A Temporal Difference GNG-based Approach for the State Space Quantization in Reinforcement Learning Environments

GNG + Q(lambda)

Davi C. L. Vieira
Instituto Federal de Sergipe – IFS
Email: davi.vieira@ifs.edu.br

Paulo J. L. Adeodato
Centro de Informática – UFPE
Email: pjla@cin.ufpe.br

Paulo M. Gonçalves Jr.
Instituto Federal de Pernambuco – IFPE
Email: paulogoncalves@recife.ifpe.edu.br

***Abstract*—The main issue when using reinforcement learning algorithms is how the estimation of the value function can be mapped into states. In very few cases it is possible to use tables but in the majority of cases, the number of states either can be too large to be kept into computer memory or it is computationally too expensive to visit all states. State aggregation models like the self-organizing maps have been used to make this possible by generalizing the input space and mapping the value functions into the states. This paper proposes a new algorithm called TD-GNG that uses the Growing Neural Gas (GNG) network to solve reinforcement learning problems by providing a way to map value functions into states. In experimental comparison against TD-AVQ and uniform discretization in three reinforcement problems, the TD-GNG showed improvements in three aspects, namely, 1) reduction of the dimensionality of the problem, 2) increase the generalization and 3) reduction of the convergence time. Experiments have also show that TD-GNG found a solution using less memory than TD-AVQ and uniform discretization without loosing quality in the policy obtained.**

Keywords*—Reinforcement Learning, Adaptive State Space Partitioning, Growing Neural Gas, Q-Learning

I. Introduction

Reinforcement learning algorithms are widely used in tasks where the expected behavior is not known. In this kind of problem, the only information available is a reinforcement signal given by a critic [1]. It is the critic's responsibility to evaluate and encode in a reward signal the effect of actions taken by the agent. This reward signal indicates how "pleasurable" it is to perform a particular action at a given state [2]. The agent should learn a sequence of actions that maximizes the reward signal it receives in the long run. Actions that lead to states which give the largest immediate rewards may lead to others which do not [2].

The learning process of reinforcement learning algorithms relies on estimating value functions for each state-action pair. The value functions are estimations of the expected future rewards that can be received by the agent. In tasks with small state spaces, these value functions can be stored in a table addressed by state-action pair. This approach however suffers from the curse of dimensionality [3] where the search space grows exponentially as the number of states and actions increases. Another problem with this approach is the long time spent by the agent to fill all table entries, since the agent needs to continuously visit all states to make better estimates of the value function [2].

pb de tabular Q

Models that either approximate functions or perform some type of state aggregation can be combined with reinforcement learning algorithms as a way to reduce the large state space problem. While reinforcement learning algorithms estimate the value function of a state, models like multilayer perceptron and growing neural gas generalize the input space and perform the mapping of the value function into states.

One type of function approximation is CMAC [4], known as tile-coding due to Sutton and Barto [2]. This model was extensively used with success in many complex large domain environments which include Dribble [5], Cart-pole [6], Keep-away [7], Acrobot [2], Mountain Car [2] and many others. In this approach, the state space is quantized in regions called tiles forming a grid of tiles that is called a tiling. This approach can contain various tilings which are overlapped by an offset of $\frac{1}{c}$, where c is the number of tilings used. For each state, c overlapped tiles are activated and the value function is obtained by summing all the involving tiles [2]. This approach has some drawbacks that have already been reported by Whiteson *et al.* [8]. In their paper, two main limitations are exposed [8]: the first is that tile-coding requires a human designer to figure out the model, and the second is related to the degree of generalization that is fixed throughout the learning process. Research points out that performance increases only when generalization is gradually reduced over time [9]. Furthermore Vieira *et al.* [5] have shown that the amount of memory used increases with the number of states hence, imposing difficulties on convergence for very large spaces.

State aggregating algorithms like Kohonen maps [10] and Growing Neural Gas [11] can be used in reinforcement learning problems on large domains. Mountain car [12], robot navigation [13]–[15], puddle world [16], cart centering [17] and arm-control [18], [19] are examples of that nature. Basically most of these models are made by two Self Organizing Maps (SOM) networks; one is used to map the state space and the other is used for searching the action space [12], [18], [19].

One drawback when using SOM is the need to determine the number of nodes; too many nodes can slow down the learning considerably while too few nodes may not be enough to find a good solution. The goal is to have new nodes being added in the course of learning until some degree of convergence is reached. Several papers have been published focussed on this aspect. In [16], Baumann *et al.* presents the GNG-Q+ model to partition the state space by aggregating

GNG Q+: approche qui ajoute des noeuds
si chgmts dans politique

similar states. In this approach, nodes maintains a table which contains the actions value of the region represented by its centroid. At each iteration, Q-Learning is applied to the current approximation and, if an update causes a change in the policy, then a local error variable is incremented. The addition of new nodes occurs in regions where the local error variable exceeds a predefined threshold. One problem with this approach is related to the movement of existing nodes and to the addition of new nodes. To gather sufficient information about where to either add or move nodes, this process occurs only when an episode is finished. Thus, the node movement and the refinement process tend to be very slow, once that many steps will be necessary until a satisfactory representation of the state space be reached. This characteristic slows down the training phase considerably and can be a serious constraint to its application in real world.

TD AVQ:
TODO

Another approach that can adaptively partition the state space was presented by Lee *et al.* [20]. In this work, a vector quantization called TD-AVQ is used to adaptively partition the state space by tracking the accumulated reward received by the agent in one region before it activates a different region. If this accumulated reward surpasses a predefined threshold θ and the actual state s is at a distance greater than a constant parameter ρ, the state space is partitioned with the addition of a new codeword at s. The Q-Learning is not applied while a different region is not activated and in the meanwhile the rewards are discarded and only transition rewards are considered. In this case, the agent will not know the real cost to reach a different region. This problem potentially prevents the use of this approach in environments where different rewards can be received in the same episode as, for example, in puddle world.

pb de TD AVQ
dans puddel world

This paper presents TD-GNG, a new single layered algorithm based on Growing Neural Gas (GNG) that only grows when required. At each iteration, the algorithm looks for nodes that have value functions saturated and add new ones in places nearby. Thus, nodes with saturated value function tends to stabilize leading the network to good solutions. At this point no nodes are added. In TD-GNG, there is no need of the environment model. It operates in on-line fashion, therefore there is no need to store the experience. Experiments have shown that the proposed algorithm was capable of finding good solutions with few nodes. Also, the growth was well controlled leading the network to a good representation of the input space and value functions.

This paper is organized as follows. Section II presents some background of Temporal Difference algorithms that were used to build the proposed model, which is described in Section III. The testbed environments are presented in Section IV. Experiments on Section V compare the performance of TD-GNG, TD-AVQ and uniform discretization on three well known environments with continuous state. Section VI presents final considerations and the conclusions.

II. Temporal Difference

In reinforcement learning problems, the agent must estimate a value function for each visited state and perform actions that leads to states where the value function is maximal. In these states, the agent will receive larger rewards. In other words, the agent must learn a behavior policy π that maximizes the sum of future rewards r over the time t, i.e., $V^{\pi}(s_t) = \sum_{k=0}^{\infty} r_{t+k}$. The value function V^{π} reflects the expected return that will be received by following a policy π. A discount factor denoted by γ can be used to bound the return value in such a way that if $\gamma < 1$, the infinite sum will have a finite value. The agent will take future rewards into account more strongly as γ approaches one [2].

One way to estimate the value function is by Temporal Difference methods [21]. Learning by Temporal Difference refer to models that can learn without waiting for a final outcome. This kind of learning is very useful in problems where the experience does not break into episodes. In this kind of problems, interaction continues indefinitely, making it impracticable to wait until the end to start the learning process [2]. Q-Learning [22] is a temporal difference algorithm that can incrementally estimate the value function of a state-action pair by computing

$$Q(s,a) = Q(s,a) + \alpha[r + \gamma max_b Q(s', b) - Q(s,a)] \quad (1)$$

after every transition from a non-terminal state. In Equation 1, s' is the next state following s, a is the action performed in s, b is the greedy action in s', α is a step-size constant that indicates the learning rate, r is the reward received in s and γ is a constant parameter that indicates the discount factor.

To accelerate the learning process, a mechanism called Eligibility traces [23] can be combined with temporal difference algorithms. This mechanism consists in propagating a portion of the reward received in one state to all predecessors states, in the sense that recent states receive more credit or blame then old states. The implementation of this mechanism is made with the addition of a memory (e) with the same size as the Q-table and addressed by state-action pair. At each time step, the eligibility of the current state-action pair is incremented by one and all others are decremented by a fraction, a constant parameter λ.

The pseudo-code of Q-Learning with Eligibility traces ($Q(\lambda)$) is presented on Algorithm 1.

III. Proposed Algorithm

The proposed algorithm combines the GNG network with a modified $Q(\lambda)$ algorithm. The modified $Q(\lambda)$ is used for computing the value functions while the modified GNG algorithm performs the mapping of these values into the state space. The algorithm works by dividing a large set of states into groups, each having the estimation of the reward of regions represented by their centroid points. Each group is represented by one node that has its own action-table and e-table. The action-table has the estimation of the reward for each possible action and the e-table is used for eligibility traces.

The algorithm consists of 3 stages called Adaptation, Refinement, and Behavior and Learning, described in the following subsections.

comment les noeuds (centroide) sont deplaces

A. Adaptation

Growing Neural Gas (GNG) is an incremental network model able to learn the important topological relations in a given input vector space by continuously adding new nodes into its network which starts with two connected nodes [11].

Algorithm 1: Pseudo-code of Q-Learning with eligibility traces as described in [2].

```
Data: Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all $s$, $a$
repeat for each episode:
    Initialize $s, a$;
    repeat for each step of episode:
        Take action $a$, observe $r, s'$;
        Choose $a'$ from $s'$ using policy derived from $Q$(e.g., $\epsilon$-greedy);
        $a^* \leftarrow \mathtt{argmax}_b Q(s', b)$ (if $a'$ ties for the max, then $a^* \leftarrow a'$);
        $\delta \leftarrow r + \gamma Q(s', a) + 1$;
        $e(s,a) \leftarrow e(s,a) + 1$;
        forall pair $(s,a)$ do
            $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$;
            if $a' = a^*$ then
                $e(s,a) \leftarrow \gamma\lambda e(s,a)$
            else
                $e(s,a) \leftarrow 0$
            end
        end
        $s \leftarrow s'$;
        $a \leftarrow a'$;
    until $s$ is terminal;
until;
```

At each iteration, nodes can be connected, moved towards an input signal or added after a certain number of iterations. Nodes are either placed in regions that are not well represented or removed from regions where the activity is too low. A local accumulated error variable and edges are used for these purposes.

gng classic: how nodes move

In the modified GNG algorithm, nodes are moved towards regions where the probability of receiving rewards are high. These regions are identified by evaluating the value function with the condition $r + \gamma\mathtt{argmax}_a Q(w', a) > \mathtt{argmax}_b Q(w, b)$ and moving nodes a fraction towards it. This will result in a behavior where nodes tend to move towards the goal state making a path to it.

As nodes represent knowledge about each region stored in their action-tables, removing them means loss of that knowledge. Thus, nodes aren't removed in the proposed model. No change was made in the manner of how nodes are connected. The algorithm is activated when a new representation of the environment's state is perceived by the agent. The closest node, according to the Euclidean distance, to this state is called the winner and it is the one that has the right to respond to it. The output of the winner node will be the next action to be performed by the agent. Algorithm 2 presents the proposed modified GNG algorithm.

B. Refinement

For the addition of new nodes, a threshold is used in places represented by nodes where actions value exceeds it. Instead of using a fixed threshold, the algorithm estimates the maximum value that actions value can assume. If a discount factor is used action-value functions will have a limit that can be calculated

deplacement des noeuds, creation des arcs

Algorithm 2: Modified GNG algorithm.

```
Function GNG(s, s') is
    Input: actual state $s$ and next state $s'$
    Find the nearest node $w$ and the second-nearest node $l$ to $s$;
    Find the nearest node $w'$ to $s'$;
    if $r + \gamma\mathtt{argmax}_a Q(w', a) > \mathtt{argmax}_b Q(w, b)$ then
        Move the winner node $w$ and all its neighbors $n$ a fraction $e_w$ and $e_n$ toward the state $s$;
    end
    if $w$ and $l$ are connected by an edge then
        Set the age of this edge to zero;
    else
        Create a new edge between $w$ and $l$;
    end
    Increment the age of all edges emanating from $w$;
    Remove edges with an age larger than $a_{max}$;
    Decrease parameters $e_w$ and $e_n$;
end
```

using the equation,

$$Q(s,a) \leq \left| \frac{r_t}{(1-\gamma)} \right| \tag{2}$$

Equation 2 is only true if r is constant over time which is true only in special cases, e.g, the agent gets stuck in a region and is unable to get out since the number of nodes are insufficient to represent a good solution. In this case, the action-value function will saturate and no learning will occur. This can be avoided if new nodes are added in these regions before the saturation of the action-value function by defining a threshold described by the following equation,

$$\theta = \left(r_{t+1} + \gamma^1 r_{t+2} + \cdots + \frac{\gamma^{n-1} r_{t+n}}{(1-\gamma)} \right) c \tag{3}$$

where c is a constant parameter with a value between 0 and 1. This parameter defines the maximum value that an action-value function can assume before new nodes are added.

C. Behavior and Learning

One of the main problems when combining reinforcement learning algorithms with models that discretize the state space is the loss of the convergence property. Figure 1 (a) shows a particular case on a small gridworld where reinforcement learning algorithms are unable to find a solution. The environment was split in 4 regions numbered and delimited by two solid thick lines. In the fourth region, to reach the goal two different actions which depends on the agent's position are needed: at position $(3,4)$ the best action would be to go right while at position $(4,3)$ the best action would be to go up. However, this behavior is impossible to reach since these two particular states are in the same region and therefore will share the same action-value function. Instead of performing a uniform discretization, the environment can be split in a more efficient way as shown on Figure 1 (b). In this case, the environment was split in 2 regions providing a more simple and efficient partition schema of the state space, whereas the optimal behavior can now be reached.

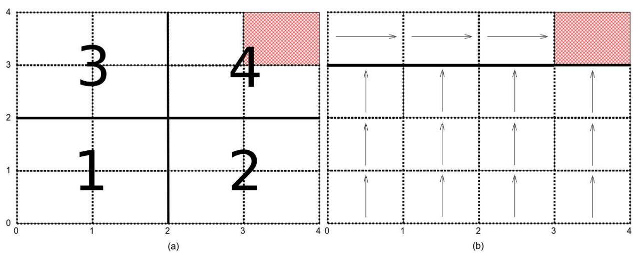


Fig. 1. Uniform and a non-uniform discretization on small gridworld.

In the example of Figure 1 (a), to reach the goal the agent needs to know where it came from to determine its next action. For example, if agent reaches the region 4 from state $(1, 3)$ the next action must be to go right but if it came from state $(3, 1)$ the next action must be to go up. In this case, the agent has a "dependence of path", in contrast with the "independence of path" of environments that have the Markov property [2]. This problem can happen in the majority of reinforcement learning algorithms, which discretize the state space. In order to minimize the loss of the Markov property, the proposed approach prevents the agent from using different actions in the same region. For this to happen, learning only occurs when either a different region is activated or the episode ends. If one region is activated for a long period of time, than its value function will saturate and then a new node will be added. This helps the agent keep track of actions that lead to different regions and, meanwhile, the agent accumulates the rewards. This modification in the Q-Learning algorithm turns it to a special kind of algorithm called n-step TD prediction [2],

pb discretisation S:
perte markov

$$Q(w, a) = Q(w, a) + \alpha[R^{(n)} - Q(w, a)]e(w, a) \quad (4)$$

a comprendre ...

In Equation 4 the terms $Q(w, a)$ and $e(w, a)$ mean the action-table and eligibility traces memory of node w, respectively. The term $R^{(n)}$ is the n-step return that can be expanded to $\sum_{i=1}^{n} \gamma^{i-1} r_{t+1} + \gamma^n \mathtt{argmax}_b Q(w', b)$, where n is the number of steps until either the agent reaches a different region activated by w' instead of that activated by w or the episode ends. The decay of eligibility traces is determined by the number of the n steps until the change of the current active region as described bellow,

$$e_t(w, a) = \begin{cases} 1 & \text{if } w = w_t \text{ and } a = a_t \\ (\gamma\lambda)^n e_{t-1}(w, a) & \text{otherwise} \end{cases} \quad (5)$$

for all w, a.

Algorithm 3 shows the complete algorithm and in Figure 2 the learning scheme is presented.

IV. Testbed Environments

This section describes three testbed environments used to evaluate the proposed algorithm. The first testbed is called Mountain Car. In this task, a car is placed between two hills without enough energy to reach the goal state at the top of the right hill. The only way to reach the goal state is driving the car up the left hill as a way to get more energy. Figure 3 illustrate the Mountain Car environment.

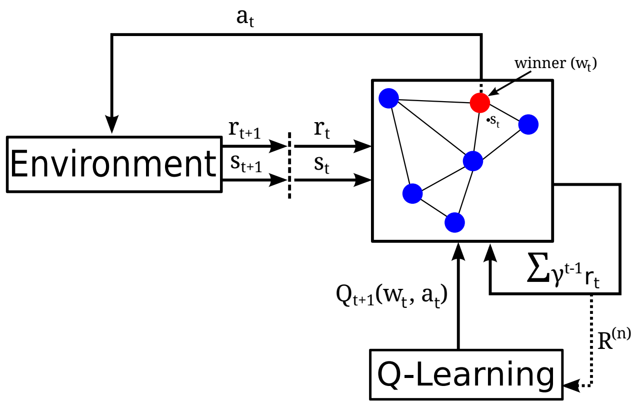


Fig. 2. Learning scheme of TD-GNG.

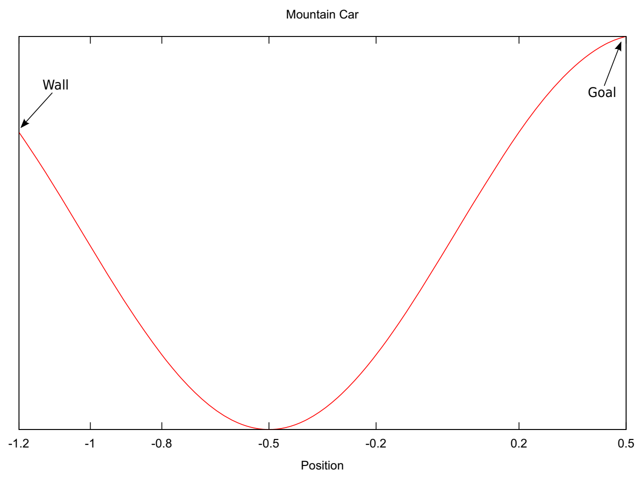


Fig. 3. Mountain Car.

The agent can perform two actions, which leads the car to left ($a_t = -1$) or right ($a_t = 1$), a reward of -1 is given on all time steps until it reaches the goal position in which it receives a reward of 0. The state of the environment is given by two continuous variables that describe the position (x_t) and velocity ($\overline{x}_t$) of the car and are updated by the following equations as described by Singh and Sutton [24],

$$x_{t+1} = x_t + \overline{x}_{t+1} \quad (6)$$
$$\overline{x}_{t+1} = \overline{x}_t + 0.001a_t - 0.0025 \times \cos(3x_t) \quad (7)$$

and bounded by

$$x_{t+1} = \min(\max(-1.2, x_{t+1}), 0.5) \quad (8)$$
$$\overline{x}_{t+1} = \min(\max(-0.07, \overline{x}_{t+1}), 0.07) \quad (9)$$

When the car reaches the top of the left hill, the velocity is reset to zero, which simulates the presence of an inelastic wall. If the car reaches the top of the right hill, then the episode is terminated. Each episode starts at a random state.

In the second task, the agent is situated in a continuous two-dimensional state environment called puddle world. The objective of the agent in this task is to reach the goal region

Algorithm 3: TD-GNG

```
repeat for each episode
    Initialize $w, a$;
    repeat for each step of episode:
        Take action $a$, observe $r, s'$;
        Find the nearest node $w'$ to $s'$;
        $R \leftarrow R + \gamma^n r$;
        $n \leftarrow n + 1$;
        if $R + \gamma^n Q(w', a) \leq \left(R + \frac{\gamma^n r}{1-\gamma}\right) c$ then
            Place at position $s'$ a new node $k$;
            $w' \leftarrow k$;
        end
        if $w \neq w'$ then
            Choose $a'$ from $w'$ using policy derived from $Q(e.g., \epsilon - greedy)$;
            $a^* \leftarrow \mathtt{argmax}_b Q(w', b)$ (if $a'$ ties for the max, then $a^* \leftarrow a'$);
            $\delta \leftarrow R + \gamma^n Q(w', a^*) - Q(w, a)$;
            $e(w, a) \leftarrow 1$;
            forall pair $(w, a)$ do
                $Q(w, a) = Q(w, a) + \alpha \delta e(w, a)$;
                if $a' = a^*$ then
                    $e(w, a) \leftarrow (\gamma\lambda)^n e(w, a)$
                else
                    $e(w, a) \leftarrow 0$
                end
            end
            $n \leftarrow 0$;
            $R \leftarrow 0$;
            $a \leftarrow a'$;
        end
        GNG($s, s'$);  // refers to Algorithm 2
        $w \leftarrow w'$;
    until $s$ is terminal;
until;
```

at right bottom corner with 0.05 of size along each of the two dimensions. At each time step, the agent can move in four directions (left, right, up and down) and is rewarded by -1 except if is in the puddle which receives an additional penalty. As described in [25], these penalties were -400 times the distance into the puddle (distance to the nearest edge). The puddles were 0.1 in radius and were located at center points (.1, .75) to (.45, .75) and (.45, .4) to (.45, .8). The episode is terminated and the agent receives a reward of 0 if its location is in the goal region. A random Gaussian noise with standard deviation 0.05 is also added into movements to add more difficulty to the task. The environment described is shown in Figure 4.

The third and last environment used in our experiments is a 20x20 two-dimensional maze as showed on Figure 5. In this task the agent has to solve a maze by reaching the goal state fixed at position (19, 2). At each time step, the agent receives a reward of -1 until it reaches the goal state where it receives a reward of 0. The agent starts at a random position and can perform four actions (left, right, up, down), if one of these actions moves it into the wall or out the limits of the world, the position of the agent is not changed and will receive a reward of -1.

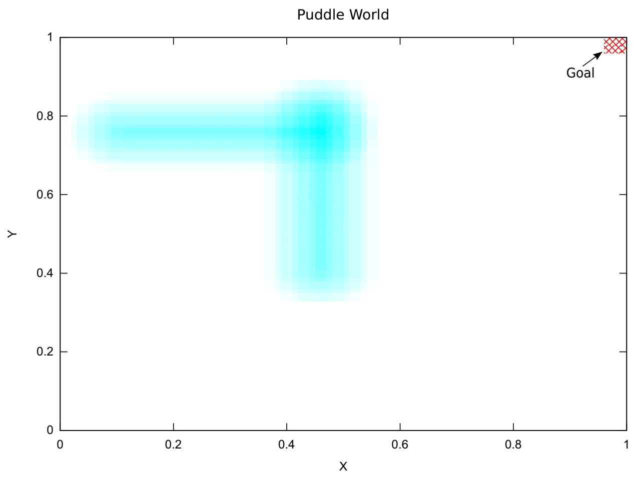


Fig. 4. Puddle World.

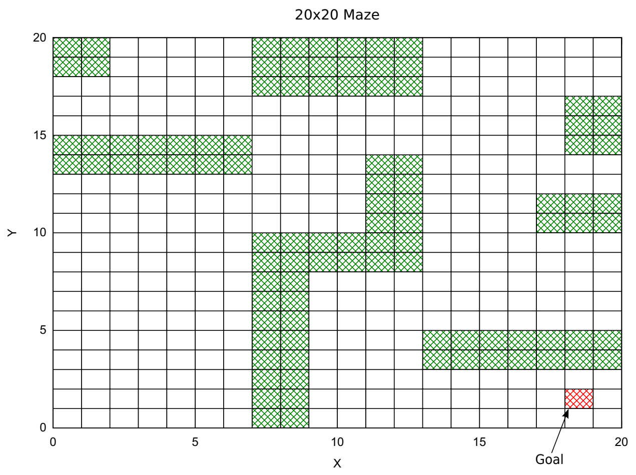


Fig. 5. 20x20 Maze.

V. Results and Discussion

To evaluate the proposed model, performance was measured on the three experimental environments described in Section IV and compared with uniform discretization and TD-AVQ algorithm with and without merging codewords. In mountain car and 20x20 maze environments, the performance metric was the number of steps until the agent reach the goal state. In the puddle world task, the accumulated reward was the performance metric chosen since the shortcut path may not be the best because it is possible for the agent to reach the goal state through puddles with larger penalties. All experiments were run ten times and the performance was averaged and plotted.

For TD-GNG, the following parameter settings were applied in all experiments: $\alpha = 0.1$, $\gamma = 0.999$, $E = 0.1$, $\lambda = 0.9$, $e_w = 0.01$, $e_n = 0.001$, $a_{max} = 100$, $\theta = 0.5$, with

trace: nb de pas jusqu'à but ou accumulated reward + taille du modèle (nb de features)

dicretisation uniforme: experte

exception of the mountain car environment were $E = 0$ was set. In this environment, a long sequence of actions is needed to reach the goal and a random action can ruin all this sequence, so exploration has not been used. For uniform discretization, several configurations for the discretization of the state space were tested. Best results were found when the state space was discretized in 8×8 regions for mountain car and 20×20 regions for puddle world and maze environments. Also, several configurations for α, λ, E and γ were tried and best initial results were found when $\alpha = 0.1$, $\lambda = 0.9$ and $\gamma = 0.95$. The E parameter was set to 0.1 in 20x20 maze and puddle world and 0 in the mountain car environment. After searching the parameters space of TD-AVQ, the following settings were common on the three environments: $\alpha = 0.1$, $\gamma = 0.95$, $E = 0$ for mountain car and $E = 0.1$ for the remaining, and $\lambda = 0.9$. The parameters θ and ρ were 5 and 0.1 for puddle world, and 15 and 0.01 for 20x20 maze and mountain car.

Figures 6 to 8 shows the averaged performance and the growth of the models on mountain car, 20x20 maze and puddle world. In general, all experiments show that TD-GNG achieved a better performance than TD-AVQ and uniform discretization. In the beginning of the learning process, when few nodes are representing the state space, all adaptive algorithms take more time to reach the goal. Only after the first episodes in the learning process when new features have already been added and the state space is well represented, the model's performance improves considerably. This happens with most adaptive models which start with a coarse representation of the state space.

In all experiments TD-GNG was capable of finding a near optimal policy in less than 50 episodes. The TD-AVQ with and without merge, reached similar results in mountain car task but in 20x20 Maze the convergence was reached after 250 episodes. As expected, TD-AVQ was unable to find a good policy in the puddle world task and, in some episodes, the agent was unable to reach the goal and the episode needed to be restarted. This can be seen when the accumulated rewards have some peaks as shown in Figure 8. The uniform discretization showed the worst performance in all experiments, even in the first episodes when the adaptive methods did not have yet an adequate representation of the input space. In uniform discretization, some similar regions may have been separated, causing an increase in the number of distinct regions to visit therefore making the algorithm spend more time to find a near-optimal policy than in the proposed model. Random Gaussian noise added to the agent actuator in the puddle world task showed that TD-GNG was robust against it, which is an important characteristic, since it is a very common type of noise affecting actuators and sensors in the actual world.

In the proposed model nodes are added very quickly in the beginning of learning because the actual number of nodes at start are insufficient to represent the input space and therefore its action-value functions saturate more quickly, but after some episodes, the growth tends to converge and the performance begins to improve. In all experiments, the amount of nodes was less than the number of codewords used by TD-AVQ, even when merging of codewords was used. The amount of nodes used by TD-GNG to solve the 20x20 maze (see Figure 9 (a)) was approximately 80 from the total of 302 valid states, a reduction of approximately 74% without loosing

TABLE I. MEMORY (KB) SPENT BY TD-GNG, TD-AVQ WITH AND WITHOUT MERGING IN LEARNING FOR 500 STEPS OVER 10 RUNS.

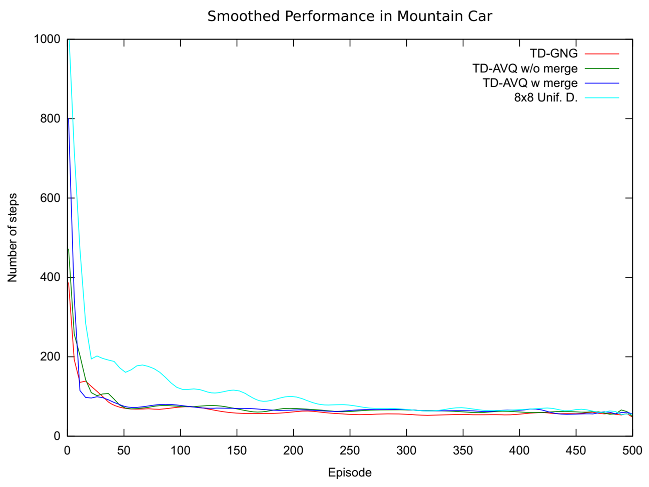
		Memory usage	
Algorithm	**Environment**	**Memory average**	**Standard deviation**
TD-GNG	Mountain Car	0.2 kb	0.01
	Puddle-World	1 kb	0.05
	Maze 20x20	1 kb	0.11
TD-AVQ with Merging	Mountain Car	0.36 kb	0.04
	Puddle-World	0.75 kb	0.03
	Maze 20x20	1.85 kb	0.1
TD-AVQ without Merging	Mountain Car	0.46 kb	0.04
	Puddle-World	0.78 kb	0.03
	Maze 20x20	2.12 kb	0.06
Uniform Discretization	Mountain Car	0.8 kb	0
	Puddle-World	6 kb	0
	Maze 20x20	6 kb	0

performance. In TD-AVQ model without merging, the amount of codewords used for the 20x20 maze was near 140 and near 120 when merging codewords was used but there was a slightly decrease in the performance when merging of codewords was used. In the mountain car problem, that have continuous state variables, the proposed model was also able to make a good representation of state space with few nodes that were smaller than the number of codewords used by TD-AVQ. In average, the numbers of nodes utilized by the proposed model were 25 and 80 nodes against 59 and 50 codewords utilized by TD-AVQ without merging, and against 46 and 49 codewords utilized by TD-AVQ with merging for mountain car and puddle world (see Figure 9 (b)), respectively. In puddle world, TD-AVQ uses less codewords than nodes but the performance was relatively worse than in TD-GNG.

Another point investigated was the total amount of memory used, presented at Table I. In the training process both TD-AVQ methods and uniform discretization spent more memory than TD-GNG in two of the three tested environments. Since TD-AVQ uses more codewords than nodes, the memory it used was greater than TD-GNG did. The same happens with the uniform discretization, where the state space was partitioned in more regions than nodes. However, TD-GNG was quicker in reaching a near-optimal policy than the other tested methods in all experiments carried-out. This indicates that the proposed model maps the input space more efficiently.

VI. CONCLUSION

Empirical results have shown that the algorithm proposed in this paper improves the applicability of reinforcement learning in complex environments by reducing the size of the state space. With the aggregation of similar states, the proposed model was capable of solving problems with both continuous and discrete state variables, reaching a good representation of the state space with few nodes representing a much larger amount of states. Despite having achieved similar performances on the mountain car problem, the proposed TD-GNG has solved it consuming much less computational resources than the TD-AVQ did. This feature allows its use in embedded applications where the amount of memory is critical. In the 20x20 maze, using less resolution than TD-AVQ and uniform discretization, the proposed algorithm reached convergence more quickly. In the puddle world problem, while the TD-AVQ was incapable of converging, the TD-GNG has converged in less than 200 episodes. The growth of the network seems to be well controlled, showing to converge as the



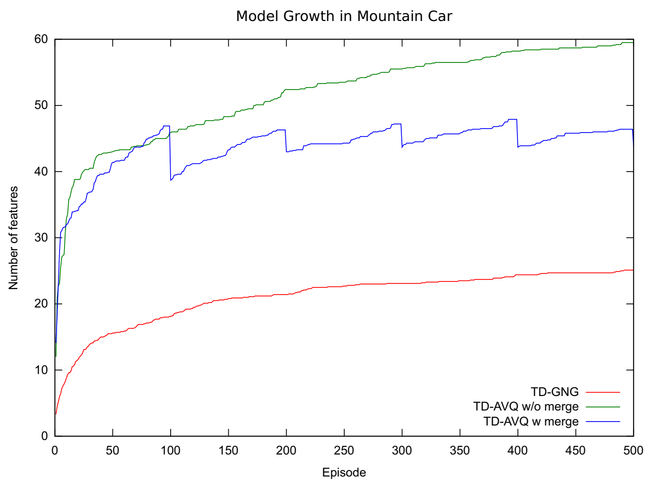
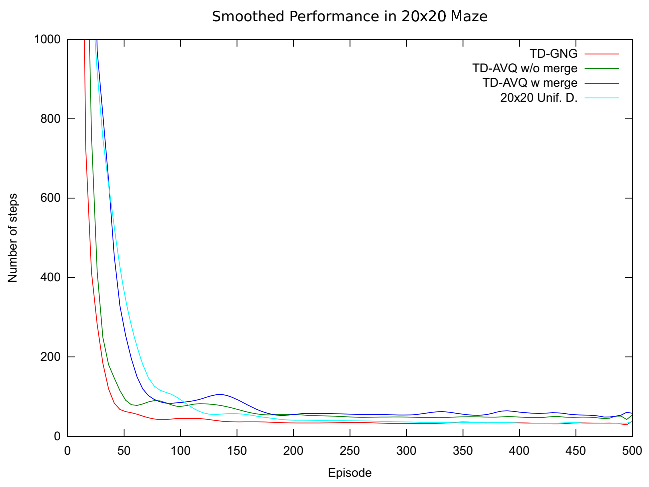


Fig. 6. Performance and Model Growth in mountain car.



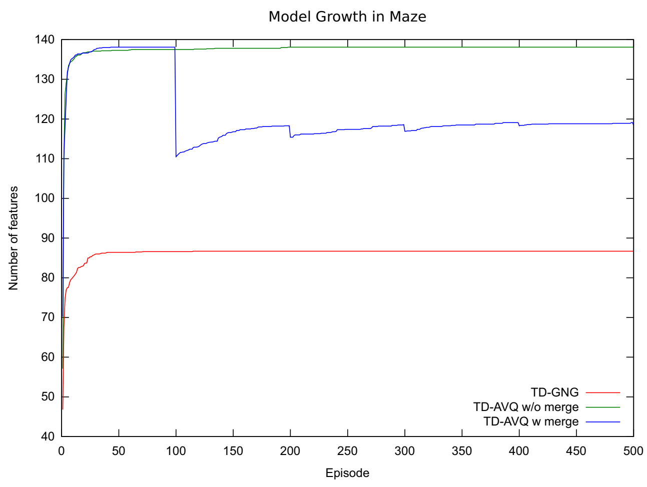
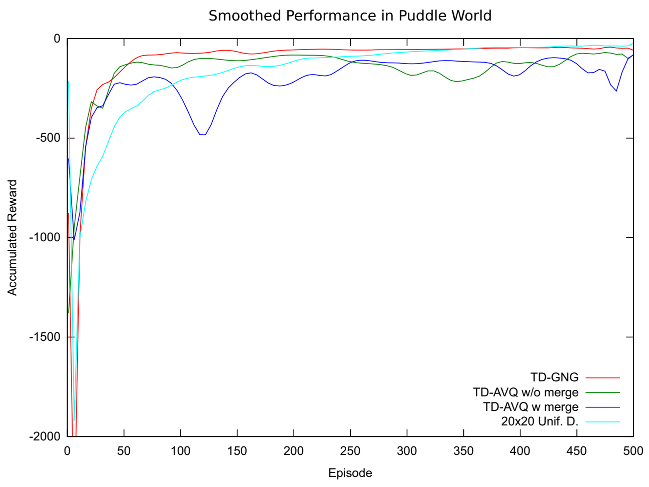


Fig. 7. Performance and Model Growth in 20x20 maze.



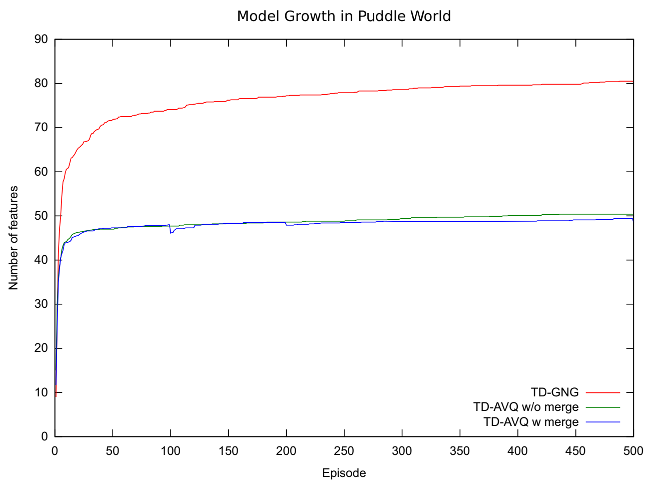
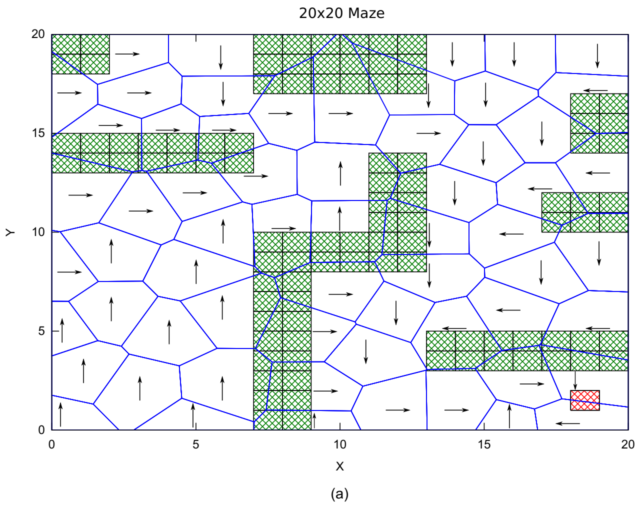


Fig. 8. Performance and Model Growth in puddle world.



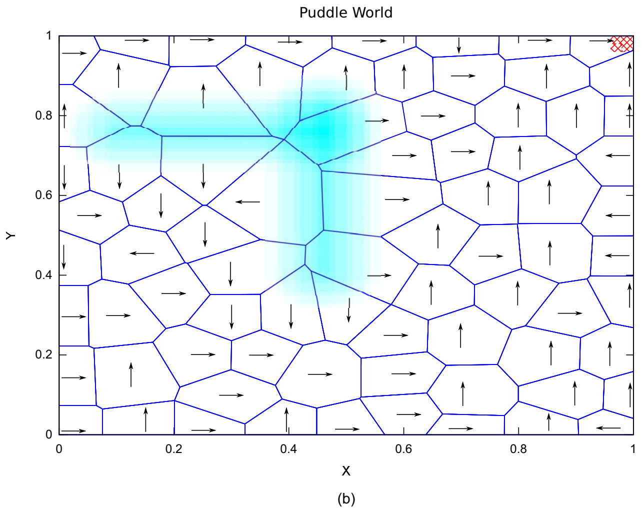


Fig. 9. State space partition performed by TD-GNG in Puddle World and 20x20 Maze environments.

performance increases. The proposed algorithm has shown to be less sensitive to variation in parameters such that in all experiments the same settings were used. In all experiments the TD-GNG algorithm has shown to be capable of reducing the dimensionality of the problem, increasing the generalization, and reducing the convergence time of RL algorithms.

References

[1] B. Widrow, N. K. Gupta, and S. Maitra, "Punish/reward: Learning with a critic in adaptive threshold systems," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. SMC-3, no. 5, pp. 455–465, 1973.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge University Press, 1998, vol. 1, no. 1.

[3] R. Bellman, *Dynamic Programming*, 1st ed. Princeton, NJ, USA: Princeton University Press, 1957.

[4] J. S. Albus, "A new approach to manipulator control: The cerebellar model articulation controller (cmac)," *Dynamic Systems, Measurement and Control*, vol. 97, no. 3, pp. 220–227, 1975.

[5] D. C. de Lima Vieira, P. J. L. Adeodato, and P. M. Gonçalves, Jr., "Improving reinforcement learning algorithms by the use of data mining techniques for feature and action selection," in *IEEE International Conference on Systems, Man and Cybernetics*, ser. SMC '10. Los Alamitos, CA, USA: IEEE Computer Society, October 2010, pp. 1863–1870.

[6] C. shin Lin and H. Kim, "Cmac-based adaptive critic self-learning control," *IEEE Transactions on Neural Networks*, vol. 2, no. 5, pp. 530–533, September 1991.

[7] P. Stone, R. S. Sutton, and G. Kuhlmann, "Reinforcement learning for robocup soccer keepaway," *Adaptive Behavior*, vol. 13, no. 3, pp. 165–188, November 2005.

[8] S. Whiteson, M. E. Taylor, and P. Stone, "Adaptive tile coding for value function approximation," University of Texas at Austin, Tech. Rep. AI-TR-07-339, 2007.

[9] A. A. Sherstov and P. Stone, "Function approximation via tile coding: Automating parameter choice," in *Abstraction, Reformulation and Approximation*, ser. Lecture Notes in Computer Science, J.-D. Zucker and L. Saitta, Eds. Springer Berlin Heidelberg, 2005, vol. 3607, pp. 194–205.

[10] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, September 1990.

[11] B. Fritzke, "A growing neural gas network learns topologies," *Advances in Neural Information Pprocessing Systems*, vol. 7, pp. 625–632, 1995.

[12] H. Handa, "State space construction of reinforcement learning agents based upon anticipated sensory changes," in *IEEE International Joint Conference on Neural Networks*, vol. 2, 2004, pp. 1115–1120.

[13] J. Provost, *Reinforcement learning in high-diameter, continuous environments*. ProQuest, 2007.

[14] A. P. S. Braga and A. F. R. Araújo, "A topological reinforcement learning agent for navigation," *Neural Computing & Applications*, vol. 12, no. 3-4, pp. 220–236, 2003.

[15] H. M. Gross, V. Stephan, and M. Krabbes, "A neural field approach to topological reinforcement learning in continuous action spaces," in *IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on Neural Networks*, vol. 3, 1998, pp. 1992–1997.

[16] M. Baumann and H. Kleine Büning, "State aggregation by growing neural gas for reinforcement learning in continuous state spaces," in *10th International Conference on Machine Learning and Applications and Workshops*, ser. ICMLA '11, vol. 1, 2011, pp. 430–435.

[17] M. Abramson, P. Pachowicz, and H. Wechsler, "Competitive reinforcement learning in continuous control tasks," in *In Proceedings of the International Joint Conference on Neural Networks (IJCNN*, 2003.

[18] A. J. Smith, "Applications of the self-organising map to reinforcement learning," *Neural Networks*, vol. 15, no. 8-9, pp. 1107–1124, 2002.

[19] H. Montazeri, S. Moradi, and R. Safabakhsh, "Continuous state/action reinforcement learning: A growing self-organizing map approach," *Neurocomputing*, vol. 74, no. 7, pp. 1069–1082, March 2011.

[20] I. S. Lee and H. Y. Lau, "Adaptive state space partitioning for reinforcement learning," *Engineering Applications of Artificial Intelligence*, vol. 17, no. 6, pp. 577 – 588, 2004.

[21] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.

[22] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[23] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge, 1989.

[24] S. P. Singh and R. S. Sutton, "Reinforcement learning with replacing eligibility traces," *Machine Learning*, vol. 22, no. 1-3, pp. 123–158, January 1996.

[25] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," *Advances in Neural Information Processing Systems 8*, pp. 1038–1044, 1996.