithms to this domain [8]. However, the DCOP framework requires that the reward of every action combination be known *a priori*, making it difficult to handle non-stationary traffic distributions. In contrast, our previous work has extended the DCOP framework to the Distributed Coordination of Exploration and Exploitation [17] (DCEE) framework. In order to address the importance of dynamic and unknown rewards, DCEE algorithms take a multiagent approach towards balancing exploiting known good configurations with exploration of novel action combinations to attempt to find better rewards.

Another popular AI approach to traffic problems is to apply Reinforcement Learning [16] (RL) algorithms. Several RL algorithms have previously been applied to the control of traffic lights [10], such as Q-Learning [1, 14], SARSA [18], SCQ-Learning [9], and others [3]. Drawing from these studies, we will build an RL algorithm to benchmark our DCEE methods against.

We evaluate our ideas by implementing DCEE and RL algorithms in a modified version of the traffic simulator from UT-Austin [7]. We limit the scope of our experiments to Manhattan traffic grids. Our objective metrics are the average delay of all vehicles over time and the rate of vehicles passing through the grid.

2. BACKGROUND

This section provides background on the traffic optimization problem, as well as the two approaches used in this paper's experiments.

2.1 Traffic Optimization

Given this problem domain's affinity to an exponential growth in complexity (in terms of the number of possible configurations and eventualities that the agents – traffic lights, commuters – can give rise to), it comes as no surprise that many works have attempted to address improving traffic signal performance. The 1970s saw the development of SCOOT (Split, Cycle and Offset Optimization Technique) in the UK [12]. This system features a central computer system to monitor a series of intersections, attempting to minimize the sum of the average queues and number of vehicle stops. Notably, SCOOT makes use of a model of the traffic flow based on Cyclic Flow Profiles (average one-way flow of vehicles past a fixed point on the road) measured real-time using sensors. While the system has been observed to register a 12 percent improvement over fixed-time systems, it is not readily amendable to scaling due to the need for centralized control. Another related system is SCATS (Sydney Coordinated Adaptive Traffic System), which uses multiple levels of control, segregated by scale: from local, regional, to central. However, grouping signals into subsystems to be managed by higher levels are not done automatically, and thus incurs setup costs for expansion and changes.

Aside from these well known responsive control systems, there also exist adaptive systems such as RHODES [11], which features more involved sensor systems, models, allowing them to do away with explicit cycle length definitions. RHODES also has a hierarchical architecture, with the lowest level control making immediate second-by-second decisions on signal phase and durations. The middle and highest level form a feedback loop with this lowest level to predict demands and flows over longer period of time. Despite this sophistication, the system still requires heavy use of models, which can take time to perfect, as well as requiring the setup of the hierarchy.

When selecting a traffic optimization system to deploy, it is always important to consider what sensors (inputs) are required, what knowledge must be built into the system, how adaptive the system is, and what metrics (outputs) will be optimized. As AI researchers, we are most interested in methods which use low-cost sensors, have minimal knowledge requirements, can quickly adapt to changes in traffic patterns, and can work to optimize many different metrics.

2.2 DCEE

When formulating multi agent problems, there is a spectrum from centralized (one agent gathers all of the information, make a decision, and distributes this decision) to decentralized (all agents have their own local state and do not coordinate with others) decision making. The DCEE framework strikes a balance between these two extremes by allowing agents to form "neighborhoods;" each agent shares information and coordinates with only a limited set of agents. Such shared coordination should improve the total reward relative to each agent disregarding the state of all other agents, while requiring many fewer messages and computational resources than full centralization.

Formally, a DCEE problem [17] consists of:

1. a set of variables, $V = x_1, x_2, ..., x_n$, where $x_i \in D_i$

2. a set of agents, each controlling a variable from V (in the general case, one agent could control multiple variables)

3. an (initially unknown) reward function $f_{ij} : D_i \times D_j \rightarrow R$, which gives the cost of a binary constraint $(x_i \leftarrow d_i, x_j \leftarrow d_j)$, where $d_i \in D_i, d_j \in D_j$

4. a set time horizon $T \in \mathbb{N}$

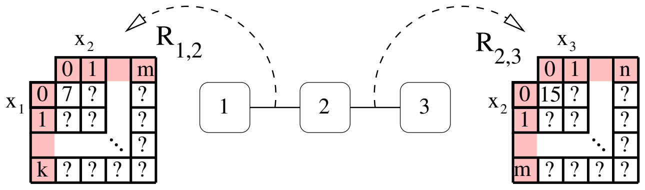


Figure 1: This figure shows an example 3-agent DCEE. Each agent controls one variable and the settings of these three variables determine the reward of the two constraints (and thus the total team reward).

5. a set of assignments of values to variables $A_0, ..., A_T$ to be processed sequentially by the agents. Each assignment A_t is a tuple $(x_{1t} \leftarrow d_{1t}, x_{2t} \leftarrow d_{2t}, ..., x_{nt} \leftarrow d_{nt})$

The goal is to maximize the total reward during the time horizon:

$$R = \sum_{t=0}^{T} \sum_{x_i, x_j \in V} f_{i,j}\left(d_{i,t}, d_{j,t}\right)$$

The simplest cases of DCEE problems make use of binary constraints between pairs of agents (see Figure 1 for an example). As mentioned earlier, the reward function f is initially unknown, and must be empirically estimated by trying out different variable assignments with an agent's neighbors. Communication among agents in a neighborhood is essential so that each agent can build up its mapping of binary constraints to (approximate) rewards for each of its neighbors.

An important factor that shapes coordination among neighboring agents is the concept of *k-movement*, where at most k agents can change their variables simultaneously in a neighborhood every round. Larger k values allows for more joint moves, but sometimes this can decrease total team performance [17].

This paper focuses on the class of static estimation (SE) DCEE algorithms. The *k=1* SE-Optimistic algorithm allows a single agent to change variable(s) per neighborhood. For instance, in Figure 1, if agent 2 changes its variable setting, agents 1 and 3 must remain fixed. Alternatively, if agent 1 changes its variable, agent 2 must remain fixed, but agent 3 could choose to change. The algorithm is *optimistic* in the sense that it estimates that it will receive the maximum reward on every constraint if it picks an unexplored configuration. This results in the behavior that 1) every agent wishes to change configurations on every round, 2) the algorithm will always be exploring the environment (practically speaking, since our domain premise is that there are too many configurations to exhaustively cover during the given time frame), and 3) agents with the worst performance per neighborhood will be allowed to explore. On every *round*, every agent will measure the reward between itself and all of its neighbors. It will then use Algorithm 1 to decide which agent (per neighborhood) can choose a new assignment.

$getMaxGainAndAssignment$ is a function that returns a variable assignment that maximizes the difference between total expected reward across all binary constraints of the agent and its current reward. For the case of SE-Optimistic algorithms, this will always be an unexplored position, because the agents are optimistic that such positions will have a very high potential reward. In this manner, for each neigh-

Algorithm 1 K-1 Algorithm Pseudocode

```
for each neighbor i do
    Send variable assignment, reward matrices to i
    Receive variable assignment, reward matrices from i
end for
$g, a \leftarrow getMaxGainAndAssignment()$
Send Bid $g$ to all neighbors
Receive Bids from all neighbors
$G \leftarrow max(\textbf{Bids})$
if g > G then
    UpdateAssignment($a$)
end if
```

borhood, the agent with the worst performance across its binary constraints with neighbors will get to change.

For the *k=2* SE-Optimistic algorithm, each neighborhood can allow up to two agents performing joint movement to change configurations (see Algorithm 2). Being an optimistic algorithm, it again assumes that any unexplored binary constraint will yield the maximum reward. Communication is more involved as the agents must first look around their neighborhood to see with whom they will likely make the strongest bidding pair (by assuming that the neighbor's neighbors would not change while evaluating the combined rewards). Each agent then sends an *OfferPair* message to their prospective partner, and those pairs that successfully match each other as the best in their vicinity will use that projected gain to compete with their respective neighbors' bids. Should an agent not get a reply from her desired partner, though, she would have to evaluate her potential reward as would an agent in the *k=1* scenario, then bid using that value.

Algorithm 2 K-2 Algorithm Pseudocode

```
for each neighbor $i$ do
    Send variable assignment, reward matrices to $i$
    Receive variable assignment, reward matrices from $i$
end for

$g, p, a \leftarrow getMaxGainAndAssignmentForPair()$
    Send OfferPair to agent $p$
$doPair \leftarrow False$
for all OfferPair received do
    if agent requesting to pair is $p$ then
        Send Accept to agent $p$
        $doPair \leftarrow True$
    end if
end for
Attempt to receive Accept message
if (not received Accept message from $p$)
or (not doPair) then
    $p \leftarrow \emptyset$
    $g, a \leftarrow getMaxGainAndAssignment()$
end if
Send Bid($g$,$p$) to all neighbors
Receive Bids from neighbors except $p$
$G \leftarrow max(\textbf{Bids})$
if g > G then
    $changing \leftarrow True$
else
    $changing \leftarrow False$
    if $p \neq \emptyset$ then
        Send ProhibitVariableChange to $p$
    end if
end if
Receive messages from neighbors
if changing and $p \neq \emptyset$ then
    if received ProhibitVariableChange from $p$ then
        $changing \leftarrow False$
    end if
end if
if changing then UpdateAssignment($a$)
end if
```

2.3 Reinforcement Learning

The second approach we investigate in this paper is Reinforcement Learning. Reinforcement learning (RL) [16] is a machine learning paradigm that is aimed at learning (near-) optimal agent behavior through interactions with an environment. This environment is typically formulated as a Markov decision process (MDP), which is a tuple $\langle S, A, T, R \rangle$, where S is the set of possible states of the environment, A the possible actions, T the dynamics of the environment (specified as state transition probabilities), and R the reward function (which attributes a utility to state transitions).

2.3.1 SARSA

One popular RL algorithm is SARSA [13]. It is a model-free method that estimates an action-value function, $Q(s, a)$, measuring the expected return of taking action a in state s from experience. After each state transition, it updates its estimates according to:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

r_t represents the reward at time t for transitioning from state s_t to s_{t+1}. a_t is the action that caused that transition, and a_{t+1} is the action that will be taken in state s_{t+1}. Under certain conditions [15], these Q-value estimates converge to the true Q-values in the limit, and an optimal policy can be followed by taking the action with the highest Q-value in every state.

The problem considered in this paper is a multiagent system, and thus the agents will be learning in the presence of other agents. This renders the problem non-stationary for learning agents that do not coordinate or take each other into account, and proofs guaranteeing convergence to the optimal policy of single-agent algorithms are invalidated. Still, independent learners often perform well, notwithstanding their lack of coordination [4], and therefore we will take this approach, keeping the algorithm as simple as possible.

2.3.2 Tile Coding

SARSA, and other temporal difference methods, are often implemented with look-up tables for the Q-values. However, when applying these algorithms to problems with large state and action spaces, or even continuous ones, memory requirements become an issue. Furthermore, an agent would need to visit every state-action pair multiple times to account for a potentially stochastic environment. Therefore, generalization techniques are a necessity. Tile Coding [2, 16] is one form of function approximation where the state-space is partitioned multiple times, i.e., into multiple tilings. Each tiling divides the space into a number of disjoint sets, or tiles, and when a state is visited, it is mapped to exactly one tile in each tiling. Instead of directly estimating the Q-value of each state s and action a, the Q-function is decomposed into a sum of weights for each tile:

$$Q(s,a) = \sum_{i=1}^{n} b_i(s,a) w_i$$

where n is the number of tiles and b_i is 1 or 0, depending on whether the tile is activated by state s and action a. Whenever a regular Q-value update would be executed, the weight of each activated tile is updated instead. Mapping states to different tilings and decomposing the Q-function into a linear combination of tilings allows the generalization of experience between states that are similar. The more tiles two states share, the more generalization will occur.

3. EXPERIMENTAL SETUP

This section introduces the traffic simulator used in experiments. It also details the setup used for DCEE and SARSA experiments.

3.1 The AIM Simulator

The Autonomous Intersection Management (AIM) simulator is developed by the Learning Agents Research Group at the University of Texas at Austin. A microscopic traffic simulator, AIM mainly supports a Manhattan topology of North-South and East-West multi-lane roads joined by a number of intersections. The primary vision of this project is to investigate a future where autonomous vehicles and intelligent traffic intersections would eliminate the need for traffic lights altogether. As part of their benchmark, however, the team has also implemented ordinary traffic lights into the system. For the purpose of our study, we made exclusive use of this benchmark feature as the backdrop against which traffic lights will be tested. Even though the traffic-light system in AIM is implemented using the same message-passing foundation, the inherited efficiency in which vehicles navigate an intersection, accelerate, and decelerate, as well as subtle details including variable vehicle sizes, have convinced us that this is an attractive simulator for our purpose. Figure 2 shows a screenshot of a 2×2 intersection setup in the AIM simulator.

Our setup involves single-lane two-way streets forming a 2×2 matrix of four intersections. Thus, each road has two spawn points where new vehicles can enter the system. Each spawn point uses a Poisson process to determine the spawn time for the next vehicles; all spawn points share the same rate parameter, λ. Because each direction has one lane, a newly spawned vehicle will not pass other cars, nor does it perform U-turns. Upon creation, each vehicle is assigned a destination, uniformly chosen among the seven possible exit points of the system; the vehicle then follows the shortest path to reach its designated exit point.

3.2 DCEE Experiments

In the setup for the DCEE experiments, we make use of a special traffic light signal scheme that relies on the AIM simulator's definition of an "active phase." Two parameters specify precisely such an active phase: the green offset and green duration in seconds (see Figure 3). For simplicity, the majority of our experiments will have the active phase length fixed at 60 seconds. We then associate each intersection

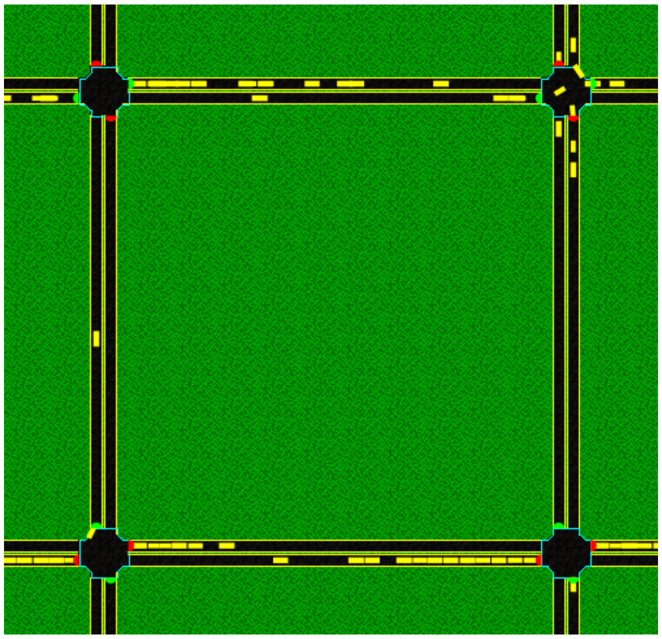


Figure 2: A screenshot of a four-intersection layout in AIM with two-way single-lane traffic.

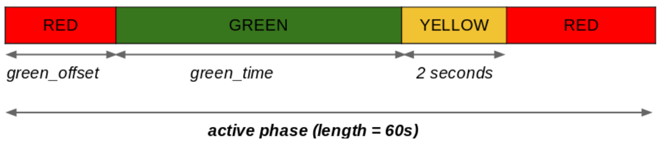
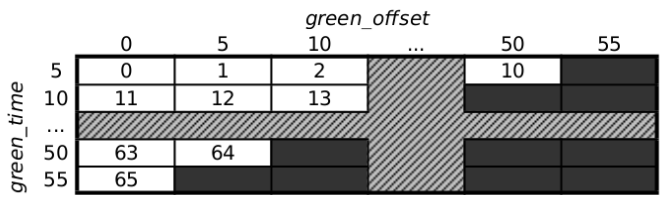


Figure 3: Traffic light configuration that makes up an "active phase" in the simulator.

with a DCEE agent, letting each agent control exactly one variable — its *signal scheme index*. This index enumerates all possible traffic signal configurations, running from 0 to 65 (see Figure 4). Thus, index 0 maps to the tuple (0, 5), which is an active phase with no leading red, and five seconds of green time (followed by two seconds of yellow and 53 seconds of red). To translate the next index, we attempt to increase green time by a five-second interval, while keeping the total active phase length. Should this not be possible, we instead increase the offset by five seconds, and reset green time to five seconds. This results in a triangular translation table, stopping at (55,5), for a total of 66 possible combinations.

Once the "active phase" has been determined, the entire signal layout for each direction (North-South, East-West) will be specified as shown in Figure 5. Note that at any moment, only one direction is considered to be active (running the active phase signal scheme), and the other direction is inactive, running the complementary signal scheme.

Agents evaluate the rewards of binary constraints with a neighbor by measuring the average travel time of a fixed number of cars traveling on the stretch of road connecting two agents. The total team reward at each round is then the weighted average of this average travel time, scaled by the volume of traffic on each stretch of road linking a pair of agents. For our static estimation agents, we set the unexplored reward to be 0 seconds, an unattainable value for



green_time \ green_offset	0	5	10	...	50	55
5	0	1	2	▨	10	■
10	11	12	13	▨	■	■
...	▨	▨	▨	▨	▨	▨
50	63	64	■	▨	■	■
55	65	■	■	▨	■	■

Figure 4: The signal scheme index for each DCEE agent, and its corresponding (*green_offset*, *green_time*) value, when active phase length is fixed at 60 seconds. Note that *green_offset* and *green_time* increase at five-second intervals.

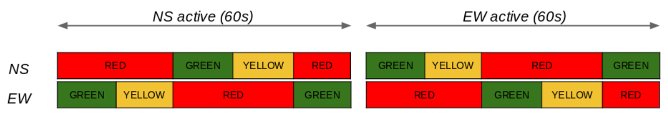


Figure 5: The full signal scheme for an intersection, dependent on a given active phase. From left to right is the direction of time: the calculated active phase is active for North-South in the first 60 seconds, before switching to East-West in the next 60 seconds. The whole signal scheme repeats after 120 seconds total.

travel time. To allow numerically higher reward value to be more desirable, we negate the actual measurements. Average travel time serves as a reasonable metric to optimize, as it correlates with other commonly used metrics that gauge traffic light performance, such as queue length and throughput. Thus, by letting the agents optimize for higher rewards in this context, we are letting the intersections work toward lower average travel time along the lanes between them, thus improving traffic light performance. After the agents have changed their signal schemes, we allocate a cool-down period of 600 seconds to allow the effect of the new signal schemes to have an impact on the traffic condition. Only after this cool-down period do the agents begin to evaluate rewards.

3.3 SARSA Experiments

We apply SARSA with tile coding to the traffic light problem by giving control of each intersection to a SARSA agent. The SARSA setup is as follows:

Every two seconds, the agents are presented with only two possible actions:

1. Do nothing, or
2. Change the green light to the other direction (e.g., from North-South to East-West).

Note that when changing the lights, a short period of yellow occurs, stopping traffic in all directions.[1]

The state space is made up of three variables:

1. The number of seconds since the most recent light change.
2. The number of seconds since the second-to-last light change.
3. The ratio between the accumulated waiting times in both directions.

[1]In preliminary experiments, we had not implemented the yellow light. Because the AIM simulation does not allow cars to crash into each other, the learned (and also optimal) policy kept switching the green light between the different directions very quickly, allowing cars traveling North-South into the intersection at the same time as cars traveling East-West!

The first variable ensures that the agents will be able to learn a repeated schedule of fixed length. The inclusion of the second state variable allows the agents to learn asymmetrical schedules, which are useful when the traffic load is higher in one direction than in the other. The third state variable allows the agent to adapt to the slight variations that occur in the general traffic pattern, by conditioning learning on the traffic load in each direction. This traffic load is measured by counting the time each car has lost when approaching the intersection (actual time - optimal time at maximum speed allowed), summed over all the cars. Technically, the variable is a bit more than simply the ratio. We define $\text{wait}_{\text{green}}$ to be the accumulated waiting time for all cars in the green direction, and wait_{red} to be the accumulated waiting time for all cars in the red direction. While a single ratio $\frac{\text{wait}_{\text{green}}}{\text{wait}_{\text{red}}}$ behaves as needed when $\text{wait}_{\text{green}}$ is larger than wait_{red}, for all combinations where wait_{red} is larger, values are $\in (0, 1)$. This asymmetry makes a simple ratio not suited for linear tile-coding, and having two variables ($\frac{\text{wait}_{\text{green}}}{\text{wait}_{\text{red}}}$ and $\frac{\text{wait}_{\text{red}}}{\text{wait}_{\text{green}}}$) would address this problem, but increases the size of the state space unnecessarily, as we can encode this information in one variable (see Algorithm 3).

Algorithm 3 Calculating the ratio between accumulated waiting times in both directions

```
if $\text{wait}_{\text{green}} > \text{wait}_{\text{red}}$ then
    $\text{var}_3 = log(\frac{\text{wait}_{\text{green}}}{\text{wait}_{\text{red}}})$
else
    $\text{var}_3 = -log(\frac{\text{wait}_{\text{red}}}{\text{wait}_{\text{green}}})$
end if
```

If we ensure that $\text{wait}_{\text{green}}$ and wait_{red} are at least one, then var_3 will be 0 when accumulated waiting times in both directions are exactly the same, and positive or negative when the waiting time is greater in the direction that has green or red light respectively. This ensures that, as opposed to using a simple ratio, this variable is symmetric around 0 and allows easy generalization around 0 with linear tile coding. The logarithm captures small differences in traffic when the load is very similar in both directions, while compressing the differences when the traffic is asymmetric. Tile size is 100 for the first two variables, and 1 for the third variable; 32 tilings are used over the three variables.

Note that including this accumulated waiting time is not unrealistic. Modern road infrastructures include cameras and other sensors that are able to keep track of the traffic. Furthermore, the use of floating car data is increasing. This is a method used to calculate traffic speed based on signals from cellular and GPS devices in cars, providing real-time information on traffic.

The reward that agents receive is the negation of the total accumulated waiting time for all cars approaching their intersection. Maximizing this reward will reduce the average waiting time. Although the aim in the traffic problem is to optimize the whole traffic system, the reward agents receive is only local. Calculating the global reward by broadcasting the local rewards is costly, and furthermore, employing global reward introduces a credit assignment problem: was the change in reward the effect of my own action, or that of another agent. This may in fact decrease team performance [5].

The action-selection strategy used is ϵ-greedy: when an action must be selected in state s, the action with the highest estimated Q-value is selected with probability $1 - \epsilon$, and with probability ϵ a different action is randomly selected. Furthermore, ϵ is decreased over time ($\epsilon = 0.9998^t$, t the t-th decision step, which occurs every 2 seconds), to increase exploitation of the acquired knowledge. The discounting factor γ, as well as the replacing eligibility traces' decay λ, is set to 0.9, .

3.4 Remarks

The setup for the DCEE and RL approaches differs. First, their action spaces are different. Second, RL incorporates state which DCEE does not. Third, DCEE algorithms can choose among 60s long schedules, while the RL approach involves decisions every 2s. Direct comparisons between the approaches's action selection will be inexact. However, it is not our purpose to evaluate these algorithms in that respect, but rather, for each approach we chose the most fitting setup for the traffic problem, and want to compare these algorithms' performance at their best. The DCEE setup is built to quickly learn good policies matching the general traffic pattern, while the RL setup is built to help SARSA adapt to the general traffic pattern, as well as to short-term variations in the pattern, which requires more experience.

4. RESULTS

This section describes the empirical comparison between SARSA and the DCEE algorithms described in previous sections. We run the simulator with a 2×2 grid for $90,000$ seconds, in which we measure the agents' performance in terms of the average delay per car and the total throughput. These metrics are the two most important measures used by engineers to compare control policies. While throughput indicates how many cars a system can process in a given time period, average delay gives an indication of how the system's performance affects individual cars' travel time. As we will see in the experiments, different control policies can have approximately the same throughput, yet with largely differing delays.

Before the agents are allowed to learn, we implemented a $7,500$ second warm-up period, in which the agents are forced to select random actions, to allow the system to fill with cars and reach an equilibrium. This warm-up period is not graphed in the figures.

We will evaluate two settings with different traffic loads. The first experiment will be parametrized to generate an average of 10 cars per minute, per entry link. In the second experiment, an average of 30 cars will be generated each minute, a much higher traffic level. Figures 6 and 7 visualize the performance of SARSA, K1 and K2, as well as two control algorithms (random RL and random DCEE, i.e., the RL and DCEE setups but with random action-selection) on the low traffic setting. Experiments are averaged over 100 runs for statistical significance, and errorbars show one standard deviation. We can see that the situation described before occurs in this low-traffic setting. It is not necessary

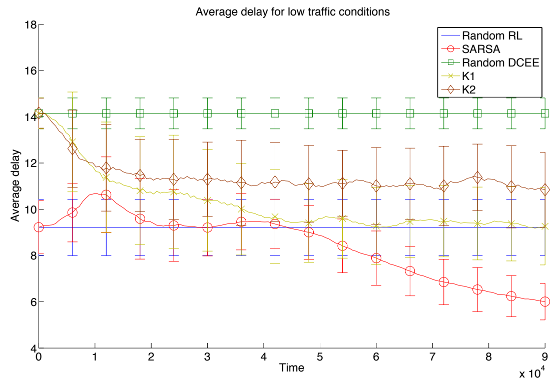


Figure 6: Average delay for a low traffic level (10 cars spawned per minute at each entrance).

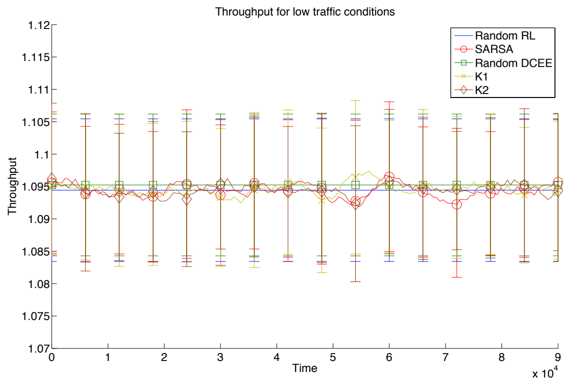


Figure 7: Throughput for a low traffic level (10 cars spawned per minute at each entrance).

to optimize the throughput of cars in this system as there are so few of them, but the algorithms can clearly improve the average delay. Notably, SARSA performs much better than K1 and K2, as it is not limited to the (too long for low traffic) 60 second schedules, but incorporates the actual traffic situation in its state, and adapts to that. K1 can only approach the level of random RL, while K2 performs even worse than K1, even though it coordinates more. This apparent discrepancy is due to a phenomenon coined *the team uncertainty penalty* [17], in which agents with few neighbors actually achieve higher performance with lower levels of coordination (e.g., K1), while agents with many neighbors can achieve higher performance with higher levels of coordination (e.g., K2).

Let us now consider the higher traffic setting experiment, visualized in Figures 8 and 9. Both throughput and average delay can be greatly improved upon in this setting. SARSA starts with very poor performance for both measures compared to DCEE algorithms, but is able to learn a great deal, and in the end it outperforms both DCEE algorithms as these are stateless and can not adapt to the local, ephemeral fluctuations in the traffic pattern. The power of the DCEE framework is demonstrated by the performance of K1 and K2 as measured by average delay. Between two adjacent changes in signal schemes, the agents spend 600

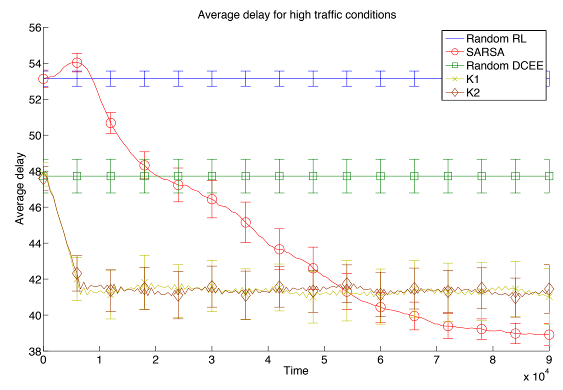


Figure 8: Average delay for a high traffic level (30 cars spawned per minute at each entrance)

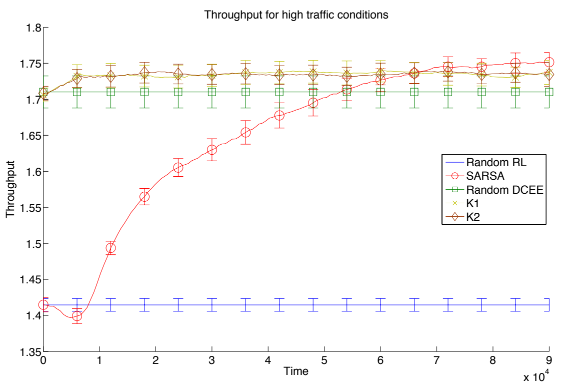


Figure 9: Throughput for a high traffic level (30 cars spawned per minute at each entrance)

seconds for the cool-down period, and about 600 seconds for collecting data to measure average travel times. Thus, they are both able to significantly optimize the traffic flow of the system in only eight decision steps (around the 10,000 second mark), with $66^4 = 1.9 \times 10^7$ possible schedule combinations to choose from. Furthermore, throughput is only slightly improved as it already is at a near-optimal level for any possible schedule, yet SARSA is again able to outperform the DCEE algorithms in the long run. Lastly, it is interesting to note that in both experiments, SARSA first decreases its performance as compared to random, before it is able to improve again.

5. DISCUSSION

The results described in the previous section show that the DCEE algorithms are particularly well suited at quickly finding good solutions, while the RL approach proved to be more adaptive in the long run. Besides showing that more coordination can be detrimental to performance, the results suggest we try giving the DCEE algorithms the option to choose schedules with varying lengths, to make better adaptation to the traffic level possible. For example, shorter schedules can be beneficial in low traffic conditions. Figures 10 and 11 show the delay and throughput for K1 choosing from various 60 second schedules as before, and K1 choosing from a larger range of schedules with lengths varying from 10 seconds to 100 seconds. It is clear from the graphs that randomly choosing among the much larger number of schedules (16,370 available signal scheme choices instead of only 66) yields worse performance, as indicated by the relative initial performance levels. K1 shows more learning for the variable schedules, but does not converge to performance as good as that with more limited options. Still, it also converges after only eight decision steps (one every 1200 seconds), and reaches impressive performance given the huge number of possible combinations $16370^4 = 7.18 \times 10^{16}$). One possible reason why the variable phase length version never quite converges to the same performance level as that of the fixed version is due to the large number of nonoptimal signal schemes. Since the agents choose new positions to explore randomly, they are more likely to encounter solutions that are not as good as those found by restricting to a 60 second active phase length. However, it should be the case that the best performance reached by the former is higher than that by the latter.

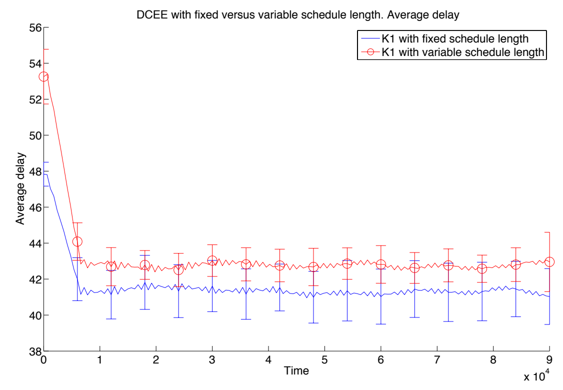


Figure 10: This graph shows the average delay for a high traffic level, comparing K1 with fixed 60 second schedules and K1 with variable length schedules.

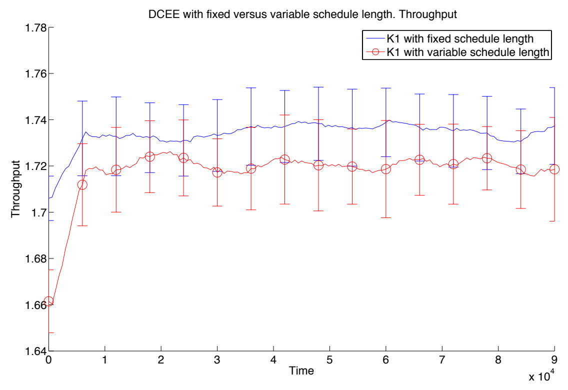


Figure 11: This graph shows the throughput for a high traffic level, comparing K1 with fixed 60 second schedules and K1 with variable length schedules.

6. CONCLUSIONS AND FUTURE WORK

The DCEE framework was specifically developed to address real-world situations where problems need to be solved in a distributed way, and on-line performance must quickly reach high quality. This forces the solvers to strike a tight balance between exploring new actions and exploiting known good actions. In this paper, we applied DCEE algorithms to the problem of coordinating traffic light control, and showed that, compared to an algorithm from the popular Reinforcement Learning (RL) approach, they are able to very quickly achieve a large improvement in performance. Also, our previous results concerning the team uncertainty penalty are confirmed in this setting [17], showing again that more coordination among agents is not necessarily beneficial.

The main advantage of the RL approach used in this paper is its ability to incorporate relevant state-information that allows the agent to better adapt to the problem. This allowed SARSA to outperform the DCEE algorithms, although requiring much more experience to reach a similar quality of performance.

Future work will extend this work in several ways. First, we are currently implementing isolated traffic signal optimization, a benchmark used by civil engineers that involves analytically calculating traffic schedules based on given traffic flow levels. Second, we intend to implement the work from [9], which can be considered to be the state-of-the art RL approach. Benchmarking our current algorithms against these two new ones will allow us to better evaluate the importance of coordination, showing how far off the performance of our current approaches is from that of idealized controllers, and potentially inspire further multiagent learning algorithms. Third, we plan to implement more advanced DCEE algorithms, in particular the Balanced Exploration family, which should enable better exploitation behaviors. Fourth, we are investigating a multi-objective optimization approach, in order to optimize both delay and throughput metrics simultaneously. Recent work on multi-objective reinforcement learning is promising for this [19].

7. ACKNOWLEDGMENTS

Tim Brys is funded by a Ph.D grant of the Research Foundation-Flanders (FWO), and performed a research visit to Prof. Matthew E. Taylor at Lafayette College, funded by a *Short Stay Abroad* grant also from the FWO. This work was supported in part by NSF IIS-1149917.

8. REFERENCES

[1] B. Abdulhai, R. Pringle, and G. Karakoulas. Reinforcement learning for true adaptive traffic signal control. *Journal of Transportation Engineering*, 129(3):278–285, 2003.

[2] J. Albus. *Brains, behavior and robotics*. McGraw-Hill, Inc., 1981.

[3] 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*, volume 5, page 65, 2005.

[4] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(2):156–172, 2008.

[5] Y.-H. Chang, T. Ho, and L. P. Kaelbling. All learning is local: Multi-agent learning in global reward games. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, Cambridge, MA, 2004. MIT Press.

[6] N. T. O. Coalition. National traffic signal report card, executive summary. http://www.ite.org/reportcard/ExecSummary.pdf, 2012.

[7] K. Dresner and P. Stone. A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence Research*, 31:591–656, Mar. 2008.

[8] R. Junges and A. Bazzan. Evaluating the performance of dcop algorithms in a real world, dynamic problem. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 599–606. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[9] L. Kuyer, S. Whiteson, B. Bakker, and N. Vlassis. Multiagent reinforcement learning for urban traffic control using coordination graphs. *Machine Learning and Knowledge Discovery in Databases*, pages 656–671, 2008.

[10] Z. Liu. A survey of intelligence methods in urban traffic signal control. *IJCSNS International Journal of Computer Science and Network Security*, 7(7):105–112, 2007.

[11] P. Mirchandani and F.-Y. Wang. Rhodes to intelligent transportation systems. *Intelligent Systems, IEEE*, 20(1):10 – 15, jan.-feb. 2005.

[12] D. Robertson and R. Bretherton. Optimizing networks of traffic signals in real time-the scoot method. *Vehicular Technology, IEEE Transactions on*, 40(1):11 –15, feb 1991.

[13] G. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering, 1994.

[14] M. Shoufeng, L. Ying, and L. Bao. Agent-based learning control method for urban traffic signal of single intersection. *Journal of Systems Engineering*, 17(6):526–530, 2002.

[15] S. Singh, T. Jaakkola, M. Littman, and C. Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308, 2000.

[16] R. Sutton and A. Barto. *Reinforcement learning: An introduction*, volume 1. Cambridge Univ Press, 1998.

[17] M. E. Taylor, M. Jain, P. Tandon, M. Yokoo, and M. Tambe. : Distributed on-line multi-agent optimization under uncertainty: Balancing exploration and exploitation. *Advances in Complex Systems (ACS) 14(03)*, pages 471–528, 2011.

[18] T. Thorpe and C. Andersson. Vehicle traffic light control using sarsa. 1997.

[19] K. Van Moffaert, M. M. Drugan, and A. Nowé. Scalarized multi-objective reinforcement learning: Novel design techniques. In *Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*. IEEE, 2013.