Two Steps Reinforcement Learning

Fernando Fernández,* Daniel Borrajo†
Universidad Carlos III de Madrid Avda/ de la Universidad 30, 28911-Leganés, Madrid, Spain

When applying reinforcement learning in domains with very large or continuous state spaces, the experience obtained by the learning agent in the interaction with the environment must be generalized. The generalization methods are usually based on the approximation of the value functions used to compute the action policy and tackled in two different ways. On the one hand by using an approximation of the value functions based on a supervized learning method. On the other hand, by discretizing the environment to use a tabular representation of the value functions. In this work, we propose an algorithm that uses both approaches to use the benefit of both mechanisms, allowing a higher performance. The approach is based on two learning phases. In the firs one, a learner is used as a supervized function approximator, but using a machine learning technique which also outputs a state space discretization of the environment, such as nearest prototype classifier or decision trees do. In the second learning phase, the space discretization computed in the firs phase is used to obtain a tabular representation of the value function computed in the previous phase, allowing a tuning of such value function approximation. Experiments in different domains show that executing both learning phases improves the results obtained executing only the firs one. The results take into account the resources used and the performance of the learned behavior.

1. INTRODUCTION

The reinforcement learning task (RL) can be define as the automatic acquisition of an action policy by some agent in a given environment. The learning phase is define by two main properties. The firs one is that learning is performed following a trial and error interaction of the learning agent with its environment. Through this interaction, the agent obtains the experience required for learning to solve a task. The second property refers to the information included in that experience, which is typically based on states, actions, and a reinforcement signal. Such a reinforcement signal indicates to the agent how good the execution of the last action to solve the task has been[1].

Model free RL algorithms[2] are based on the computation of the action-value function, *Q(s,a)* for each policy π. This function is define as the expected reward that will be received by an agent if it is located in a state s, the firs action that

*Author to whom all correspondence should be addressed: e-mail: ffernand@inf.uc3m.es.
†e-mail: dborrajo@ia.uc3m.es.

it executes is action a, and then, it follows the action policy π. If the optimal Q function is computed, the optimal action policy can be computed too, so most RL literature focuses on computing and representing such a function.

The Q-learning algorithm[3] is one of the most widely used ones for computing the action-value function. Given any experience tuple of the type $\langle s, a, s', r \rangle$, where s is a state, a is an action, s' is the state achieved when executing a from s, and r is the immediate reinforcement received, it updates the Q function following Equation 1.

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')] \tag{1}$$

This equation uses the discount parameter γ, following a discounted infinit horizon criteria of optimality. Also, it introduces a learning parameter α. If α is 1, the equation is known as the deterministic update function of Q-learning, and it is simplifie as shown in Equation 2.

$$Q(s, a) \leftarrow r + \gamma \max_{a'} Q(s', a') \tag{2}$$

Q-learning updates the value of Q for a define action and state, so the update only affects them, and no other states and/or actions. The optimal Q function is progressively approximated while updates are performed. One problem of this approach is that if we have very large or continuous state and/or action spaces, performing enough updates for every state and action so that the Q function is correctly approximated requires a large amount of experience.

Given that this approach is unpractical in some cases (because typically, the amount of experience is limited), it is preferable that the updates performed for an action and a state modify the value of Q, not only for them, but also for similar states and actions. Implementing this approach consists of findin a function approximator, $\hat{Q}_\theta$, whose output depends on a reduced set of parameters θ. The idea is that a modificatio in one of these parameters produces a different output of the approximator for different states and actions. So, small modification in θ, which are due to a new approximation value of $\hat{Q}_\theta$ for a define action and state, may produce changes in the $\hat{Q}_\theta$ approximation for a large set of actions and states.

The motivation of this work is to fin a method to compute the action-value function in model free RL. The method assumes a reduced set of actions and finit trials, where positive and discrete rewards can be obtained only when a goal area is achieved. It is based on findin discretizations of the state space that are adapted to the value function being learned. In this case, we learn the action value function, $Q(s, a)$, instead of the state value function, $V(s)$, learned in dynamic programming.[4] The method is based on two learning phases. The firs one is a model free version of the smooth value iteration algorithm,[5] which we have called iterative smooth Q-learning (ISQL). This algorithm, which executes an iterative supervised learning of the Q function, can be used to obtain a state space discretization too. This new discretization is used in a second learning phase, which we have called multiple discretization Q-learning (MDQL), to obtain an improved policy. Both learning phases, ISQL and MDQL, compose the algorithm that we have called two steps reinforcement learning (2SRL).

The paper is organized in two folds. The firs one begins in the following section with a brief summary of the generalization problem, introducing some of the main approaches used, but deepening in variable resolution discretization methods. Sections 3 and 4 show two approaches based, one on supervised learning function approximation (ISQL), and the other on state space discretizations. Section 5 discusses the disadvantages and limitations of both approaches with some initial experimentation.

The second fold of the paper begins in Section 6, where the 2SRL method is presented, describing the improvement over the results obtained by previous approaches, as it is shown in Section 7. Section 8 shows the experiments performed over the car on the hill domain. Also, it describes some comparative results of the new method with the results obtained by variable resolution discretizations (VRD)[6] over the Acrobot domain. Lastly, Section 9 presents the conclusions and further research.

2. RELATED WORK

Function approximation of the value functions in reinforcement learning[7] has been addressed mainly in two different ways. The firs one consists of discretizing the state space to obtain a reduced and discrete one, so tabular representations of the value functions can be used. In this case, each of the parameters in θ define a different region of the state. Uniform discretization has achieved good results,[8] but only in low-dimensional domains, where a high resolution does not increase the number of states too much. A very large state space produces unpractical computational requirements in dynamic programming methods, and unpractical amounts of experience in model-based methods.[9]

Variable resolution methods[10] are applied to increase the resolution of the discretization only in the areas where it is required, i.e., in the areas where differences in the value function and/or in the policy are found.[6] A kd-tree is typically used to represent the state space, so each leaf contains the approximation of the value function for the represented state. These leaves can be split into two, whenever is needed, following different criteria. A model-free version of this approach has also been introduced.[11] In that case, the leaves of the kd-tree contain the L $Q_{a_i}(s)$ values, for $i = 1, \ldots, L$, one for each action, instead of a single $V(s)$ value. The problem of this approach is that the split mechanism of a leaf depends on differences on the $Q_i(s)$ value of any action, so a resolution that could be good for an action may be increased because the need of different actions. To solve this, one discretization per action should be required, as it will be introduced later in this paper. New developments on VRD also include the action space in the discretization process, generating a discretization of the "joint" space,[12] and hence, automatically discretizing the action space too.

The discretization approach is used in Section 3, where the VQQL algorithm is described. However, we will show that discretization methods may obtain poor results if they are not supervised by the action value function being learned, mainly because they can lose the Markov property.[13] Continuous U trees has also been used for discretizing the state space, avoiding violations of the Markov assumption.[14]

The second approach for learning the Q function consists of the supervised learning of tuples $\langle s, a, q_{s,a} \rangle$, where s is a state, a is an action, and $q_{s,a}$ is the $\hat{Q}$ value approximated for that state and action. Any supervised learning method can be used, each of them definin a set of parameters θ to be computed.[15–17] In the literature, several related work based on this approach can be found, as the application of neural networks for playing Backgammon.[18] In this case, the set of parameters θ to compute is the architecture of the network and the weights of the different links among the neurons, so that the neural network outputs a good approximation of the optimal Q function. Instance-based methods[19] can also be applied, as well as locally weighted regression,[20] or self organizing maps.[21] This approach is studied in Section 4, where we defin the iterative smooth Q-learning (ISQL) algorithm, based on the smooth value iteration algorithm,[5] which will be tested using different function approximators.

3. A DISCRETIZATION-BASED APPROACH

The firs approach for function approximation that we present here is based on the discretization of the state space. An instance of this approach is the VQQL model, define next.

3.1. VQQL

The VQQL algorithm (vector quantization for Q-learning)[22,23] is based on the unsupervised discretization of the state space. We use vector quantization methods, and more specificall , the generalized Lloyd algorithm (GLA),[24] also called k-means. This method generates a set of prototypes that together with the nearest neighbor rule define Voronoi regions.[25] Each of these regions clusters a set of states of the original state space representation. Figure 1 shows how to represent the action policy following this approach.

The use of vector quantization and the generalized Lloyd algorithm to solve the generalization problem in RL algorithms requires two consecutive phases:

Learning the quantizer. Designing the vector quantizer, $D(s)$, is composed of B prototypes from the input data obtained from the environment during exploration.

Learning the Q function. Once the vector quantizer is designed (i.e., the environment is clustered into B different states), the Q function must be learned, generating the Q table composed of B rows and a column for each action.

The VQQL algorithm was firs applied in the simulation league within the RoboCup domain.[26] The vector quantization technique takes advantage of the statistical characteristics of the domain to reduce its size, considering only relevant areas from the whole domain. The algorithm has been applied to learn the ball interception skill of a goalie,[22] and in cooperative multi-robot observation of multiple

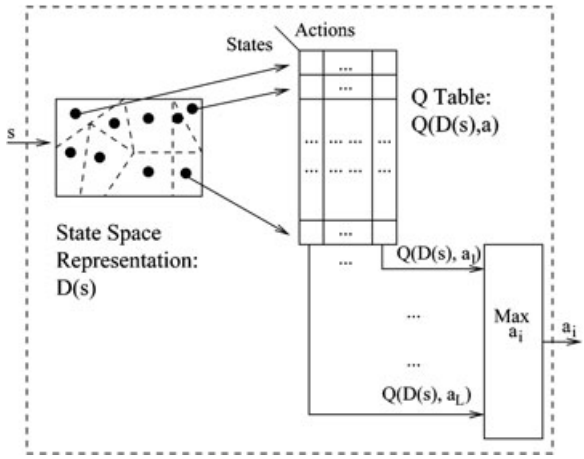

Figure 1. Function approximation with VQQL.

moving targets (CMOMMT),[27] to obtain collaborative behaviors in a multirobot observation task.[23,28] These experiments show that the nearest prototype approach used to discretize the state space produces better results than uniform discretizations. However, it still has two main problems. The firs one is that the right number of prototypes or regions must be decided by the designer. Depending on the problem, a higher or lower resolution could be required, and different learning and test processes are required to fi that parameter.

The second one is the loss of the Markov property,[13] given that the discretization is unsupervised, so regions that break the Markov property may be introduced, as it is described next.

3.2. Loosing the Markov Property: Nondeterminism Introduction

When the state space is discretized, it is possible that the new state space discretization does not satisfy the Markov condition. The loss of the Markov property has as its main consequence the introduction of nondeterminism in the domain,[13] as described in the following example. The classical exploration domain or grid world, as shown in Figure 2a, is a domain where a robot has to learn to arrive to the goal area from random initial positions in a continuous state space. For simplicity reasons, we will use a version where robot actions are limited to follow cardinal points, i.e. "go East," "go North," "go West," and "go South." The grid size is 5×5, and all actions produce a motion of size one. If the robot tries to execute an action that ends out of the grid, the robot is forced to stay in the same position. This situation can be considered as a blocked situation or a cycle of size one. Longer cycles could be found if the robot oscillates among several states.

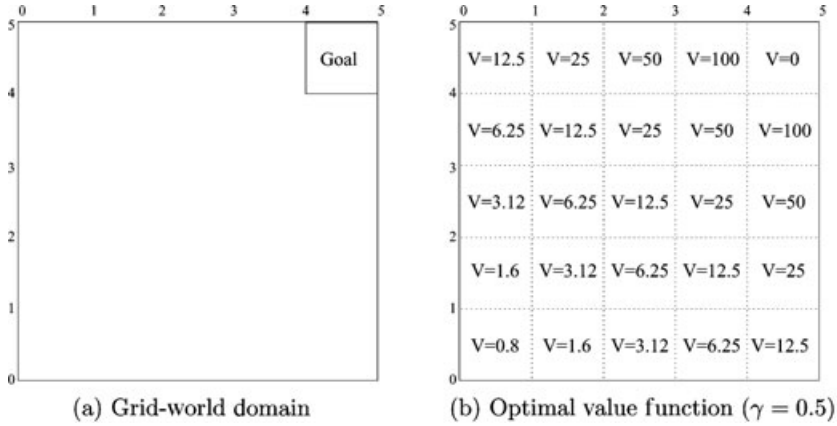

Figure 2. Grid-world domain. (a) Shows the goal area and (b) the optimal value function for $\gamma = 0.5$.

If we use a reinforcement function that returns a maximum reward of 100 when the robot arrives to the goal area and 0 otherwise, and we use a discount factor of $\gamma = 0.5$, for each executed action, the optimal value function for each cell is shown in Figure 2b. At the same time, the limits of the value function present an optimal state space representation that does not lose the Markov property nor introduces nondeterminism. The limits of the different regions are define by the discontinuities of the value function, so in this case, only 25 discretized states are needed. The execution of any action from any state in the same region always produces a state transition to the same region, receiving the same reward.

Although this is a deterministic domain, it is easy to show that obtaining a state space representation that keeps the Markov property and, hence, the determinism in the environment without the use of any prior knowledge is not an easy task. For instance, since "a priori" we might not know the number and geometry of the regions, instead of introducing 5×5 regions, a 6×6 discretization could have been used. The result is illustrated in Figure 3. This approximation of the environment loses the Markov property and introduces a high nondeterminism because, even if the execution of the same action from the same region always achieves the same fina region, in some executions the goal could be achieved, while not in others. So immediate rewards can be different in different executions, resulting in a non-Markovian and nondeterministic behavior.

The main problem of this nondeterminism introduction is that the optimal value functions cannot be correctly approximated, and hence, optimal policies for such a problem may not be learned either.[13] Next section shows a method that theoretically outputs state space discretizations which maintain the Markov property.

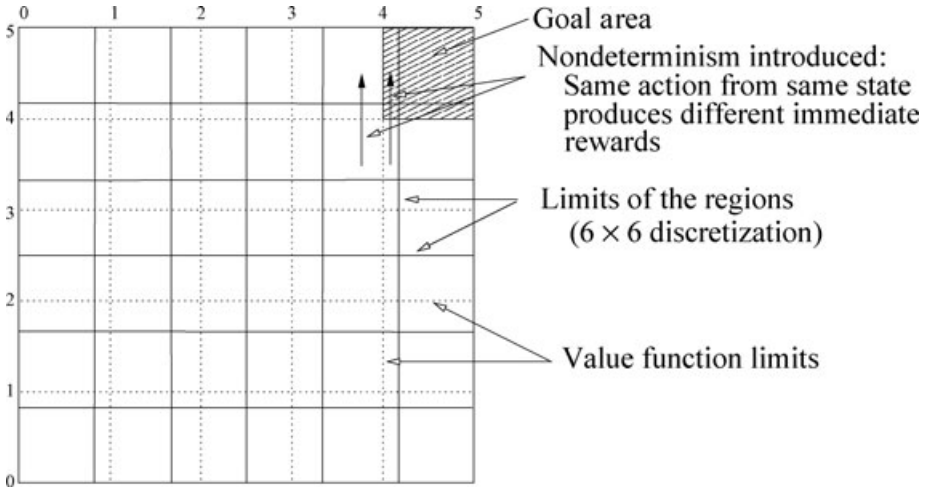

Figure 3. Example of uniform discretization with 6 × 6 regions in the grid-world domain.

4. A RL ALGORITHM BASED ON A SUPERVISED LEARNING FUNCTION APPROXIMATOR

Smooth value iteration[5] is define as a version of the discrete value iteration,[4] where a function approximator is used instead of the tabular representation of the value function, so the algorithm can be used in the continuous state space case. That algorithm requires a model of the environment. However, we have transformed it into a model-free version, following the ideas introduced for the ENNC-QL algorithm[9,29] and the fitte Q iteration algorithm.[30] In this case, instead of having a model of the environment as input, it needs a set of experience tuples obtained from the interaction of the agent with the environment. Furthermore, the cost-to-go approximation of the original smooth value iteration can be changed to a delayed reward one. Thus, a maximum reward, $r_{\max}$, is received only when the goal is achieved. A discount parameter, γ, is also applied. This version, which we call *iterative smooth Q-learning* (ISQL), is shown in Figure 4. The update equation of the Q function used is based on Q-learning.

The algorithm assumes a discrete set of L actions, and hence, it will generate L function approximators, $Q_{a_i}(s)$. Using one function approximation per action is not required really, given that the action could be introduced as an input of the function approximator, as it is done with the state.[20,31] However, this solution typically introduces the problem of combining heterogeneous feature spaces (the action and the state spaces) that, depending on the function approximation used, might require feature weighting methods,[32] and that could mitigate the possible generalization advantages of this approach.

The algorithm requires a collection of experience tuples, T. Different methods can be applied to perform this exploration phase, from random exploration to human-driven exploration.[33] In each iteration, from the initial set of tuples, T, and using

Iterative Smooth Q-Learning

- Inputs:
  1. A state space X and a goal area $G \subset X$
  2. A discrete set of L actions, $A = \{a_1, \ldots, a_L\}$
  3. A collection T of N experience tuples of the kind $< s, a_i, s' >$, where $s \in X$ is a state where action a_i is executed and s' is the resulting state
- Generate L initial approximators of the action-value function $\hat{Q}^0_{a_i} : X \rightarrow \mathcal{R}$, and initialize them to return 0
- $iter = 1$
- Repeat
  - For all $a_i \in A$, initialize the learning sets $T^{iter}_{a_i} = \emptyset$
  - For j=1 to N, using the j^{th} tuple $< s_j, a_j, s'_j >$ do
    - $c_j = \begin{cases} r_{max} & \text{if } s'_j \in G \\ \max_{a_r \in A} \gamma \hat{Q}^{iter-1}_{a_r}(s'_j) & \text{otherwise} \end{cases}$
    - $T^{iter}_{a_j} = T^{iter}_{a_j} \cup \{< s_j, c_j >\}$
  - For each $a_i \in A$, train $\hat{Q}^{iter}_{a_i}$ to approximate the learning set $T^{iter}_{a_i}$
  - $iter = iter + 1$

  Until r_{max} is propagated to the whole domain
- Return $\hat{Q}^{iter-1}_{a_i}, \forall a_i \in A$

Figure 4. *Iterative smooth Q-learning* algorithm.

the approximators $\hat{Q}^{\text{iter}-1}_{a_i}(s)$, $i = 1, \ldots, L$, generated in the previous iteration, the Q-learning update rule for deterministic domains can be used to obtain L training sets, $T^{\text{iter}}_{a_i}$, $i = 1, \ldots, L$, with entries of the kind $\langle s_j, c_j \rangle$, where c_j is the resulting value of applying the Q update function to the training tuple j, whose state is s_j.

In the firs iteration, $\hat{Q}^0_{a_i}(s)$ are initialized to 0, for $i = 1, \ldots, L$, and all $s \in S$. Thus, when the respective c_j are computed, they depend only on the possible values of the immediate reward, r. If we suppose the r values are always 0, except when a goal state is achieved, where a maximum reward of $r_{\max}$ is obtained, the only two values for c_j in the firs iteration are 0 and $r_{\max}$. This is illustrated in Figure 5 for the domain presented in Section 3.2. The figur shows that for goal areas, the Q value is 0. For actions "Go East" and "Go North," instances with $c_j = r_{\max}$ appear, while with the other two actions, all the state space stays with a value of zero (no state can achieve the goal with a "Go West" nor a "Go South" action).

If we suppose that we are able to perfectly approximate the sets $T^0_{a_j}$, in the following iteration, there will be experience tuples $\langle s, a_i, s' \rangle$ for which some $\hat{Q}^1_{a_i}(s')$ will be $r_{\max}$, so examples of the kind $\langle s, \gamma^1 r_{\max} \rangle$ will appear, and a new approximator will be learned with this new data, as it is shown in Figure 6. By repeating this process

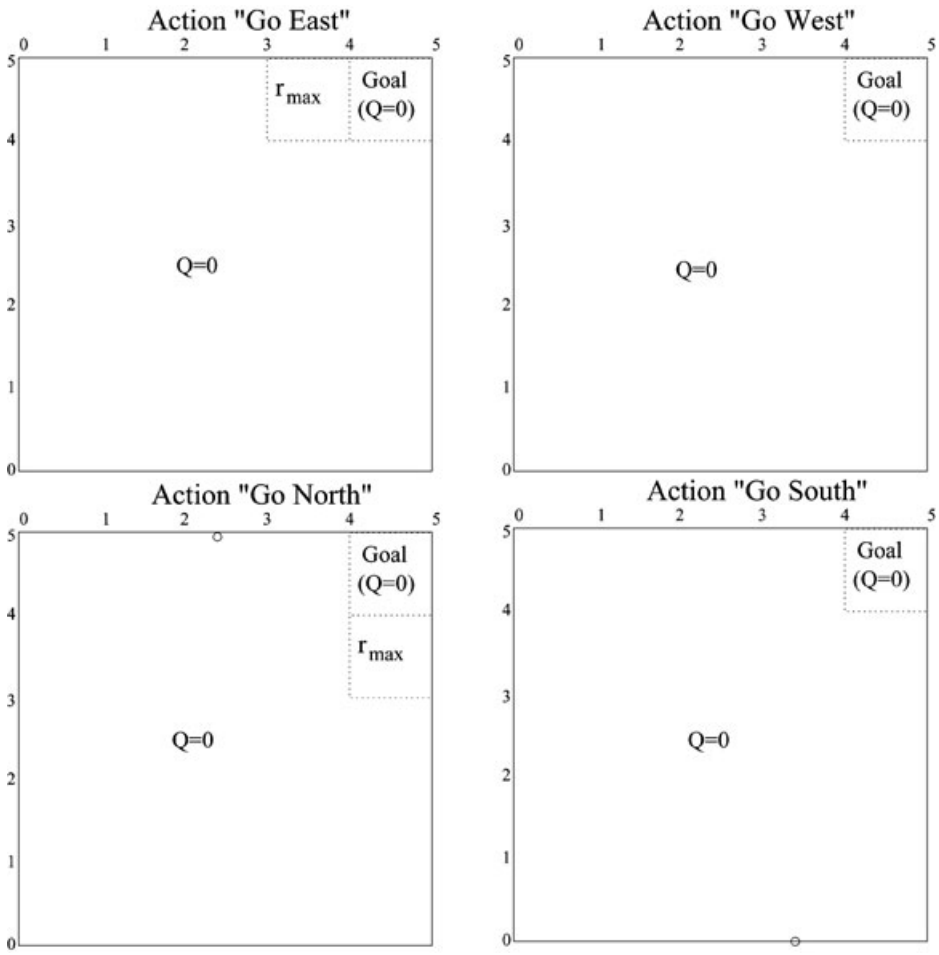

Figure 5. Expected Q function learned after firs iteration of ISQL.

iteratively, the whole domain will be learned from examples of the kind $\langle s, \gamma^{\text{iter}} r_{\max} \rangle$, for $iter = 1$ to the number of iterations executed, as define in Figure 7.

Another difference with the original algorithm is the end condition. Smooth value iteration uses as end condition that the c_j values computed remain the same in two consecutive iterations for all the state transitions. However, that end condition might not be achieved, depending on the function approximator used.[5] The end condition of our variant is that the reward has been propagated from the goal areas to the whole domain, i.e., there is no state with the initial Q value of zero except the goal area. This end condition is only useful if the goal areas can be reached from any position of the state space. If this property is not ensured, the number of iterations required is the length of the optimal path (measured as the number of executed actions) from the furthest position of the environment to the goal, say max_path.

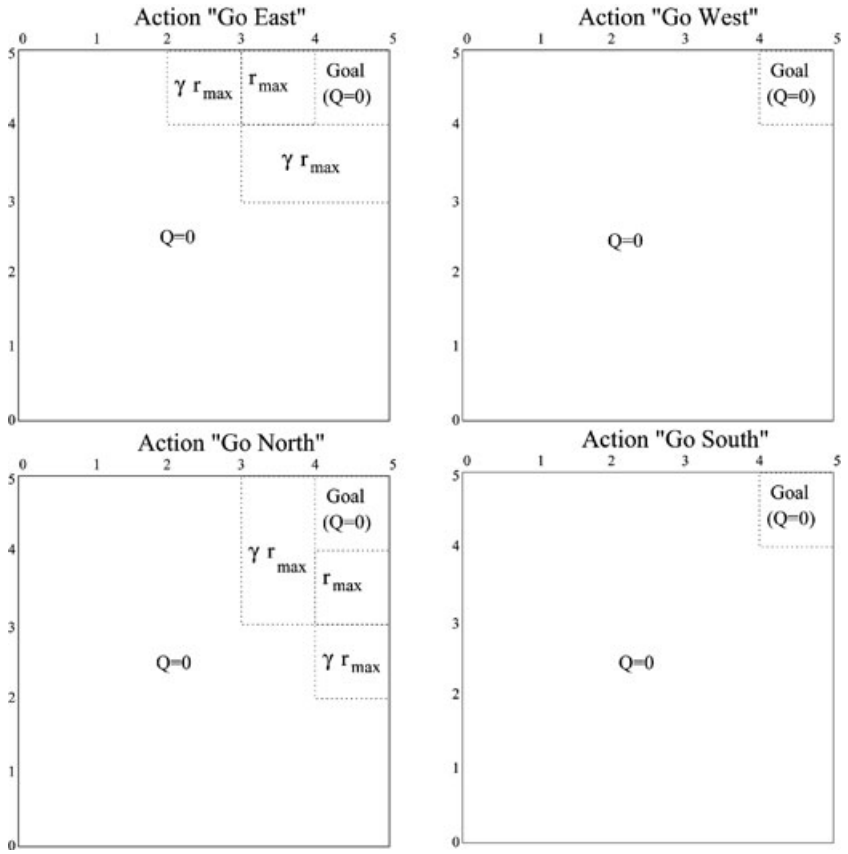

Figure 6. Expected Q function learned after second iteration of ISQL.

So, theoretically, the number of iterations should be, at least, max_path. In the example, this number is 9, and the optimal discretization for each action is given in Figure 7.

As in the case of,[5] we assume absorbing goal states reachable from every state in the domain, deterministic state transitions, and a limited set of discrete actions of a define size. Furthermore, if we assume discrete time, and actions taking one unit of time to be executed, the reinforcement learning problem can be addressed as a problem where the value functions can only take a value in a discrete set $\{\gamma^i r_{\max}, i = 1, \ldots, max\_path\}$. Therefore, we can apply a supervised learning algorithm with a discrete set of classes. So we can differentiate among different regions of the original state space, only by verifying the class to which they belong. From a variable resolution discretization point of view, the optimal discretization of the domain for each action (given that each action requires a different approximation) is given by the limits generated in the corresponding action value function computed by the iterative smooth Q-learning algorithm, if the assumptions previously introduced hold.

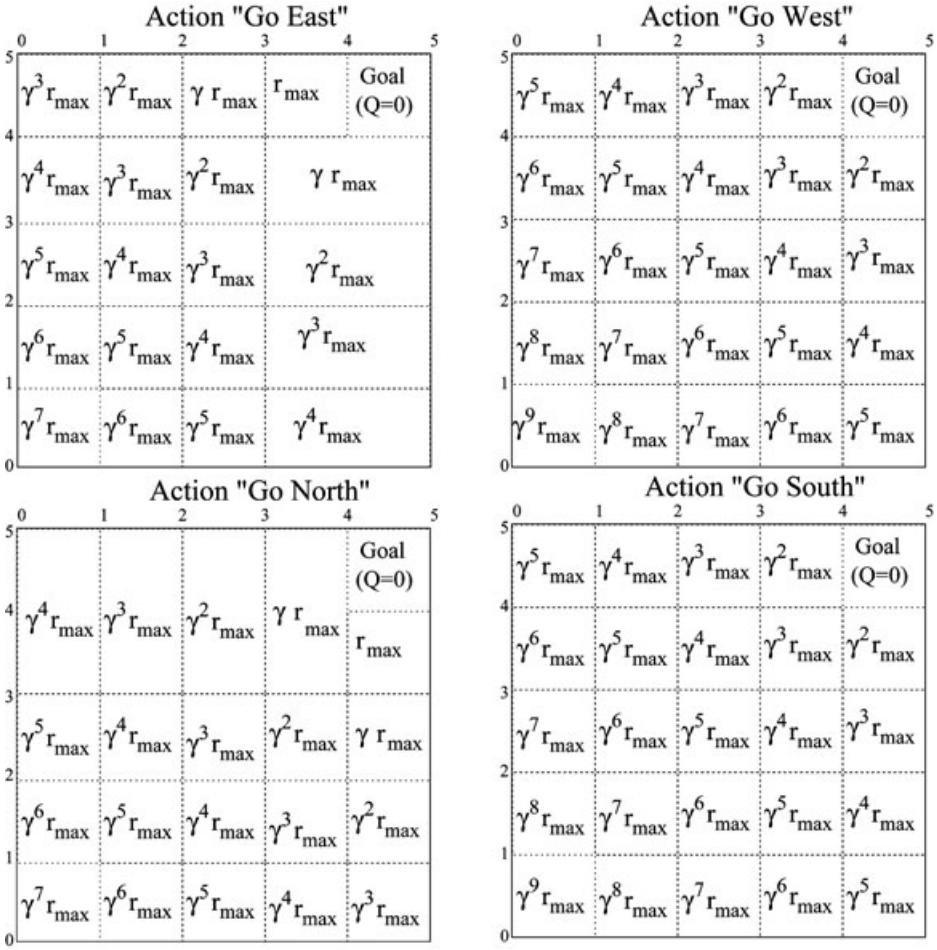

Figure 7. Expected Q Function learned after nine iterations of ISQL, that is the optimal Q function. Borders of the Q-function show a Markovian discretization of the state space.

Furthermore, an important issue of the discretizations obtained is that they hold the Markov property, in the sense that if we execute an action from any point in a region, we will always obtain the same reward, and the same V value for the achieved state, which means that the states in the same region will always receive the same Q value update.

If the domain is stochastic, the algorithm can be applied as well without any modification From a practical point of view, stochastic behavior means that learning examples $< s, c >$, generated for learning the function approximators in each iteration, will not exactly follow the distribution shown in Figures 5 to 7, but some of those examples will be out of its region. However, these examples could be considered as noise in the data of the supervised learning task, so the only

requirement is that the supervised learning function approximator used is able to absorb that noise. Since most ML-supervised induction techniques consider noisy inputs, this is not a tight constraint.

5. EXPERIMENTS AND RESULTS

The previous two sections have presented different RL approaches that represent two main research lines, and that will be used in the following experimentation. On the one hand, discretization-based methods have been applied using two discretization approaches: uniform and vector quantization.

On the other hand, the ISQL algorithm, based on the supervised approximation of the value functions has been used. In this case, three different approximation methods have been applied. C4.5 is an algorithm that builds decision trees,[34] and we will call this ISQL-C4.5. PART obtains decision rules from the tree generated by C4.5,[35] and we will call this ISQL-PART. We have used the implementation of C4.5 and PART provided by the WEKA tool.[36] Lastly, we have applied the ENPC algorithm (evolutionary nearest prototype classification)[37,38] a nearest prototype approach that has as its main property that the number of prototypes to use is automatically computed. We will call this ISQL-ENPC.

The reason for selecting these algorithms is that they work well with the default parameter values, so they can be easily applied within automatic processes, as the ISQL algorithm requires. Furthermore, they learn by dividing the space in regions: regions are each leaf of the decision tree, each rule of the decision rules, or the Voronoi regions define by the prototypes and the nearest neighbor rule, respectively. This is an advantage that will be exploited further, as it will be shown in the following section.

The domain used for experimentation is an offic navigation domain, shown in Figure 8. It consists of a robot moving inside an offic area. This area is represented by walls, free positions, and goal areas, all of them of size 1×1. The whole domain is $N \times M$ (24×21 in our experiments). The possible actions that the robot can execute are "go North," "go East," "go South," and "go West," all of size one. The robot knows its location in the space through the continuous coordinates (x, y) provided by some localization system. Furthermore, the robot has an obstacle avoidance system that blocks the execution of actions that would hit walls. Two goal areas have been located in two different rooms, and the robot is expected to achieve one of them.

We can introduce in this domain several levels of noise in the perception of the robot localization. For instance, we could inject a 10% of noise, that means that if the robot is in the (x, y) position, the perception that it has of its position is $(x + n_x, y + n_y)$, where n_x and n_y are random variables following a uniform distribution in the range $(-0.10, 0.10)$. Following this approach, several comparisons have been done for noise levels of 0% (deterministic domain), 5%, 10%, and 20% of noise.

In all the experiments, training data is obtained from 4,000 trials. Each trial consists of trying to reach the goal area in a maximum of 10 actions from random initial positions in the continuous state space. The 10 actions are not enough to reach

Figure 8. An offic navigation domain.

the goal from most of the initial positions in the domain, but it is a small value to avoid the robot wastes a long time in the same room. Tests have been performed over 4,000 trials, each of them with a maximum of 100 actions (value higher than the required one to reach the goal from the furthest position in the domain).

On each trial, if after the execution of the 100 actions the goal is not achieved, the trial is considered failed. Next, different experiments performed with different methods are described.

5.1. Uniform Discretizations

To learn the policy, and once the discretizations are available, the Q-learning update equation was used to learn the policy, with a value of $\gamma = 0.9$ (widely applied in the RL literature). To improve the results obtained without increasing the exploration trials, the experience tuples obtained are used up to 20 times in an iterative process. Furthermore, to improve the convergence, the α parameter is set to 0.2 in the firs iteration, and it is decreased by 0.01 in each step, until the 0 value is achieved. These values, the best ones we have obtained empirically, produce good results that could be nonoptimal.

Figure 9 shows the obtained results. The x-axis shows the size of the discretization used. In this case, action policies for discretizations of size 400, 504, 676, 1024, 2116, 3721, 6400, and 9801 states have been learned. These sizes are obtained by using the same number of levels for each coordinate of the domain, except for the size 504, which is an optimal discretization of 24×21, i.e., the right size of the domain, taking into account that action movement is one (the Markov property is not lost). The y-axis shows the percentage of successful trials, i.e., trials where the robot has been able to achieve one of the two goal areas, starting in a random initial position of the whole domain. Furthermore, different results are shown depending on the noise level.

The figur shows the optimal solution achieved for the deterministic domain (noise level set to 0%) and 504 states, obtaining a 100% of successful trials. This expected result, however, requires a perfect knowledge of the domain, and that the user define the discretization to the optimal 24×21 size. However, if the domain

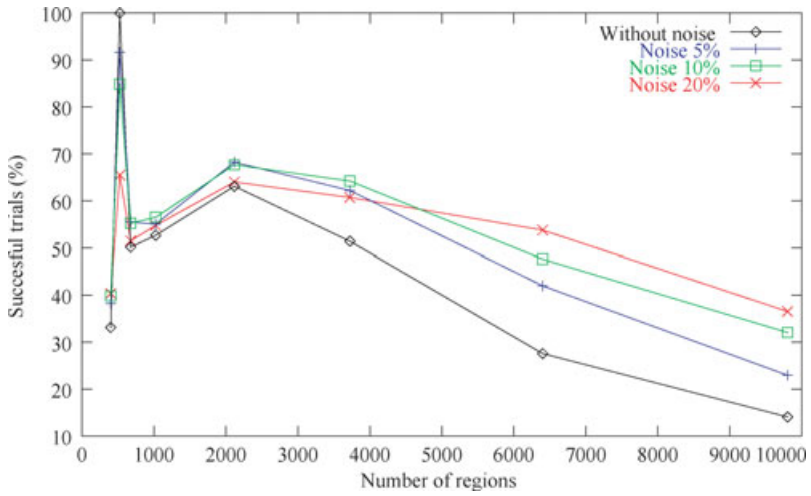

Figure 9. Percentage of successful trials with uniform discretizations of different sizes and different noise levels.

is stochastic, this situation changes as a function of the noise level. So, with a 5% of noise, the success decreases down to 91.67%; with a noise of 10%, it decreases down to 84.85%; and with a noise value of 20%, it decreases down to 65.45%. This is due to the random behavior introduced in the domain by the noise component, which may do the agent believes that it is in a region where it actually is not.

Without taking into account this handmade solution, Figure 9 shows that for small sizes of 400, 676, and even 1024 states, the results are poor, and they do not achieve a 60%. However, for 2116 and 3721 states, the results improve and a 68.2% is achieved for a noise value of 5%. However, with higher sizes, the performance decreases again, given that the number of regions is very high, resulting in a poor use of the experience. A larger amount of experience is expected to increase the results of the high-resolution discretizations.

When the domain is noisy, the robot typically obtains better results than when the domain is not. This is due to the random component that the noise introduces, which improves the exploration of the robot. Furthermore, the higher the resolution of the discretization is, the more probability the robot has to get out of a loop.

5.2. VQQL

The VQQL algorithm define a previous step for learning the state space discretization of the original state space. A random exploration is done over the environment, and the experience tuples obtained will be used to learn the policy too. Thus, the experience used for learning the model is exactly the same than the one used in the uniform case of the previous experimentation.

Figure 10 shows the set of prototypes obtained with GLA for 1024 states and the domain without noise. For different noise levels, different discretizations

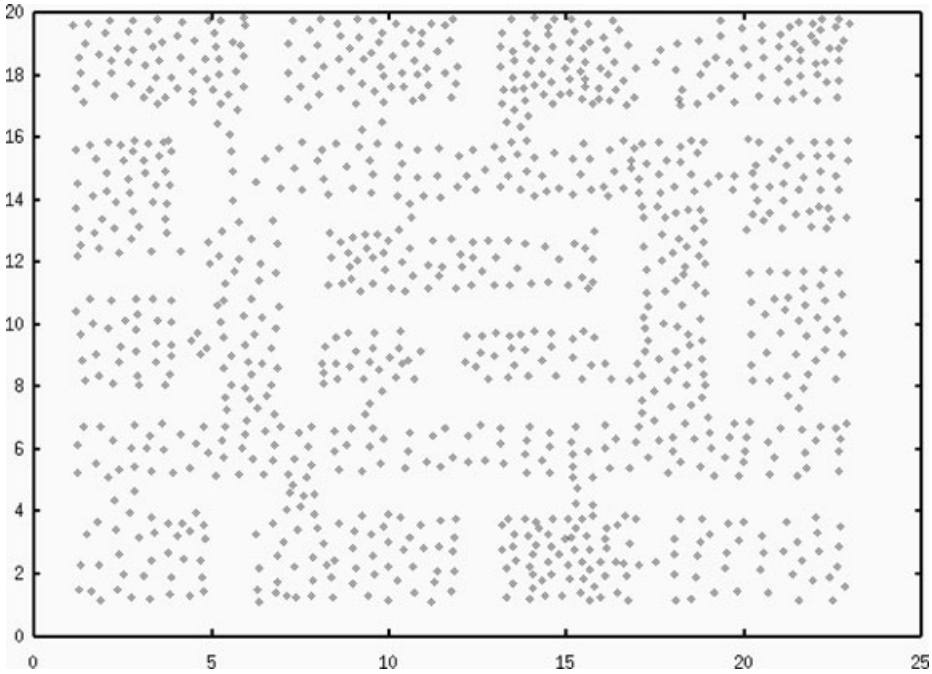

Figure 10. Prototypes of the discretization obtained by GLA of size 1024.

are obtained. Prototypes are only located in relevant areas of the domain, i.e., in those areas where the robot can walk, and hence, areas where the robot must make decisions about its following actions. Thus, having prototypes only in these areas makes all prototypes useful.

Given that the number of prototypes is an input parameter of the algorithm, we have executed it for the same number of states than in the uniform case. Once the state space discretizations are computed, action policies can also be computed. Figure 11 shows the percentage of successful trials where the robot has been able to reach the goal area. To learn the action policy, the same learning parameters as in the uniform case are used.

Results are similar to the ones presented in the uniform case, with two main differences. First, in the VQQL case, the optimal solution is not obtained, and second, without taking into account the optimal solution, values close to the 70% are achieved, but with a smaller number of regions.

5.3. Iterative Smooth Q-Learning with ENPC

The main difference of using the ISQL-ENPC algorithm with respect to using uniform discretizations and VQQL is that the number of regions is automatically defined so repeating the experimentation with different discretization sizes is not

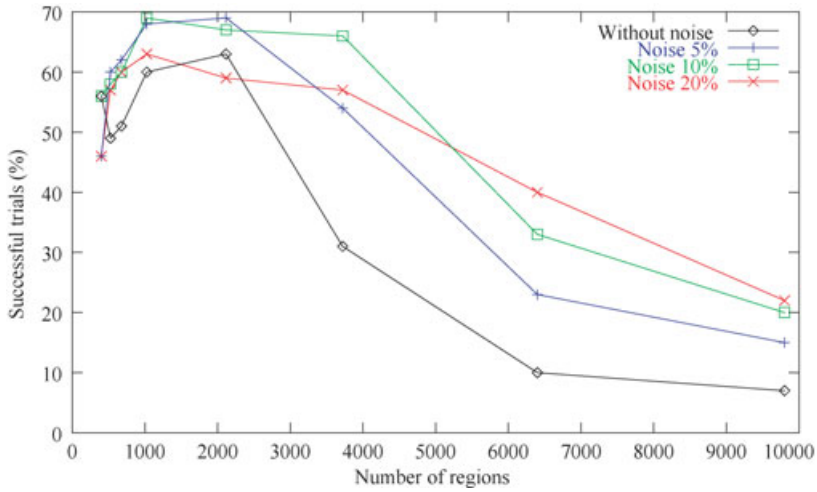

Figure 11. Percentage of successful trials with VQQL for different numbers of regions and noise levels.

needed. Figure 12 shows the performance obtained in each of the iterations of the ISQL algorithm with ENPC.

The experiment has been performed for a total of 44 iterations. For each iteration and noise level in the domain, the success of the learned policy is shown. The figur shows that the results achieved range from 25% to 45%, obtaining poor results because of the errors propagated from the goal areas to the whole domain. Figure 13 shows the evolution of the number of prototypes used while the ISQL-ENPC algorithm is executed, showing the average number of the four discretizations (one for each action).

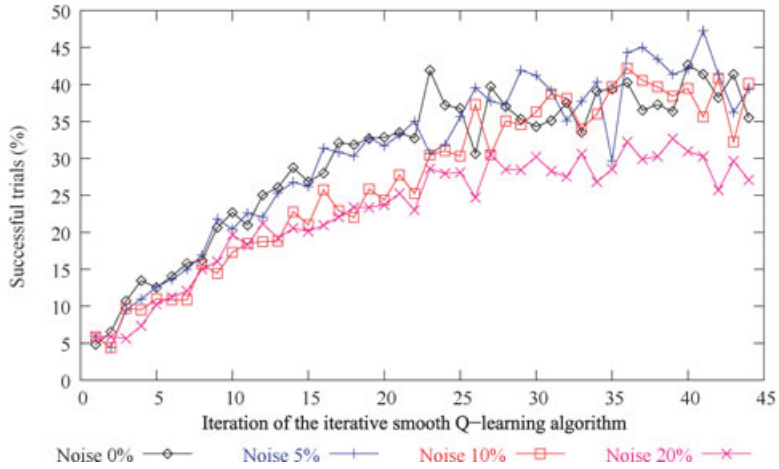

Figure 12. Percentage of successful trials using iterative smooth Q-learning with ENPC.

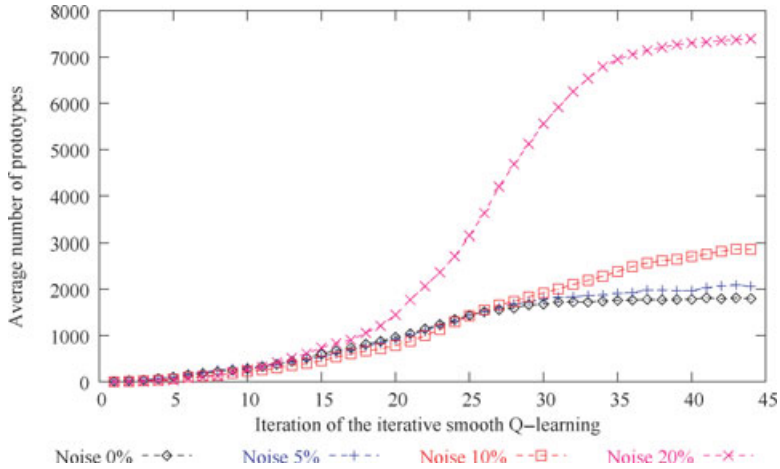

Figure 13. Number of prototypes generated with ENPC-QL for different noise levels.

The main effect of introducing noise in the data is an overfittin problem of the ENPC algorithm to this noisy data, and the introduction of a higher number of prototypes to increase classificatio rates. However, even when the number of prototypes increases very much, as with a noise level of 20%, where the number of prototypes is higher than 7,000, the navigation problem does not seem to be solved worse than with lower sizes, as shown in Figure 12.

In Section 4, we define the end condition to be the number of iterations required to propagate the reinforcement in the goal areas to the rest of the domain. In this example, the number of prototypes seems to stabilize in iteration 25 for noise levels of 0% and 5%, respectively, whereas with noise values of 10% and 20%, that occur in iteration 28. Differences among them are due to errors in the propagation of the reinforcement from the goal.

5.4. Iterative Smooth Q-Learning with C4.5 and PART

The same experiment has been performed for the iterative smooth Q-learning algorithm, using C4.5 and PART as the function approximators. The results are shown in Figure 14, for a learning process with the same parameters than in the previous experiments. The figur represents in the x-axis the noise level, for the previously used values of 0%, 5%, 10%, and 20%. The y-axis shows the percentage of successful trials.

The figur shows how ISQL-C4.5 is very sensitive to the noise of the domain, more than the previous ones. So, for the deterministic domain, the success percentage achieved is 64.8%, but when the noise is increased, this value decreases down to the 22.4%. These results show that depending on the noise of the domain, the parameters of the C4.5 algorithm must be modifie from the predefine ones to support a higher noise (by modifying the pruning phase of the algorithm). This imposes the user

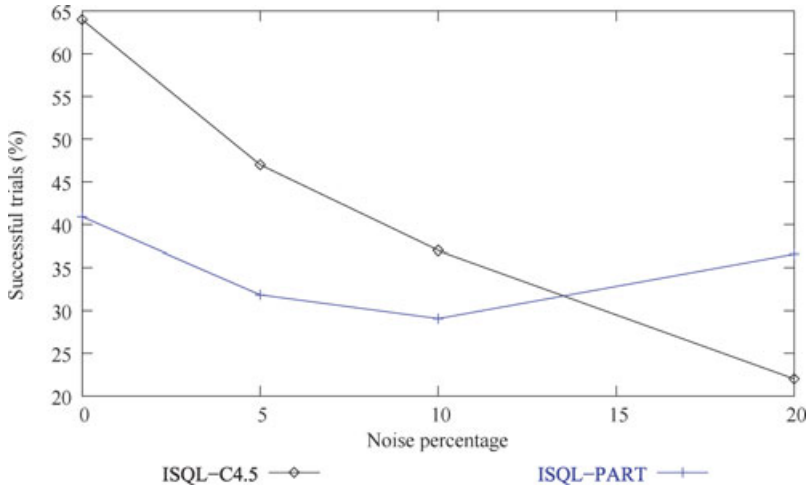

Figure 14. Percentage of successful trials with C4.5 in iterative smooth Q-learning.

the burden of tuning the parameters, given that different domains may require different parameters. The number of iterations of the algorithm executed is 29, 27, 27, and 29 for each noise level. This number corresponds to the iterations where the reinforcement was propagated to the whole domain. The differences among them are produced by the accumulated error in the propagation of the reinforcement from the goal areas, as it is the case of ISQL with ENPC.

In the case of ISQL-PART, results are poorer, probably because of the higher pruning capability of the algorithm for the default parameters. So results are around 40% of success for the deterministic domain, and lower for higher noise levels.

5.5. Conclusions

Table I shows the success obtained by all methods, without taking into account the optimal handmade uniform discretization. Table II shows the number of regions generated by each algorithm.

The tables show that VQQL achieves, in general, similar success levels to uniform discretizations. However, it typically requires a smaller number of states.

Table I. Percentage of success of different methods in the offic navigation domain.

Noise	Uniform	VQQL	ISQL PART	ISQL C4.5	ISQL ENPC
0%	63.08	63.75	40.92	**64.8**	36.77
5%	68.2	**69.65**	31.85	47.1	35.62
10%	67.65	**69.27**	29.07	37.6	35.07
20%	**64.0**	63.05	36.6	22.4	28.52

Table II. Number of regions created by different methods in the offic navigation domain.

Noise	Uniform	VQQL	ISQL PART	ISQL C4.5	ISQL ENPC
0%	2116	2116	487	498.5	1437.25
5%	2116	2116	504.25	731	1443
10%	2116	1024	440	649	1737.25
20%	2116	1024	389	552.5	5123.75

In a bidimensional domain, this might not be very important, but it can be critical in other domains with higher dimensional state spaces, where vector quantization discretizations are very powerful.[28] ISQL-C4.5 achieves similar solutions when the domain is deterministic, but its performance decreases drastically when noise increases. ISQL-PART obtains worse results, maybe because of the higher pruning capability, represented by a smaller number of rules generated, specially when noise is introduced in the domain. ISQL-ENPC obtains poor results similar to the ISQL-PART. In general, the poor results of ISQL, in contrast to discretization-based methods, can be due to two facts. The firs one is that the errors in the propagation of the reinforcements from goal areas to the rest of the domain produce very poor policies. The second one is that these errors could increase by the nondeterminism of the domain, and that supervised learning algorithms such as PART or ENPC typically work worse in noisy domains.

However, there is a useful result of ISQL. When applied, for instance, with ENPC, it also outputs a state space discretization for each action, define by the prototypes of the classifie . In the case of PART, the discretization is define by each generated rule that can be considered as a region in the space. Last C4.5 generates a decision tree, whose leaves also defin a discretization of the state space.[14] It might be that these state space discretizations are more accurate than discretizations obtained without taking into account the value function, as it is the case of uniform discretizations or VQQL. Therefore, the idea that is extended in the following sections is to use these discretizations in a second learning phase to improve the approximation obtained.

6. TWO STEPS REINFORCEMENT LEARNING

Several splitting criteria can be used for incrementally discretizing the state space in deterministic dynamic programming problems.[6] They tried to introduce more regions in those areas where value and/or policy approximation is harder, i.e., areas of the state space where discontinuities of one or both functions are found, depending on the criteria. These discontinuities can be understood as areas where very different values of the value function can be found. In the same way, ISQL tries to differentiate the max_path different values of the value function, obtained from $\{\gamma^i, i = 1, \ldots, max\_path\}$, as define in Section 4.. So, it is trying to discriminate among different regions of the state space with different values of the action value

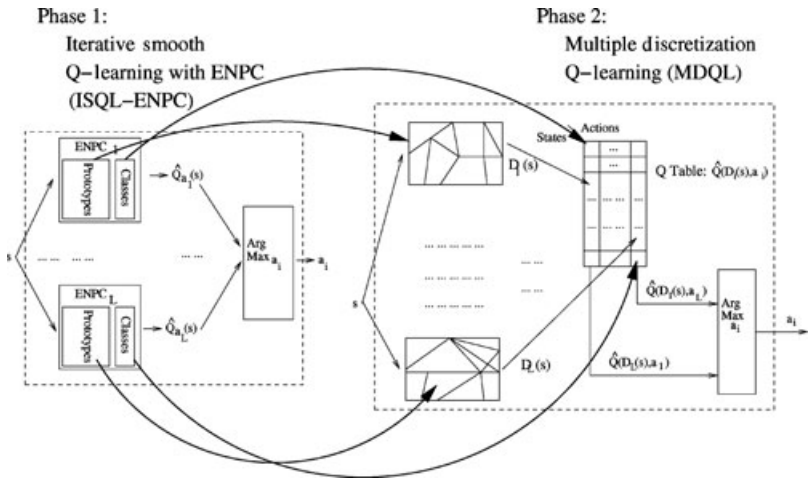

Figure 15. The two steps of the 2SRL algorithm.

function. The difference is that, in this case, the resolution is define by max_path, and so, by the size of the actions, instead of a criteria define by the designer, such as policy disagreement or value nonlinearity.[6]

Definin the resolution based on the size of the actions, and hence, the number of actions required to solve the task, provides one main advantage: It is a parameter very easy to understand by the designer of the system. Furthermore, if algorithms like ENPC[38] or C4.5[34] are used in the ISQL phase, no initial coarse discretization must be provided by the user, as it is reported in Munos and Moore[6] and, Reynolds[39] given that both algorithms begin with discretizations of size 1.

The idea of the algorithm that we call two steps reinforcement learning (2SRL), is that, even if iterative smooth Q-learning obtains poor policies, if the function approximator used for each action outputs also a state space discretization, these discretizations may correctly defin the limits of the action value function (Figure 15).

The discretizations obtained in this firs phase can be used to tune, in a second phase, the action value function obtained in the previous phase, following a multiple (one per action) discretization-based approach. We have called this second phase multiple discretization Q-learning (MDQL). This is different than multi grid methods[40] used to maintain different discretization levels at the same time, typically in a hierarchy.

The need of this new phase is motivated by the two assumptions of ISQL define in Section 4 that might not be true: the domain may not be deterministic, and we probably do not have enough experience, nor perfect approximators to correctly compute the optimal action value function, as the experimentation of the previous section has shown.

Figure 6 illustrates the structures required to represent the action policy used by this method in its two phases. The left box shows the structures required by iterative smooth Q-learning (in this case, with ENPC). In the right box, the structures used

for representing the action policy in the multiple discretization Q-learning phase are shown, requiring one state space discretization for each action. This phase is not equal to the VQQL scheme, where only one discretization is used, as shown in Figure 1, and it is different to other VRD methods. For instance, Moore and Munos approximate the state value function, and hence, they only need one discretization, and Reynolds uses the same discretization for all the actions when it approximates the action value function.

The figur also shows how to move from the firs scheme to the second one. Prototypes of the approximator $ENPC_i$ will be used as the discretization $D_i(s)$, and the classes of $ENPC_i$ will be located in column i of the generated Q table, providing an initialization of the Q table in the second learning phase. Each $ENPC_i$ approximation may have a different number of prototypes, so the number of rows of the table is given by the maximum number of prototypes of the L approximators, and zero values can be used to complete the columns with less prototypes.

Once the translation from the firs scheme to the second one is done, a new learning phase can be executed. A main difference with the firs one is that now, the stochastic Q-learning update function described in Equation 1 will be used instead of the deterministic one (Equation 2), because the state space discretization obtained in the firs phase may not be totally accurate, as it indeed happens as it was shown in Section 5.

The second learning phase can be executed with new experiences or with the same ones used in the firs phase. In the experimentation performed next, the second approach is applied using the same experiences several times, as it was performed when uniform discretizations and *VQQL* were applied.

7. EXPERIMENTS WITH 2SRL

This experimentation receives as input the experimentation described in Section 5. 2SRL is executed in two phases, ISQL and MDQL. The firs one was executed in Section 5. for the offic navigation domain. In this section, we show the experiments performed with 2SRL, when using both ENPC (2SRL-ENPC) and PART (2SRL-PART) as function approximators. The learning process is the same than the followed for VQQL and uniform discretizations.

For 2SRL-ENPC, results are summarized in Figure 16, which shows the performance for each noise level. Also, the results shown in Figure 12 for ISQL-ENPC have been included for comparison reasons. As in that case, results for each iteration of the ISQL phase are shown.

The figur shows that in only 20 iterations, for all noise levels, the behaviors learned achieve a range of 70%–80%, typically 10% higher than the behaviors learned with uniform and VQQL discretizations, and 30% higher than executing only the ISQL algorithm. So, it shows that, even if ISQL does not achieve good results, the discretizations that it computes by using ENPC as a function approximator achieve a better approximation of the Q function in the MDQL phase. Furthermore, the sizes of the discretizations are computed automatically by the ENPC algorithm, providing this method with another advantage.

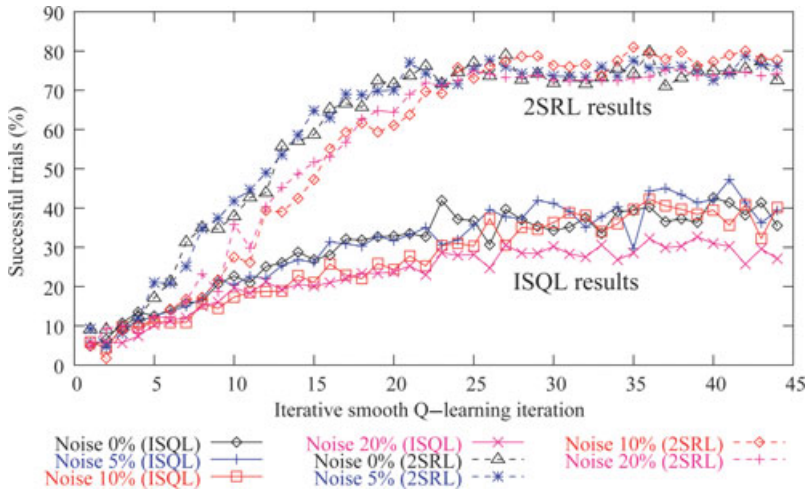

Figure 16. Percentage of successful trials when executing the multiple discretization Q-learning phase.

The MDQL phase has also been executed over the discretization generated by decision rules (PART) previously obtained. Table III shows the different results achieved for all the methods applied, but without taking into account the optimal handmade uniform discretization.

The table shows how 2SRL improves the results of executing only the ISQL phase, both for ENPC and for PART. However, with PART, results are not so good as for ENPC. This happens probably because it generates a small number of regions, as it was shown in Table II, so the limits in the value function are not define as well as in the ENPC case.

As a conclusion, 2SRL, specially when it uses ENPC, is able to solve the offic navigation problem in an efficien way, achieving better results than other methods without the need of definin the size of the discretization. This is done not only for a deterministic case, but for cases with a strong stochastic component, with a

Table III. Comparative results of the different methods in the offic navigation domain.

	Noise Level (%)			
Algorithm	0	5	10	20
Uniform	63.08	68.2	67.65	64.00
VQQL	63.75	69.65	69.27	63.05
ISQL-PART	40.92	31.85	29.07	36.6
2SRL PART	68.95	58.62	61.85	56.45
ISQL-ENPC	36.67	35.62	35.07	28.52
2SRL ENPC	**77**	**75.4**	**78.62**	**73.45**

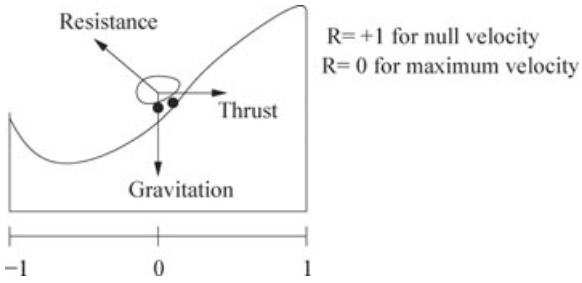

Figure 17. Graphical representation of the car on the hill domain.

noise value of 20%, where the results are close to the deterministic case, and even higher than for the optimal 24×21 uniform discretization, that achieved a 65.45% of success.

8. ADDITIONAL EXPERIMENTATION

In this section, we extend the experimentation performed over the 2SRL algorithm, introducing comparisons with other approaches.

8.1. Car on the Hill

This domain, widely used in the RL community,[13] consists of pushing a car until it reaches the top of a mountain, preferably with the lowest speed, as Figure 17 shows. It is a bidimensional domain, with an x component referring to the position of the car, and a v component referring to the velocity of the car. The x component value ranges from -1 to $+1$, whereas the velocity ranges from -4 to 4.

From the RL point of view, the goal of the car is to obtain a maximum reward in each trial. Each immediate reward depends on the state of the car, i.e., on its position and velocity, as define in the reinforcement function in Equation 3.

$$r(x, v) = \begin{cases} 0 & \text{if } -1 \leq x < 1 \\ f(v) & \text{if } x \geq 1 \end{cases} \tag{3}$$

where $f(v)$ is a linear function that returns a positive reward only when the car arrives to the goal, i.e., the top of the mountain ($x \geq 1$). Maximum reinforcement of $+1$ is obtained when the velocity is 0, and a null reward when the velocity is maximum (-4 or 4). There are two actions in this domain, pushing the car with a force of 4 or -4, in the direction shown by the figur (thrust).

Training data is obtained from 4,000 trials. Each trial tries to reach the goal area from a random location by executing a maximum of 150 actions. Tests have been performed over 1,000 trials, each of them with a maximum of 150 actions.

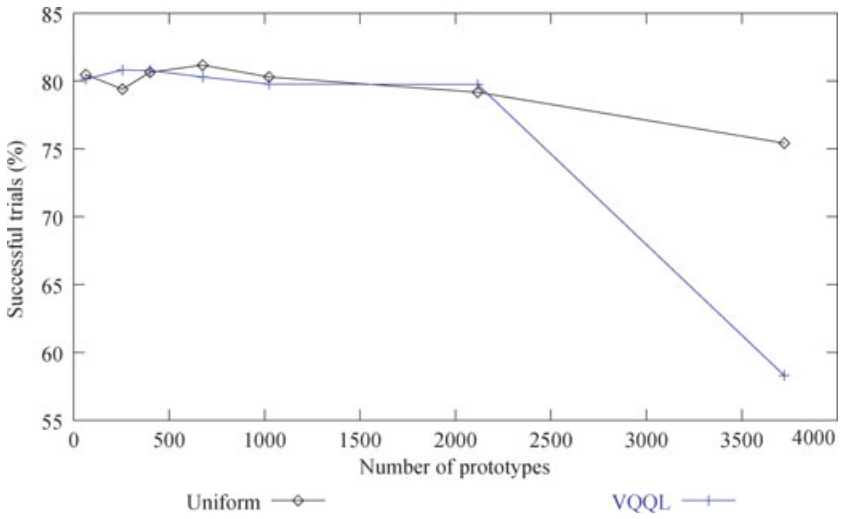

Figure 18. Percentage of successful trials with uniform discretizations and VQQL.

8.1.1. Uniform Discretizations and VQQL Results

Since we do not know "a priori" what is the optimal discretization size, state space representations of different sizes have been used. In the VQQL case, the GLA algorithm must be applied to obtain such state space discretizations. In the uniform case, the state space discretizations are hand made, using the same number of discretization levels for each component, x and v. The process performed for learning the action policy is the same than the one applied for the offic navigation domain. Figure 18 shows the percentage of successful trials for uniform discretizations and VQQL, using the same sizes.

The figur shows that uniform discretizations and VQQL achieve success percentages of around 80%, for small state space representations (256 and 400 states). This value is optimal, given that there are initial positions from which the car exits the figur from the left ($x \leq -1$), independently of the policy it follows (around 20% of the initial positions). These results are worse for bigger state spaces, where the use of the experience acquired from the 4,000 learning trials is not exploited because the state space being learned is so large. Thus, if the percentage of successes is taken into account, these two methods obtain good results for most of the state space sizes.

However, if the end velocity is observed, the results achieved above cannot be considered as good. Figure 19 shows the average speed that the car has when arriving to the goal for the uniform and VQQL cases. The figur shows that an average velocity of arriving to the goal smaller than 1 is achieved only when the number of prototypes is high (for 2116 states). This shows that, although with a small number of states, we can obtain policies that make the car reach the goal area in a high percentage of the trials, the velocity of arriving to this goal area may also

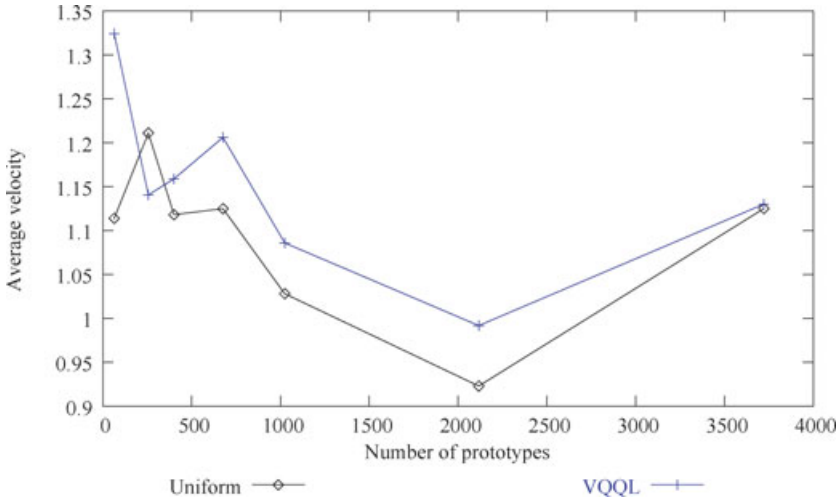

Figure 19. Average velocity of arriving to the goal with uniform discretizations and VQQL.

be very high, if the number of states is not big enough. This shows the difficult of findin the right size of the state space of these approaches.

8.1.2. 2SRL

The application of ISQL to the car on the hill domain requires only one consideration. The algorithm assumes a unique positive reward at the end of a trial that must be propagated to the rest of the environment. However, in this domain, the reinforcement function is a linear function that takes a continuous set of values from 0 to 1. To solve this problem, the reinforcement function has been discretized (for all techniques) to the known set of values that the value function can take, i.e., the succession of values define by $\gamma^i r_{\max}$, for $i = 1, \ldots, \infty$. We will explore in the future whether this limitation is a strong one, and, in that case, how to overcome it. Furthermore, for speeds higher than 2, the reinforcement achieved is considered zero. Also, the end condition of the algorithm cannot be applied, given that there are areas in the domain from where the goal area cannot be reached, so the algorithm is executed 49 iterations (empirically enough to propagate the reinforcements from the goal to the rest of the domain). Once these modification in the reinforcement function are performed, the application of both phases of 2SRL (ISQL and MDQL) is direct.

The learning process is the same than the one followed for VQQL and uniform discretizations. The evolution of the learning performance is shown in Figure 20. For comparison reasons, we also include the results obtained by executing only the firs step of the algorithm (ISQL-ENPC).

ISQL-ENPC is able to reach the goal in almost 70% of the trials, after at least 35 iterations. So, using the ENPC algorithm as a function approximator, the results obtained on successful trials are worse than the ones obtained by uniform discretizations and VQQL.

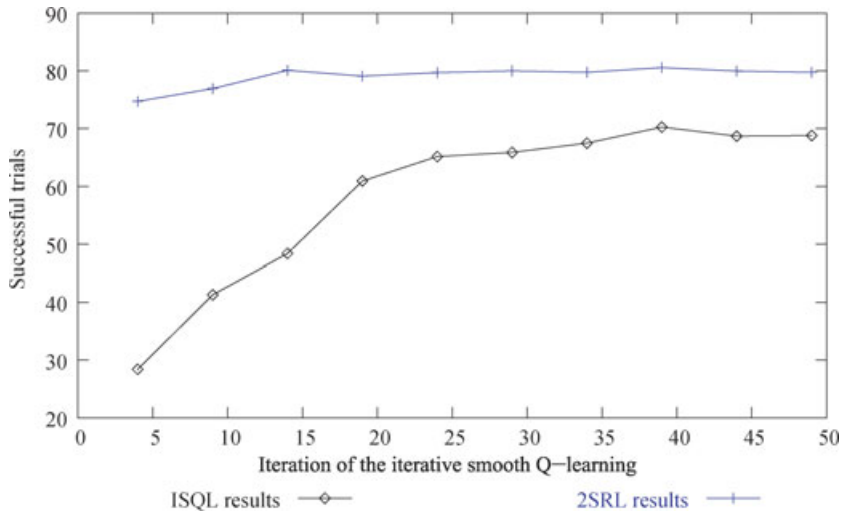

Figure 20. Percentage of successful trials when executing 2SRL-ENPC in relation to just executing the ISQL-ENPC phase.

For 2SRL, we can see that in only 14 iterations, values close to 80% are achieved, so successful results are obtained faster (convergence rate), and the results are improved (around a 10%) with respect to the result of the ISQL phase. Figure 21 shows how these advantages are extrapolated to the average velocity, achieving results below 1 consistently.

From these results, we can draw two main conclusions. Uniform discretizations obtain very good results, similar to the ones obtained with VQQL. So, in this domain, very results can be achieved with both methods, taking into account that a right number of regions must be used to obtain the best results. Iterative smooth Q-learning (ISQL) is not able to achieve these results due to the errors propagated from the initial approximations of the Q function to the rest of the domain. The results using the two phases of 2SRL are also close to the optimal. Moreover, this approach, from the variable resolution discretization point of view, has an advantage: the resolution of the discretization is automatically computed in the ISQL phase.

8.2. Acrobot

The experiments in the Acrobot domain have two goals. On the one side, to test the 2SRL method in a domain with a higher dimensional state space, four dimensions in this case. On the other side, to compare the results obtained by 2SRL with other VRD algorithms, specificall with the results reported in. Munos and Moore.[6] Therefore, we have simulated the experiments performed in that work, using the implementation of the domain provided at Remi Munos Web page.[a]

[a]http://www-2.cs.cmu.edu/∼munos.

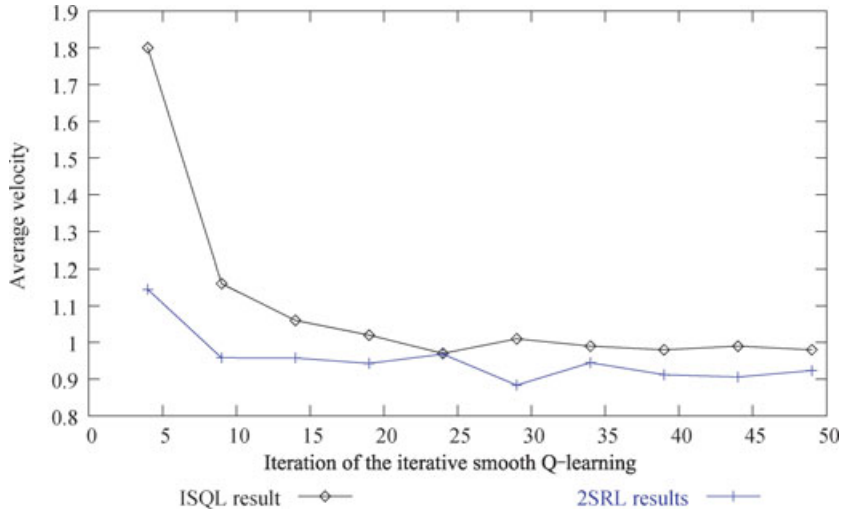

Figure 21. Average velocity when reaching the goal, executing 2SRL-ENPC and ISQL-ENPC.

This problem consists of a two-link arm with one actuator at the elbow, as it is shown in Figure 22. The state is composed of four variables, θ_1 and θ_2, and the velocities of the two bars. The goal is to locate the Acrobot in its inverted vertical position in the minimum time. That position is define by a narrow range, $\frac{\pi}{16}$ on both angles around the vertical position define by $\theta_1 = \frac{\pi}{2}$ and $\theta_2 = 0$. When that location range is achieved, the system receives a reward of $+1$. Otherwise it receives a reward of 0.

The firs step was to execute the system for the uniform discretization case, with different discretization sizes. The values used for the parameters were $\gamma = 0.93$, and a time step of 0.01. However, actions are considered to have a length of 0.5, given that the action size using a time step of 0.01 is very small for changes on the discretized state (even for high-resolution discretizations). The learning method is the same one than in previous experiments. Experience tuples are obtained from 8000 exploration trials initiated in random locations of the state space. Each of those trials has a length of 20 actions. The test is performed over 256 trials of a maximum

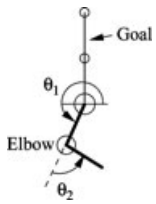

Figure 22. Acrobot domain.

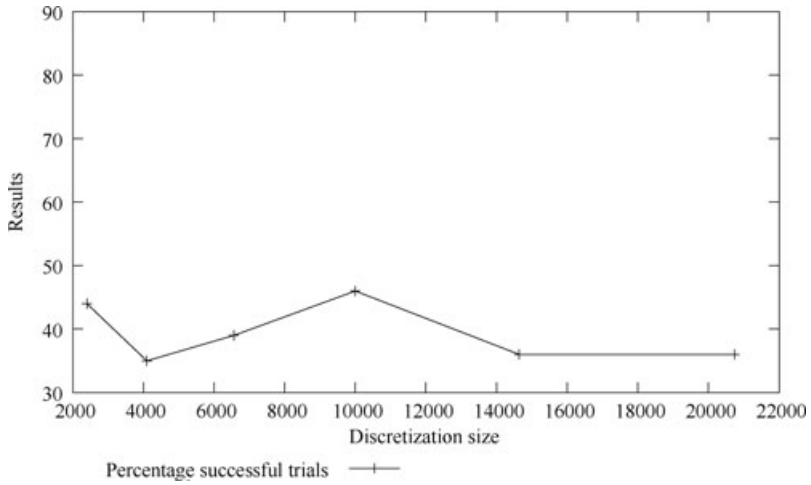

Figure 23. Results on the acrobot domain using uniform discretizations.

length of 40 s, following the parameters applied in. Munos and Moore[b] Figure 23 summarizes the results.

The figur shows the percentage of trials that achieved the goal. The figur shows that the best result is obtained for 10,000 states. That makes a different with the results for VRD reported in Munos and Moore.[6] In model free methods, as in this case, increasing the state space not always increases the performance, so a balance must be found between the state space size and the amount of experience used to learn. Opposite to that, VRD is a model-based method, which find the optimal policy for the discretization that it obtains, and it always improves performance when the discretization size is increased.

Furthermore, the results are not directly comparable with the results described for VRD in Munos and Moore.[6] Uniform discretization described in this work follows a finite-di ference method, meaning that the same region always returns the same V or Q value for all the states in the region. VRD follows a finite-elemen approach, so the same region may produce different values depending on the exact position in the region of the state that we are evaluating.[41] These different schemes produce different dynamics, so performances are hardly comparable.[c]

Once uniform results are obtained, we have executed the two phases of the 2SRL algorithm. Figure 24 shows the percentage of successful trials. These values are shown for the different approximations obtained in the different iterations of the ISQL-ENPC algorithm, as well as for 2SRL-ENPC.

The figur shows that, executing the two phases of the algorithm, the average number of successful trials improves the values over the ISQL case, in more than a

[b]Tests were performed over 1024 trials to improve statistical properties of the results, and then mapped to the case of 256 trials for comparison.

[c]Thank Rémi Munos for the discussion and comments on this topic.

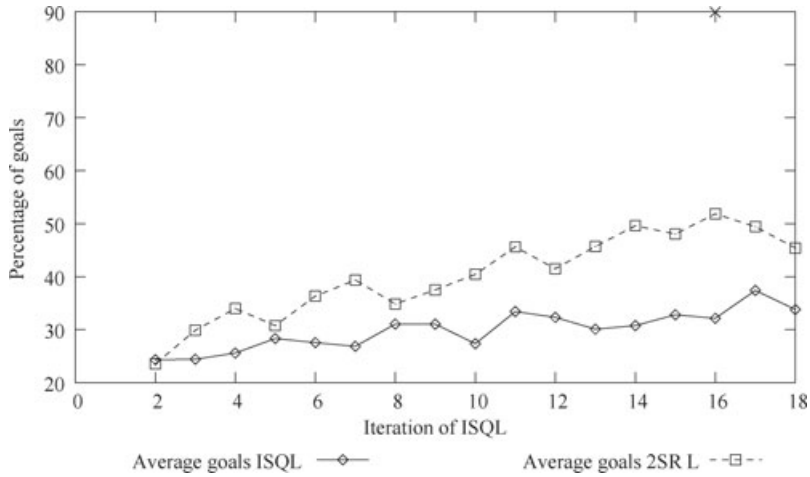

Figure 24. Average number of goals obtained by ISQL-ENPC and 2SRL-ENPC.

10%. The number of iterations executed was only 18, because of the big increment in the number of prototypes, shown in Figure 25. The figur shows that the increment of the number of prototypes generated by ISQL-ENPC is exponential with the number of iterations, and in 18 iterations is above 25,000. This high number introduces strong execution restrictions on the algorithm, given that the ENPC implementation used has a complexity in classificatio which is linear with the number of prototypes, instead of logarithmic, which could be obtained with a more efficien implementation (using, for instance, kd-trees[42]).

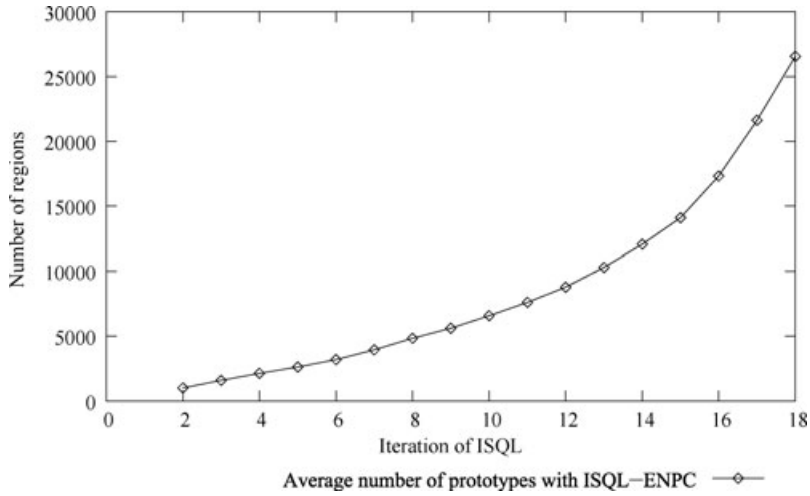

Figure 25. Number of prototypes obtained by ISQL-ENPC.

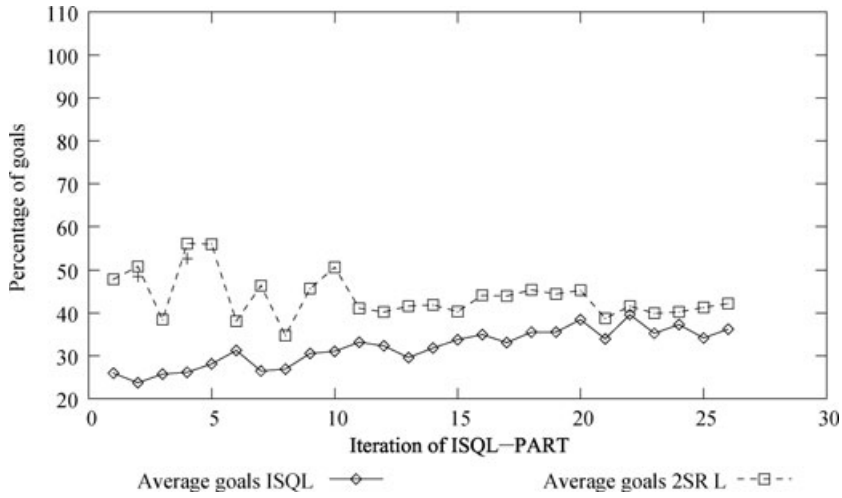

Figure 26. Average number of goals obtained by ISQL-PART and 2SRL-PART.

The same experiment is repeated but using PART decision rules instead of ENPC for approximating the action value function to obtain the state space discretization. Results are summarized in Figures 26 and 27. As in the previous case, Figure 26 shows the improvement in performance when executing the two phases of the algorithm. Furthermore, Figure 27 also shows that the number of regions generated usually grows exponentially with the number of iterations, even in this

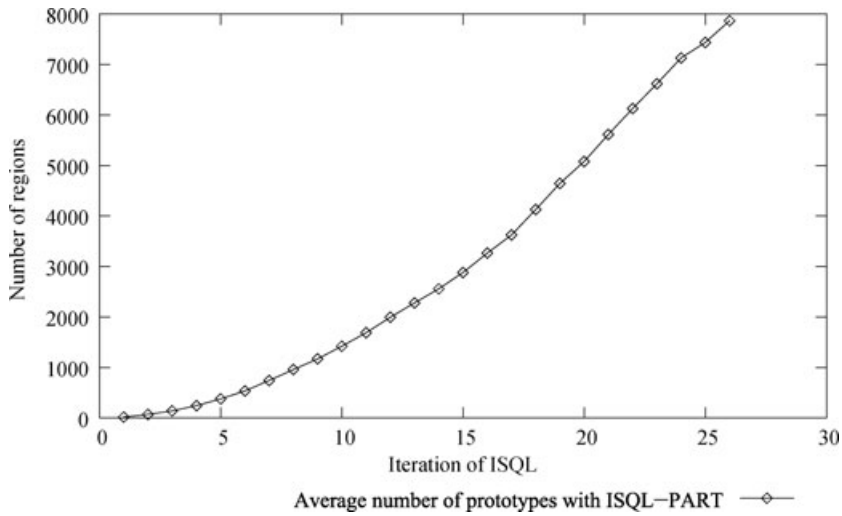

Figure 27. Number of regions obtained by ISQL-PART.

case, the number of regions obtained is much lower than when using ENPC. Also, the number of regions is very small when compared to the results reported in Munos and Moore,[6] which uses values one order of magnitude bigger.

9. CONCLUSIONS AND FUTURE WORK

We have presented a new RL method for domains with continuous state spaces based on the supervised discretization of such state spaces. This new method, called 2SRL, is composed of two main learning phases based on two approaches: supervised function approximation, using the iterative smooth Q-learning algorithm, and state space discretization methods, using the multiple discretization Q-learning.

The algorithm has two main advantages. First, the two phases have shown better results over methods based on only one of the approaches. So, this method can be applied to approaches which only execute the firs phase. For instance, the Q-RRL algorithm[43] is a relational reinforcement learning algorithm very similar in structure to the iterative smooth Q-learning algorithm presented here, but using relational data instead of feature-based data. Furthermore, Q-RRL uses a logical Q-tree to approximate a Q-table, so a state space discretization is obtained in the same way that was presented in this work for C4.5 tree or the decision rules. So, a second learning phase could be added over that state space representation obtained after executing Q-RRL to tune the Q-learning approximation obtained. The second advantage is that the number of parameters and/or knowledge that must be introduced in order the algorithm to work correctly is very low, and hence, easily applicable to new domains.

The relationship among losing the Markov property when using state space discretizations and the nondeterminism introduction in domains originally deterministic has been shown. So, losing the Markov property can be understood as a problem of nondeterminism introduction that could be efficientl tackled. In this sense, the ISQL algorithm is based on the idea that, when generating an action policy, the nondeterminism in the domain can be considered noise of a deterministic domain if the function approximator used is able to absorb that noise. This is very useful when definin the limits among the different regions of the state space with strong borders in the value function, obtaining very good action policies, as it was shown for the offic navigation domain.

Another difference with previous work based on state space discretizations is that they typically use the same discretization for each action. Here, we show that in model free methods where the action-value function, $Q(s, a)$, must be approximated, it is possible that each action requires a different number of resources or state space discretizations. This introduces the idea of using one state space discretization per action. But, the discretizations are simpler than the required ones if only one new state space is computed. We would like to compare both approaches in the future.

Additional research will also be the study of the use of different exploration and exploitation strategies to improve the usefulness of the algorithm in real domains, where initial explorations may not be easily performed. The effects of these strategies to the 2SRL algorithm must be studied, as well as efficien methods for obtaining the

function approximation of the Q function in the firs learning phase. For instance, concepts like the "support" set of states, or the "rollout" tests, could be introduced in the ISQL phase, as it was introduced in smooth value iteration.[5]

Acknowledgments

This research was partially conducted while the firs author was visiting Carnegie Mellon University from the Universidad Carlos III de Madrid, supported by a generous grant from the Spanish Ministry of Education and Fulbright. Both authors were partially sponsored by the Spanish MEC project TIN2005-08945-C06-05 and regional CAM-UC3M project number CCG06-UC3M/TIC-0831.

References

1. Sutton RS, Barto AG. Reinforcement learning: An introduction. Cambridge, MA: MIT Press, 1998.
2. Kaelbling LP, Littman ML, Moore AW. Reinforcement learning: A survey. Int J Artif Intelli Res 1996;4:237–285.
3. Watkins CJCH. Learning from delayed rewards. PhD thesis, Cambridge University, Cambridge, England, 1989.
4. Bellman R. Dynamic programming. Princeton, NJ. Princeton University Press; 1957.
5. Boyan JA, Moore AW. Generalization in reinforcement learning: Safely approximating the value function. Adva Neural Inform Process Syst 1995;7.
6. Munos R, Moore A. Variable resolution discretization in optimal control. Machine Learning 2002;49(2/3):291–323.
7. Santamaría JC, Sutton RS, Ram A. Experiments with reinforcement learning in problems with continuous state and action spaces. Adaptive Behavior 1998;6(2):163–218.
8. Munos R. A study of reinforcement learning in the continuous case by the means of viscosity solutions. Machine Learning 1999;40:265–299.
9. Fernández F, Borrajo D. On determinism handling while learning reduced state space representations. In: Proc Eur Conf on Artificia Intelligence (ECAI 2002), Lyon (France); July 2002. pp 280–284.
10. Moore AW. Variable resolution dynamic programming: Efficientl learning action maps in multivariate real-valued spaces. In: Proc Eighth Int Machine Learning Workshop, 1991.
11. Reynolds SI. Reinforcement learning with exploration. PhD thesis, University of Birmingham, 2002.
12. Monson CK, Wingate D, Seppi KD, Peterson TS. Variable resolution discretization in the joint space,. In: Proc Int Conf on Machine Learning and Applications (ICMLA 2004), 2004.
13. Moore AW, Atkeson CG. The parti game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. Machine Learning 1995;21(3):199–233.
14. Uther WTB. Tree based hierarchical reinforcement learning. PhD thesis, Carnegie Mellon University, August 2002.
15. Bertsekas DP, Tsitsiklis JN. Neuro-Dynamic programming. Bellmon, Athena Scientific MA 1996.
16. Tsitsiklis JN, Van Roy B. Feature-based methods for large scale dynamic programming. Machine Learning, 1996;22:59–94.
17. Baird LC. Gradient descent for general reinforcement learning. Neural Infor Process Syst 1998; 11.
18. Tesauro G. Practical issues in temporal difference learning. Machine Learning, 1992;8:257–277.

19. Aha D. Lazy learning. Boston, MA: Kluwer Academic Publishers; 1997.
20. Smart WD, Kaelbling LP. Practical reinforcement learning in continuous spaces. In: Proc Int Conf of Machine Learning, 2000; pp 903–907.
21. Touzet C. Neural reinforcement learning for behaviour synthesis. Robotics and Auton Syst, 1997;22:251–281.
22. Fernández F, Borrajo D. VQQL. Applying vector quantization to reinforcement learning. In: RoboCup-99: Robot Soccer World Cup III, Lecture Notes in Artificia Intelligence, vol 1856 Springer Verlag; Berlin; 2000. pp 292–303.
23. Fernández F, Parker L. Learning in large cooperative multi-robot domains. Int J Robot Automation 2001;16(4):217–226.
24. Lloyd SP. Least squares quantization in PCM. IEEE Trans Infor Theory 1982;28:127–135.
25. Gersho A, Gray RM. Vector quantization and signal compression. Boston, MA: Kluwer Academic Publishers, 1992.
26. Kitano H, Tambe M, Stone P, Veloso M, Coradeschi S, Osawa H, Matsubara H, Noda I, Asada M. The Robocup synthetic agent challenge. In: Proc Fifteenth Int Joint Conference on Artificia Intelligence (IJCAI97), San Francisco, CA, 1997. pp 24–49.
27. Parker LE. Distributed algorithms for multi-robot observation of multiple moving targets. Auton Robots 2002;12(3):231–255.
28. Parker LE, Touzet C, Fernández F. Techniques for learning in multi-robot teams, In: Robot teams: from Diversity to polymorphism A. K. Peters Publishers, 2002; pp 191–236.
29. Fernández F, Borrajo D, Parker L. A reinforcement learning algorithm in cooperative multi-robot domains. J Intel Robot Syst 2005;43(2–4):161–174.
30. Ernst D. Tree-based batch mode reinforcement learning. J Machine Learning Res 2005;6:503–556.
31. Forbes JRN. Reinforcement learning for autonomous vehicles. PhD thesis, Graduate Division of the University of California at Berkeley, 2002.
32. Modha DS, Spangler WS. Feature weighting in k-means clustering. Machine Learning 2003;52:217–237.
33. Smart WD. Making reinforcement learning work on real robots. PhD thesis, Department of Computer Science at Brown University, Providence, RI, May 2002.
34. Quinlan JR. Induction of decision trees. Machine Learning 1986;1(1):81–106.
35. Frank E, Witten IH. Generating accurate rule sets without global optimization. In: Proc Fifteenth Int Conf on Machine Learning 1998.
36. Witten IH, Frank E. Data mining. Practical machine learning tools and techniques with Java implementations. San Francisco, CA Morgan Kaufmann; 2000.
37. Fernández F, Isasi P. Automatic findin of good classifier following a biologically inspired metaphor. Comput Inform 2002; 21(3):205–220.
38. Fernández F, Isasi P. Evolutionary design of nearest prototype classifiers J Heuristics 2004;10(4):431–454.
39. Reynolds SI. Adaptive resolution model-free reinforcement learning: Decision boundary partitioning. In: Proc Int Conf Machine Learning 2000; pp 783–790.
40. Anderson C, Crawford-Hines S. Multigrid Q-learning. Technical report, Colorado State University, Boulder, CO, 1994.
41. Kushner HJ. Numerical methods for stochastic control problems in continuous time. SIAM J. Control Optim 28, 1990.
42. Shibata T, Kato T, Wad T. K-d decision tree: An accelerated and memory efficien nearest neighbor classifie . In: Proc Third IEEE Int Conf on Data Mining 2003; pp 641–644.
43. Dzeroski S, De Raedt L, Driessens K. Relational reinforcement learning. Machine Learning 43, 2001.