An extension of a hierarchical reinforcement learning algorithm for multiagent settings

Ioannis Lambrou, Vassilis Vassiliades, and Chris Christodoulou

Department of Computer Science,
University of Cyprus, Nicosia 1678, Cyprus
{cs07li2,v.vassiliades,cchrist}@cs.ucy.ac.cy

Abstract. This paper compares and investigates single-agent reinforcement learning (RL) algorithms on the simple and an extended taxi problem domain, and multiagent RL algorithms on a multiagent extension of the simple taxi problem domain we created. In particular, we extend the Policy Hill Climbing (PHC) and the Win or Learn Fast-PHC (WoLF-PHC) algorithms by combining them with the MAXQ hierarchical decomposition and investigate their efficiency. The results are very promising for the multiagent domain as they indicate that these two newly-created algorithms are the most efficient ones from the algorithms we compared.

Keywords: Hierarchical Reinforcement Learning, Multiagent Reinforcement Learning, Taxi Domain

1 Introduction

Reinforcement learning (RL) suffers from the curse of the dimensionality, where the addition of an extra state-action variable increases the size of the state-action space exponentially and it also increases the time it takes to reach the optimal policy. In order to deal with this problem, there exist three general classes of methods that exploit domain knowledge by: i) using function approximation to compress the value function or policy and thus generalise from previously experienced states to ones that have never been seen, ii) decomposing a task into a hierarchy of subtasks or learning with higher-level macro-actions, and iii) utilising more compact representations and learn through appropriate algorithms that use such representations.

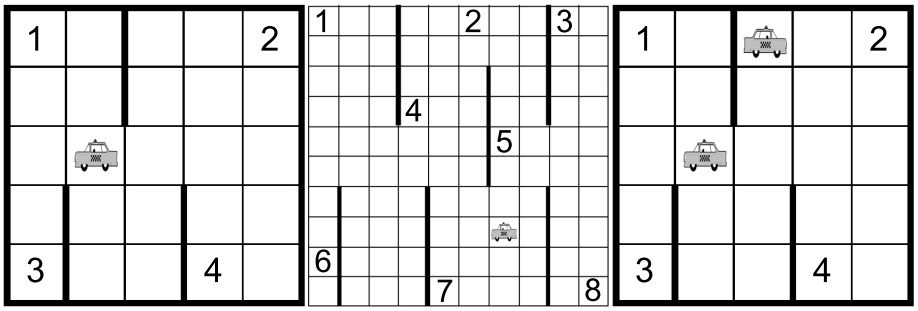
Hierarchical methods reuse existing policies for simpler subtasks, instead of learning policies from scratch. Therefore they may have a number of benefits such as faster learning, learning from fewer trials and improved exploration [4]. This area has attracted an influx of research work. Singh [17] presents an algorithm and a modular architecture that learns the decomposition of composite Sequential Decision Tasks (SDTs), and achieves transfer of learning by sharing the solutions of elemental SDTs across multiple composite SDTs. Dayan and Hinton

[3] show how to create a Q-learning [20] managerial hierarchy in which high level managers learn how to set tasks to their sub-managers who, in turn, learn how to satisfy them. Kaelbling [10] presents the Hierarchical Distance to Goal (HDG) algorithm, which uses a hierarchical decomposition of the state space to make learning achieve goals more efficiently with a small penalty in path quality. Wiering and Schmidhuber [21] introduced the HQ-learning algorithm, a hierarchical extension of Q-learning, to solve partially observable Markov decision processes. Parr [14] developed an approach to hierarchically structuring Markov Decision Process (MDP) policies called Hierarchies of Abstract Machines (HAMs), which exploit the theory of semi-MDPs [18], but the emphasis is on simplifying complex MDPs by restricting the class of realisable policies, rather than expanding the action choices. Sutton et al. [18] consider how learning, planning, and representing knowledge at multiple levels of temporal abstraction, can be addressed within the mathematical framework of RL and MDPs. They extend the usual notion of action in this framework to include options, i.e., closed-loop policies for taking actions over a period of time. Overall, they show that options enable temporally abstract knowledge and actions to be included in the RL framework in a natural and general way. Dietterich [4] presents the MAXQ decomposition of the value function, which is represented by a graph with Max and Q nodes. Max nodes with no children denote primitive actions and Max nodes with children represent subtasks. Each Q node represents an action that can be performed to achieve its parent's subtask. The distinction between Max nodes and Q nodes is needed to ensure that subtasks can be shared and reused. Andre and Russell [1] extend the HAM method [14] and produced the Programmable HAM (PHAM) method that allows users to constrain the policies considered by the learning process. Hengst [8] presents the HEXQ algorithm which automatically attempts to decompose and solve a model-free factored MDP hierarchically. Ghavamzadeh and Mahadevan [7] address the issue of rational communication behavior among autonomous agents and propose a new multi-agent hierarchical algorithm, called COM-Cooperative Hierarchical RL (HRL) to communication decision. Mehta et al. [11] introduced the multi-agent shared hierarchy (MASH) framework, which generalises the MAXQ framework and allows selectively sharing subtask value functions across agents. They also developed a model-based average-reward RL algorithm for that framework. Shen et al. [16] incorporate options in MAXQ decomposition and Mirzazadeh et al. [13] extend MAXQ-Q [4] producing a new algorithm with less computational complexity than MAXQ-Q. Mehta et al. [12] present the HI-MAT (Hierarchy Induction via Models and Trajectories) algorithm that discovers MAXQ task hierarchies by applying dynamic Bayesian network models to a successful trajectory from a source RL task.

Learning in the presence of multiple learners can be viewed as a problem of a "moving target", where the optimal policy may be changing while the agent learns. Therefore, the normal definition of an optimal policy no longer applies, due to the fact that it is dependent on the policies of the other agents [2]. In many cases, the aim is for a multiagent learning system to reach a balanced state, also known as equilibrium, where the learning algorithms converge to stationary

policies. Oftentimes a stationary policy can be deterministic, however, there are situations where we look for stochastic policies. An example from game theory, where pure strategy (i.e., deterministic policy) equilibria do not exist, is the famous Rock-Paper-Scissors game. In this game, the solution is to converge to a mixed strategy (i.e., stochastic policy) equilibrium where the agents play each action with probability 1/3. Therefore, in multiagent learning settings it is sometimes beneficial for an agent to keep an estimate of its stochastic policy and update it accordingly during the course of learning. This estimate of the stochastic policy is used by the algorithms PHC (Policy Hill Climbing) [2], which is an extension of Q-learning, and WoLF-PHC (Win or Learn Fast-PHC) [2], which is an extension of PHC (see Section 2.3). The WoLF-PHC was empirically shown to converge in self-play to an equilibrium even in games with multiple or mixed policy equilibria [2].

In this paper, we explore a novel combination of the hierarchical method MAXQ with the multiagent RL (MARL) algorithms PHC and WoLF-PHC. We evaluate the performance of Q-learning [20], SARSA (State Action Reward State Action) [15], PHC [2], WoLF-PHC [2], MAXQ-Q [4], as well as the two newly created algorithms MAXQ-PHC and MAXQ-WoLF-PHC, in two single-agent taxi domains [4, 5] (shown in Figures 1a and 1b) and a multiagent extension of the taxi domain we created (shown in Figure 1c).



(a) Taxi domain (b) Extended taxi domain (c) Multiagent taxi domain

Fig. 1: Single-agent and multiagent taxi domains. (a) Original 5×5 domain (adapted from [4]), (b) Extended 10×10 domain, with different wall distributions and 8 passenger locations and destinations (adapted from [5]), (c) Multiagent domain, where there are 2 or more taxis and passengers.

The rest of the paper is organised as follows. Section 2 describes the methodology we followed with a description of the taxi domain, the MAXQ hierarchical decomposition for this domain, and the novel combination of the MAXQ hierarchy with the algorithms PHC and WoLF-PHC. The results are presented and analysed in Section 3, and finally in Section 4, we summarise the conclusions and briefly discuss some issues related to this work.

2 Methodology

For all RL algorithms we investigate in this paper, apart from the PHC and WoLF-PHC variants, ϵ-greedy or Boltzmann exploration is utilised. An ϵ-greedy exploration selects a random action in a given state with a probability ϵ, otherwise it selects the action with the highest Q-value. A Boltzmann exploration selects an action a_i from state s with probability $p(a_i)$ given by equation 1:

$$p(a_i) = \frac{e^{Q(s,a_i)/t}}{\sum_{a \in A} e^{Q(s,a)/t}} \quad (1)$$

where $Q(s, a_i)$ is the value of state s and action a_i, A is the action set, and the temperature t is initialised with a temperature τ_0 at the beginning of an episode, and in subsequent steps is decreased exponentially by a multiplication with a cooling rate.

2.1 Taxi Problems

In the simple taxi problem [4] there is a 5×5 gridworld with four specially-designated locations marked with numbers 1 to 4 (Figure 1a) which represent the possible passenger locations and destinations. The taxi problem is episodic. Initially, in each episode, each taxi starts in a randomly chosen state and each passenger is located at one of the 4 locations (chosen randomly). The episode continues with the execution of the wish of the passenger to be transported to one of the four locations (also chosen randomly). The taxi must go to the passenger's location, pick up the passenger, go to the destination location and put down the passenger there. The episode ends when the passenger is delivered successfully. In the extended version of the simple taxi problem [5], there is a 10×10 gridworld with eight possible passenger locations and destinations marked with numbers 1 to 8 (Figure 1b). The task is the same as the simple taxi problem with the only difference being apart from the different wall distributions, the higher scale due to the larger grid and the increased number of destinations. The multiagent taxi problem (MATP) (Figure 1c) is an extension of the simple taxi problem, that we created to evaluate the MARL algorithms. In this version there exist two or more taxis and two or more passengers. Each taxi could carry at most one passenger at a time. The task is again the same with the only difference being that the episode ends when all the passengers are delivered successfully.

In the single-agent taxi problems there are six primitive actions: (a) four navigation actions that move the taxi one square, North, South, East, or West, (b) a Pickup action, and (c) a Putdown action. In the MATP there is an extra "idle" action (d), which does not modify the position of any agent. Each action is deterministic. There is a negative reward of -1 for each action and an additional reward of +20 for successfully delivering the passenger. There is a negative reward of -10 if the taxi attempts to execute the Putdown or Pickup actions illegally. If a navigation action would cause the taxi to hit a wall, the

position of the taxi does not change, and there is only the usual negative reward of -1. In the MATP version, two or more taxis cannot occupy the same cell at the same time. If they collide, they will get an additional negative reward of -20. Then there would be a random selection of which taxi would stay at the current position and the other taxis would go back to their previous positions. If one of the colliding taxis has chosen one of the actions (b), (c) or (d), then surely that taxi would be the one which will stay at the current position and the other taxis would go back to their previous positions. MATP can be seen as a scenario where the state is composed of $n + 2k$ state-variables (where n is the number of taxis and k the number of passengers): the location of each taxi (values 0-24), each passenger location, including the case where the passenger is in the taxi (values 0-4, 0 means in the taxi) and each destination location (values 1-4). Thus there will be $25^n \times 5^k \times 4^k$ states.

Fitch et al. [6] have also explored a multiagent version of the simple taxi domain. In their work, they specifically had two taxis and two passengers and an extra primitive action for picking up the passengers (pickup passenger one and pickup passenger two), whereas our version has only one pickup actions. In [6] the four navigation actions are stochastic: with 92.5% probability they move the taxi in the intended direction and with 7.5% probability they move the taxi randomly in one of the other three directions, compared to our version where the navigation actions are deterministic. Moreover, Fitch et al. [6] consider two versions of the problem, one with and another without taxi collisions. In the version with collisions the taxis cannot occupy the same cell at the same time as is the case in our MATP. In particular, in their MATP version, after a collision, all taxis stay at their previous locations, in contrast with our taxi domain where one taxi always stays at its current position. Finally, a significant difference is that in [6], the agents have full-observability over the joint state-action space, and this is reflected in the Q-value estimates of the agents; in our MATP however, we allow full-observability only over the joint state space, meaning that the agents can observe the state, but cannot observe the other agents' actions. While this reduces the size of the Q-tables, it could make the problem harder.

2.2 MAXQ hierarchical decomposition in the taxi domain

Dietterich [4] proposed a hierarchical extension of Q-learning, called MAXQ-Q. MAXQ-Q uses the MAXQ hierarchy to decompose a task into subtasks. The MAXQ graph can effectively represent the value function of any hierarchical policy. All taxi problems have the same task as described above and share the same hierarchical structure. This structure contains two sub-tasks, i.e., "get the passenger" and "deliver the passenger", while both of them involve navigation to one of the four locations. The top of the hierarchy starts with the MaxRoot node which indicates the beginning of the task. From the MaxRoot node, the agent has the choice of two "actions" that lead to the MaxGet and MaxPut nodes, corresponding to the two sub-tasks above. From the MaxGet node the agent has the choice of two "actions", the pickup primitive action and the navigate sub-task, while from the MaxPut node the agent has the choice of the navigate sub-

task and the putdown primitive action. The navigate sub-task of MaxGet and MaxPut leads to a MaxNavigate node which in turn leads to the four navigation primitive actions of north, east, south and west.

In the graph of the MAXQ hierarchy, each of the Max nodes is a child of a Q node (apart from MaxRoot) that learns the expected cumulative reward of executing its sub-task (or primitive action) which is context dependent. In contrast, the Max nodes learn the context independent expected cumulative reward of their sub-tasks (or primitive actions), which is an important distinction between the functionality of the Max and Q nodes.

2.3 Multiagent extensions that use the MAXQ hierarchy

A multiagent extension of Q-learning to play mixed strategies (i.e., probabilistic policies) in repeated and stochastic games was proposed in [2]. This extension is known as Policy Hill Climbing (PHC) and works by maintaining a policy table in addition to the Q-table. The policy table can be seen as the probability of choosing an action in a given state, therefore for each state, the sum of all policy values is equal to 1. After the Q-values are updated with the Q-learning rule, the policy values are updated using a learning rate parameter δ based on the chosen action, and subsequently normalised in a legal probability distribution. An extension of the PHC algorithm was additionally proposed in [2] that uses the Win or Learn Fast (WoLF) principle, i.e., learn quickly when losing and be cautious when winning. This is achieved by varying the learning rate δ of the policy update. More specifically, two learning rates are used, $\delta_{winning}$ when winning and δ_{losing} when losing, where $\delta_{losing} > \delta_{winning}$. The WoLF-PHC algorithm, uses a separate table to store an estimate of the average policy over time (see [2] for details), and to calculate when the agent is winning or losing. An agent is winning when the expected performance of its current policy is greater than the expected performance of its average policy. It has to be noted that a stochastic policy (i.e., a mapping of states to distributions over the action space) is needed in these algorithms so that the agents are able to reach mixed strategy equilibria, while an estimate of the average stochastic policy over time is needed to check whether the agent is losing or winning.

A hierarchical extension of Q-learning was proposed with the MAXQ decomposition in order to learn the respective value function for each node, thus learning a hierarchical policy [4]. As described above, the PHC and WoLF-PHC algorithms maintain the current policy (and the average policy in the case of WoLF-PHC) along with the value functions. Therefore, a possible combination of the MAXQ hierarchy with the PHC or WoLF-PHC algorithms would most probably enable the learning of a hierarchical multiagent policy. This is the basis of the rationale of the approach we followed in this paper, where we implemented this combination. More specifically, in the case of MAXQ-PHC, each Max node contains additionally the policy of the respective node, while in the case of MAXQ-WoLF-PHC, the average policy was also included. Thus, the selection of "actions" in each node is done based on the stored policy, rather than

an ϵ-greedy or a boltzmann exploration schedule. To the best of our knowledge such a combination has not previously been reported in the literature.

3 Results

3.1 Single-Agent Tasks

The Taxi Problem (TP) was run for 50 trials with 3000 episodes per trial and the Extended Taxi Problem (ETP) was run for 30 trials with 5000 episodes per trial. When we use ϵ-greedy exploration, ϵ takes the value 0.1 and when we use Boltzmann exploration, the initial temperature τ_0 takes the value 80 and the cooling rate the value 0.98, except for the MAXQ-Q algorithm; more specifically in the TP, we used a cooling rate of 0.9 at all nodes, while in the ETP, we used cooling rates of 0.9996 at MaxRoot and MaxPut, 0.9939 at MaxGet, and 0.9879 at MaxNavigate. These cooling rate values are the same as the ones used by Dietterich [4] for the TP; we observed, however, that they are more effective in the ETP rather than in the TP in our case. We did some experimentation with these values, but more needs to be done to fine-tune them. For both TP and ETP experiments we used a discount factor γ of 0.95 for all algorithms, and a step size α of 0.3 for Q-learning, SARSA, PHC and WoLF-PHC algorithms and 0.5 for MAXQ-Q, MAXQ-PHC and MAXQ-WoLF-PHC algorithms. The policy learning rate δ was set to 0.2 for the PHC and MAXQ-PHC algorithms. For the WoLF-PHC and MAXQ-WoLF-PHC algorithms, we set $\delta_{winning}$ to 0.05 and δ_{losing} to 0.2 in the TP experiments. In the ETP experiments, these values were set to 0.01 and 0.04 for the WoLF-PHC algorithm, while for the MAXQ-WoLF-PHC were set to 0.1 and 0.4 respectively. These values were found after some preliminary experimentation. Table 1 shows the ranking of the algorithms based on the median of the number of primitive time steps required to complete each successive trial averaged over 50 and 30 trials for TP and ETP respectively, for each algorithm.

As it can be seen, the tested algorithms have different rankings in the two problems. The most efficient algorithm for TP is SARSA using Boltzmann exploration and the most efficient algorithm for ETP is MAXQ-Q using ϵ-greedy exploration with a median of 11.46 and 64 respectively. This is due to the fact that in ETP the state-action space is bigger than the state-action space of TP, so the agents need more time to learn. Using the MAXQ hierarchical method we speed up learning, so the MAXQ-Q is the most efficient algorithm for ETP. We also notice that, both Q-learning and MAXQ-Q have better results when using an ϵ-greedy exploration, in contrast with SARSA which has better results when using Boltzmann exploration. This may be the case because Q-learning and MAXQ-Q are off-policy algorithms, whereas SARSA is an on-policy algorithm. This could also indicate that a more thorough investigation is needed to fine-tune all parameters.

On a closer inspection, when the MAXQ-Q algorithm is paired with a Boltzmann exploration in the TP, the average number of primitive time steps over the last 100 episodes (over all trials) converges around 34.6, whereas when ϵ-greedy

Table 1: Ranking of algorithms in the single-agent taxi problem. The efficiency of each algorithm is ranked according to the median of the number of primitive time steps required to complete each successive trial averaged over 50 and 30 trials for TP and ETP respectively. The most efficient algorithm scored 1 and the least efficient scored 10.

Ranking	Taxi Problem		Extended Taxi Problem	
	Algorithm	Median	Algorithm	Median
1	SARSA(boltzmann)	11.46	MAXQ-Q(ϵ-greedy)	64
2	MAXQ-Q(ϵ-greedy)	11.86	SARSA(boltzmann)	94.2
3	Q-learning(ϵ-greedy)	12.1	Q-learning(ϵ-greedy)	95.17
4	SARSA(ϵ-greedy)	12.9	MAXQ-WoLF-PHC	117.18
5	WoLF-PHC	14.35	MAXQ-PHC	125.93
6	PHC	14.74	SARSA(ϵ-greedy)	130.45
7	MAXQ-PHC	16.64	PHC	163.6
8	MAXQ-WoLF-PHC	16.88	MAXQ-Q(boltzmann)	169.28
9	Q-learning(boltzmann)	21.88	Q-learning(boltzmann)	179.98
10	MAXQ-Q(boltzmann)	35.62	WoLF-PHC	220.63

exploration is used, the average number of primitive time steps over the last 100 episodes (over all trials) converges around 11. This might suggest that in the case of Boltzmann exploration, regardless of the task, more fine-tuning needs to be done to the exploration schedule, while decreasing step sizes could provide an additional advantage. It has to be noted that the exploration schedule of each Max node should be optimised separately.

The PHC and WoLF-PHC algorithms are in the middle of the ranking for TP which means that they have respectable results for TP, even if they were initially created to be used in MARL problems. Moreover, the WoLF-PHC with median of 14.35 is better than the PHC algorithm with median of 14.74. This was expected because WoLF-PHC is an extension of the PHC algorithm which tries to improve it. Something not expected is that PHC seems to be better than WoLF-PHC for ETP. When plotting the graphs of average steps for each episode, we observed that the WoLF-PHC graph is smoother than the PHC graph. Therefore, the median metric of PHC has a lower value than the median value of WoLF-PHC because of oscillations on the PHC curve for average steps that occasionally reach a lower number than the respective WoLF-PHC curve.

Finally, we observe that the hierarchical algorithms MAXQ-PHC and MAXQ-WoLF-PHC are more efficient for ETP rather than for TP. That was expected, as hierarchy speeds up learning, so agents learn faster using these algorithms for ETP where the state-action space is bigger than the state-action space for TP.

Table 2: Ranking of algorithms in the multiagent taxi problem. The efficiency of each algorithm is ranked according to the median of the number of primitive time steps required to complete each successive trial averaged over 10 trials. The most efficient algorithm scored 1 and the least efficient scored 7.

Ranking	Algorithm	Median
1	MAXQ-WoLF-PHC	102
2	MAXQ-PHC	103
3	WoLF-PHC	133.1
4	PHC	153.4
5	Q-learning (ϵ-greedy)	705.4
6	MAXQ-Q(ϵ-greedy)	901
7	SARSA (ϵ-greedy)	1623.9

3.2 Multiagent Tasks

The multiagent taxi problem (MATP) was run for 10 trials with 1000000 episodes per trial. A preliminary experimentation showed that Boltzmann exploration parameters could not be obtained easily, therefore we used an ϵ-greedy exploration. Exploration parameter ϵ was set to 0.1 in all algorithms except for MAXQ-Q, for which $\epsilon = 0.35$ due to the fact that it could not finish a single trial with lower values. For Q-learning, SARSA, PHC and WoLF-PHC algorithms we used $\alpha = 0.3$ and $\gamma = 0.95$. For the PHC algorithm $\delta = 0.2$, and for WoLF-PHC $\delta_{winning} = 0.05$ and $\delta_{losing} = 0.2$. For MAXQ-PHC and MAXQ-WoLF-PHC algorithms $\alpha = 0.5$ and $\gamma = 0.9$. We also used a $\delta = 0.2$ for MAXQ-PHC, and $\delta_{winning} = 0.05$ and $\delta_{losing} = 0.2$ for MAXQ-WoLF-PHC. In the case of MAXQ-Q, we used $\alpha = 0.05$ and $\gamma = 0.95$. Table 2 shows the ranking of the algorithms with regards to the median of the number of primitive time steps required to complete each successive trial averaged over 10 trials, for each algorithm.

As we can see, the most efficient algorithms are the hierarchical, multiagent algorithms MAXQ-WoLF-PHC and MAXQ-PHC, with medians 102 and 103 respectively. The WoLF-PHC algorithm follows with a median of 133.1, and then the PHC algorithm with a median of 153.4. The least efficient algorithms are the single-agent ones, i.e., Q-learning, MAXQ-Q and SARSA with medians 705.4, 901 and 1623.9 respectively. Figure 2 illustrates the performance of the algorithms in terms of average number of steps over time. From the results in Figure 2 we notice that the initial average number of steps in all hierarchical methods is significantly lower, compared to the corresponding non-hierarchical methods. In addition, we notice that all multiagent algorithms have a significantly lower average number of steps than the single-agent algorithms during all episodes. Interestingly, the behaviour of MAXQ-Q becomes worse over time and this deserves to be investigated further. The results of Figure 2 clearly show that our proposed algorithms are the most efficient for the MATP, due to the combination of the strengths of the MAXQ decomposition and the multiagent

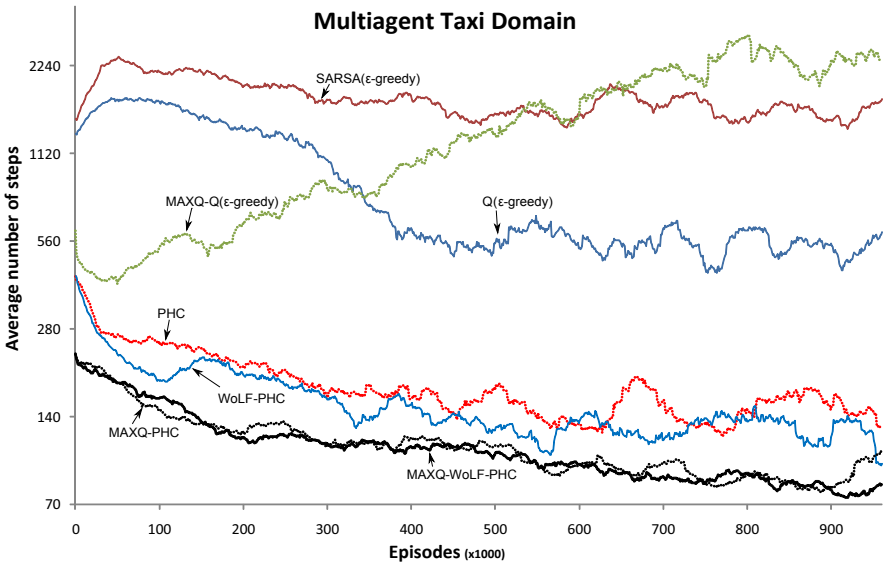


Fig. 2: Multiagent taxi domain results.

learning component that comes from maintaining an estimate of the stochastic policy and updating it accordingly. The single-agent algorithms have the worst performance as expected. The results of SARSA are worse than the results of Q-learning and this might be due to the fact that SARSA does not try to approximate a greedy policy (like Q-learning), but instead to approximate a "target policy" that is the same as its "behaviour policy", which can be disadvantageous in multiagent settings.

4 Discussion and Conclusions

In this paper, we investigated and compared some single-agent RL (SARL) algorithms using the taxi domain and the extended taxi domain, and some MARL algorithms using a multiagent extension of the taxi domain we created.

In particular, for the single-agent part, we first compared Q-learning and SARSA using both ϵ-greedy and Boltzmann exploration. We observed that Q-learning is more efficient with ϵ-greedy exploration compared to SARSA, which is more efficient with Boltzmann exploration. Then we compared two multiagent algorithms, PHC and WoLF-PHC, and we have observed that in the simple TP they have good results, even if they are algorithms for multiagent problems. In addition, WoLF-PHC is more efficient than PHC as we expected, because WoLF-PHC is an extension of PHC, which has been designed to improve it. Subsequently, we investigated and compared hierarchical algorithms and more specifically, algorithms based on the MAXQ decomposition. By trying the

MAXQ-Q algorithm for TP and ETP we have observed that MAXQ-Q is more efficient with ϵ-greedy exploration than with Boltzmann exploration. Finally, we extended the PHC and WoLF-PHC algorithms with the MAXQ decomposition and noticed that even if the results are not so good for TP, they improved for ETP where the state-action space increases. For the multiagent part, we first tried the Q-learning, SARSA and MAXQ-Q algorithms using ϵ-greedy exploration and observed that agents do not learn well, something we expected since these are algorithms for SARL. Then we compared the PHC and WoLF-PHC algorithms and observed that WoLF-PHC is more efficient than PHC. Finally, we compared the MAXQ-PHC and MAXQ-WoLF-PHC algorithms and observed that MAXQ-WoLF-PHC is slightly more efficient than MAXQ-PHC. Certainly a direct comparison with other studies using a multiagent taxi domain cannot be made, as their results are reported in a different way (see for example [6]).

In conclusion, our results have demonstrated that as the state-action space grows, the MAXQ decomposition becomes more useful, because MAXQ speeds up learning. Thus, agents learn faster when they use the MAXQ decomposition in both SARL and MARL problems. Additionally, in MARL problems, the introduction of the WoLF-PHC mechanism creates more adaptive agents.

Combining agents which use different algorithms could be further explored (heterogeneous settings). This would demonstrate how the behaviour of each agent would change and if agents can cooperate with other agents that use different algorithms. In the algorithms we used, the agents do not communicate. It would be worthwhile implementing an algorithm which gives the opportunity for agents to communicate, such as WoLF-Partial Sharing Policy (WoLF-PSP) [9] which is an extension of WoLF-PHC. This algorithm could undergo hierarchical extension using the MAXQ decomposition. Moreover, on-policy versions of the PHC and WoLF-PHC algorithms could be implemented and investigated, that combine the on-policy sampling mechanism of the SARSA algorithm, and the policy update mechanisms of the PHC and WoLF-PHC algorithms respectively. Furthermore, they could be extended hierarchically using the MAXQ decomposition. Additionally, the combination of methods that automatically find the hierarchical structure of the task (such as [8], [12]), with the WoLF-PHC mechanism could be explored as well. Finally, directed exploration methods [19] could be utilised and compared with the undirected exploration methods we used in this study (i.e., Boltzmann and ϵ-greedy).

It has to be noted that the curse of dimensionality is still a major problem for the work presented in this paper, since only lookup tables have been used rather than any form of function approximation. To illustrate the explosion of the state space, let us consider using the multiagent task in the extended taxi problem domain. A simple calculation would require 2.703GB of RAM per table (Q-table, policy or average policy), per agent. In addition, as it is well known, the explosion of state-action space does cause generalisation problems. For these reasons, it would be very beneficial if state abstraction or function approximation approaches are employed to our study above in order to reduce the number of parameters needed to achieve generalisation in such large state-action spaces.

References

1. Andre, D., Russell, S.J.: In: Leen, T.K., Dietterich, T.G., Tresp, V. (eds.) Advances in Neural Information Processing Systems 13 (NIPS '00), pp. 1019–1025. MIT Press, Cambridge, MA (2001)
2. Bowling, M.H., Veloso, M.M.: Artificial Intelligence 136(2), 215–250 (2002)
3. Dayan, P., Hinton, G.E.: In: Hanson, S.J., Cowan, J.D., Giles, C.L. (eds.) Advances in Neural Information Processing Systems 5 (NIPS '92), pp. 271–278. Morgan Kaufmann, San Francisco, CA (1993)
4. Dietterich, T.G.: Journal of Artificial Intelligence Research 13, 227–303 (2000)
5. Diuk, C., Cohen, A., Littman, M.L.: In: Cohen, W.W., McCallum, A., Roweis, S.T. (eds.) Proceedings of the 25th International Conference on Machine Learning (ICML '08). pp. 240–247. ACM, New York, NY (2008)
6. Fitch, R., Hengst, B., Suc, D., Calbert, G., Scholz, J.B.: In: Zhang, S., Jarvis, R. (eds.) AI 2005: Advances in Artificial Intelligence. Lecture Notes in Computer Science, vol. 3809, pp. 164–175. Springer (2005)
7. Ghavamzadeh, M., Mahadevan, S.: In: Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3 (AAMAS '04). pp. 1114–1121. IEEE Computer Society, Washington, DC, USA (2004)
8. Hengst, B.: In: Proceedings of the 19th International Conference on Machine Learning (ICML '02). pp. 243–250. Morgan Kaufmann, San Francisco, CA (2002)
9. Hwang, K.S., Lin, C.J., Wu, C.J., Lo, C.Y.: In: Huang, D.S., Heutte, L., Loog, M. (eds.) Advanced Intelligent Computing Theories and Applications. Lecture Notes in Computer Science, vol. 4681, pp. 422–432. Springer (2007)
10. Kaelbling, L.P.: In: Proceedings of the 10th International Conference on Machine Learning (ICML '93). pp. 167–173. Morgan Kaufmann, San Francisco, CA (1993)
11. Mehta, N., Tadepalli, P., Fern, A.: In: Driessens, K., Fern, A., van Otterlo, M. (eds.) Proceedings of the ICML'05 Workshop on Rich Representations for Reinforcement Learning. pp. 45–50. Bonn, Germany (2005)
12. Mehta, N., Ray, S., Tadepalli, P., Dietterich, T.: In: Proceedings of the 25th International Conference on Machine Learning (ICML '08). pp. 648–655. ACM, New York, NY, USA (2008)
13. Mirzazadeh, F., Behsaz, B., Beigy, H.: In: Proceedings of the International Conference on Information and Communication Technology, 2007 (ICICT '07). pp. 105–108 (2007)
14. Parr, R.: Hierarchical control and learning for Markov decision processes. Ph.D. thesis, University of California at Berkeley (1998)
15. Rummery, G.A., Niranjan, M.: On-line Q-learning using connectionist systems. Tech. Rep. CUED/F-INFENG/TR 166, Cambridge University (1994)
16. Shen, J., Liu, H., Gu, G.: In: Yao, Y., Shi, Z., Wang, Y., Kinsner, W. (eds.) Proceedings of the 5th International Conference on Cognitive Informatics (ICCI '06). pp. 584–588. IEEE (2006)
17. Singh, S.P.: Machine Learning 8, 323–339 (1992)
18. Sutton, R.S., Precup, D., Singh, S.: Artificial Intelligence 112, 181–211 (1999)
19. Thrun, S.B.: Efficient Exploration In Reinforcement Learning. Tech. Rep. CMU-CS-92-102, Carnegie Mellon University, Pittsburgh, PA (1992)
20. Watkins, C.J.C.H.: Learning from delayed rewards. Ph.D. thesis, University of Cambridge (1989)
21. Wiering, M., Schmidhuber, J.: Adaptive Behavior 6, 219–246 (1998)