Hierarchical reinforcement learning using path clustering

Paulo Gil
Instituto de Telecomunicações,
ISCTE-IUL – University Institute of Lisbon
Lisbon, Portugal
paulo.diniz.gil@ieee.org

Luís Nunes
Instituto de Telecomunicações,
ISCTE-IUL – University Institute of Lisbon
Lisbon, Portugal
luis.nunes@iscte.pt

Abstract— **In this paper we intend to study the possibility to improve the performance of the Q-Learning algorithm, by automatically finding subgoals and making better use of the acquired knowledge. This research explores a method that allows an agent to gather information about sequences of states that lead to a goal, detect classes of common sequences and introduce the states at the end of these sequences as subgoals. We use the taxi-problem (a standard in Hierarchical Reinforcement Learning literature) and conclude that, even though this problem's scale is relatively small, in most of the cases subgoals do improve the learning speed, achieving relatively good results faster than standard Q-Learning. We propose a specific iteration interval as the most appropriate to insert subgoals in the learning process. We also found that early adoption of subgoals may lead to suboptimal learning. The extension to more challenging problems is an interesting subject for future work.**

Keywords-hierarchical reinforcement learning; Q-Learning; performance; subgoals.

I. INTRODUCTION

Reinforcement Learning (RL) is a family of Machine Learning methods where a sequence of actions that leads to a goal is rewarded when the goal is achieved. The RL method will learn by trial and error, which are the best actions at each state to achieve the main goal and collect the best rewards. RL algorithms change the way they respond to the same inputs over time, improving the collected reward. Hierarchical reinforcement learning decomposes a reinforcement learning problem into a hierarchy of sub-problems (sub-tasks) such that higher-level tasks invoke branch tasks as if they were subroutines. This work aims to solve small/medium scale discrete RL problems efficiently by using path clustering to enable its hierarchical decomposition. Dietterich's taxi problem [1] is a standard in RL, especially in testing Hierarchical RL solutions and it was chosen to serve as the instantiation of our work. Originally this problem was solved using Q-Learning (QL) [2]. This algorithm was subsequently extended to make use of subgoals and options (QL+O) [3, 4] to maximize learning performance and minimize the number of actions that the agent takes to achieve its goal. We define the subgoals informally, as bottlenecks, or doorways between state regions. Options contain as a set of initial states (from which they can start) a policy to reach a given subgoal.

A Policy is a mapping from states to actions. RL methods specify how the agent changes its policy as a result of its experience.

II. PREVIOUS WORK

Hierarchical Reinforcement Learning is the designation given to RL methods that divide the problem into sub-problems such that solving each of the sub-problems is more efficient than solving the main problem directly. The most important advantage of hierarchical decomposition is the reduction in computational complexity of each sub-problem. The overall problem can be represented more compactly and sub-problems can be managed independently increasing its reusability and allowing to speed-up the learning. The implementation of subgoals discovery and options creation mechanisms followed the works described in [3, 4, 5]. Subgoals enable the agent to learn and use partial policies. Partial policies are comprised of consecutive actions that lead the agent from the current state to the next relevant subgoal. These sets of primitive actions are defined in [1] as temporally abstract actions. In [3] authors present the theory that support options and its benefits. Diverse Density is the method used in [4] that enables the agent to discover subgoals based on similarities between paths. Our work proposes an alternative method of discovering subgoals by classifying and intersecting paths. Taxi Problem was used to experiment the described implementation. Different approaches to this problem can be found in [1, 6, 7] and other interesting Hierarchical RL approaches are described in [8, 9].

III. METHODOLOGY

The Taxi-Problem is one of the most explored problems on Machine Learning. This problem's environment is composed of a five by five grid, containing walls between some specific positions (shown in Fig.1). Walls limit the actions between neighboring states, since the agent can't transpose those walls. The taxi-agent must pick-up a passenger and takes it to its final destination. At the beginning of each episode, the passenger and destination are randomly placed in or near the grid corners. The taxi can be initially located at any position in the grid. An episode starts with the unoccupied taxi. The taxi-agent must move to the passenger's position, execute a pick-up action, move to the passenger's destination and execute the drop-off action. The episode ends when the occupied taxi reaches the

All the authors are with Instituto de Telecomunicações, IT-IUL - Edifício II - Piso 0 Av. das Forças Armadas, 1649-026 Lisboa, Portugal and ISCTE- IUL, Av. das Forças Armadas, 1649-026 Lisboa, Portugal

destination and drops-off the passenger successfully. Initially, the taxi agent decides a percentage of its actions randomly, using ε-greedy. Repeating episodes a considerable amount of times, the agent should learn an action-choice policy that is adequate to improve the collected reward for each episode of this problem.

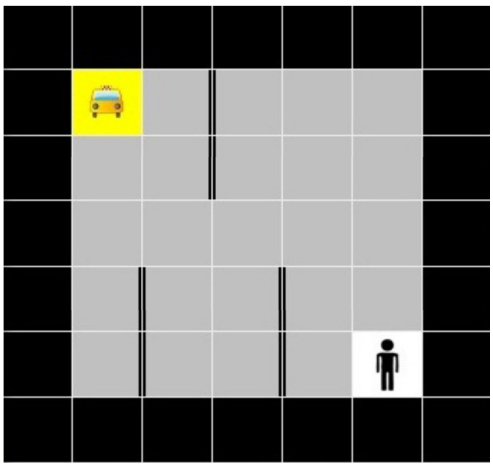

Figure 1. Example of a figure caption. (figure caption)

Initially, the agent is able to perform six distinct actions, i.e., moving actions, referring to move to any of the adjacent positions, pick-up and drop-off actions. This six are referred to as primitive actions. Upon reaching the state where the passenger is located, the taxi agent must perform the pick-up action in order to take it. In the same way, when the taxi agent has the passenger and reaches the state where the agent wants to be left, the agent needs to perform the drop-off action. Anytime the drop-off action is executed correctly, the main goal is reached and the episode ends, this is the condition of success that the agent seeks. A substantial reward is given to the agent only in case of success, and small negative rewards are given at each step. To speed-up the look-up of states a one-to-one hashing function was devised to translate state descriptions to integers.

A. General Q-Learning Process

QL is one of the most explored approaches to implement RL algorithms. The agent must initially interact with the environment through exploratory actions to promote its learning process. Using ε-greedy a part of the initial actions is random, enabling the agent to explore a large number of states and preventing the initially explored states to have an advantage and keep being selected as best paths to the goal. As the number of iterations increases the agent will have more information and choose better actions. The ε factor, that controls the exploration rate, is decreased during training, increasing the number of times the agent chooses the best action according to the current policy.

Briefly the QL algorithm will update a table containing the quality of each available action for each state. After each action, the agent should receive a reward that depends on the consequences of that action. Each state-action pair *(s, a)* is characterized by a quality. The quality of a specific *(s, a)* pair is proportional to its contribution to reach the main goal. The quality of the last *(s, a)* pair preformed, $Q(s,a)$, will be updated by [2] :

$$Q(s,a) = (1- \alpha) Q(s,a) + \alpha(r_{s,a} + \gamma max_a^t Q(st,at)). \quad (1)$$

Our implementation follows the classic QL to calculate the quality of a specific *(s, a)* the following simulation parameters were used: Learning rate $\alpha = 0.9$, and the discount factor $\gamma = 0.125$. These simulation parameters are the same as the ones used in [5] that achieved good results in a similar sized problem. Following the reward values described in [1], we used $r_{s,a} = 20$, when s = main goal state and a = drop-off action (final action that takes the agent to its main goal), $r_{s,a} = -10$ every time agent attempts to perform illegal pick-up or drop-off actions and $r_{s,a} = -1$ otherwise. Note that invalid actions are not executed but they are given negative reinforcement.

B. Subgoals

Subgoals can be described as important states between the agent's current state and the main goal. Comparing it to a real world situation, a subgoal could be seen as a bridge between two towns, the Taxi is located in town A and the passenger is waiting for the taxi in town B. The Taxi needs to cross it to reach the passenger's location.

Some classes of paths may include more subgoals than others. It depends on which side of the environment the passenger is located and which side he wants to be dropped-off. Our method to discover and use subgoals follows the ones described in [1, 4, 5, 10]. First of all we need concise data referring to the paths covered by the agent. As mentioned above, paths are lists of states. A path contains the states covered in single episode, excluding looping states and the main goal state. Every time the agent reaches a new state, a process verifies if it is already contained in the current list of covered states. If this condition is true, all states contained within the loop path are eliminated from the list. Paths are classified and intersected to discover mandatory states that will be subgoal candidates. Since it is likely that there are different subgoals between the initial state and the main goal, paths are classified in different classes. In this problem, path classification is easily solved because the class of each path depends only on passenger's initial and goal locations. There are however, techniques described in [5] to decide the class of a path in other problems where the class of a path cannot be determined by state characteristics alone.

1) Classes of Paths: One of the most decisive factors for the discovery of subgoals is the differentiation between classes of paths. Considering two episodes, in which the passenger's initial or goal location is different, even if all other parameters are the same, the states in the path will never be the same because these locations are part of the state. Thus, there are 16 classes of paths that should be identified and each path must be compared and intercepted only paths in the same class. These 16 different classes are the 16 possible combinations of the 4 different positions where the passenger can be initially located and the four different positions where it wants to be dropped-off. The following condition is used to group

different paths in classes: two paths having at least one state in common belong to the same class. It's verifiable that however different two paths may be, if they belong to the same class, they have, at least, one state in common, the state where the passenger is taken.

2) Path Intersection: Having distinguished and grouped the different classes of paths, it's possible to discover subgoals by intersecting those paths. The result of the interception process between all paths in the same class is a list of candidate subgoals. Depending on the exploration of the problem-state so far false positive subgoals may appear. False positive subgoals are states that could be present in all the paths of the same class however they are not mandatory states to reach the main goal. Its emergence is related to insufficient exploration performed until the moment the intersection process is triggered. If the exploration performed is sufficient to cover most of the environment states, no false positive subgoal should appear in this problem. Thus, it is extremely important to determine the subgoal discovery activation point. An example of a true subgoal is coordinate (3, 3) for passenger's start and goal positions in opposite parts of the map, when the taxi is holding the passenger.

C. Options

Options can be generically described as abstract and complex actions that can be executed by the agent, leading it directly to a subgoal, provided that the option contains the current state in its initial set. Having stored all paths relating to every class and their respective interceptions, which are actually the subgoals, we have all the necessary information to implement options. This implementation is divided in four important phases. Initially we need to create specific learning structures for each subgoal containing all states that report to it. These structures are composed of an initial set (states which lead to a given subgoal, i.e., the set of states for which the nearest subgoal is the same). The stop condition in this case was simplified to be true only when the subgoal is reached. We also need to add the option's initial quality to the main Q-Learning structure. Thereby the agent may trigger complex abstract actions over every state where one is available. Some states might not have abstract complex actions, for example, states located in loop areas or states that have not yet been covered by the exploration process.

1) Option's Learning Structure: The option's learning structure is identical to the original learning structure. Initially it is necessary to add states in the initial set to an option. For each subgoal a single Q-learning structure is created, containing the corresponding states and the qualities of theirs actions. These qualities are directly copied from the original Q-Learning structure. At the end of this process the specific learning structures associated to each subgoal are no more than general learning structure subsets. From now on these specific learning structures will evolve autonomously, i.e., each learns its own policy. We also created a correspondence method that labels the relation between states and correspondent specific learning structure, to expedite the option's matching process.

2) General Learning Structure Update Process: This process aims to update the states in the general learning structure for which new non-primitive actions are now available. Abstract complex actions are added to all states identified as being in the option's initial set. For a state, x, and its correspondent subgoal state, y, the x state option's quality is calculated by the arithmetic average of all maximum qualities from all states in the initial set of the option that leads to y. This option's quality calculation process allows the agent to reach subgoals quickly when starting from states far from the subgoal. This happens because the option's quality is often the highest for all possible actions in state x when the subgoal is far. In states close to the subgoal, the quality of the best primitive action will be superior to the option's quality, thus primitive actions will be preferred when the agent is close to the subgoal. This process of updating the option's quality is performed not only when the options are inserted in the learning mechanism but every time an option is executed.

3) Specific Q-learning Process: In general terms this process is identical to III-A and has the same quality assignment method. However there are some differences in the reward values attributed and the reinforcement conditions. As already explained, the final states of specific learning structures are subgoals. So for the rewards we used a moderate positive reward, $r_{s,a} = 10$ for the final action, that takes the agent to the correspondent subgoal state and $r_{s,a} = -1$ otherwise. It's not necessary to give strong negative rewards since there are no impossible actions included in these specific learning structures.

4) Complex Q-learning Process: When subgoals discovery is activated, options are included in the learning mechanism. With this approach the choice of actions should include one more action for all states where options are available. As before, when random actions or primitive actions are chosen, only the General Q-leaning Structure is updated. Every time an option is chosen to be executed the learning mechanism must keep the precise option's entry state. Entering the option, actions are chosen based on the Specific Q-learning Structure until the subgoal is reached. Following a Specific Q-learning Structure, the agent's temporary main goal is to reach the subgoal state. Following the same reasoning, General Q-learning Structure and the specific one are updated as described in III-A and III-C3. When the subgoal state is reached, positive reward is attributed to the last (s, a) pair relative to the specific Q-learning structure, as described in III-C3. A neutral reward is attributed to the same pair in the General Q-learning Structure as described in III-A and the state where the option was started is also updated in the General Q-learning structure with the new quality value for the (s, a) pair, s corresponding to the option's entry state and a corresponding to the option itself. This update follows the process described in III-C2. The main objective of Complex Q-learning Process implementation is

the achievement of a better learning performance compared to the General Q-learning Process. Moreover we want to investigate the learning process efficiency improvement of the Complex Q-learning Process in comparison with the General Q-learning Process.

IV. RESULTS

As a first experimental approach the number of environment states was limited to easily analyze the subgoal discovery results. So, the first experiments were limited to episodes which the taxi is initially located at (1, 1) coordinate, the passenger at (5, 5) and the ultimate goal at (5, 1). This limitation forces all paths collected to belong to the same class. This specific class was chosen for the first experiment because it is one of the most complexes. An initial exploration rate of 0.9 was used for these experiments and the subgoals discovery activation point was set for iteration number 100. The subgoals discovery results can be seen in Table 1.

TABLE I. SUBGOALS DISCOVERED WITH INITIAL EXPLORATION RATE OF 0.9 AND SUBGOALS DISCOVERY MECHANISM TRIGGERED AT THE 100TH ITERATION

Subgoal coordinate	Passenger Taken	Passenger's state	Main Goal
(3, 2)	False	(5, 5)	(5, 1)
(3, 3)	False	(5, 5)	(5, 1)
(5, 5)	False	(5, 5)	(5, 1)
(3, 3)	True	(5, 5)	(5, 1)
(3, 2)	True	(5, 5)	(5, 1)
(3, 1)	True	(5, 5)	(5, 1)
(4, 1)	True	(5, 5)	(5, 1)

These results are exactly what we were expecting. This experiment's purpose was to obtain all subgoals of a specific class, avoiding non-relevant states. In some experiments there is a trend for the appearance of superfluous states that can be seen as False-Positives. This occurs when a low exploration rate is used or when the subgoals discovery process is prematurely triggered. True subgoals, when discovered, will always prevail, no matter how many iterations or path interceptions are made. It is relevant to understand that every class of paths has at least one subgoal, the state where the taxi takes the passenger. This peculiar state serves as a passage between the first 25 environment states and the other 25 remaining states. Without exceptions the agent must walk through this passage to reach the main goal. When this point is reached we can assume that half of the path was traveled and the agent knows that he already has the passenger. From this point on, the states that the agent will cover, having the same coordinates, are different from the ones already covered in this episode because currently the agent has the passenger. In this manner, when the agent passes through this special state it arrives in a different environment which is complementary to the previous one.

The proper composition of general Q-learning and specific Q-learning structures also deserved special attention. Extensive conformity experiments were performed. From this point onwards all experiments were executed with no environment limitations or restrictions. This way the combined study of all classes of paths and all possible episodes for the classic Taxi-Problem is ensured. Performing experiments with all classes of paths makes it impossible to guarantee a concise environment exploration, so that the appropriate subgoals can be discovered. Another concern is how to decide the best subgoal discovery activation point. It is therefore important that these mechanisms might be triggered in time to help maximize the learning process. This research work objectives are not restricted to demonstrate that our extension accelerates the learning process but we also search for the combination of factors that produce the best results. With this in mind we decided to investigate the General Q-learning mechanism results and its behavior differences using distinct exploration rate values.

It is observable in Fig. 2 that lower exploration rates are recommended to quickly achieve a good performance. Results show that for low exploration rates, the agent tends to start using its learned actions at an early stage. This induces a very significant decrease in the number of iterations to reach the main goal at this stage. As the number of iterations increases, the optimization process continues and at the 1000th iteration the lower exploration rates already have near values to the best verified in this experiment. Good performance for higher exploration rates is time consuming and the optimum values are only reached after 10000 iterations and they are worse than the ones achieved by lower exploration values.

This experiment demonstrates that for relatively small scale problems and considering only the general QL method, a better learning performance is achieved making use of low exploration rates.

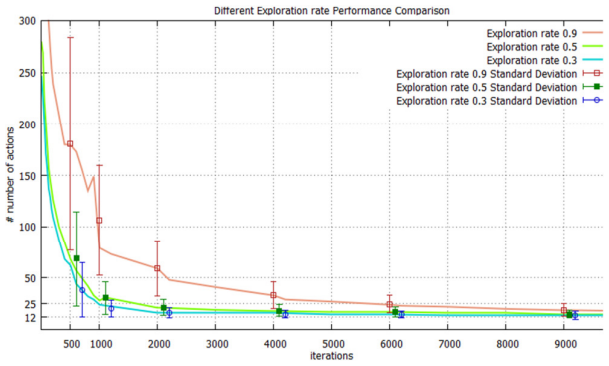

Figure 2. Example of a figure caption. Example Monte Carlo simulation with 30 runs for General Q-Learning with diverse exploration rates, exploration rate discount factor = 0.9, reducing the exploration rate each 1000 iterations.

Reaching the crucial phase of our experiments we tried to assert the extension of the benefits of QL+O. We consider that an entirely optimized learning process must be fast to obtain a good performance and tend towards the best performance. In real-time applications, the initial training delay is often intolerable. As referred above, since there are only 50 states within any single class a reduced number of iterations is necessary to visit all states in a class. However, to visit all states of all classes of paths in the Taxi Problem at least 16 iterations are necessary. So, some classes could be visited for the first time in advanced iterations. This fact considerably restricts the efficient subgoal discovery at early iterations. Each of these experiments was repeated 30 times. The subgoal discovery activation point was the most difficult parameter to

configure in these new experiments. Results in Fig. 3 indicate a performance improvement granted by the activation of the extension. However, different activation points result in different performance improvement characteristics.

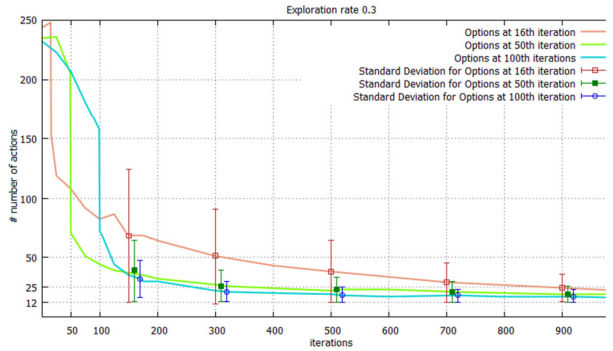

Figure 3. Monte Carlo simulation with 30 runs for each learning mechanism, exploration rate = 0.3, exploration rate discount factor = 0.9, reducing the exploration rate each 1000 iterations.

The effects triggered in the learning process for options at the 16th iteration (16QL+O) are very promising at first, however as the number of iterations increases the benefits fade. This is largely because it is almost impossible to cover all 16 classes of paths at the16th iteration, $P \approx 1^{-6}$ and the subgoals discovered referring to the classes of paths covered may not be the best ones. These subgoals discovery and options activation initially accelerate the agent's performance, however long term performance will be affected due to its premature activation.

The performance improvement triggered by the options inclusion at the 50th (50QL+0) and 100th (100QL+O) iterations are much more interesting. These two performance lines present identical characteristics however there are subtle differences between the two.

For 50QL+O, the performance gain starts earlier, this way it reaches a good performance faster. For 100QL+O, the performance gain starts later but it converges faster to an almost perfect performance.

After testing many different activation points and their results we conclude that the optimum point to activate should be between the 50th and 100th iteration. From now on we decided to compare the performance differences between general QL, referenced as No Options, and QL+O activated at optimum points. As we can observe in Fig. 4 the lines referring to QL+O present a better performance than general QL.

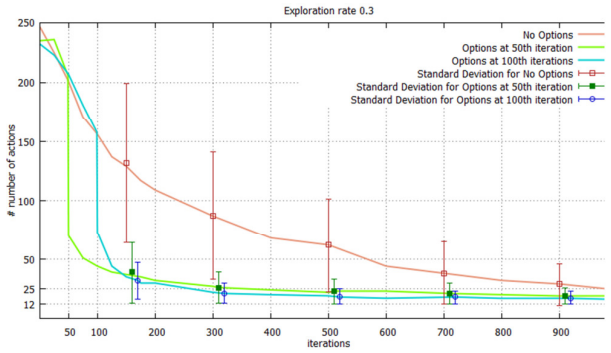

Figure 4. Monte Carlo simulation with 30 runs for each learning mechanism, exploration rate = 0.3, exploration rate discount factor = 0.9, reducing the exploration rate each 1000 iterations, version 2.

Inspecting the different performances at the 150th iteration there is a visible average gain around 72% for 50QL+O and 75% for 100QL+O. Performance differences at the 300th iteration are almost identical. Standard deviation differences between the QL+O and the general QL are also interesting to observe. At the 150th iteration, the standard deviation maximum point for 50QL+O is equal to general QL standard deviation minimal point. In its turn standard deviation maximum point for 100QL+O is even lower. It's common to verify disjoint intervals for iterations near the options' activation point. These results demonstrate the initial boost of performance triggered by QL+O.

As the number of iterations increases the performance gains become less noticeable, nevertheless at the 1000th iteration the performance of general QL is still worse than the QL+O's performance. Another important fact is related to the point where different implementations reach a near-optimal average number of actions. Considering 25 the near-optimal average number, it is interesting to note that the QL+O lines reach this point around iteration 300. In turn, general QL reaches the same point at the 1000th iteration. So there is a gap of 700 iterations between these implementations' performance.

V. CONCLUSIONS

In this paper we intended to improve the performance of the Q-Learning algorithm, by making use of subgoals and options. We expected to speed up the agent's learning process over the discreet environment of taxi-problem. We considered that an optimized learning process must be fast to obtain a good performance and tend towards the best performance. In real-time applications, the initial training delay is often intolerable. So, we also searched for the combination of factors, as learning parameters and subgoal discovery activation point, which produce the best results.

Initially more attention was given to the discovery of subgoals that could efficiently optimize our learning effort. Actions learned through interaction within a specific class of paths are meaningless to the remaining classes. So, simple learning transfer between classes is not a valid solution. There are a total of 800 different states distributed among 16 classes of paths. True subgoals when discovered will never disappear

due to the action of subgoals discovery process. False-Positive subgoals may appear due to insufficient exploration performed at subgoals discovery activation point.

Regarding to option's creation process, some states may not acquire abstract actions, i.e., options. In accordance with this restriction, those states aren't represented in any specific learning structure. This is explained by the fact that those states are located in looping areas, not being considered for option's creation. For (s, a) qualities of a random specific learning structure, the action with higher quality value within a random state is the one who brings the agent closer to its specific subgoal. It is interesting that during the creation of specific learning structures, the quality values of some (s, a) pairs are discarded. Those pairs are relative to invalid actions. It is also important to mention that we used different reward values and reinforcement conditions for specific learning structures. So each specific learning structure will evolve autonomously.

Inspecting the results, our options enable the agent to improve their performance. However the options optimization depends, initially, on the exploration level attained when the subgoal discovery is triggered, i.e., the quality of the subgoals discovered. We can conclude that the quality and effectiveness of the subgoals and options are intrinsically related to the amount of different states that the agent already covered. So it is clear that the quality of subgoals and options is somehow relative, as it must be balanced with the accurate point to trigger the subgoal discovery, under penalty of not bringing significant benefits to the learning process. Furthermore, triggering the extension at early stages implies that subgoals and options used are not the best, which leads to some immediate improvement that may not last long.

Performing intensive experiments, we pointed out the optimum iteration range to trigger the options mechanism in this problem. It's important to remember that this iteration range is relative to a set of parameters that granted the best results. Other simulation parameters may result in different optimum points to trigger the options mechanism.

In conclusion the key issue for the effectiveness of the extension is the ideal point to include it in the learning process. Adjusting the exploration rate and the extension's activation point, in order to get the best performance, we were able to achieve an initial learning improvement around 75%. It is important to notice that this experiment's parameters allow the extension to still get the most optimized solution. The achieved results are the same order as [4].

As future work, we intend to extend our method to larger size discrete problems and to the continuous domain.

ACKNOWLEDGMENT

The authors wish to thank Dr. Luís Ducla Soares, Dr. Pedro Sebastião and Dr. Rui Lopes of ISCTE-IUL for valuable comments and discussion.

REFERENCES

[1] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," J. Artif. Intell. Res. (JAIR), vol. 13, pp. 227–303, 2000. [Online]. Available: http://www.informatik.uni-trier.de/~{}ley/db/ journals/jair/jair13.html#Dietterich00

[2] C. Watkins and P. Dayan, "Technical note: Q-learning," Machine Learning, vol. 8, no. 3, pp. 279–292, May 1992. [Online]. Available: http://dx.doi.org/10.1023/A: 1022676722315

[3] R. S. Sutton, D. Precup, and S. P. Singh, "Between mdps and semi-MDPs: A framework for temporal abstraction in reinforcement learning," Artificial Intelligence, vol. 112, no. 1-2, pp. 181–211, 1999. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary? doi=10.1.1.45.5531

[4] A. Mcgovern and A. G. Barto, "Automatic discovery of subgoals in reinforcement learning using diverse density," in ICML, C. E. Brodley, A. P. Danyluk, C. E. Brodley, and A. P. Danyluk, Eds. Morgan Kaufmann, 2001, pp. 361–368. [Online]. Available: http://dblp.uni-trier.de/rec/bibtex/conf/icml/McGovernB01

[5] D. F. Jardim, "Hierarchical reinforcement learning: Learning sub-goals and state-abstraction," Master's thesis, ISCTE- University Institute of Lisbon, 2010.

[6] B. Hengst, "Generating hierarchical structure in reinforcement learning from state variables," in Lecture Notes in Artificial Intelligence. Springer, 2000..

[7] F. Mirzazadeh, B. Behsaz, and H. Beigy, "A new learning algorithm for the maxq hierarchical reinforcement learning method," in Information and Communication Technology, 2007. ICICT '07. International Conference on, march 2007, pp. 105 –108.

[8] A. G. Barto and S. Mahadevan. (2003) Recent advances in hierarchical reinforcement learning. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/ summary?doi=10.1.1.4.6238

[9] O. Simsek, A. P. Wolfe, and A. G. Barto, "Identifying useful subgoals in reinforcement learning by local graph partitioning," in Proceedings of the 22nd international conference on Machine learning, ser. ICML '05. New York, NY, USA: ACM, 2005, pp. 816–823. [Online]. Available: http://doi.acm.org/10.1145/1102351.1102454

[10] R. Parr and S. Russell, "Reinforcement learning with hierarchies of machines," in Advances in Neural Information Processing Systems, M. I. Jordan, M. J. Kearns, and S. A. Solla, Eds., vol. 10. The MIT Press, 1997. [Online]. Available: http://citeseer.ist.psu.edu/parr97reinforcement.html