Exploring Continuous Action Spaces with Diffusion Trees for Reinforcement Learning

Christian Vollmer, Erik Schaffernicht, and Horst-Michael Gross

Neuroinformatics and Cognitive Robotics Lab
Ilmenau University of Technology
98693 Ilmenau, Germany
christian.vollmer@tu-ilmenau.de
http://www.tu-ilmenau.de/neurob

Abstract. We propose a new approach for reinforcement learning in problems with continuous actions. Actions are sampled by means of a diffusion tree, which generates samples in the continuous action space and organizes them in a hierarchical tree structure. In this tree, each subtree holds a subset of the action samples and thus holds knowledge about a subregion of the action space. Additionally, we store the expected long-term return of the samples of a subtree in the subtree's root. Thus, the diffusion tree integrates both, a sampling technique and a means for representing acquired knowledge in a hierarchical fashion. Sampling of new action samples is done by recursively walking down the tree. Thus, information about subregions stored in the roots of all subtrees of a branching point can be used to direct the search and to generate new samples in promising regions. This facilitates control of the sample distribution, which allows for informed sampling based on the acquired knowledge, e.g. the expected return of a region in the action space. In simulation experiments, we show how this can be used conceptually for exploring the state-action space efficiently.

Keywords: reinforcement learning, continuous action space, action sampling, diffusion tree, hierarchical representation.

1 Introduction

Reinforcement learning in continuous domains is an area of active research. Conventional algorithms are only proven to work well in environments where action space and state space are both discrete [1]. To extend those algorithms to continuous domains a common approach is to discretize the state space and the action space and apply discrete algorithms [2]. This, however, usually reduces the performance of the approaches [3]. One major issue when applying reinforcement learning to continuous domains is the lack of techniques to represent and update knowledge over continuous domains efficiently. Several successful approaches have been proposed that represent knowledge by means of parametric function approximators [3] or sample-based density estimation.

In this work, we present a novel approach to reinforcement learning in continuous action spaces, based on action sampling. In action-sampling-based approaches, the agent stores knowledge by means of a set of discrete samples, which are generated successively by a certain technique, one per learning step, and executed and evaluated thereafter by the agent. To store knowledge efficiently, those samples have to be concentrated on regions with high interest. Therefore, the sampling-technique has to use the knowledge acquired so far, to make the sampling process as informed as possible. In our approach, actions are sampled by means of a diffusion tree, which organizes samples from a continuous space and knowledge about the underlying domain in a hierarchical structure. Higher levels in the hierarchy represent knowledge about bigger regions in the action space. Evaluation of knowledge is done by recursively walking the tree from its root to its leaves. In a balanced tree, evaluation therefore is efficient. While walking down the tree, the stored knowledge is used to control the sample distribution. In this paper, we only outline the theoretical concept and validate it in a proof-of-concept manner. Further research has to be done to proof the full validity of the approach for real-world applications.

This paper is organized as follows. Section 2 briefly introduces the state of the art in sampling-based approaches to reinforcement learning. As a basis of our approach the Dirichlet Diffusion Tree is introduced in section 3. Our proposed algorithm is described in section 4. Section 5 shows results of two simple experiments conducted to conceptually validate our approach. Conclusions and an outlook to future work are stated in section 6.

2 State of the Art

Much research has been done in the field of reinforcement learning in continuous domains. In this section, we will outline a few techniques, strongly related to our proposed approach. Our algorithm belongs to the group of sampling-based approaches. Algorithms of that group typically represent knowledge by means of samples drawn from the underlying domain.

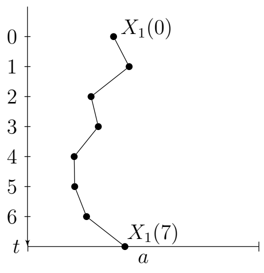
In [4] an approach is presented that extends the traditional dynamic programming to continuous action domains. However, the state space remains discrete. Values for states are stored in a table, one value per state. The policy is also represented as a table, where for every state an action is stored. Multilinear interpolation is used to compute values in the continuous state domain. In every iteration of the presented algorithm, a sweep through the whole state space is done where for every state a new action and a new value is computed. Therefore, an action is being sampled uniformly for every state. If the action is better than the previously stored one w.r.t. the expected return, the old action is discarded and the new one is stored instead. Unfortunately, this approach is not suited for real-time exploration and learning, due to the computational cost for the sweeps. Also sampling actions uniformly does not incorporate any knowledge about promising actions for a state seen so far and, thus, is inefficient for fast exploration. In [5,6] the idea of sampling actions is extended to a so-called tree-based sampling approach. For a state, a set of action samples is drawn. For every action the resulting successor state is simulated. In that simulated state again a set of action samples is drawn and again the next state is evaluated. That way a look-ahead tree is built. Based on that tree the expected long-term return of an action in the current state can be estimated. For this approach a generative process model is required, which narrows the applicability in practice. In [7] a sampling-based actor-critic approach is presented which operates on a discrete state space. For every state a set of action samples is maintained. With every action sample an importance weight is associated. Together, all samples for a state approximate a probability density function (PDF) over the continuous action space for that state. New action samples are drawn from that distribution by means of importance sampling. The weight of a sample is set proportional to the expected return of that action. Therefore, the approximated PDF has high values where actions are promising w.r.t. the expected return and thus are sampled and executed more often.

3 Mathematical Foundations

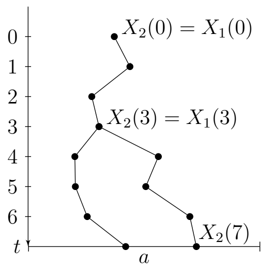
In this section, the necessary mathematical foundations will be introduced. We start with a brief definition of our notation for reinforcement learning and then introduce the formalism of the Dirichlet Diffusion Tree, which serves as a basis for our approach.

Reinforcement Learning: Our proposed approach is based on the idea of Q-Learning [1], a well known approach to reinforcement learning. The reader is assumed to be fairly familiar with this topic. We refer to [8] for a good and comprehensive introduction. In the following our notation of Q-Learning will be defined. The state of the agent will be denoted by $s \in S$, actions will be assumed to be equal for all states and will be noted by $a \in A$. The reward function is given by by $r = r(s,a) : S \times A \rightarrow \mathbb{R}$. Estimated action-values are defined by $\hat{Q}(s,a) = r(s,a) + \gamma\ \hat{V}(s')$. Where $\hat{V}$ is the estimated state value and γ is the discounting factor.

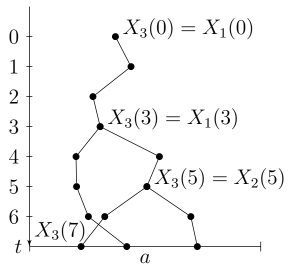
Dirichlet Diffusion Tree: Our approach is based on the idea of the Dirichlet Diffusion Tree (DDT) introduced in [9], in particular on the construction of such a tree, which will be outlined in the following (see Fig. 3. In a DDT samples are generated sequentially, each one by a stochastic diffusion process of duration $t = D$. The time evolution of a sample i is represented by a random variable $X_i(t)$ with $t \in [0, D]$. The start location of the first sample is set to $X_1(0) = 0$. The location of the sample an infinitesimal time step dt later is determined by $X_1(t + dt) = X_1(t) + N(t)$, where $N(t)$ is multivariate Gaussian with zero mean and covariance $\sigma^2 I dt$. The values $N(t)$ for distinct values of t are i.i.d., thus the time evolution of $X_1(t)$ is a Gaussian process. Lets call the so generated path X_1 (see Fig. 1(a)). For the second sample the start point of the new diffusion process, the path X_2, is set to the start point of the first one, hence $X_2(0) = X_1(0)$. The second sample then shares the path of the first sample up to a randomly sampled



(a) First path X_1.



(b) Second path X_2.



(c) Third path X_3.

Fig. 1. Evolution of a Dirichlet Diffusion Tree for three successively sampled paths with a length of $D = 7$. The first path (*left*) is sampled by accumulation of gaussian increments. The second path (*middle*) diverges from the first at time $t = 3$. The third path (*right*) shares the first part with the first path then goes along the second path and diverges at time $t = 5$.

divergence time T_d, where it diverges from the first path and goes its own way, which is again determined by a Gaussian process (see Fig. 1(b)). Thus for $t \leq T_d$ the paths are the same and for $t > T_d$ they are different. T_d is a random variable and is determined by a divergence function $a(t)$. The probability of diverging in the next infinitesimal interval dt is given by $p(T_d \in [t, t + dt])dt = a(t)dt$, where $a(t)$ is an arbitrary monotonically increasing divergence function (see [9] for details). As a result the probability of divergence increases monotonically in time during the diffusion process. Lets assume X_2 diverged from X_2 at time $T_d = t_0 = 3$.

Now the third path X_3 is being sampled. Lets assume, the point of divergence of the third path is $t_1 > t_0$, i.e. X_3 diverges later the X_2 did and $X_1(t) = X_2(t) = X_3(t)$ for $t \in [0, t_0]$,. Thus, when the process reaches $t_0 = 3$ a decision has to be made whether it should follow X_1 or X_2 until it diverges at $t = t_1 = 5$ (see Fig. 1(c)). This decision is done by randomly choosing from one of the branching paths with probability proportional to the number of previous times the respective path was chosen. Thus paths that have often been chosen before, are more likely to be chosen again. The concept of preferring what has been chosen before is called reinforcement of past events by [9] and is one of the main reasons which motivates the use of the DDT in our work. [9] further introduces an additional way to implement this concept by reducing the probability of divergence from a path X_i proportional to the number of times the path has been travelled before. Thus it is less likely to diverge from a path that has been used by many samples before.

After generating N paths $X_1, \ldots, X_N$, the values $X_1(D), \ldots, X_N(D)$ represent the set of samples generated if the DDT is viewed as a black-box sampling technique. We call those values final samples, as they are the final outcome of each diffusion process.

4 Our Algorithm

The algorithm proposed here borrows heavily from the idea of the diffusion tree and thus is called DT-Learning, where DT stands for diffusion tree. Like most other sampling-based approaches it operates on a discrete state space $S = \{s_i\}_{i=1,\dots,N_s}$. To represent values and actions, we maintain a diffusion tree for every state, where the domain of the samples is the action space of the agent. The following paragraph introduces the structural elements that make up a diffusion tree as used in our approach.

Structural Elements of Our Diffusion Tree: Unlike the continuous notion of the diffusion tree as presented in [9], the paths of our diffsion tree are discrete in time and consist of a sequence of concrete samples of the diffusion process, which we further call nodes. Further, we extend the notion of the diffusion tree by a structural element called segment, which comprises the set of nodes from one divergence point to another (see Fig. 2).

Let c be a segment and let $c[i]$ be the i-th node of c. That way, the segments themselves comprise a tree structure, where a segment has one parent segment and arbitrarily many child segments. One particular segment has no parent segment and is called the root segment. Segments without child segments are called leaf segments. The last node of a leaf segment is also a leaf node of the entire tree. In order to ease notation we will use a functional notation for attributes of an entity (a tree, a segment, or a node) in the following. Let $rt(s)$ be the root segment of the tree of state s. Let $pa(c)$ be the parent segment of a segment c and let $ch(c)$ be the set of child segments of segment c. In case c is a leaf segment, $ch(c) = \emptyset$. Let further $leaf(c)$ denote the last node of a segment c. If c is a leaf segment, $leaf(c)$ is also leaf node of the tree. A leaf node of the tree represents a final sample from the underlying domain. All intermediate nodes of all segments in the tree are just a byproduct of the sampling and have no particular use. Put differently, if we interpret the diffusion tree as a black-box sampling mechanism which just generates samples in the action space, we would only see the final samples represented by the leaf nodes. The remaining tree structure would be hidden in the box.

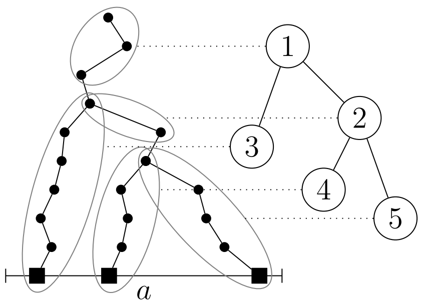


Fig. 2. Abstraction of a diffusion tree (*left*) to a tree of segments (*right*). Nodes in the diffusion tree make up a segment (*ellipses*). The segments themselves form a tree (*right*). Segment 1 is the root segment, segments 3,4, and 5 are leaf segments. The rectangular leaf nodes (*left*) are the final action samples, placed continuously in the action space.

Hierarchical Representation of Knowledge: Besides the structural relations, several elements carry further information as attributes. The attribute $counter(c)$ counts the number of paths that share the segment c, i.e. the number of paths that went c before they diverged and went their own way. The attribute $val(c)$ carries the q-value of a segment. The q-value of a segment is our way of representing the estimated long-term return of a state or state-action pair and is defined recursively as follows. The value of a leaf segment c of the tree in state s is $val(c) = \hat{Q}(s, a)$, where $a = leaf(c)$ is the final action sample of the segment. The quantity $\hat{Q}(s, a)$ is the estimated long term return, when executing action a in state s and is obtained in the real-time run when the agent enters the resulting successor state s' and is given by $\hat{Q}(s, a) = r(s, a) + \gamma\hat{V}(s')$. The value of a non-leaf segment c is defined by the maximum value over all it's children. By applying this rule recursively bottom-up the value of root segment of state s becomes the maximum value of all action samples generated by the diffusion tree in that state and thus $val(rt(s)) = \hat{V}(s)$ is the expected long term return for state s when acting greedy, i.e. always executing the action that maximizes expected long-term return.

Controlled Exploration by Informed Sampling: In order to direct our search for good action samples we need to control our action sampling process. We do this by controlling the divergence time and by controlling the choice of path to go at a divergence point. For the first one, we use the approach from the original DDT, which is decreasing the probability of divergence from a segment c with increasing counter $counter(c)$. This way we implement the principle of reinforcement of past events. For the latter one, we will describe our approach in the following.

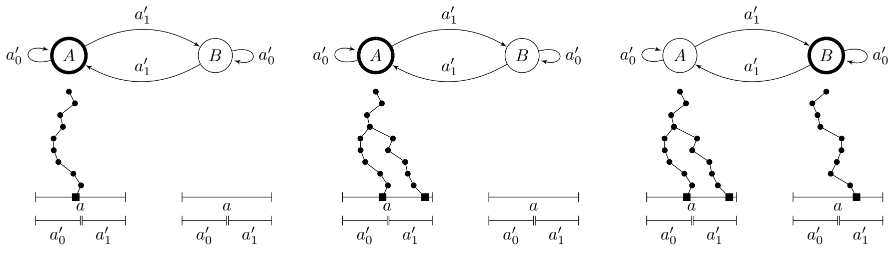
The information available at a branching point $leaf(c)$ is the set of children c' of the segment c and all information those children are attributed with, in particular each one's $val(c')$, which represents the expected action-value of the region covered by the subtree of c'. Based on that information, we can make a decision about which path to choose in numerous ways, each with different effects on the resulting sample distribution. The original heuristics of [9] is to randomly choose a child with probability proportional to the child's counter. This heuristic results in an accumulation of samples in regions where already many samples are, because counters of segments leading to those regions are high. However, to facilitate efficient exploration we wish to accumulate samples in regions with high expected long-term return instead. A straight forward approach to implement this idea is to deterministically choose the child with the maximum value. This will ultimately lead to accumulation of samples in regions with high expected long-term return. However, this statement is only valid, if the tree has 'seen' values in all promising regions of the underlying domain, i.e. it has some samples evenly distributed over the underlying domain. If we choose this heuristic right from the start of the learning process, the tree will concentrate its samples to local optima it encounters in the first few sampling steps. A common way to circumvent this issue in conventional approaches is to choose actions randomly at the beginning of the learning process which accounts for the uncertainty of knowledge about the utility of the actions and to increase the trust on the knowledge obtained by decreasing the random proportion in decision making over time. To implement this idea we use Boltzmann Selection, where the probability of choosing a child is given by $p_c = \exp\left(val(c)/\tau\right)/\sum_{c' \in ch(pa(c))} \exp\left(val(c')/\tau\right)$. Thus, at the beginning of the learning process we set τ to a high value to account for the uncertainty of knowledge. Choices will be made purely randomly and final samples will be evenly spread over the action space. Over time we decrease τ, and thus the choice will be increasingly deterministic to account for the increasing certainty of the acquired knowledge about high expected return.

Algorithmic Description: Algorithm 1 shows the pseudocode of our approach. Knowledge is acquired by incrementally building diffusion trees in the states. Every time the agent visits a state, it generates a new path (line 2) in the diffusion tree and thereby samples an action a to be executed. In the beginning

Algorithm 1. DT_LEARNING(s).

```
 1: repeat
 2:    $c \leftarrow SAMPLE\_PATH(s)$
 3:    $a \leftarrow leaf(c)$
 4:    execute $a$, observe result state $s'$ and reward $r$
 5:    $PROPAGATE\_UP(c, r, val(rt(s')))$
 6:    $s \leftarrow s'$
 7: until $s$ is goal state
procedure $SAMPLE\_PATH(s)$
 8: if $rt(s) = 0$ then
 9:    $rt(s) \leftarrow$ sample new segment starting at t=0 and a=0
10:    return $rt(s)$
11: else
12:    $c \leftarrow rt(s)$
13:    loop
14:       $d \leftarrow$ sample divergence time $\in [start(c), D]$ // with $start(\cdot) \equiv$ start time
15:       if $d \leq end(c)$ then // with $end(\cdot) \equiv$ end time
16:          $c' \leftarrow$ sample new segment starting at t=d and a=c[d]
17:          $pa(c') \leftarrow (c)$ and $ch(c) \leftarrow ch(c) \cup \{c'\}$
18:          return $c'$
19:       else if $d > end(c)$ then
20:          $c \leftarrow$ choose child $c' \in ch(c)$ by Boltzmann Selection
procedure $PROPAGATE\_UP(c, r, v)$
21: $val(c) \leftarrow r + \gamma \cdot v$
22: repeat
23:    $c \leftarrow pa(c)$;
24:    $e \leftarrow r + \gamma\ v - val(c)$
25:    if $e > 0$ then
26:       $val(c) \leftarrow val(c) + \alpha\ e$ // with $\alpha \equiv$ learning rate
27: until $c$ has no parent
```

of a run the diffusion trees in all states are empty, i.e. they have no path. On the first visit of a state s the agent generates the first path, which will be the first segment c of the tree in s and thus $rt(s) = c$ (line 9). The leaf node of c represents the final action sample a and thus $leaf(rt(s)) = a$ (line 3). The agent will now execute a leading into state s', observe the reward $r(s, a)$ (line 4) and update the value of the three in s (line 5) by first setting the value of c according to value update equation (line 21) and then recursively updating the value of the parents (line 22). When entering a state with a tree that has at least one segment, we walk down the tree by sampling a divergence time (line 14) and choosing between children (line 20) until divergence (line 15). Figure 3 shows a run of an agent in a world with two states and two actions.



(a) Step t. (b) Step $t + 1$. (c) Step $t + 2$.

Fig. 3. Successive sampling of paths. The upper part of each figure shows the state transition graph of a simple abstract world with two discrete states and two discrete actions, where the current state is painted with a thick line width. Below the states A and B the diffusion trees of those states are shown. The interval lines below the trees illustrate the mapping from continuous action samples to the two discrete actions utilized in the selected exemplary application.

5 Experiments

In order to validate our approach we conducted two experiments in simulation. The experiments serve to validate the value of informed sampling against uninformed sampling. Therefore we compare two algorithms, DT-Learning (DTL) and a simple random scheme we call Random Sampling Q-Learning (RSQL). In RSQL, with probability ν an action-sample is drawn uniformly in every state and kept in case its resulting estimated return is greater than the return of the best action-sample kept so far for that state. With probability $1 - \nu$ the best action obtained so far is executed. The parameter ν is set to a value near one at the beginning and is decreased over time to account for the uncertainty of knowledge in the beginning. Thus, RSQL is the simplest sampling scheme possible as it is as uninformative as possible while still fulfilling all necessities of the Q-learning framework.

The task in the first experiment is to find the shortest path from a start location to a goal location in a grid world. The states space consists of the two-dimensional locations in the grid. The actions a' in a gridworld consist of the five

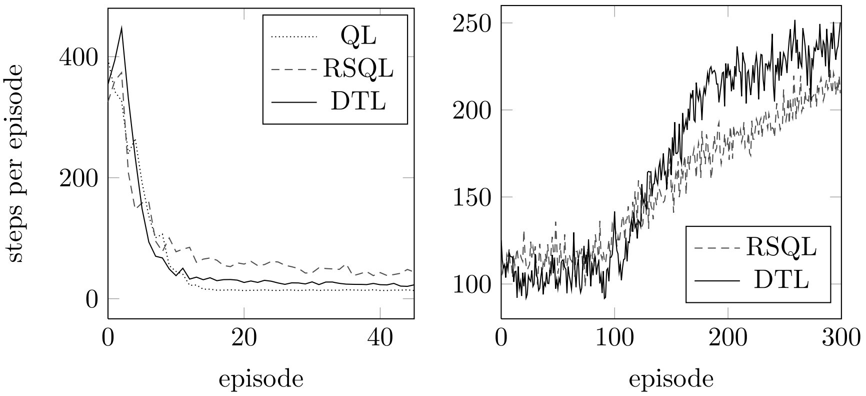


Fig. 4. Performance of the algorithms DT-Learning (*DTL*), RSQ-Learning (*RSQL*), and Q-Learning (*QL*) on the two test tasks to reach a goal cell (*left*) and to stabilize a pendulum (*right*)

choices to go up, down, west, east and to stay, i.e. $a' \in \{0, \dots, 4\}$. To apply the action-continuous approaches, their continuous outputs $a \in [0, 5]$ are mapped to those five actions by $a' = \lfloor a \rfloor$. The agent receives a positive reward when it enters the goal cell and a negative one, when it bumps into a wall. We chose this discrete world, because it is simple and facilitates easy analysis of the key properties of our algorithm. We evaluated the average number of steps until the agent reaches the goal point during a number of successive learning episodes, where the agent keeps its knowledge over the different episodes. Figure 4 (left) shows the results, averaged over 10 trials each. We applied Q-learning (QL) in its original action-discrete fashion, to serve as a base line for comparison. As can be seen the convergence of both sampling-based algorithms is worse than Q-learning. This is because Q-Learning, working with the five discrete actions, is naturally the best fit for this task. The convergence of DTL is better than the one of RSQL, due to DTL sampling more actions in regions with high expected return, whereas RSQL acts ignorant about the knowledge obtained earlier and thus generates samples that lead into walls with relatively high probability.

In a second experiment we tested our algorithm on the task to stabilize a pendulum in an upright position. To ease the task, the starting position for every episode is the upright position. During an episode the number of steps is counted until pendulum crosses the horizontal position. The two-dimensional state space, consisting of angle $\phi \in [0, 2\pi]$ and angular velocity $\omega \in [-10\frac{rad}{s}, 10\frac{rad}{s}]$, was discretized into 41 equally sized intervals per dimension. The action space was the angular acceleration $A = [-10\ Nm, 10\ Nm]$. Figure 4 (right) shows the results of the two algorithms RSQL and DTL. As can be seen DT-Learning converges slightly faster. Again, this is due to the more efficient exploration resulting from controlled sampling of actions in regions with higher expected return. We omitted Q-Learning here, because the necessary discretization of the action space would render the results incomparable.

6 Conclusion

In this work we presented an approach for reinforcement learning with continuous actions. We were able to show the benefits of informed sampling of actions by efficiently using hierarchically structured knowledge about values of the actions space. The computational cost of sampling an action is of logarithmic order in the number of action samples, as is typical for tree-based approaches. In comparison to a very simple, uninformed sampling scheme our approach showed better convergence rates. However, some open issues remain. Due to the discretization of the state space, there is a discontinuity in the value of a particular action between two states. This could be handled by an interpolation between two trees. Another issue concerns the aging of information in unused parts of the trees. Because memory requirements for our approach are relatively high, a technique must be found to prune subtrees based on the utility of their contained information. These issues will be subject to further research.

References

1. Watkins, C.J., Dayan, P.: Q-learning. Machine Learning 8, 279–292 (1992)
2. Gross, H.M., Stephan, V., Boehme, H.J.: Sensory-based robot navigation using self-organizing networks and q-learning. In: Proceedings of the 1996 World Congress on Neural Networks, pp. 94–99. Psychology Press, San Diego (1996)
3. Gaskett, C., Wettergreen, D., Zelinsky, A., Zelinsky, E.: Q-learning in continuous state and action spaces. In: Australian Joint Conference on Artificial Intelligence, pp. 417–428. Springer, Heidelberg (1999)
4. Atkeson, C.G.: Randomly sampling actions in dynamic programming. In: 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007), pp. 185–192 (2007)
5. Kearns, M., Mansour, Y., Ng, A.Y.: A sparse sampling algorithm for near-optimal planning in large markov decision processes. Machine Learning 49, 193–208 (2002)
6. Ross, S., Chaib-Draa, B., Pineau, J.: Bayesian reinforcement learning in continuous pomdps with application to robot navigation. In: 2008 IEEE International Conference on Robotics and Automation (ICRA 2008), pp. 2845–2851. IEEE, Los Alamitos (May 2008)
7. Lazaric, A., Restelli, M., Bonarini, A.: Reinforcement learning in continuous action spaces through sequential monte carlo methods. In: Platt, J., Koller, D., Singer, Y., Roweis, S. (eds.) Advances in Neural Information Processing Systems, vol. 20, pp. 833–840. MIT Press, Cambridge (2008)
8. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. The MIT Press, Cambridge (March 1998)
9. Neal, R.M.: Density modeling and clustering using dirichlet diffusion trees. In: Bayesian Statistics 7: Proceedings of the Seventh Valencia International Meeting, pp. 619–629 (2003)