Multi-Agent Reinforcement Learning for Coordinated Block Transportation

Maikel Withagen (S1867733) [a] Steven Bosch (S1861948) [a]
Thomas Derksen (S2016346) [a]

[a] *University of Groningen, Faculty of Mathematics and Natural Sciences, Nijenborgh 9, 9747 AG, Groningen*

Abstract

1 Introduction

One of the returning issues in machine learning is that it is often hard to decide which learning algorithm should be used for a certain application. Over the years, a lot of valid learning algorithms have emerged, each with their own advantages and disadvantages. Some algorithms might be faster, while others might be more precise, and an algorithm that is better in one area usually performs worse in another. Matching the right task with the right algorithm is not always a straightforward task.

In this paper we will look at the problem of multi-agent box-transportation. For this task, two virtual robots will have to cooperate to transport a box to a certain goal area, while avoiding certain obstacles. This is a problem that has already garnered some attention in machine learning research. In 2006, Wang and de Silva tried to solve this problem with reinforcement learning techniques [4]. They compared the performance of a Q-learning algorithm with the performance of team Q-learning in a dynamic environment. In 2010, a team at the University of Deft also showed how you could use Q-learning, team Q-learning and the WoLF-PHC algorithm to accomplish a similar task in a static environment [1].

We will also compare the Q-learning and team Q-learning algorithm, since they seem most suitable for a task like this. For our initial task, two agents have to transport a box in a two dimensional static environment. We will compare the performance of our two reinforcement learning algorithms by looking at the time it takes for them to converge to the optimal solution (given that they find it).

Secondly, we will also expand the task by adding another goal area or adding additional agents. **However, we still have to see if this is doable.**

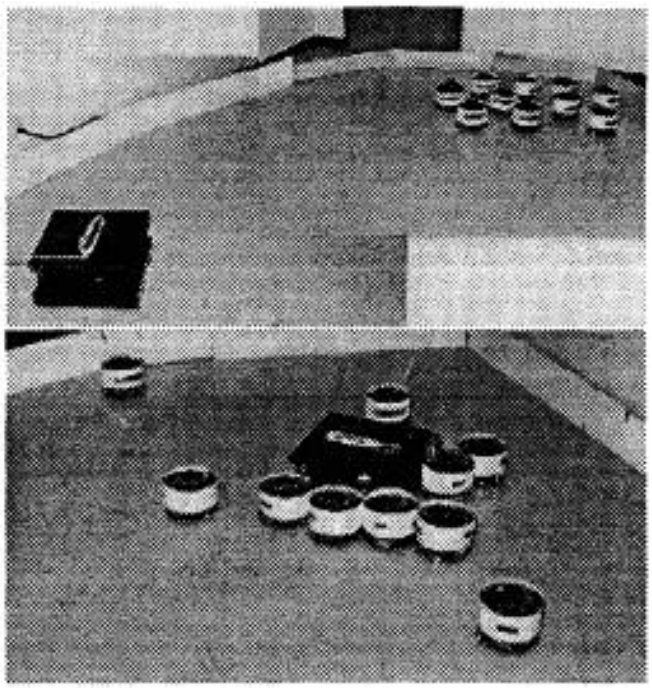
2 Application

The box-transportation problem is significant in both the theoretical and practical domain. From a theoretical point of view, it is a relatively simple problem with clear boundaries. Although there are multiple versions of this problem, the basic idea is intuitively easy to understand. Because of its simplicity, it is not hard to extend it to more complicated, real-life situations. It is also a good task to observe the effects of different learning algorithms since, in

the more simple, stripped down versions of this task, there are not a lot of parameters that influence the outcome. Finally, it is an interesting way of studying cooperation between multiple autonomous agents.

Although we use a simulation in this paper, the task has successfully been completed by physical robots. For example, in [2] a team of two Pioneer 2 mobile robots was used to physically push a box to a goal area while a third robot served as an observer. In [3] they took a different approach, where they used a swarm of smaller custom robots to collectively move a box to a designated spot (as can be seen in figure 1).

Figure 1: Robots from [3] locating a lit box



It is not hard to come up with practical applications for these kind of systems. In the future, they might for example be useful in construction, or to clear the rubble after a building has collapsed.
Our extentions are also interesting, for obvious reasons.

3 Method

To compare the various learning algorithms, we have written a multi-agent simulation in Python in which agents have to move an object to a certain goal area. In this section, the various components of the simulation will be explained.

3.1 The Environment

The environment consists of an **x by y** grid in which the agents can move around. There are a number of different types of cells:

Free cells These are cells that agents can freely move to. Once an agent or a block moves to a free cell, this cell becomes occupied. They are denoted by dots in the visual representation

of the simulation.

Walls These cells are occupied, so an agent can never move to them. They are initialized at the start of the simulation and will not change during the simulation. There are outer walls, which are on the edges of the grid, and some walls inside the grid to make the environment more complex. They look like hash tags in the visualization.

Block The block is an item that has to be transported to a certain goal area by the agents. It occupies one cell, and agents cannot move through it. It can only be moved if it is grasped by all of the agents present in the simulation, and those agents move in the same direction. If the block reaches the goal, an epoch ends. In the simulation, it is denoted by a B.

Goal The goal cell is basically the same as a free cell, with the exception that if the block reaches the goal a reward is given to the agents and the run of the simulation ends. Agents can freely move on and over the goal, just like with free cells. The goal is represented by a B in the simulation.

Agent We'll talk about these in more detail in the next subsection. They occupy one cell, so other agents cannot move through each other. In the simulation they are represented by A's. In **FIGURE X** you can see how the visual representation of the simulation looks.

3.2 The Agents

For our initial simulation, we start out with two agents. At the start of a run, the agents are spawned at their respective start locations which are defined in the algorithm. Each step, an agent can perform one of five actions: **Stay**, **Left**, **Right**, **Up**, **Down** or **Grab**. The actions mostly speak for themselves. The agent can stay where it is, move in one of four directions or grab the block. The *grab* action can be performed at any step, but will only do something when an agent is next to the block. Once an agent has grabbed the block it will not let go. The move actions *left, up, right* and *down* only do something when the cell the agent wants to move to is actually free.

Each step, an agent chooses the best action based on its learning algorithm. Then, the agents perform their actions and update the values related to their learning algorithms (usually these are the values in their Q-tables). This process repeats itself until the block has reached the goal. After that, a new run starts and the agents are reset to their starting positions. There are two ways for an agent to get a reward: they can either grab the block to get a reward of 10, or they can get the block to the goal for a reward of 100. These rewards are used to calculate the Q-values, which we will talk about more in the next section.

3.3 The Learning Algorithms

For this paper, we compared two machine learning algorithms: Q-learning and team Q-learning. Both are reinforcement learning algorithms, and they are based on the same principles. The goal of both algorithms is, in our case, to make sure that the agents find the optimal policy that will result in the highest reward. It can be found through a value iteration method.

Q-learning was first introduced by Watkins in 1989 [5]. It is a model free reinforcement learning technique for multi-agent systems. In our simulation, the agents begin with a Q-table filled with zeros. Each step, they choose one of the five actions with certain probabilities, which are based on the different Q-values. The probabilities are calculated as follows: $P(a_k) =$

$\frac{e^{Q(s,a_k)/\tau}}{\sum_{l=1}^{m} e^{Q(s,a_l)/\tau}}$ In this case, s is the current state, m is the total number of Q-values for the current state, a_k is the k^{th} action and τ is a parameter that gradually decreases during the simulation. After an action is chosen and a new state s' is reached, the value in $Q(a_k)$ is updated as follows: $Q(s, a_k) = (1-\varepsilon)Q(s, a_k)+\varepsilon(r+\beta maxQ[s', a'])$ (where ε is the learning rate between 0 and 1). From this formula, it becomes apparent that in order to calculate the Q-value for a certain action in a certain state, both the outcome of that action and the outcome of the best possible action in the next state are considered. After each epoch, to make sure the algorithm converges, we update τ: $\tau_i = \tau_i - (\tau_1 - 0.1/N)$, where τ_i is the value of τ-value in epoch i and N is the total number of epochs.

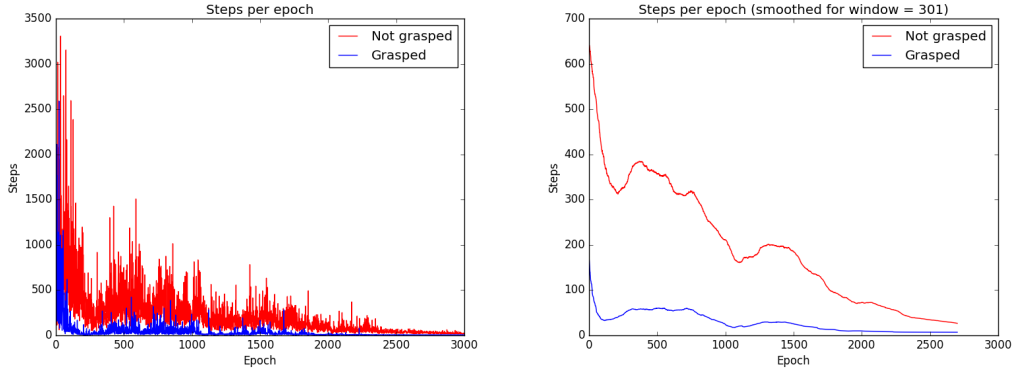
After the agents have reached the goal with the block, their Q-tables are transferred to the next run. This should result in the fact that the agents learn to choose the path that gains them the highest reward. One notable thing in our simulation is that the agents have two separate Q-tables: one for when the agents have grasped the block, and one for when they have not. This is necessary because the best actions change when the agents start moving the block. Thus, they have to optimize two seperate paths.

Team Q-learning is based on the same principles as Q-learning, and is actually an extension of the Nash Q-learning algorithm. The primary difference is that in team Q-learning, the agents share a single Q-table, and the Q-values are based on their joint actions. In this case, each step the best action pair is chosen based on the current position of both of the agents. So, after both agents have taken an action, the Q-value for the joint position and joint action of the agents is still updated with the same formula, but in this case, $Q(s, a_k)$ is the Q-value for the shared action a_k in the combined state s. The probability that a certain action is taken is calculated in a way that is very similar to Q-learning. The main difference is that the probabilities for shared actions are calculated, instead of probabilities for the actions of the individual agents. So, in team Q-learning the agents share a Q-table and action probabilities. With this algorithm, it is like the agents try to cooperate with each other.

3.4 The simulation

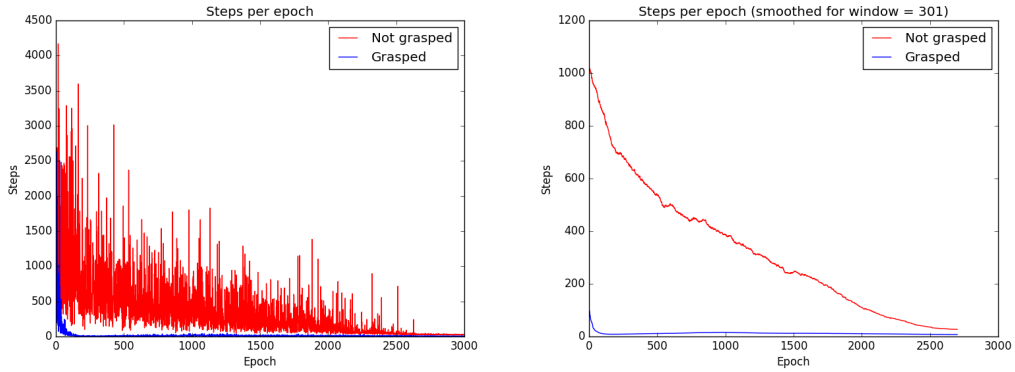
The previous subsections explained how the individual parts of our simulation work. However, there is still an question that needs to be answered: how do all these pieces fit together to form a coherent simulation? There are two different algorithms, one for Q-learning and one for team Q-learning. Although they are quite similar, there are some significant differences. In this subsection, the basis of the algorithms will be explained, and the differences will be highlighted.

The program is executed with two arguments: one for the way in which the actions are chosen (because we can also use the epsilon-greedy strategy, although we do not use it for this paper) and one for the environment. The environment can either be simple, in which case it is just a five by five grid without inner walls, or complex. The complex world is the world that can be seen in **FIGURE X**. The outer loop of the algorithms loops through the number of runs, which is usually one. In it, τ and a few other parameters are initialized. Then, the algorithms loop through the right number of epochs, which are divided in a training and a test part. Each epoch begins by resetting the world and putting the agents back in their starting positions. Then, it goes into a loop that runs until the goal is reached. Each step, the agents choose an action based on the Q-values for the state they are in. In team Q-learning, a set of actions is chosen, while in Q-learning, both of the agents have to individually choose an action. Although we do not use it for this paper, the actions can also be chosen with the epsilon-greedy strategy. Usually, an action (or a set of actions) is chosen with a probability relating to its Q-value, as we saw in the previous subsection. After the actions have been chosen, the world is updated. For this to be



(a) Single Q

(b) Single Q (smoothed)



(c) Single Q

(d) Single Q (smoothed)

Figure 2: Number of steps over 3000 epochs

done succesfully, both of the algorithms have to check a few things: do the agents move in to a space that is occupied? Have the agents grasped the block? If they have grasped the block, are they moving in the same direction? All of these things have to be checked to see if the action can be succesfully performed or if one or more of the agents stay in their current state.

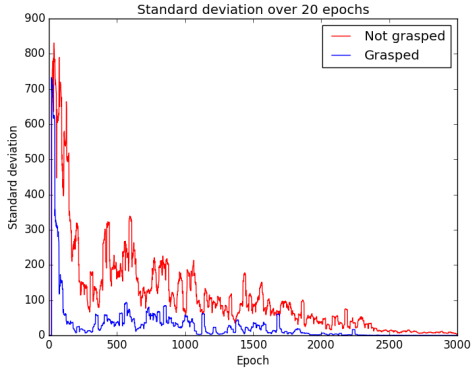
4 Results

Figure 2 shows the number of steps taken per epoch by two agents using single or team agent Q-learning for a run of 3000 epochs. Each plot shows a graph of the number of steps it takes for the agents to reach and grab the block and a graph of the number of steps it takes them to move the block to the goal. We measured the difference in performance between the two Q-learning algorithms by both speed of convergence and optimality of the learned path.

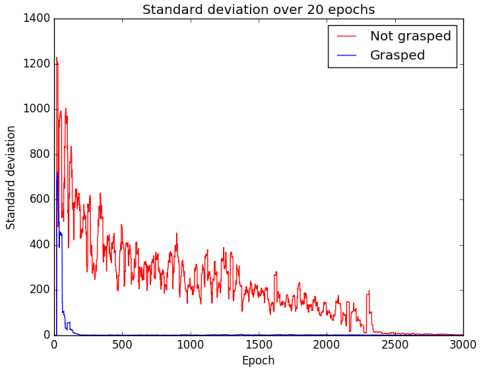
4.1 Speed of convergence

To measure the speed of convergence we used as a convergence criterion that the standard deviation of 20 consecutive epochs be smaller than a set value.[1]. Figures 3a and 3b show the standard deviations for both the single and team learning algorithm. The simulation was run 30 times per learning algorithm and every run the number of epochs it took to reach the convergence criterion was counted. Figures 3c and 3d show the results of these counts.

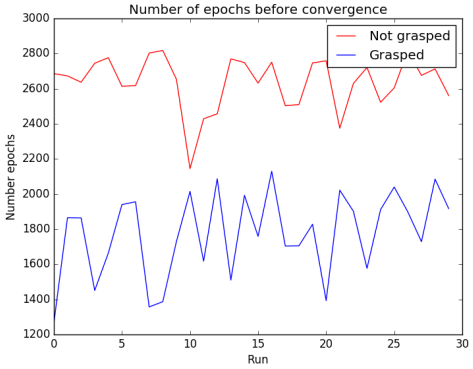
[1]For these results $\sigma_{not_grabbed} < 7$ and $\sigma_{grabbed} < 2$



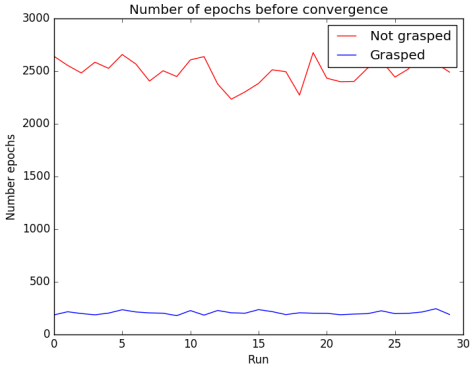
(a) Single Q



(b) Team Q



(c) Single Q



(d) Team Q

Figure 3: (a and b) Standard deviation over 20 epochs in a run of 3000 epochs. (c and d) Epochs before convergence.

To test whether the difference was significant we performed an unpaired t-test on the not grabbed, grabbed and summed results. The t-test tested highly significant for the not grabbed samples with $m_{single} = 2636.6$ and $m_{team} = 2500.0$: $t(55.331) = 3.8914$, $p < 0.001$. It also tested highly significant for the grabbed samples, with $m_{single} = 1776.767$ and $m_{team} = 205.9$: $t(29.279) = 35.443$, $p < 0.001$. Finally, the t-test tested highly significant for the added results, with $m_{single} = 4413.367$ and $m_{team} = 2705.9$: $t(44.634) = 36.169$, $p < 0.001$.

From these results we can conclude that single Q-learning is converged slightly faster when the block is not grabbed, but considerably slower when the block is grabbed. In total this results in the team Q-learning being quite faster overall, with an average difference of approximately 1707 epochs. Note that this difference is taken over the sum of the epochs for grabbed and not grabbed samples. Since the algorithms do both the grabbed and not grabbed part during one epoch, you might also consider just the difference between the samples with the highest number of epochs, which in both cases is the not grabbed one. This difference was still measured highly significant, but is much smaller: ca. 137 epochs.

5 Discussion

References

[1] Lucian Busoniu, Robert Babuska, and Bart de Schutter. Multi-agent reinforcement learning: An overview. In Dipti Srinivasan and Lakhmi Jain, editors, *Innovations in Multi-Agent Systems and Applications*, chapter 1, pages 183–221. Springer Berlin Heidelberg, Berlin, 2010.

[2] Brian P. Gerkey and Maja J. Mataric. Pusher-watcher: an approach to fault-tolerant tightly-coupled robot coordination. In *International Conference on Robotics and Automation*, pages 464–469, 2002.

[3] C. Ronald Kube and Hon Zhang. The use of perceptual cues in multi-robot box-pushing. In *International Conference on Robotics and Automation*, pages 2085 – 2090, 1996.

[4] Ying Wang and Clarence de Silva. Multi-robot box-pushing: Single-agent q-learning vs. team q-learning. In *International Conference on Intelligent Robots and Systems*, 2006.

[5] Christopher John Cornish Hellaby Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.