5th International Workshop on Evolutionary and Reinforcement Learning for Autonomous Robot Systems (ERLARS 2012)
Montpellier, France, August 28 2012

Improved State Aggregation with Growing Neural Gas in Multidimensional State Spaces

Michael Baumann[1] and **Timo Klerx**[2] and **Hans Kleine Büning**[2]

Abstract. Q-Learning is a widely used method for dealing with reinforcement learning problems. However, the conditions for its convergence include an exact representation and sufficiently (in theory even infinitely) many visits of each state-action pair—requirements that raise problems for large or continuous state spaces. To speed up learning and to exploit gained experience more efficiently it is highly beneficial to add generalization to Q-Learning and thus enabling the transfer of experience to unseen but similar states. In this paper, we report on improvements for *GNG-Q*, an algorithm that solves reinforcement learning problems with continuous state spaces and simultaneously learns a proper abstract state space. This approach combines Q-Learning and growing neural gas (an adaptive vector quantizer) to compute a state space abstraction. It starts with a coarse resolution that is gradually refined based on information achieved during learning. We improve the dealing with the non-determinism that may emerge in abstracted state spaces, suggest a new refinement strategy and propose a new criterion to decide when a refinement is necessary. Furthermore, we argue that this criterion offers an implicit local stopping condition for changes made to the approximation. Additionally, we employ eligibility traces to speed up learning. We evaluate the improved method in continuous state spaces with up to four dimensions and compare the results with several approaches from literature. Our experiments confirm that the modifications highly improve the efficiency of the abstract state space and that our approach is well competitive with existing methods.

1 Introduction

In reinforcement learning (RL), an agent has to learn a policy to solve a given problem—just from interaction with a (probably unknown) environment. It does so by trying its available actions in the different states of the environment and uses the (maybe delayed) scalar feedback—the reward—from the environment to update its estimation of the policy. The environment has to offer rewards in a way that the agent can learn a useful behavior by maximizing these rewards over time. RL problems are often modeled as Markov decision processes (MDP) [17] and can be solved with temporal difference methods that usually store the learned behavior in a tabular representation for each possible combination of states and actions. One well-known RL algorithm is Q-Learning [21] (for detailed information on other approaches we refer to [18, 17]) that is proven to converge to an optimal policy under several conditions [20]. These conditions include to visiting each state-action pair infinitely often. This requirement is fraught with problems in large or continuous state spaces that can be found in realistic settings. To use aforementioned table-based methods, continuous state spaces have to be discretized—a step that often needs domain knowledge to find a proper resolution of the approximation.

Large state spaces suffer from two severe problems: First, the *curse of dimensionality* (the search space grows exponentially in the number of states) induces high memory requirements. Second, the large amount of state-action pairs inhibits the agent to gather enough knowledge for each possible state as the probability to experience a certain state more than once decreases as the size of the state space increases. One way to cope with these problems is the use of generalization—i.e. transfer knowledge to unseen but similar states—that can e.g. be achieved by aggregating states [11]. For detailed overviews of other approaches, we refer to [4] or [19].

In this paper we analyze the *GNG-Q* approach [1] and investigate its behavior on a continuous state RL task: An agent is situated in a 2-dimensional environment and has to learn the shortest path from any position to the goal (cf. Sec. 5). Furthermore, we investigate two multi-dimensional problems: a d-dimensional continuous world and the acrobot swing up problem [16].

The idea of *GNG-Q* is to learn the behavior and its representation in parallel using a combination of Q-Learning and the unsupervised growing neural gas (GNG) vector quantizer [9]. *GNG-Q* assumes that similar states need similar behavior and computes a Voronoi tessellation of the state space and treats all states in one region equally. The core idea of *GNG-Q* is as follows: The approximation is initially very coarse and is refined in regions that contain incompatible states. In each learning step, the agent uses the current approximation to update its estimated policy. Simultaneously, changes in the learned policy point out regions that have to be refined. Thus, an *abstracted state space* is built by aggregating compatible states and this abstraction is adjusted based on the interaction during learning without knowing the environment in advance.

Our contribution is as follows: 1. We argue, that abstracted state spaces may introduce non-determinism and adapt the Q-update to better cope with this situation. 2. A new operation for refining regions of states is introduced. 3. We provide new criteria for the refinement and adaptation of the approximation and argue how these criteria lead to an implicit stopping condition for adjustments on the approximation. 4. Eligibility traces are incorporated to speed up learning. 5. We experimentally evaluate the influences of the parameters in the approach and compare its performance to other approaches. The enhancements in the updated algorithm called *GNG-Q*$^+$ lead to a significant decrease in the size of the approximation and an improved regulation of the refinement and adaptation. Furthermore, our adjusted use of the edges in the graph offers a means to model the state transition function for the abstracted states and allows to removing

[1] International Graduate School “Dynamic Intelligent Systems”, University of Paderborn, 33095 Paderborn, Germany, email: mbaumann@upb.de
[2] University of Paderborn, 33095 Paderborn, Germany

dead regions. Our experiments confirm, that *GNG-Q^+* is well competitive with other approximative approaches and that *GNG-Q^+* is able to efficiently compute a useful policy in parallel with a compact state space approximation. The proposed approach operates on-line and does not need the underlying model of the considered reinforcement learning problem. Additionally, it does not need to incorporate domain knowledge and the approximation offers flexible and adaptive shapes. After learning, the policy can be stored very efficient as only the Voronoi centers of the approximation and the associated action values are needed. The mapping of one state of the original state space to its abstraction is realized by a nearest neighbor rule and is thus very fast and easy to implement.

2 Related Work

Another approach that uses vector quantization was introduced in [11]. There, an adaptive vector quantizer is used to partition the state space while the agent is learning. The partitioning is carried out based on proximity in the state space and similarity regarding the rating of actions generated by Q-Learning. The approximation is refined if the reward accumulated in one region is exceeding some threshold and a predefined minimal distance to all neighboring code words is kept. In this approach, the centers of the created regions are not able to move and domain knowledge is required to determine useful values for the thresholds used. Fernández et al. [8] present the VQQL model that consists of the generalized Lloyd algorithm for vector quantization and Q-Learning. It uses vector quantization to obtain a set of codebook vectors that represent the state space. In a subsequent step, Q-Learning is used to learn a policy based on this reduced representation. The key difference to the *GNG-Q* and *GNG-Q^+* approaches is that Fernández et al. construct the state space representation independent from learning. Ratitch and Precup [13] also follow a similar approach as they place units as centers for an approximation. Their goal is to add a new unit if for the current state the number of nearby units is below some threshold.

Konidaris et al. [10] approximate continuous environments with the Fourier basis, a method that employs Fourier series to approximate the optimal value function. They compare their approach empirically to various other basis function approaches and conclude that its performance is similarly good. Unfortunately, this approach seems to be rather run time consuming.

Adaptive Tile Coding [22] is an approach that learns a policy in parallel with an appropriate state space abstraction. Starting with a very coarse approximation consisting of just one tile, it is refined based on information (e.g. based on the policy) from learning. The refinement operation splits one tile evenly in half, which will often lead to problems as it may happen that many splits are needed until incompatible states are separated. Another Tile Coding approach was presented in [12] where a genetic algorithm was used to decide upon refinements allowing unevenly splits. A drawback of this algorithm may be that it is executed for several resolutions of the approximation.

3 State Space Abstraction in Reinforcement Learning

This section presents the theoretical model of state space abstraction and shows, how the agent uses the transition and reward functions of the original Markov decision process (MDP) to update the abstracted MDP. Additionally, we describe the *GNG-Q* approach that we adapt in Sec. 4.

3.1 Theoretical Model

Single agent reinforcement learning problems are usually modeled as Markov decision processes (MDP). In this work, we consider continuous state spaces in deterministic environments with discrete time steps and discrete actions. Thus, we define a MDP as $\mathrm{M} = (S, A, \mathrm{T}, r)$ where the transition function $\mathrm{T} : S \times A \to S$ returns the succeeding state $\mathrm{T}(s_t, a_t) = s_{t+1} \in S$ after performing action $a_t \in A$ in state $s_t \in S$. The reward function $r : S \times A \to \mathbb{R}$ reflects the immediate merit of this execution.

Q-Learning [21] is one frequently employed algorithm to learn an optimal policy $\pi^\star$. It approximates the action-value function $Q^\star(s, a)$ that expresses the expected accumulated reward for performing action a in state s and following an optimal policy afterwards. The agent incrementally updates its approximation $\widehat{Q}$ of the action-value function $Q(s_t, a_t)$ during interaction with the environment: it executes a_t in s_t and observes the succeeding state s_{t+1} and the reward $r_t = r(s_t, a_t)$. The approximation is then updated according to $\widehat{Q}_{t+1}(s_t, a_t) := (1 - \alpha_t)\widehat{Q}_t(s_t, a_t) + \alpha_t\big[r(s_t, a_t) + \gamma \max_{a' \in A} \widehat{Q}_t(s_{t+1}, a')\big]$ with the learning rate α and discount factor γ. Q-Learning is proven to converge to the true Q-function given that each state-action pair is updated infinitely often, an exact representation of the policy is used and the learning rate α_t fulfills $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$ [20]. As pointed out earlier, large state spaces introduce severe issues and it is thus highly beneficial to introduce some kind of generalization [17]. Amongst many others (for detailed overviews see [19] or [4]), one approach to deal with large state spaces is the use of *state space abstractions*. Following [19], we define an abstract state space as follows:

Definition 1 (State Space Abstraction) *Let $\mathrm{M} = (S, A, \mathrm{T}, r)$ be a deterministic Markov decision process. We define the corresponding* abstracted MDP $\widehat{\mathrm{M}} = (\widehat{\mathcal{S}}, A, \mathrm{T}, r)$ *where $\widehat{\mathcal{S}}$ is a partition of the actual state space S and usually $|\widehat{\mathcal{S}}| \ll |S|$ holds. Each abstract state $\widehat{\mathbf{s}} \in \widehat{\mathcal{S}}$ is defined as a set $\widehat{\mathbf{s}} := \{s \mid \psi(s) = \widehat{\mathbf{s}},\ s \in S\}$ where the* abstraction-function *ψ is a mapping $\psi : S \to \widehat{\mathcal{S}}$ that maps each state of S to one of the states of the abstracted state space $\widehat{\mathcal{S}}$. Thus, ψ provides a partition of S with $\bigcup_{\widehat{\mathbf{s}} \in \widehat{\mathcal{S}}} \widehat{\mathbf{s}} = S$ and $\widehat{\mathbf{s}}_1 \cap \widehat{\mathbf{s}}_2 = \varnothing$, $\forall\, \widehat{\mathbf{s}}_1 \neq \widehat{\mathbf{s}}_2 \in \widehat{\mathcal{S}}$.*

The value functions in RL for an abstract MDP $\widehat{\mathrm{M}}$ can be learned from interactions with the original MDP M: The agent observes a state $s_t \in S$ and performs action a_t that takes it to the subsequent state $s_{t+1} = \mathrm{T}(s_t, a_t)$ and results in a reward $r(s_t, a_t)$. This information can be used e.g. in a Q-Learning update for the abstracted MDP [19]:

$$\begin{aligned}\widehat{Q}_{t+1}(\psi(s_t), a_t) &:= (1 - \alpha_t)\widehat{Q}_t(\psi(s_t), a_t) \\ &\quad + \alpha_t\Big[r(s_t, a_t) + \gamma \max_{a' \in A} \widehat{Q}_t(\psi(s_{t+1}), a')\Big] \qquad (1)\end{aligned}$$

Note, that the update for one abstract state $\widehat{\mathbf{s}}$ affects all states $s \in S$ that are abstracted to $\widehat{\mathbf{s}}$, i.e. all states s for which $\psi(s) = \widehat{\mathbf{s}}$ hold. This is a major advantage as one update affects several states and each (maybe unseen) state is treated equally as any other state abstracted to the same abstract state.

3.2 Growing Neural Gas State Quantizer

One approach to learn an approximation of the state space while performing reinforcement learning was presented in [1]. This approach learns the behavior and its representation in parallel: An adaptive vector quantizer (growing neural gas (GNG), [9]) is used to approximate the state space and in each learning step, Q-Learning is executed on the current approximation. The goal is to find a partition of the state space in regions such that each region contains states that can be treated equally. *GNG-Q* (summarized in Alg. 1) uses information collected during learning to adjust the approximation. Thus, the abstraction function ψ and the agent's approximation $\widehat{Q}$ for the abstracted MDP are learned in parallel.

GNG-Q aggregates states to so called *state regions* that are similar regarding some measure and require the same behavior. All states in one state region are treated equally and share one Q-vector. These regions are built using a growing neural gas: A set of units $n \in N$ called *neurons* are positioned in the state space and a Voronoi tessellation is created by a nearest neighbor rule that assigns any state $s \in S$ to the nearest neuron $nn(s) = \operatorname{argmin}_{n' \in N} \mathrm{d}(s, n')$ using some distance measure d. The abstraction function ψ is thus defined by the set N_t of neurons at time t and the nearest neighbor rule.

In each learning step, the nearest and second nearest neurons n_1, n_2 to the current state s_t are determined and both neurons are connected with a *neighborhood connection*. These connections are equipped with an age that is used to remove outdated connections. We call such connected neurons n_1 and n_2 *topological neighbors*.

The goal of the *refinement* is to relieve regions with incompatible states by splitting them. In the growing neural gas approach, a local *error* is introduced for each neuron. In the *GNG-Q* approach, the error is a counter for the changes in the policy in the respective region. Initially, the approximation has a very coarse resolution and in each learning step, Q-Learning is applied to the current approximation. Regions that need refinement are identified by monitoring changes in the policy learned so far: Every time, a Q-update causes that $\operatorname{argmax}_a \widehat{Q}_t(n_1, a) \neq \operatorname{argmax}_a \widehat{Q}_{t+1}(n_1, a)$ holds, the error for the current state's region is increased as this means, that the agent would now prefer a different action for this region. The approximation is periodically refined by adding a new neuron in the region with the highest error because this is evidence that the region consists of states that have to be treated separately to obtain a good policy.

In the generic growing neural gas approach, neurons are *moved* in order to adapt to the probability distribution by which the samples of the input space are drawn. In RL one has to deal with sequences of samples as the agent interacts with the environment in such a way that it iteratively transitions from one state to a subsequent state. Thus, the network in *GNG-Q* would try to follow the trajectories of the agent. In order to prohibit this behavior and to supply a static approximation during each episode, an additional set is introduced for each region: The so-called *regional states* R_n store the states, the agent visited in n's region during the current episode. After each episode, each neuron n is moved a small portion ϵ_b towards the centroid of R_n. Additionally, each topological neighbor of n is moved a much smaller portion $\epsilon_n \ll \epsilon_b$ towards the centroid of R_n. Thus, the positions of the neurons adapt to states visited during the last episode.

4 Revised Growing Neural Gas Q-Learning

This section argues that non-determinism may occur in the abstract state space although the original MDP is deterministic. We argue how to better deal with this non-determinism, show how to add eligibility traces to speed up learning, investigate the role of neighborhood connections and introduce criteria for the movement and the refinement as well as a revised refinement method. Finally, we sum up the new algorithm called *GNG-Q*$^+$ in Alg. 2.

Algorithm 1: GNG-Q (the numbers in braces indicate changes made in *GNG-Q*$^+$)

```
foreach episode do
    while episode not finished do
        observe $s_t$, perform $a_t$, observe $s_{t+1}$ and $r(s_t, a_t)$
        use current approximation to compute $\psi(s_t)$, $\psi(s_{t+1})$
        update Q-estimation according to Eq. 1                    (1)
        update neighbor connections and error values              (2)
        if network still adapts then insert a new neuron every
        $\lambda$'th iteration                                    (3)
    if network still adapts then
        adapt $n$ and $n$'s topological neighbors to centroid of $n$'s
        regional states $R_n$                                     (4)
```

4.1 Dealing with Induced Non-Determinism

Although we consider deterministic environments—the state transition function as well as the reward function is deterministic—the aggregation of states can introduce non-determinism in the abstracted MDP: Consider the situation in a shortest path scenario depicted in Fig. 1: If the agent performs the action "go right" in one of the states abstracted by the region $\widehat{\mathrm{s}}_1$, then, depending on its location in this region, the subsequent state can be in region $\widehat{\mathrm{s}}_2$ or $\widehat{\mathrm{s}}_3$. As the agent updates its estimates of $\widehat{Q}(\widehat{\mathrm{s}}_1, \rightarrow)$ depending on the Q-vector of the succeeding state, the Q-values are prone to oscillate. This problem occurs, if there are at least two states s_1, s_2 in one region $\widehat{\mathrm{s}}$ that result in states that are abstracted by different abstract states after performing the same action, formally: $\exists s_1 \neq s_2 \in \widehat{\mathrm{s}}, \exists a \in A : \psi(\mathrm{T}(s_1, a)) \neq \psi(\mathrm{T}(s_2, a))$. This kind of non-determinism can be caused by irregular shaped regions (as in the *GNG-Q* approach, cf. Fig. 1(a)) but may also occur whenever transitions between differently sized abstract states are possible (cf. Fig. 1(b)). In [8], this non-determinism is also called "the loss of the Markov property" as the subsequent state $\widehat{\mathrm{s}}_{t+1}$ now not only depends on $\widehat{\mathrm{s}}_t$ but also on the states visited before time t.

To improve the behavior with this non-determinism in the abstract MDP, we equip *GNG-Q*$^+$ with a learning rate α_t that depends on the time step t such that $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$ hold. Note, that this is the same condition on the learning rate as given in [20] for the convergence of Q-Learning. Following the idea of [7], such a learning rate can be constructed as $\alpha_t = \frac{1}{1+\mathrm{visits}(\widehat{\mathrm{s}},a)^\omega}$ where $\mathrm{visits}(\widehat{\mathrm{s}}, a)$ is the number of executions of action a in the abstracted state $\widehat{\mathrm{s}}$ and ω is a constant to regulate the decrease of the learning rate over time. To fulfill the condition above, $0.5 < \omega \leq 1$ has to hold. This learning rate decreases the influence of updates of each $(\widehat{\mathrm{s}}, a)$ over time and helps to reduce the oscillation of the Q-values. The original *GNG-Q* did not consider this non-determinism and used a constant learning rate. Thus, the Q-update *(1)* in *GNG-Q*$^+$ is adapted to use a learning rate α_t as described.

Abstracting states transforms the learning problem in something close to a partially observable MDP (POMDP) [15, 19]. Unfortunately, this improvement does not provide a solution for the POMDP, but it introduces more stability to the learning process.

4.2 Adding Eligibility Traces

One way to speed up learning in reinforcement learning problems is the use of eligibility traces [6]. They offer a means to distribute immediate reward to all state-action pairs (s, a) that have been visited

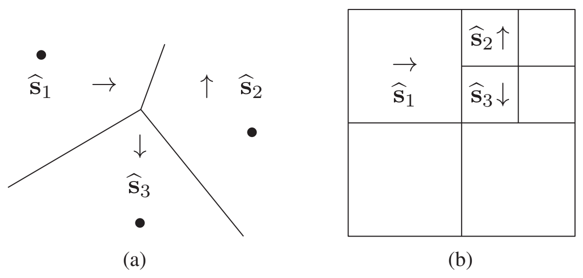

$\widehat{s}_1$ $\rightarrow$
$\uparrow$ $\widehat{s}_2$
$\downarrow$ $\widehat{s}_3$
(a)
$\widehat{s}_2\uparrow$
$\rightarrow$ $\widehat{s}_1$
$\widehat{s}_3\downarrow$
(b)

Figure 1. Induced non-determinism in different approximation schemes: The action $\rightarrow$ in $\widehat{s}_1$ may lead to different succeeding states depending on the actual state that is abstracted to $\widehat{s}_1$.

during the current episode according to their eligibility $e(s,a)$. This counter is increased by 1 every time the action a_t is performed in the current state s_t and if a_t is the action with the highest Q-value for state s_t. If a_t is not the maximal action, the eligibility traces for all state-action pairs are reset to zero. Additionally, each $e(s,a)$ is decayed by a factor $\lambda \in [0,1]$ for all state-action pairs:

$$e_{t+1}(s,a) = \begin{cases} \gamma\lambda(e_t(s,a)+1) & \text{if } s = s_t \text{ and } a = a_t = a^\star \\ 0 & \text{if } a_t \neq a^\star \\ \gamma\lambda e_t(s,a) & \text{if } s \neq s_t \text{ or } a \neq a_t \end{cases}$$

with $a^\star = \operatorname{argmax}_{a'} \widehat{Q}_t(s,a')$. Thus, reward or punishment can be credited for all state-action pairs that were "responsible" for it. If the agent performs an exploratory action (i.e. an action that has not the highest Q-value for the current state), the eligibility traces are cut off.

In every update, the temporal difference error δ_t is computed as $\delta_t = r - \widehat{Q}_t(s_t,a_t) + \gamma \max_{a'} \widehat{Q}_t(s'_t,a')$ between the current state s_t and the succeeding state s_{t+1}. This value is added to the Q-value of every state-action pair:

$$\widehat{Q}_{t+1}(s,a) = \widehat{Q}_t(s,a) + \alpha_t\delta_t e_t(s,a),\ \forall s \in S, \forall a \in A$$

The method used here is called *Watkins's Q(λ)* [17]; for a comparison of other approaches see [17, 6]. The transformation of this approach to neurons is straightforward: For each neuron n, we use $e_t(n,a)$ to express the eligibility for this neuron-action pair (compare Alg. 2).

4.3 Neighborhood Connections

In each update, the nearest neuron n_1 to the current state s and the nearest neuron n'_1 to the subsequent state are connected if $n_1 \neq n'_1$. Thus, the neighborhood connections abstract the transitions of the original MDP to the abstracted MDP: Each connection between two abstract states $\widehat{s}_1$ and $\widehat{s}_2$ implies that an action performed in a state s_1 with $\psi(s_1) = \widehat{s}_1$ resulted in a state s_2 with $\psi(s_2) = \widehat{s}_2$ (or vice versa as the connections are undirected). Every time, a neuron n_1 is the nearest neuron to the current state, the age of all connections (n_1, n') connecting n_1 with its neighbors n' is increased. Furthermore, a new connection (n_1, n'_1) is created with n'_1 representing the nearest neuron of s_2. If this connection already exists, its age is reset to zero. If the age of any connection exceeds age_{max}, the connection is removed as this is evidence, that this connection is outdated: Consider the latest age_{max} neuron-action pairs (n,a) then there was no action that lead from n_1 to n'_1 (or vice versa). Thus it is save to assume, that the approximation changed in a way that there is no action that leads from any state abstracted by n_1 to any state abstracted by n'_1 (or vice versa). If the deletion of connections results in isolated neurons, these may be removed as well, as they tend to be unreachable following the same argumentation.

Contrary to *GNG-Q* and the generic growing neural gas algorithm that use the neighborhood connections to adapt the nearest neuron and all its topological neighbors, *GNG-Q*$^+$uses the neighborhood connections to determine "dead" abstract states, i.e. abstract states that have not been visited for a long time (cf. Alg.1, *(2)*). Reasons for this could be changes in the environment or changes in the approximation due to adaptations or refinements. Additionally, the neighbor connections in *GNG-Q*$^+$ could be used to model an abstract transition function $\widehat{\mathrm{T}} : \widehat{\mathcal{S}} \times A \rightarrow \widehat{\mathcal{S}}$ for the abstracted MDP $\widehat{\mathrm{M}}$.

4.4 Adjusting the Approximation

After each episode, the approximation can be adjusted by two operations: The *adaptation* moves the neurons in such a way that they represent the state space as good as possible and the *refinement* is used to split regions that contain incompatible states. In the following, we will present changes to these operations and state new conditions for their application.

In the *GNG-Q*$^+$ approach, each neuron is moved by ϵ_b in the direction of the centroid of its regional states. Contrary to the *GNG-Q* approach, the positions of a neuron's topological neighbors (cf. Alg. 1, *(4)*) are not changed in order to increase the stability of the approximation learned so far. Thus, in *GNG-Q*$^+$ states visited in one region only affect the center of this particular region.

Especially, we only move a neuron n if its associated error value $\text{error}(n)$ is larger than a small threshold Δ (e.g. $\Delta = 1$). The intention is that we do not want to move a neuron, which is well positioned and has a useful Q-vector. It is intuitive to consider the error of a neuron for this purpose as this value is increased every time the policy in its region changes. Thus, the performance of neurons with high error values may increase by repositioning them whereas neurons whose local policy has not changed often recently shall keep their position. This behavior can be seen as parameter exploration as discussed e.g. in [14].

As each region is only assigned one Q-vector for all its contained states, it is important, that only compatible states are aggregated. During the learning steps, the policy is monitored and every time, the local policy in one region changes, the associated error is increased. In [1], after each λ_{insert} steps, the region of the neuron with the highest error is refined, unless a specific stopping condition is met.

In our approach, we refine the approximation *after* an episode, if $\sum_{n \in N_t} \text{error}(n) > |N_t|$ holds and at least λ_{insert} episodes have passed since the last refinement. Thus, the refined approximation can be adapted for some time and Q-vectors on the new approximation can be learned accordingly. Of course, one could refine the approximation whenever the sum of all errors is larger than zero. However, this might cause a too fine approximation, as sometimes a change in the policy is inevitable. The motivation for the condition above is, that on average each neuron is "allowed" to change its policy once per episode.

The refinement is done by cloning the neuron n_e with the highest error and perturbate n_e and the new neuron n^+ by a small amount to ensure that their initial positions differ slightly. The new neuron n^+ is initialized with the Q-vector of n_e and connected to the same topological neighbors. After this refinement, the error values of all neurons are reset to zero, to reflect the fact that the abstraction function ψ has changed. In *GNG-Q*$^+$ the insertion (*(3)* in Alg. 1) is performed after one episode.

The condition stated above implicitly provides a *stopping crite-*

rion for adjustments of the abstracted state space: If the errors of all neurons are small, this is evidence, that the overall policy has not changed often since the last insertion and the current policy can be expressed sufficiently with the current resolution. The *GNG-Q* uses an external measure to decide, when the approximation is fine enough, whereas *GNG-Q*$^+$ uses the above criteria on the error to decide when the approximation should be refined or moved.

As we change the number and the positions of the neurons over time, we also change the abstraction function because ψ is defined by the positions of the neurons and the used distance measure. After moving the neurons, one state s may be abstracted by a different neuron than before because it may now be in a different region. Additionally, if a new neuron is added, the number of regions is changed and thus, states may be in a different region after the refinement, too. The refinement also changes the domain of the estimated Q-function but the influence is rather low as the Q-vector in the two new regions is the same as before. Dead regions that are deleted also change the domain of $\widehat{Q}$ but this does not influence the approximation as these regions were not visited for a long time.

4.5 Complete Algorithm

This section presents the complete algorithm of our approach (cf. Alg. 2). To deal with different sized dimensions, it is useful to scale the values of the states to be from the same interval. A common approach is to normalize a value $x \in [x_{min}, x_{max}]$ to a value $x_{scaled} \in [x^{min}_{scaled}, x^{max}_{scaled}]$ such that

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \cdot (x^{max}_{scaled} - x^{min}_{scaled}) + x^{min}_{scaled}.$$

Thus, the distance function employed in the nearest neighbor rule weights all dimensions equally and no dimension will be favored just because its values are from a larger scale. In our approach we used a normalization to the interval [0, 1].

Algorithm 2: *GNG-Q*$^+$

```
foreach episode do
    initialize state $s$
    initialize regional states $R_n = \varnothing,\ \forall n \in N$
    initialize eligibility traces: $e(n,a) = 0,\ \forall n \in N,\ \forall a \in A$
    while episode not finished do
        /* interaction with environment */
        observe current state $s_t$
        determine nearest neuron $n_1 = \mathrm{nn}(s_t)$ to current state
        select and perform action $a_t$
        observe subsequent state $s_{t+1}$
        determine nearest neuron $n'_1 = \mathrm{nn}(s_{t+1})$ to $s_{t+1}$
        /* update neurons */
        $\mathrm{visits}(n_1, a_t) \leftarrow \mathrm{visits}(n_1, a_t) + 1$
        store $s_t$ in $n_1$'s regional states: $R_{n_1} \leftarrow R_{n_1} \cup \{s_t\}$
        discount errors for all neurons
        connect neurons $n_1, n'_1$
        increase age of all neighborhood connections of $n_1$
        /* update $\widehat{Q}$ */
        $\alpha_t = \frac{1}{\mathrm{visits}(n_1,a_t)^{\omega}}$
        $\delta_t = r - \widehat{Q}_t(n_1, a_t) + \gamma \max_{a'} \widehat{Q}_t(n'_1, a')$
        $e_{t+1}(n,a) \leftarrow e_t(n,a) + 1$
        foreach neuron $n \in N$ do
            foreach action $a \in A$ do
                $\widehat{Q}_{t+1}(n,a) \leftarrow \widehat{Q}_t(n,a) + \alpha_t \delta_t e_t(n,a)$
                if $a_t = \mathrm{argmax}_{a'} \widehat{Q}_t(s_t, a')$ then
                    $e_{t+1}(n,a) \leftarrow \gamma\lambda e_t(n,a)$
                else
                    $e_{t+1}(n,a) \leftarrow 0$
        /* Monitor changes in policy */
        if $\mathrm{argmax}_a \widehat{Q}_t(n_1, a) \neq \mathrm{argmax}_a \widehat{Q}_{t+1}(n_1, a)$ then
            increase $\mathrm{error}(n_1)$
    /* Adaptation of approximation */
    foreach neuron $n \in N$ do
        if $\mathrm{error}(n) > \Delta$ then
            compute centroid $\overline{s_n}$ of regional states for neuron $n$:
                $\overline{s_n} = \frac{1}{|R_n|} \sum_{s_r \in R_n} s_r$
            adaptation of neuron $n$ to $\overline{s_n}$:
                $w_n \leftarrow w_n + \epsilon_b \cdot (\overline{s_n} - w_n)$
    /* Refinement of approximation */
    if $\sum_{n \in N} \mathrm{error}(n) > |N|$ then
        insert new neuron in most erroneous region
```

5 Evaluation

In this section, we experimentally evaluate the *GNG-Q*$^+$ approach and compare the results with those of other approaches from literature. At first, the different problem domains are described, followed by a description of the evaluation setup and the default parameter setting. Then *GNG-Q*$^+$ and *GNG-Q* are compared in a 2-dimensional continuous world. After that, *GNG-Q*$^+$ and the Fourier approach [10] are evaluated in a d-dimensional continuous world and *GNG-Q*$^+$ is used to solve the acrobot swing up control problem. We compare our results in the acrobot domain to a baseline approach and additionally to several approaches from literature.

5.1 Problem Domains

Here, we describe the two different problem domains that we chose to evaluate the performance of *GNG-Q*$^+$: The *d-dimensional continuous world* is defined and the *acrobot swing up* problem from [16] is introduced.

5.1.1 The d-Dimensional Continuous World

To test the performance of *GNG-Q*$^+$ in higher dimensional spaces, we use an extension of the continuous world employed in [1]: In the *d-dimensional continuous world* the coordinates are from $[0,1]^d \subset \mathbb{R}^d$ and the agent has to learn the shortest path from all positions to the goal located in $s_{goal} := (1, 1, \ldots, 1) \in \mathbb{R}^d$. The agent can perform actions a^+_i and a^-_i for each dimension $0 \leq i < d$ that increase or decrease the value of the i-th component by $0 < s_{step} \leq 1$, i.e. it takes a step along dimension i. Thus, the agent's action set A consists of $2 \cdot d$ actions and the state space is $S = \{(x, y) \mid x, y \in [0,1]^d\}$. At the beginning of each episode, the agent is randomly placed inside the world and if it tries to leave the world in any dimension it is positioned on the border of this dimension. To relax the goal condition, the goal is modeled as a hypercube with the edge length equal to s_{step}. For the action that leads the agent to the goal, a reward of 0 is awarded, for all other action, the reward is -1. Clearly, for $d = 2$ this environment reduces to the environment from [2] and Sec. 5.4.

5.1.2 The Acrobot Swing Up Control Problem

As additional environment we considerd the "acrobot swing up" (sometimes called double pendulum swing up) control problem as it is defined in [16].

The acrobot is a two-link under-actuated robot with the goal to move the second link above a given height. The state space consists of four continuous dimensions, namely the angles $\theta_1, \theta_2 \in [-\pi, \pi]$ and the corresponding angular velocities $\dot{\theta}_1 \in [-4\pi, 4\pi]$ and $\dot{\theta}_2 \in [-9\pi, 9\pi]$. A sample state is shown in Fig. 2. The lengths of the

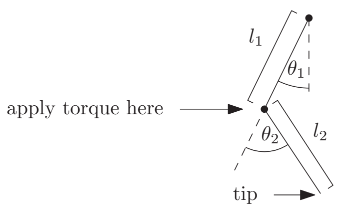


Figure 2. The acrobot from [16].

links ($l_1 = l_2 = 1$), their masses ($m_1 = m_2 = 1$), the gravity ($g = 9.8$), the goal height ($h = 1$) and the torque applied to the link are parameters that we chose in accordance to [16]. The behavior of the acrobot is calculated via formulas which can be found in [16], too. For our experiments we used a library[1] and adjusted it such that it fits the dynamics in [16].

5.2 Experimental Setup

In all experiments, one setting with a specific method and a fixed setup was simulated 100 times with different random seeds and averaged afterwards. Furthermore, the reward for the agent is always the same. If the agent reaches the goal state, it receives a reward of 0 and -1 in every other step. In each experiment, we initialized the Q-tables with zero for every state-action pair.

We divided the evaluation in a learning and a test-phase. In the learning phase we used an ε-greedy approach and learned for 10 episodes. In the following test phase, we tested the learned policy for 250 episodes in which the agent always chose the action with the highest Q-value. If the agent did not reach the goal after a fixed number of steps, this test-episode was stopped. During the evaluation phase, the agent was not allowed to learn.

For all our experiments in the continuous worlds, we chose the step size of the agent as $s_{step} = 0.05$ and the maximal number of trials in one episode as $\left(\frac{1}{s_{step}}\right)^d$. In the acrobot domain, we allowed the agent a maximal number of 5000 steps to learn and the evaluation was finished if either the goal was reached or 3000 steps have passed.

5.3 Parameter Values for Our Approach

To define useful basis values for our approach, we evaluated *GNG-Q*$^+$ for several parameter values on the 2-dimensional continuous world. In contrast to the other experiments, we only report on averages of 30 runs.

[1] http://library.rl-community.org

For the exploration strategy we used ε-greedy and experimented with $\varepsilon \in \{0.01, 0.05, 0.1, 0.15, 0.2, 0.3, 0.4\}$ and obtained the best results for $\varepsilon \in \{0.01, 0.05\}$ with $\varepsilon = 0.01$ being slightly better for all settings. The strengths of the adaptation towards the centroid of the regional states were chosen from $\{0.01, 0.025, 0.05\}$ and $\epsilon_b = 0.05$ performed best. We experimented with several exponents for the learning rate and chose $\omega \in \{0.51, 0.55, 0.6, 0.65, 0.7\}$ from which $\omega \in \{0.55, 0.6, 0.65\}$ performed best. As example, we investigated the influence of the number of episodes λ_{insert} between two refinements of the approximation and chose $\lambda_{insert} \in \{10, 20, 30, 40, 50\}$. The results are plotted in Fig. 3(a) and it can be seen, that $\lambda_{insert} \in \{30, 40, 50\}$ performs best while smaller values result in slower convergence. Note that after around 2000 episodes, all graphs reach their minimal values.

Based on theses experiments, we chose the following basic values for *GNG-Q*$^+$ (variations are mentioned correspondingly):

- decay factor for eligibility traces $\lambda = 0.9$
- number of episodes between two insertions $\lambda_{insert} = 40$
- maximal age of neighbor connections $\text{age}_{max} = 300$
- discount factor $\gamma = 0.9$
- exploration probability $\varepsilon = 0.01$
- exponent for the time dependent learning rate $\omega = 0.55$
- adaptation strength $\epsilon_b = 0.05$
- error decay $\beta = 0.9999$

5.4 Comparison between *GNG-Q* and *GNG-Q*$^+$

For the comparison of *GNG-Q* and *GNG-Q*$^+$ we employed the same scenario and the same parameter values as in [1] and thus, we used exploration rate $\varepsilon = 0.05$ and discount factor $\gamma = 0.95$ for both approaches, learning rate $\alpha = 0.1$, $\epsilon_b = 0.05$, $\epsilon_n = 0.0006$ and $\lambda_{insert} = 1000$ for *GNG-Q*. For the new approach, we used $\lambda_{insert} = 40$, $\epsilon_b = 0.05$ and for the eligibility traces $\lambda = 0.9$.

As we can see from Fig. 4, *GNG-Q* finds a good policy slightly faster in the beginning. After around 1000 episodes, *GNG-Q*$^+$ is as good as *GNG-Q* and remains more stable without high peeks, which can be seen in the zoomed section. Additionally, *GNG-Q*$^+$ needs less neurons than *GNG-Q* to represent the learned policy. Fig. 5 shows the number of neurons needed to represent the learned policy for which the number steps are shown in Fig. 4. *GNG-Q*$^+$ does not need more than 20 neurons on average with their number remaining stable whereas *GNG-Q* needs more than 100 neurons after 10000 episodes without stabilization of the number of neurons. Obviously, learning with *GNG-Q*$^+$ compared to *GNG-Q* results in a better policy while less neurons and thus less memory is needed.

Fig. 6 shows the approximation computed by one arbitrary run of *GNG-Q*. It can be seen in the left part, that the state regions are rather small and that the density of regions increases towards the goal. Although the policy (depicted in the right part of the figure) in combination with the computed approximation leads to an optimal behavior, the right part of this figure shows, that there are many redundant regions, e.g. all neighboring regions that point in the same direction. In Fig. 7(a), the state space approximation computed by one run of *GNG-Q*$^+$ together with the learned policy is shown. This result is clearly better than the one in Fig. 6 as the learned behavior for both approximation is identical but the approximation computed by *GNG-Q*$^+$ is much smaller.

Of course, the size of an approximation is not the only metric that has to be considered. However, for equal performances, a smaller approximation in number of states is definitely better than a larger one as this saves computing time and memory.

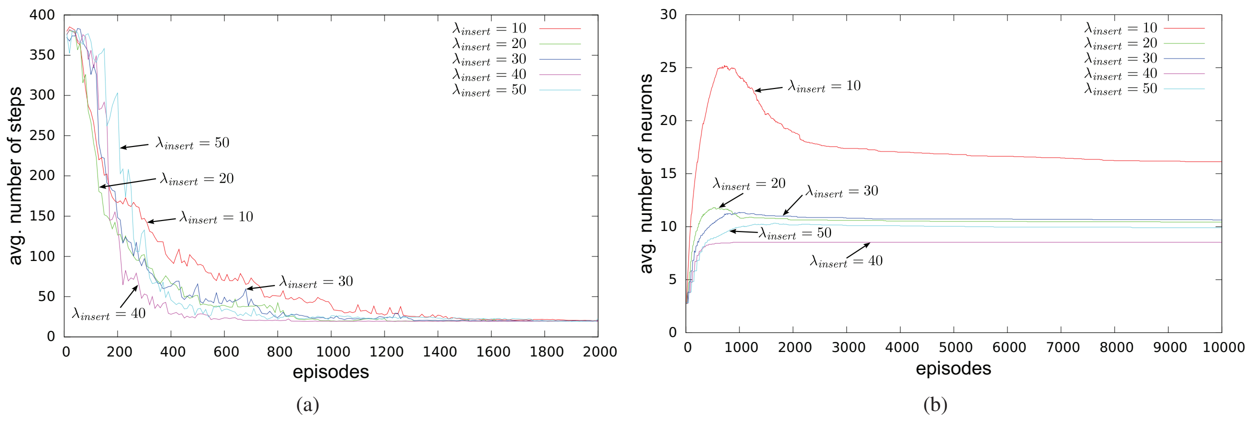


Figure 3. Average number of steps to reach the goal *(a)* and average number of neurons *(b)* with different values for λ_{insert} using *GNG-Q*$^{+}$.

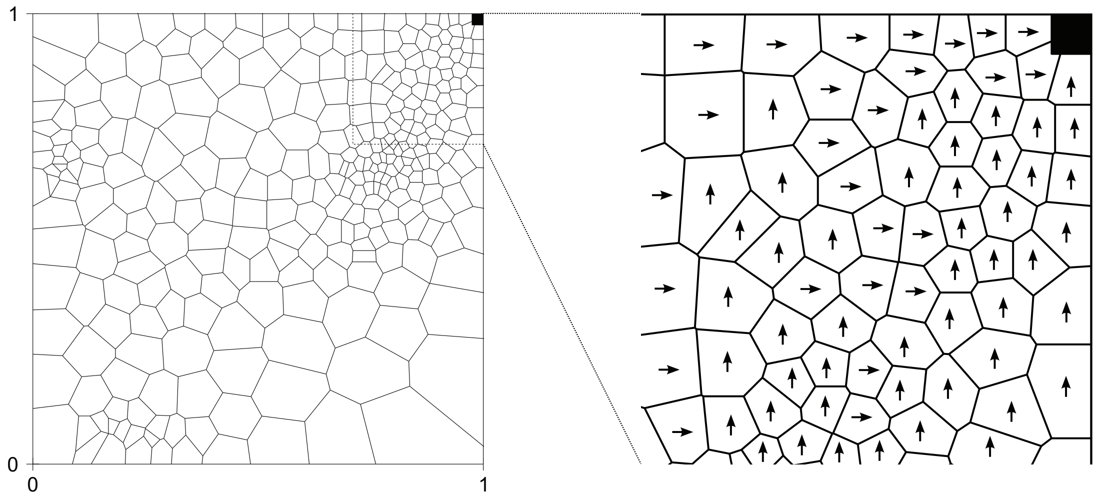


Figure 6. Approximation computed by one run of *GNG-Q*. The left part shows the abstracted state space and the right part depicts the learned policy for the dashed area.

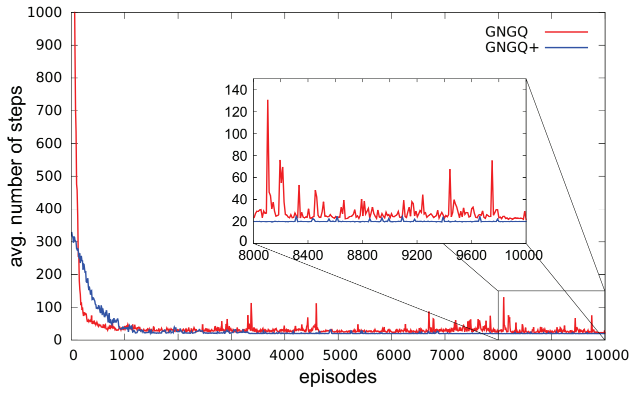


Figure 4. Average number of steps with *GNG-Q* and *GNG-Q*$^{+}$ in the scenario from [1].

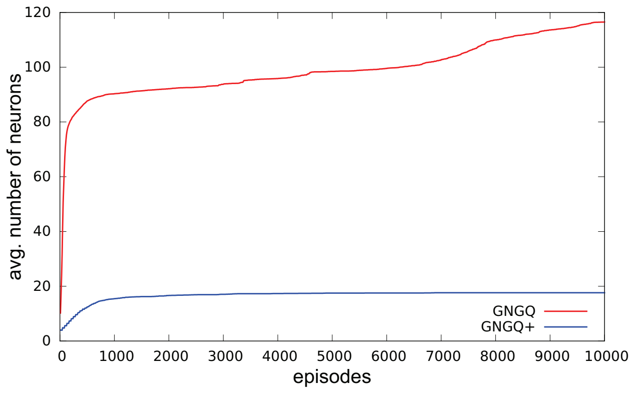
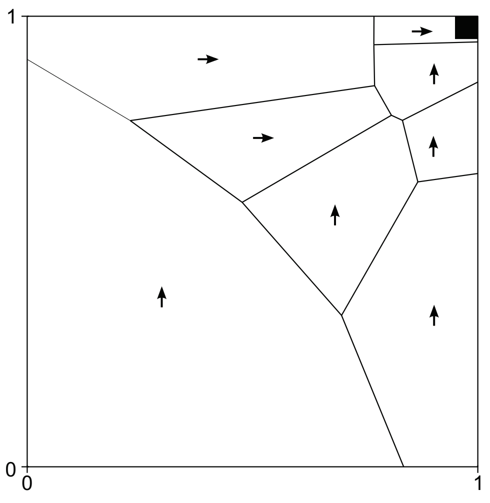


Figure 5. Size of the approximation in number of needed neurons with *GNG-Q* and *GNG-Q*$^{+}$ in the same scenario as in Fig. 4.

For the *GNG-Q*$^{+}$ approach, we investigated the distribution of the neurons in the 2-dimensional continuous world. For this, we collected the positions of all neurons from all evaluations and plotted them into the environment. The heat map in Fig. 7(b) gives an overview of the average distribution of the neurons: The opacity indicates the relation of number of neurons in this tile to the maximal number of neurons over all tiles ("the darker the tile, the more neurons fell into this particular tile").

We can see that the density of neurons increases towards the goal and that no neurons are positioned near $x = 0$ or $y = 0$. This stems from the fact, that the neurons are moved towards the centroid of all positions visited in the respective region. The distribution of neurons


(a)

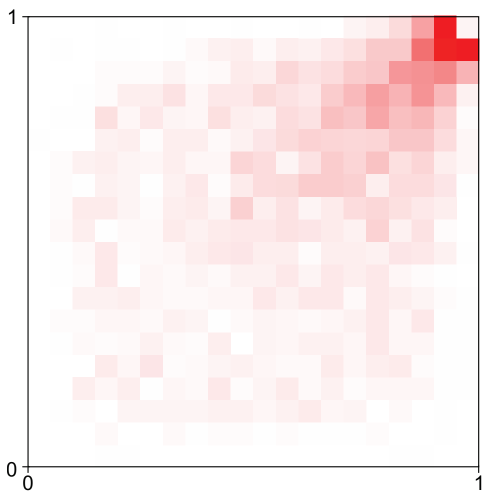

(b)

Figure 7. Exemplary state space with 8 states computed by one arbitrary run of *GNG-Q*$^+$ in (a) and heat map showing the density of average positions of all neurons from all evaluations in the 2-dimensional continuous world in (b).

in the old *GNG-Q* approach is similar but in *GNG-Q*$^+$, the number of neurons is drastically reduced (compare Fig. 5).

5.5 Experiments in Multidimensional Spaces

In this section we investigate the performance of *GNG-Q*$^+$ in two multi-dimensional environments and compare our results to approaches from literature.

5.5.1 The d-Dimensional Continuous World

Fig. 8 shows the comparison of *GNG-Q*$^+$ and the Fourier approach from [10] in a 3-dimensional continuous world with six actions. The parameters for the Fourier approach are taken from [10] for a similar environment called the “Discontinuous Room” (Fourier order = 5; $\lambda = 0.9$; $\alpha = 0.001$; $\gamma = 0.9$; $\epsilon = 0$). For the first 1000 episodes the Fourier approach outperforms *GNG-Q*$^+$. After that *GNG-Q*$^+$ remains more stable than the Fourier approach and needs less steps to reach the goal. Note that the implementation of the Fourier approach we used[2] was about 10-times slower than *GNG-Q*$^+$ (regarding computation time).

5.5.2 The Acrobot Swing Up Control Problem

We evaluated the performance of the acrobot for torque $\in \{1, 2, 5\}$ with our *GNG-Q*$^+$ approach and started from a down-hanging position, i.e. $\theta_1 = \theta_2 = \dot{\theta}_1 = \dot{\theta}_2 = 0$.

Fig. 9 shows the learning curves for the average number of steps for torque $\in \{1, 2, 5\}$. We can see that it takes about 12000 episodes to find a good policy for torque $= 1$. For torque $\in \{2, 5\}$, the learning curve does not improve significantly after 5000 episodes. After 40000 episodes the policies found by *GNG-Q*$^+$result in about 118 steps for torque $= 1$, 34 steps for torque $= 2$ and 17 steps for torque $= 5$ on average.

Fig. 10 shows the number of needed neurons for torque $\in \{1, 2, 5\}$. After 40000 episodes *GNG-Q*$^+$ uses around 60 neurons on average to approximate the state space for torque $= 1$, 30 neurons

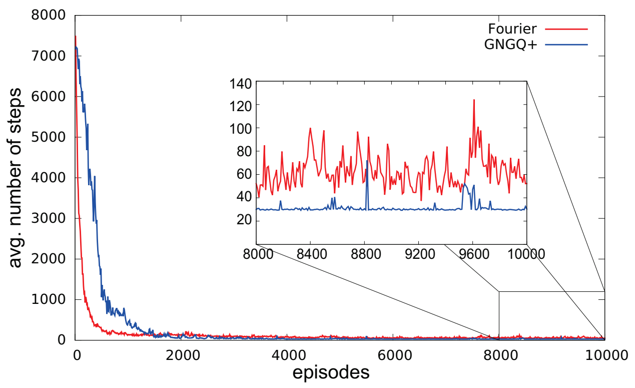


Figure 8. Average number of steps with *GNG-Q*$^+$ and Fourier based approach from [10] in a 3-dimensional continuous world.

for torque $= 2$ and 20 neurons for torque $= 5$. It seems that *GNG-Q*$^+$ would have inserted more neurons without an obvious benefit if the learning proceeded e.g. caused by exploration. As seen in Fig. 9 the policies for torque $\in \{2, 5\}$ do not improve after 5000 episodes but still, new neurons are inserted. This fact will be investigated in further research. Nevertheless, *GNG-Q*$^+$ can represent the four-dimensional state space of the acrobot problem in a very compact way with only 60 neurons on average for the hardest task.

For comparison, we employed the following baseline algorithm: We use Q-Learning on a predefined uniform discretization with a similar state space size as computed by *GNG-Q*$^+$ at convergence. Without any knowledge on the problem at hand, it is advisable to use the same resolution for each dimension. Of course, this may not be optimal as some dimensions might require a finer resolution than others. As the *GNG-Q*$^+$ approach needs approx. 61.2 neurons for torque=1, approx. 30 neurons for torque=2, and approx. 18.2 neurons for torque=5, we decided to split each dimension in 3 equal portions and thus the resulting state space has $3^4 = 81$ states. We also ran experiments for larger state spaces where the performance of this baseline approach improved but still was not satisfying. Nevertheless, this experiment should only serve as a rough comparison.

Fig. 11 shows the learning curves of the baseline approach for the

[2] http://library.rl-community.org/wiki/Sarsa_Lambda_Fourier_Basis_(Java)

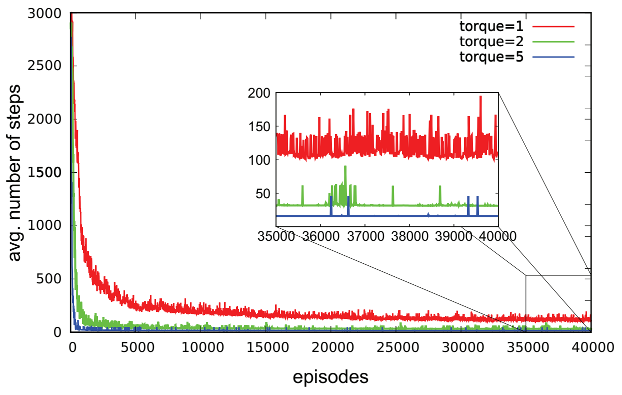


Figure 9. Comparison of the average number of steps for *GNG-Q*$^+$ in the acrobot domain with torque $\in \{1,2,5\}$.

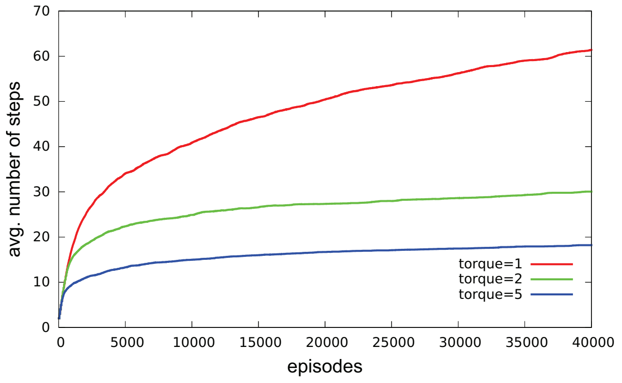


Figure 10. Comparison of the approximation size in numbers of neurons for *GNG-Q*$^+$ in the acrobot domain with torque $\in \{1,2,5\}$.

average number of steps with torque $\in \{1,2,5\}$. The learning curve for torque $= 1$ oscillates heavily, but the oscillation decreases with increasing torque. Table 1 shows the comparison in terms of mean values (μ) and standard deviation (σ) of *GNG-Q*$^+$ and the baseline approach for the last 5000 episodes. The values correspond to the learning curves from Fig. 9 and 11. Except for torque $= 5$, *GNG-Q*$^+$ outperforms the baseline approach considering the average number of steps needed to reach the goal by a factor of at least two. As already seen in Figs. 9 and 11 the learning curves of the baseline approach oscillate more than *GNG-Q*$^+$ which results in a higher standard deviation. For torque $= 5$ the baseline approach is slightly better than *GNG-Q*$^+$ on average over the last 5000 episodes, but the baseline approach needs approx. 15000 episodes to learn the good policy while *GNG-Q*$^+$ needs only 1000 episodes. Hence, *GNG-Q*$^+$ performs better than the baseline approach in most cases while needing less states. It seems that *GNG-Q*$^+$ places more neurons where a high density is necessary and only few neurons where a coarser resolution suffices.

Table 1. Comparison (mean value μ and standard deviation σ) of *GNG-Q*$^+$ and the baseline approach over the last 5000 episodes with torque $\in \{1,2,5\}$.

	GNG-Q$^+$		Baseline	
Torque	μ	σ	μ	σ
1	**117.69**	**16.53**	465.55	809.3
2	**33.18**	**5.98**	79.62	31.81
5	16.17	**2.66**	**15.25**	5.16

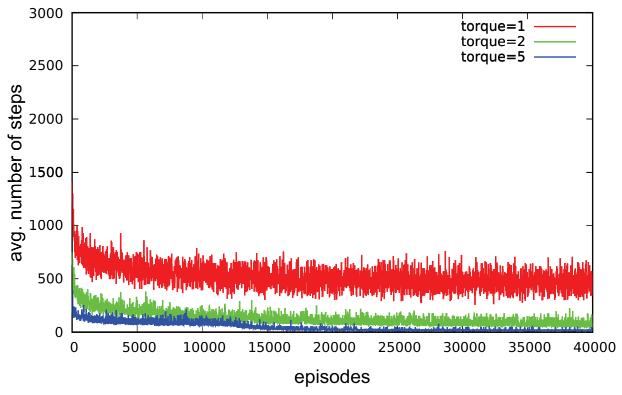


Figure 11. Comparison of the average number of steps for the baseline approach in the acrobot domain with torque $\in \{1,2,5\}$.

We also compared our results to others found in literature but it was not always possible to obtain precise values for the performance[3]. Furthermore, it was sometimes unclear which parameters values were used in the presented experiments. The *GNG-Q*$^+$ approach needs on average about 118 steps to reach the goal state from a down-hanging position. Always starting from that position the *best* policy found by *GNG-Q*$^+$ results in 72 steps. The policy found in [10] needs around 100 steps to reach the goal. Unfortunately, the start position and the torque applied are not mentioned for these experiments. The method described in [5] needs about 250 steps from the down-hanging position with torque $= 1$ whereas [3] needs at least 87 steps for the same setting. In [16], their CMAC approach needed around 85–90 steps.

5.6 Conclusion of Evaluation

In this section we compared *GNG-Q*$^+$ with *GNG-Q*, the Fourier approach, a baseline approach and other approaches from literature. The changes made to improve *GNG-Q* to *GNG-Q*$^+$ increased the performance regarding stabilization and needed neurons. Then we showed that *GNG-Q*$^+$ needs less steps at convergence than the Fourier approach. After that we investigated the difference between *GNG-Q*$^+$ and a baseline approach, resulting in more stable and in most cases better performance of *GNG-Q*$^+$. Finally, we presented results from comparable approaches.

6 Conclusion & Future Work

In this paper, we analyzed *GNG-Q*, an approach that uses a combination of Q-Learning and growing neural gas to learn a policy in parallel with a state space abstraction and proposed *GNG-Q*$^+$ that improves the former approach. The use of a non-deterministic Q-update, the incorporation of eligibility traces and the formulation of criteria for adjustments of the state space clearly improved the performance. Our evaluation showed that *GNG-Q*$^+$ is capable of learning compact state space representations in parallel with a (nearly) optimal policy in several continuous reinforcement learning problems with up to four dimensions and eight actions. Its performance is well competitive with other approaches from literature without the need of knowing the considered reinforcement learning problem beforehand.

For the future, we plan to incorporate a merging strategy to deal with the fact that Q-vectors of neighboring state regions may become

[3] We are aware that the actual results for the mentioned literature may be better than reported here and we do not want to discredit them. The values here should only show that our approach is comparable to existing methods.

similar during learning. The next step is to theoretically investigate the adapted method and to analyze convergence and optimality questions. Additionally, we will investigate how the number of neurons may be bounded further to avoid the increase caused e.g. by taking exploratory actions. Furthermore, the approach will be employed in the multi-agent reinforcement learning context. There, it will be investigated, how the proposed approach can deal with partial observability.

REFERENCES

[1] M. Baumann and H. Kleine Büning, 'State aggregation by growing neural gas for reinforcement learning in continuous state spaces', in *The Tenth International Conference on Machine Learning and Applications (ICMLA 2011)*, pp. 430–435. IEEE Computer Society, (2011).

[2] J. A. Boyan and A. W. Moore, 'Generalization in reinforcement learning: Safely approximating the value function', in *Neural Information Processing Systems 7*, pp. 369–376, (1995).

[3] K. A. Bush, *An Echo State Model of Non-Markovian Reinforcement Learning*, Ph.D. dissertation, Colorado State University, 2008.

[4] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška, 'Approximate reinforcement learning: An overview', in *Proceedings of the IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL 2011)*, pp. 1–8, (2011).

[5] A. da Motta Salles Barreto and C. W. Anderson, 'Restricted gradient-descent algorithm for value-function approximation in reinforcement learning', *Artif. Intell.*, **172**(4-5), 454–482, (2007).

[6] P. Dayan and T. J. Sejnowski, 'Td(lambda) converges with probability 1', *Machine Learning*, **14**(1), 295–301, (1994).

[7] E. Even-Dar and Y. Mansour, 'Learning rates for q-learning', *Journal of Machine Learning Research*, **5**, 1–25, (2003).

[8] F. Fernández and D. Borrajo, 'Two steps reinforcement learning', *International Journal of Intelligent Systems*, **23**, 213–245, (2008).

[9] B. Fritzke, 'A growing neural gas network learns topologies', in *Advances in Neural Information Processing Systems (NIPS) 7*, 625–632, MIT Press, (1995).

[10] G. Konidaris, S. Osentoski, and P. S. Thomas, 'Value function approximation in reinforcement learning using the fourier basis', in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, (2011).

[11] I. S. Lee and H. Y. Lau, 'Adaptive state space partitioning for reinforcement learning', *Engineering Applications of Artificial Intelligence*, **17**, 577–588, (2004).

[12] S. Lin and R. Wright, 'Evolutionary tile coding: An automated state abstraction algorithm for reinforcement learning', in *AAAI Workshops: Workshop on Abstraction, Reformulation, and Approximation (WARA-2010)*, (2010).

[13] B. Ratitch and D. Precup, 'Sparse distributed memories for on-line value-based reinforcement learning', in *Proceedings of the European Conference on Machine Learning (ECML)*, pp. 347–358, (2004).

[14] T. Rückstieß, F. Sehnke, T. Schaul, D. Wierstra, S. Yi, and J. Schmidhuber, 'Exploring parameter space in reinforcement learning', *Paladyn Journal of Behavioral Robotics*, **1**(1), 14–24, (2010).

[15] S. P. Singh, T. Jaakkola, and M. I. Jordan, 'Reinforcement learning with soft state aggregation', in *Advances in Neural Information Processing Systems 7*, (1995).

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

[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, The MIT Press, 1998.

[18] C. Szepesvári, *Algorithms for Reinforcement Learning*, Morgan and Claypool, 2010.

[19] M. van Otterlo, *The Logic of Adaptive Behavior*, IOS Press, Amsterdam, 2009.

[20] C. J. C. H. Watkins and P. Dayan, 'Q-learning', *Machine Learning*, **8**, 272–292, (1992).

[21] C. J. C. H. Watkins, *Learning from Delayed Rewards*, Ph.D. dissertation, Cambridge University, 1989.

[22] S. Whiteson, M. E. Taylor, and P. Stone, 'Adaptive tile coding for value function approximation', Technical report, University of Texas at Austin, (2007).