Application of Cooperative Q-Learning to find Optimum Path in Maze Environment

Sebastian Eger
School of Electrical and
Computer Engineering
Technical University of Munich
Munich, Germany
Email: sebastian.eger@tum.de

Martin Knopp
School of Electrical and
Computer Engineering
Technical University of Munich
Munich, Germany
Email: martin.knopp@tum.de

***Abstract*—Cooperation between agents is getting more and more important in multi agent systems.**

Cooperation especially helps in situations where agents have very different levels of learning experience. Particularly inexperienced agents benefit from the learning results of more experienced agents. However, also the more experienced agents can benefit from cooperation by, for example, learning the errors of the other agents and then being able to avoid them.

Current research focuses on the hunter-prey problem to evaluate their multi-agent reinforcement algorithms. However, in this paper we want to investigate if it is also possible to use the maze problem. The task is to find the shortest path from from start to goal.

We show that cooperation reduces the required number of steps and the resulting path length. However we also discovered that inequalities in the experience of the agents do not play an important role.

I. Introduction

In nature nearly every living organism is able to do and does cooperation. Even bacteria are doing a form of cooperation with amino acids [1]. Cooperation plays a central role in the evolution of life and without mankind would not have achieved today's technology. It is consequently logical that we want to use cooperation in systems where multiple individuals (e.g. robots) are used to solve one common problem. The main idea behind cooperation in a multi-agent system (MAS) is that agents which achieve rather bad results can learn from agents who achieve better results to improve themselves. However this does not mean cooperation has to be unidirectional. For instance, the mistakes of students can provide useful information for teachers. For evaluation often a simulation with a specific problem is set up. In most cases this is the hunter-prey problem where the agents have to catch a randomly moving target. For example, Ming Tan [3] and Majid Ahmadabadi et al. [**?**] used the hunter-prey problem to evaluate their cooperative learning approaches.

In this paper, however, we want to use a Multi-Agent System to find the shortest path in a maze. We especially focus on the competition between an Individual-Learning MAS and a Cooperative-Learning MAS. Our agents use One-Step Q-Learning [8] reinforcement learning. For each step an agent takes in the maze he gets a reward. Hence, it incrementally learns a decision policy to find the shortest path in the maze. This is done by repeatedly solving the maze.

The cooperation between our agents is based on the Weighted Strategy Sharing method introduced by Ahmadabdi et al. [2]. Here, every agent is assigned to an expertness value which represents its value of knowledge. We extend the expertness measurement by introducing *Needed Steps* and *Distance To Goal*. These measurements are especially applicable for the maze problem.

We test different cooperation types like *Learning After Steps* and *Static Q-Learning* and compare them with each other.

II. Related Work

Ming Tan [3] examined whether cooperative agents outperform independent agents. He showed that cooperation among agents is beneficial although they may learn slowly in the beginning.

Micheal Littman [4] used two agents with diametrically opposed goals to compare Q-Learning and minimax-Q algorithm. He used a game of soccer for evaluation.

Solving a maze with Q-Learning was done by D. Osmankovic and S. Konjicija [10] in MATLAB. However this was only done for a single agent system.

Luca M. Gambardella [6] used Ant-Q, a reinforcement learning approach to solve the traveling salesman problem. He has shown that Ant-Q finds very good, often optimal solutions for this problem.

Y. Shoham and R. Powers [5] discussed critically if multi-agent reinforcement learning is flawless. They reviewed four research agendas on multi-agent reinforcement learning and identified an agenda which they believe is the most promising one.

M. Wiering [7] used multi-agent reinforcement algorithms for learning traffic light controllers. His results showed that optimizing driving policies is very useful and can outperform a number of non-adaptable systems.

R. Derden, N. Friedman and S. Russell [9] extended the basic Q-Learning algorithm by maintaining and propagating probability distributions over the Q-values. This helps to balance exploration and exploitation.

III. Q-Learning

The one-step Q-Learning algorithm is one of the more widely used online learning methods in robotics [11]. The main task is to find a policy which maximizes the reward per action for solving a problem. The problem is modeled as a finite state, Markov decision process [3].

Q-Learning consists of a matrix Q, a set of actions A and a set of states S. Each action $a \in A$ leads from the current state s_t to a new state s_{t+1}. After executing an action, the agent receives a reward $r \in R$ depending on his new state.

The main update formula for Q-Learning is:

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma * \max_a Q(s,a) - Q(s,a)] \quad (1)$$

α is the learn rate and γ is the discount factor. Each time the agent executes an action it updates the Q-matrix with the received reward. To get an action for the current state, we have to find a balance between exploration and exploitation. [9].

One possible approach is the Boltzmann exploration [3]. The probability for an action a in state s is given by:

$$P(a|s) = \frac{e^{Q(s,a)/\tau}}{\sum_a e^{Q(s,a)/\tau}} \quad (2)$$

where $\tau > 0$ is the temperature specifying how randomly values should be chosen. Values of $\tau >> 0$ induce that each action has the same probability to get selected. Furthermore, the value of τ has to be set in regard to the rewards. However, by reducing the temperature, the highest-valued actions are more likely to be chosen and, in the limit $\tau \to 0$, the best action (highest Q-value) is always chosen.

This leads us to another action selection method: the epsilon-greedy method.

For this approach we explicitly differ between exploration and exploitation. Each step has a probability ϵ to do exploration (choosing a random action). Otherwise the agent does exploitation and it chooses the action with the highest Q-value:

$$a(s_t) = max_a Q(s_t, a) \quad (3)$$

Additionally, it is reasonable to reduce the exploration rate ϵ over trials or steps. By exploring the state space Q is improved gradually.

With both action selection methods there is a chance that the action cannot be executed because it would lead to a forbidden state (wall or cell occupied by agent). In that case the agent has to stay at his position, update the Q-matrix (wall reward or no reward when a robot is blocking the cell) and select a new action out of the remaining ones. This is repeated till he can move or no more actions are available. If no action can be executed he has to stay at his current position.

IV. Cooperative Q-Learning

In our case, each agent in the MAS has its private Q-matrix which it updates with the rewards the agent receives by executing actions. Additionally, an agent can use the Q-matrix to select the optimal action for its current state.

However, since in the first trials the values of the Q-matrices are still nearly the same, at this time actions are chosen randomly. By doing more and more trials the Q-matrix gets filled and the optimal policy is emerged.

Therefore, there is a chance that some agents need many steps to find the goal. However it is also possible that some agents are 'lucky' and reach the goal rather fast. This can lead to considerable differences in the values of the Q-matrices.

With Cooperative Q-Learning we want to allow agents to exchange their Q-matrix so that they can learn from each other to increase learning speed and reduce the length of the located path.

The simplest method for Q-Learning would be to just take the average of the sum of all Q-matrices. This method however makes no distinctions according to the experience of the agents.

It is logical to learn preferentially from agents which discovered a relative short path to the goal. However, also agents who needed relatively many steps to the goal could have some useful information, e.g. they probably discovered a greater area of the maze

For this reason, we want to focus on the Weighted Strategy Sharing method introduced by M.H Ahmadabdi et al. [2] in this paper. With this method each agent possesses a private expertness which is a measurement for it's experience or value of knowledge. The expertness can be computed by different conceivable measurements. However most of the measurements depend on the rewards of the agent. Measurements we used in our experiments are introduced later in this section.

The WSS-method is divided into an individual-learning phase and a cooperative-learning phase. In the individual-learning phase, each agent learns for itself by exploring the maze and updating its own Q-matrix. In the cooperative-learning phase, each agent first computes a weight for every other agent depending on its expertness and the other agents expertness. After that they all exchange their weighted Q matrices. The weights are computed by using a weighting-mechanism. As with the expertness measurement, two weighting-mechanism are introduced later in this section. The cooperative phase will be carried out when the *cooperation time* is reached.

Since in our case it is possible that multiple agents are running at the same time, we differ between two types of cooperation time: *Learning After Trials (LAT)* and *Learning After Steps (LAS)*.

In case that LAT is used the cooperative-learning phase is started when a certain number of trails are processed. Therefore cooperative learning has to be done between two trials.

In contrast, when using LAS, cooperative-learning is done during a trial after a certain number of simulation steps are processed.

Algorithm 1 Cooperative Q-Learning

```
if Individual Learning then
    if CL was done in step/trial before then
        Q = Q_coop
    end if
    s_t = GetCurState
    a_t = GetAction(s_t)
    if CheckAction(a_t) then
        s_{t+1} = GetNextState(a_t)
        r_t = GetReward(s_{t+1})
        e_i = GetEstimate(s_{t+1})
        UpdateQ(s_t, e_t, r_t)
        DoAction(a_t)
    else
        r_t = RewardWall
        e_t = 0
        UpdateQ(s_t, e_t, r_t)
    end if
else if Cooperative Learning then
    for all agents i do
        E_i = GetExpertness(i)
    end for
    W = GetWeights(E)
    Q_coop = GetQ(W)
end if
```

Depending on the cooperation time some expertness measurements are applicable or not.
Algorithm 1 shows the main Q-Learning algorithm used in our experiments.

A. Expertness measures

Expertness is a distinct measurement which represents the experience or value of knowledge of each agent. The expertness is not a fixed value and can change over steps and trials. In this section we want to look at four different measurements we used in our experiments.

1) **Absolute**
This expertness measure is the sum over the absolute Q-Learning rewards. It also considers wall rewards as a sign of experience. This expertness measure can be used for LAT and LAS cooperation.

$$E = \sum_{t=1}^{now} |r_t| \tag{4}$$

2) **Covered Steps**
This expertness measure is computed by summing over all executed steps. As the previous measurement, it is always positive but doesn't consider wall and goal rewards. Consequently, the agent which made the most steps gets the highest expertness value. Setting the agents to different speeds leads to high differences in the expertness during a trial.
This expertness measure can be used for LAT and LAS cooperation.

$$E = \sum_{t=1}^{now} t \tag{5}$$

3) **Inverse Covered Steps**
This expertness measure is computed by inverting sum over all executed steps. Therefore, it has the same properties as the Covered Steps measure only that agents which made the least steps are considered as the most experienced agents.

$$E = (\sum_{t=1}^{now} t)^{-1} \tag{6}$$

4) **Distance To Goal**
This expertness measure is calculated by inverting the agent's distance to the goal. The distance is calculated by taking the Manhattan distance between the agent's current position and the goal position. If multiple goals are available, the nearest goal is chosen. Here, the agent which is closest to the goal is considered as the most experienced agent. This expertness norm can only be used for LAS cooperation since after each trial all agents have the same experience.

$$E = |pos(agent) - pos(goal)|^{-1} \tag{7}$$

B. Weighting-mechanism

1) Learn From All (LFA)
If we assume every agent has some useful information, one simple method is using following formula

$$W_{ij} = \frac{e_j}{\sum e} \tag{8}$$

Here the agent *i* assigns a weight to every other agent *j* by dividing its own expertness through the sum of all expertness. After doing cooperation, every agent has the same Q-matrix.

2) Learn From Experts (LFE)
We only learn from the agents who have the highest expertness. This reduces the amount of communication required. We use following formula:

$$W_{ij} = \begin{cases} 1 - \alpha_i & \text{if } i = j \\ \alpha_i \frac{e_j - e_i}{\sum_{k=1}^{n}(e_k - e_i)} & \text{if } e_j > e_i \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

where α_i is the impressibility factor.

V. Simulation

To test if Cooperative Q-Learning is beneficial in terms of finding the shortest path in a maze environment, we created a simulation. The simulation was designed in respect of a high modifiability to test different configurations and parameters.

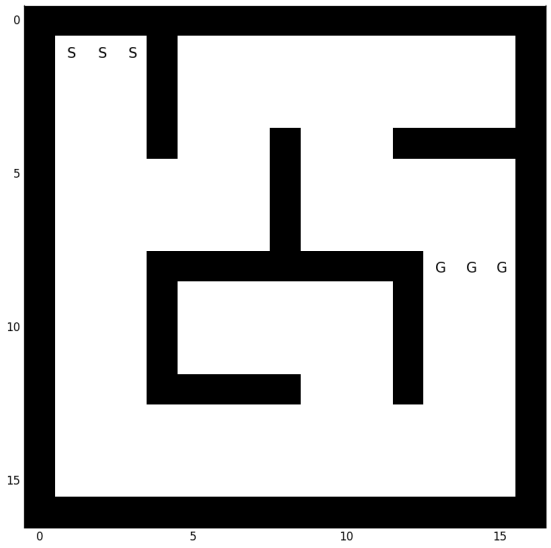


Fig. 1. Maze used for the simulation. Black cells represent walls, white represent empty cells. S: start position, G: goal position. In this case we have two possible paths with slightly different lengths.

A. Maze environment

The maze is the world in which the agents are allowed to move. Each maze consists of empty and occupied cells. An occupied cell can be either a wall or a robot. However, there is an option to allow several agents on one cell. In that case only walls are occupied cells. Agents are free to move to empty cells but are permitted to move to occupied cells. The substructure of each maze is a binary matrix (zero is an empty cell, one is a wall).

In our case start positions are located in the top left corner while goal positions can be located anywhere in the maze. The agents can either start all from the same start position or from different ones if multiple start positions are set. However it is irrelevant which goal position they reach.

The maze we created for our experiments consists of two paths to the goal positions. However, one path is slightly shorter than the other one. The purpose of this is to artificial create different levels of experiences. Also the width of the corridors are specifically set to 3 to allow multiple agents to pass simultaneously. The maze is pictured in figure 1.

B. Agents

The agents shall represent small robots who can communicate with each other, have sensors to perceive the environment and can move in several directions. Each agent has the same size of one cell in the simulation. With its simulated sensors it can detect walls in the adjacent cells. We do not consider failures of the sensors so the wall detection is perfect. Also there is only one sensor model.

Fig. 2. The blue filled cells and the green filled cells mark two complete different paths from the top left corner to the bottom right corner. Nevertheless both paths have the same length.

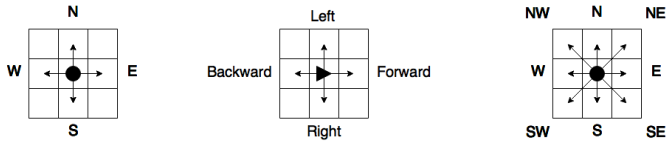


Fig. 3. Possible movement modules for the agents. Some of them have an increased state space or/and action space. Circle represents the agent. Triangle represents the agent and his orientation.

However, there are several movement models. There are two main reasons of using different movement models: First, by allowing agents only to move in the four standard directions 'North', 'East', 'South' and 'West', the variety of path lengths is heavily restricted. This is due to the Manhattan distance as illustrated in figure 2. The second reason is that we can also increase the state space by also taking the orientation into account. This leads to a more realistic movement and simulation. In figure 3 a small overview of some movement models we used for our agents is given.

C. Process Flow

The simulation executes *simulation steps* as long as not every agent has reached the goal. During a simulation step each agent has to do at least one *agent step*.

It is possible for an agent to do more than one agent step per simulation step to allow agents with different velocities. For example, if the speed parameter of an agent is set to 2, it has to execute 2 agent steps per simulation step. A more detailed explanation why different agent speeds are used is given in the next section. All agent steps are executed in sequence.

To avoid that always the same agent starts during a simulation step, the execution order is randomized in the beginning of each simulation step.

Once every agent has reached the goal, the trial is finished and a new trial is started. After the defined number of trails are carried out, the simulation run is finished.

Multiple simulation runs are performed to obtain an averaged result.

D. Realization

The simulation was implemented in Python 3.

To get an overview of the modules and simulation settings, we used a Graphical User Interface. With its help, we could

TABLE I
EQUAL EXPERIENCE

Type Of Cooperation	Coop. Time	Expertness Measure	Weighting	Average Total Steps	Average Resulting Path Length
IND	-	-	-	17244	99.6
LAT	5	Absolute	LFA	15606	52.6
LAT	5	Covered Steps	LFA	15468	63.4
LAS	30	Absolute	LFA	13293	33.2
LAS	30	Covered Steps	LFA	12826	32.6
LAS	30	Distance To Goal	LFA	16343	42.8
STA	-	Absolute	LFA	7336	47.2

IND: Individual Learning, LAT: Learn After Trials, LAS: Learn After Steps, STA: Static Q-matrix

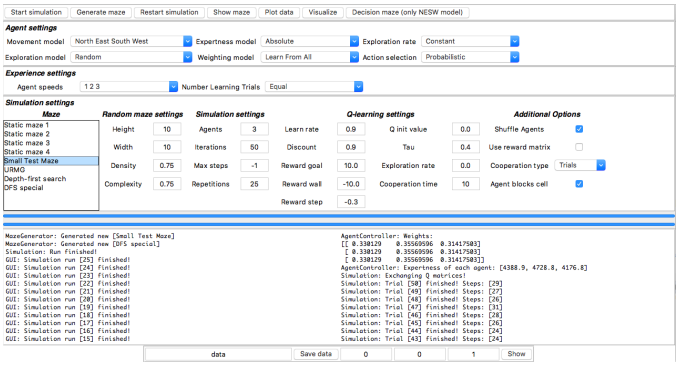


Fig. 4. GUI of the simulation. It was used to test different configurations for the agents and the simulation.

easily configure the simulation parameters and test them. The GUI is displayed in figure 4.

The examination and plots of the resulting data was done in MATLAB.

VI. CONFIGURATIONS

Since there are a lot of parameters we have to set, in this section we want to give a small overview on the most important ones because slightly different parameter can lead to severe different results. Also, we introduced some unusual parameters to test different circumstances.

A. Agent speed

Agent speed determines how many agent steps the agent can execute per simulation step. Setting the agents to different speeds leads to different expertness levels in the multi-agent system. The values are either 1 for normal speed, 2 for being twice as fast as normal and 3 for three times faster than normal.

B. Learn rate

The learn rate determines how much influence the newly learned information has. The value has to be in the range from 0 to 1. Set to 0, no new information is learned. Set to 1, all old learned information is overwritten by the newly learned information. Since our simulation is deterministic (besides random movement of the agents) we can choose a rather high value near 1. For our experiments we always set the learn rate to 0.9.

C. Discount

The discount determines how much we consider the estimation for the next state in the Q-value update process. For our experiments 0.9 is also a good value.

VII. EXPERIMENTS

With the help of the simulation we carried out multiple experiments to study how Cooperative Learning competes under different conditions and configurations. We used the learning curve of Individual Learning as a benchmark. One simulation run consists of 50 trials. A trial is finished when every agent reached a goal. Before each simulation step, the running order of the agents is randomized. Each cell of the maze, which is not a wall, can only contain one agent at the same time. We do not differ between training and test runs. Following parameters were set for every experiment:

- Learn Rate: 0.9
- Discount: 0.9
- Q Initial Value: 0
- Reward Goal: 10
- Reward Wall: -10
- Reward Step: -0.3
- Boltzmann Temperature: 0.4
- Weighting Model: Learn From All

To get an average result, we did 25 simulation runs in each case.

A. Equal Experience

To begin with, we investigated how Cooperative Learning competes when all agents have the same preconditions. Same preconditions entail that all agents have the same speed and are permitted to learn the same amount of trials. We analyze the Learning Curve as well as the Resulting Path Length (RPL). The Learning Curve consists of the executed simulation steps per trial. The sum over the Learning Curve gives us a measurement of how many total steps were needed to find the resulting path. The RPL is computed by taking the average of the last 5 trials.

In the first run, three agents were simultaneously run without any type of cooperation. We call this Individual Learning and use the results as a benchmark for the other experiments.

Table I shows the sum over the simulation steps and the RPL for LAT and LAS. Some of the prescribed expertness measure

TABLE II
DIFFERENT EXPERIENCE - SPEED

Type Of Cooperation	Coop. Time	Expertness Measure	Weighting	Average Total Steps	Average Resulting Path Length
IND	-	-	-	12201	62.6
LAS	30	Absolute	LFA	9426	31.6
LAS	30	Covered Steps	LFA	9265	29
LAS	30	Distance To Goal	LFA	11042	31.4
STA	-	Absolute	LFA	5066	40

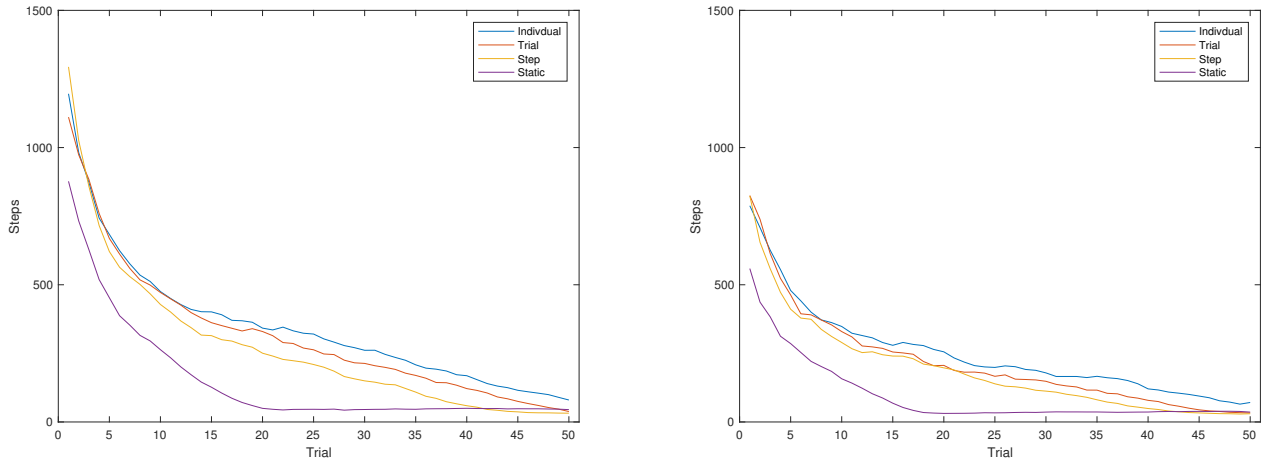
IND: Individual Learning, LAT: Learn After Trials, LAS: Learn After Steps, STA: Static Q-matrix

TABLE III
DIFFERENT EXPERIENCE - TRIALS

Type Of Cooperation	Coop. Time	Expertness Measure	Weighting	Average Total Steps	Average Resulting Path Length
IND	-	-	-	17439	155.2
LAT	6	Absolute	LFA	16088	118
LAT	6	Covered Steps	LFA	14843	101.2
LAT	6	Inverse Covered Steps	LFA	15121	91

IND: Individual Learning, LAT: Learn After Trials, LAS: Learn After Steps, STA: Static Q-matrix

Fig. 5. Learning curves for different and equal experience.



(a) Equal experience

(b) Different experience

were used. LFA was used in every case as the weighting mechanism.

The table shows that Cooperative Learning reduces the needed simulation steps and also reduces the RPL. However the results of Cooperative Learning are very different in some cases.

With a RPL of 32.6 and a total of 12826 simulation steps, LAS and Covered Steps, as the weighting measure, obtain the best results with equal experiences. Yet it is interesting that LAT with the same expertness measure achieved the worst result: 63.4 RPL.

B. Different Experience

In this set of experiments we want to investigate how Cooperative Learning competes when the agents have different experiences. To get different experiences of the agents we limit or enhance individual agents. We differ between two methods to make this possible. On the one hand we set the speed of the agents to different values. This allows agents with different experiences during a trials However, this method is useless when using LAT. This is due to the fact that a trial is only then finished when every agent has reached a goal. To get different experiences when using LAT, some agents are only allowed to do a specific amount of trials. For example, the cooperation time is set to 15 trials. Agent 1 is allowed to run all 15 trials. However, Agent 2 only runs 10 trials and Agent 1 even runs only 5. After the 15th trial all agents cooperate. This method leads to different experiences as well.

Table II shows the results for LAS and different speeds, while table III shows the results for LAT and different numbers of learning trials. Even if the number of total simulation steps could not be reduced significantly, the RPL with cooperation is considerably shorter.

C. Static Q-Learning

Both LAT and LAS use nearly the same principle: each agent has its own Q-matrix and after a specific time exchange their weighted matrices. However, another approach is to only have one static Q-matrix which shared among the agents. That means all agents use and update the same Q-matrix. One major advantage is that the information from the last moving agent is instantly available for the other agents. Therefore, agents can immediately follow or avoid the steps from the other agents.

As can be seen in figure 5 a) and b), using the static Q-matrix results in a notable reduction of simulation steps in both cases. However, the RPL is higher than with WSS: 47.2 for equal experience and 40.0 for different experience.

VIII. Conclusion and Future Work

In this paper we investigated whether Cooperative Learning is beneficial in comparison with Individual Learning in terms of finding the shortest path in a maze.

For this purpose, we did some experiments to examine different types of cooperation. The results showed that Cooperative Learning can reduce the learning time and the resulting path significantly. Although Static Q-Learning needed the least simulation steps in total, it could not find the optimal path in the end. The shortest path was found with LAS and using the WSS method.

A project for the future could be to find a combination of Static Q-Learning and WSS. A conceivable solution could be a weighted exchange of the covered path of each agent.

Furthermore, only one movement model was examined in this paper.

References

[1] Robert Axelrod. "The Evolution of Strategies in the Iterated Prisoners Dilemma" in *The Dynamics of Norms.*, Cambridge University Press, Cambridge, pp. 1-16. 1987.

[2] Majid N. Ahmadabadi, Masoud Asadpur, Seyyed H. Khodaabakhsh, Eiji Nakano. *Expertness Measuring in Cooperative Learning.* Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2261-2267. 2000.

[3] Ming Tan. *Multi-agent reinforcement learning: Independent vs. cooperative agents.* Proceedings of the Tenth International Conference on Machine Learning, pp. 330-337. 1993.

[4] Michael L. Littman. *Markov games as a framework for multi-agent reinforcement learning.* Proceedings of the Eleventh International Conference on Machine Learning, pp. 157-163. 1994.

[5] Yoav Shoham, Rob Powers, Trond Grenager. *Multi-agent reinforcement learning: a critical survey.* Technical report, Stanford University. 2003.

[6] Marco Dorigo, Luca M. Gambardella. *Ant-Q: A Reinforcement Learning approach to the traveling salesman problem.* Proceedings of the Twelfth International Conference on Machine Learning, pp. 252-260. 1995.

[7] Marco Wiering. *Multi-Agent Reinforcement Learning for Traffic Light Control.* Proceedings of the Seventeenth International Conference on Machine Learning, pp. 1151-1158. 2000.

[8] Christopfer J.C.H. Watkins, Peter Dayan. "Q-Learning" in *Machine Learning*, vol. 8, Kluwer Academic Publishers, Boston, pp. 279-292. 1992.

[9] Richard Dearden, Nir Friedman, Stuart Russell. *Bayesian Q-learning.* Proceedings of the Fifteenth National Conference on Artificial Intelligence, pp. 761-768. 1998.

[10] Dinko Osmankovi, Samim Konjicija. *Implementation of Q-Learning algorithm for solving maze problem.* Proceedings of the Thirty-fourth International Convention MIPRO, pp. 1619-1622. 2011.

[11] Ahmed, M. *Optimum Short Part Finder for Robot using Q-Learning.* Diyala Journal of Engineering Sciences, vol. 5.01, pp. 13-24. 2012.