Novel feature selection and kernel-based value approximation method for reinforcement learning

Hunor S. Jakab and Lehel Csató

Babeş-Bolyai University, Faculty of Mathematics and Computer Science

Abstract. We present a novel sparsification and value function approximation method for on-line reinforcement learning in continuous state and action spaces. Our approach is based on the kernel least squares temporal difference learning algorithm. We derive a recursive version and enhance the algorithm with a new sparsification mechanism based on the topology maps represented by proximity graphs. The sparsification mechanism – speeding up computations – favors data-points minimizing the divergence of the target-function gradient, thereby also considering the *shape* of the target function. The performance of our sparsification and approximation method is tested on a standard benchmark RL problem.

Keywords: reinforcement learning, kernel methods, function approximation

1 Introduction

Approximate reinforcement learning (RL) methods are algorithms dealing with real-world problems that are characterized by continuous, high dimensional state-action spaces and noisy measurements. In this article value functions for a given policy in continuous RL problems are estimated, also called *policy evaluation*. Linear regression models with non-linear features have been preferred [1, 6] for approximate policy evaluation, due to their good convergence properties and relatively easy to analyze. Their main drawback is the necessity to carefully design problem-specific feature functions and we address this design issue by a nonlinear extension of the algorithm using "kernelization".[1] Kernelization in turn raises the problem of complexity, over-fitting, and increased computational cost. To avoid these problems, different sparsification mechanisms are employed. The sparsification procedure influences the accuracy and the generalization capability of the function approximator. In model free reinforcement learning, the accuracy of value functions around decision boundaries is extremely important, we thus introduce a proximity-graph based sparsification mechanism which takes into account the shape of the target function. We also present a recursive version of kernel least squares temporal difference learning (KLSTD) that uses the sparsification mechanism mentioned above.

2 Notation and Background

In RL data acquisition is defined by interacting with the environment in which we want to learn: the commands and their associated rewards defines the data base. The back-

[1] A survey of kernelized value function approximation methods can be found in [10]

ground is the Markov decision process (MDP) [7], commonly used for modeling in reinforcement learning problems. An MDP is a quadruple $M\ (S, A, P, R)$, where S is the (possibly infinite) set of states, A is the set of actions, $P(s'|s,a) : S \times S \times A \to [0,1]$ describes the usually unknown transition probabilities, and $R(s,a) : S \times A \to \mathbb{R}$ is the instantaneous reward function defining the feedback to the learner. The decision making mechanism is modeled with an action selection *policy*: $\pi : S \times A \to [0,1]$ is the conditional probability distribution – $\pi(a|s)$ – of taking action a in state s. The solution to the reinforcement learning problem is the *optimal policy* π^* maximizing the expected discounted cumulative reward: $\pi^* = \operatorname{argmax}_\pi E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t\right]$. An important element of many RL algorithm is a representation of the *utility* of the state or state-action pairs as value $V_\pi(s) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t | s_0 = s\right]$ or action-value functions $Q_\pi(s,a) = E_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t | s_0 = s, a_0 = a\right]$. Value functions express the expected long term discounted reward received when starting from a given state or state-action pair and following a given policy π. Value-based RL algorithms use the state-action value function $Q(\cdot,\cdot)$ to determine an optimal greedy action selection policy.

Since this paper focuses on value approximation, we consider the action selection policy to be fixed. This restriction gives rise to a so-called Markov Reward process [9] which we will refer to as MRP. Throughout the rest of this paper for ease of exposition we will omit the policy from the notation and we will use state value functions $V(\cdot)$ but the presented algorithms equally apply for state-action value functions as well.

3 Kernel Least Squares Value Approximation

As a basis of our value approximation framework we make use of the least squares temporal difference learning algorithm (LSTD) introduced in [2]. The LSTD method is based on a generalized linear approximation to the value function using the set of feature vectors $\boldsymbol{\phi}(s) = [\phi_1(s), \ldots, \phi_d(s)]^T$: $\tilde{V}(s) = \boldsymbol{\phi}(s)^T \boldsymbol{w}$, where $\boldsymbol{w} = [w_1, \ldots, w_d]^T$ is the vector of parameters and d is the feature space dimension.

The basis of temporal difference learning methods is the incremental update of the value function based on the *temporal difference error*: $\delta_{t+1} = R_{t+1} + \gamma \tilde{V}_t(s_{t+1}) - \tilde{V}_t(s_t) = R_{t+1} - \left(\phi(s_t) - \gamma\phi(s_{t+1})\right)^T \boldsymbol{w}$ and the parameter update is:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \alpha_t \left(\delta_{t+1}\right) \phi(s_t).$$

In the limit of infinite updates, the algorithm leads to a parameter vector $\boldsymbol{w}$ that satisfies $E[\boldsymbol{\phi}(s_t)\delta_{t+1}] = 0$ [9]. Using a finite-size approximation with the sample average, we get the expression $\frac{1}{n}\sum_{t=0}^{n} \boldsymbol{\phi}(s_t)\delta_{t+1} = 0$ and substituting δ_{t+1} into the equation:

$$\tilde{A}\boldsymbol{w} = \tilde{b}, \text{ where } \tilde{A} = \frac{1}{n}\sum_{t=0}^{n} \boldsymbol{\phi}(s_t)\left(\boldsymbol{\phi}(s_t) - \gamma\boldsymbol{\phi}(s_{t+1})\right)^T, \text{ and } \quad \tilde{b} = \frac{1}{n}\sum_{t=0}^{n} \boldsymbol{\phi}(s_t) R_t.$$

When $\tilde{A}$ is invertible there is a direct solution for $\boldsymbol{w}$, as stated in [2]. The matrix $\tilde{A}$ is a sum of individual matrices the size of the feature space, and can be calculated incrementally. In [12] a kernelized version of the algorithm was presented, eliminating

the need for manual feature construction. They employ the representer theorem and the kernel trick to reformulate the calculation of $\boldsymbol{w}$ with kernel functions:

$$\boldsymbol{w}^* = \tilde{A}^{-1}\tilde{b}, \text{ where } \tilde{A} = \frac{1}{n}\sum_{t=0}^{n} \boldsymbol{k}(s_t)\left[\boldsymbol{k}^T(s_t) - \gamma \boldsymbol{k}^T(s_{t+1})\right], \quad \tilde{b} = \frac{1}{n}\sum_{t=0}^{n} \boldsymbol{k}(s_t) R_t.$$

We used $k(\cdot,\cdot)$ to denote a valid kernel function and $\boldsymbol{k}(s) = \left[k(s,s_1) \ldots k(s,s_n)\right]^T$ to denote the vector of kernel values evaluated on the data-point s and the training data-points $s_i \quad i = \overline{1,n}$. The expression for the approximated value function becomes:

$$\tilde{V}(s) = \sum_{i=0}^{n} \boldsymbol{w}_i^* k(s_i, s_t) \quad s \in S \tag{1}$$

The inversion of $\tilde{A}$ is problematic: in an on-line setting inverting $\tilde{A}$ at each step is impractical; the computation time is cubic. In the next section we present a new type of sparsification mechanism to keep the matrix $\tilde{A}$ at a small fixed size. Since the sparsification depends on the *values of the inputs*, we need a *"good"* subset, the criteria presented on the next section. To speed up online learning, we use the Sherman-Woodbury formula and express the inverse of $\tilde{A}^{-1} = C$ recursively [3]:

$$C_{t+1} = C_t - C_t \frac{\boldsymbol{k}(s_t)\left[\boldsymbol{k}^T(s_t) - \gamma \boldsymbol{k}^T(s_{t+1})\right]}{1 + \left[\boldsymbol{k}^T(s_t) - \gamma \boldsymbol{k}^T(s_{t+1})\right] C_t \boldsymbol{k}(s_t)} C_t \tag{2}$$

The optimal coefficients at time-step $t+1$ can be obtained as: $w_{t+1}^* = C_{t+1}\tilde{b}$, see [6].

4 Sparsification of the Representation

Sparsification reduces the computational cost of kernel-based algorithms by finding a small set of data-points called *dictionary*, which will be used as basis set for subsequent computation. Common sparsification methods [4] use approximate linear independence (ALD) as a criterion for discarding data-points: a new point is approximated by the linear combination of dictionary points if the approximation error, defined as $\min_\alpha \| \sum_{i=1}^{d} \alpha_i \boldsymbol{\phi}(s_i) - \boldsymbol{\phi}(s^*)\|^2$, is below a predefined threshold ν (s^* is the new data point and α_i are the variational coefficients). The drawback of the ALD-based sparsification is that it ignores the target function (the function to be estimated) therefore also ignoring the data-points relevance in estimating it.

As a major contribution of this paper we present a sparsification where the spectral information contained in the Laplacian of an on-line constructed similarity graph is used when deciding on the inclusion or removal of a new data-point to the dictionary. We start by defining the unnormalized graph Laplacian as: $L = D - A$, where A is the weighted adjacency matrix of a proximity graph $G(\mathcal{E}, \mathcal{V})$ with $\mathcal{E}$ and $\mathcal{V}$ denoting the set of edges and vertices respectively. $D_{i,i} = \sum_{j \neq i} A_{i,j}$ is a diagonal matrix containing the node degrees on the diagonal. Let us assume that the graph approximates the geodesic

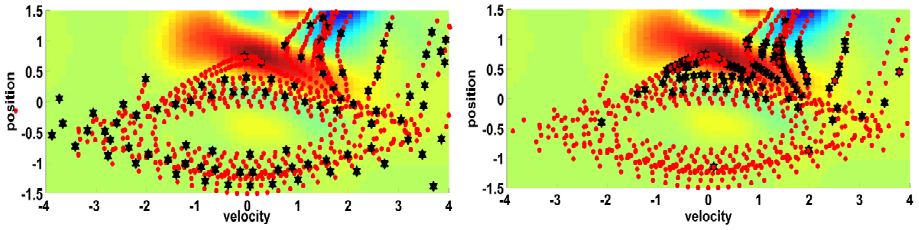


Fig. 1. Dictionary points (*black stars*) obtained after processing 5 roll-outs ($\approx$ 450 data-points) – *red dots* – for the mountain-car control problem using different sparsification techniques. Figure (a): ALD sparsification. Figure (b) our proximity graph-based sparsification. The maximum number of dictionary points was set to 100, KLSTD approximation used.

distances[2] between data-points (details on constructing proximity graphs on-line are explained in Section 5). An interesting property of the graph Laplacian is that it is symmetric, positive semi-definite, and it gives rise to the discrete Laplacian operator Δ: for every $f : \mathcal{V} \to \mathbb{R} : \Delta f(s_i) = \sum_{j=1}^{n} A_{ij} \left[f(s_i) - f(s_j) \right]$ and $\Delta f = L\mathbf{f}$ where $\mathbf{f} = \left[f(s_1) \dots f(s_n) \right]^T$ [11].

Let us denote the approximated function f and the neighbor set of a vertex s_i as $\mathrm{neig}(s_i) = \{ s_j \in \mathcal{V} | A_{i,j} \neq 0 \}$. The vertices of the graph $G(\mathcal{E}, \mathcal{V})$ are samples from the target function. To obtain a sparse dictionary we introduce a score function $\mu(\cdot)$ for a vertex $s_i \in \mathcal{V}$ as the weighted sum of the squared differences between target function values of s_i and its neighboring vertices: $\mu(s_i \in \mathcal{V}) = \sum_{v_j \in \mathrm{neig}(s_i)} A_{ij} \left[f(s_i) - f(s_j) \right]^2$. Summing up the individual vertex scores gives an overall measure about the quality of the dictionary set:

$$\sum_{s_i \in \mathcal{V}} \mu(s_i) = \sum_{i,j=1}^{n} A_{ij} \left[f(s_i) - f(s_j) \right]^2 = \mathbf{f^T L f} \tag{3}$$

This means that we apply the discrete Laplacian operator for the value function, giving the divergence of the gradient of the target functions.

Whenever the size of the dictionary reaches a maximum predefined value, we compare the quality of the dictionary set with and without a new data-point using (3) and eliminate the data-point which reduces the quality the most. Figure 1 illustrates the differences between approximate linear dependence-based sparsification and our Laplacian-based method. Performing sparsification based on the maximization of the score from (3) leads to a higher sample-density in regions where f changes rapidly while at the same time keeping sufficient number of data-points in slowly changing regions to obtain good approximation accuracy.

[2] Geodesic distance is the distance between states along the manifold determined by the transition dynamics of the MRP.

5 On-line Proximity Graph Construction

The information contained in the sequential nature of the training data in RL can be used to acquire models that reflect the transitions of the underlying MDP and the true distance between training data-points. To store this information we use so-called proximity graphs encountered in dimensionality reduction and surface reconstruction methods from point-cloud sets. Let $G(\mathcal{E}, \mathcal{V}), \quad \mathcal{V} \subset S$ be a proximity graph that is the representation of a manifold upon which the training data-points are located. Existing work [8], [11] on graph construction has focused on treating training data as either i.i.d or batch data. Since the approximation algorithm proposed by us operates on-line we present iterative versions of two well-known edge construction methods, the k nearest neighbor(kNN) and the extended sphere of influence (ϵ-SIG) methods.

The kNN edge construction strategy connects a node s_i to its k nearest neighbors $s_j \in knn(s_i)$ by an edge length equal to their spatial distance $e_{s_i,s_j} = \|s_i - s_j\|^2$. To achieve symmetry in the obtained proximity graph, we use undirected edges. As a consequence, when creating edges between data-points s_i and s_j we update the adjacency matrix as follows: $A_{i,j} = A_{j,i} = \|s_i - s_j\|^2$.

Extended Sphere of Influence Graphs $(eSIG)$*:* The extended sphere of influence graph produces a good description of non-smooth surfaces and better accommodates variations in the point sample density [8]. In this approach, edges are constructed between vertices with intersecting spheres of influence: $e_{s_i,s_j} = \|s_i - s_j\|^2$ if $R(s_i) + R(s_j) > \|s_i - s_j\|$, where $R(s_i) = \|s_i - s_k\|^2$ and s_k is the k-th nearest neighbor of s_i. The sphere of influence of a graph vertex is the sphere centered at the point, with radius given by its distance to the nearest neighbor. Disconnected components can appear with this construction, however by raising the sphere of influence radius (to be the length of the distance to the 2nd or 3rd netarest neighbor) this can be eliminated.

5.1 Updating the Graph Structure

The deletion of a vertex from the graph $G(\mathcal{E}, \mathcal{V})$ also removes the edges connecting it to its neighbors. In order to keep the information contained in the original edges about the topology we use the following pruning algorithm: Let us denote the vertex which is to be removed by s^*. After removal of s^* we get a new graph structure $G'(\mathcal{V}', \mathcal{E}')$ with the following components: $\mathcal{V}' = \mathcal{V} \setminus \{s^*\}$ and $\mathcal{E}' = \mathcal{E} \setminus \{e(s^*, s) | s \in \text{neig}(s^*)\} \cup E_{new}$ where $e(s^*, s)$ denotes an edge between the vertices s^* and s of the graph G($\mathcal{E}$,$\mathcal{V}$).
The set $E_{new} = \{e(s_i, s_j) = \|s_i, s^*\| + \|s_j, s^*\| | s_i, s_j \in \text{neig}(s^*), e(s_i, s_j) \notin \mathcal{E}\}$ contains the new edges between former neighbors of s^* with weights equal to the sum of the edge weights that connected them to s^*. Figure 2 illustrates the removal procedure on a sample graph structure obtained after interacting for 400 steps with the mountain-car environment.

The addition of the new edges serves two purposes: First the connected property of the graph is maintained, secondly it can be shown that using the previously defined update the shortest path distance matrix of the graph will remain the same except for removing the row and column corresponding to the index of s^*. In consequence the geodesic distances between the remaining graph nodes are preserved.

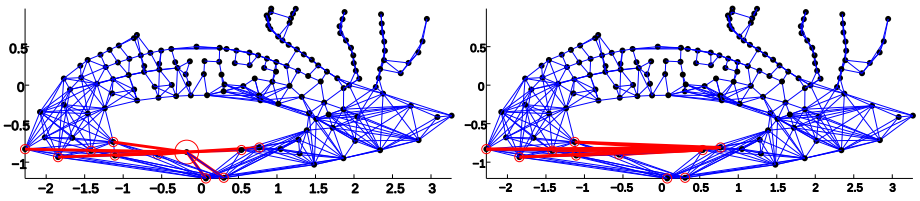
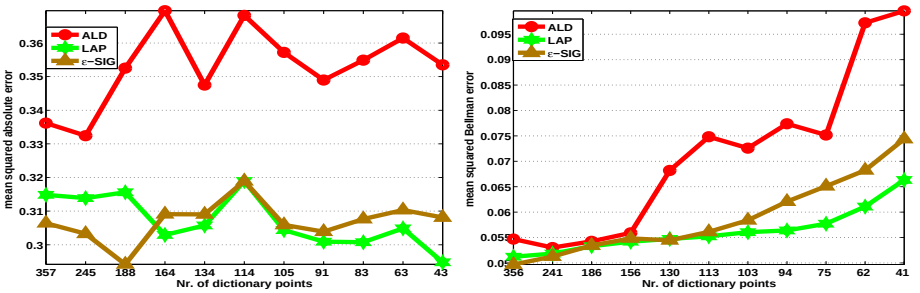


Fig. 2. Dictionary points (*black dots*) and graph structure (*blue lines*) before and after node removal. Figure (a) shows the node to be removed (*red circle*), its neighbors (*red circles with dots*) and the removable edges (*red lines*). On (b) we see the resulting graph with the newly created edges shown in red.

6 Performance Evaluation

To measure the performance of our policy evaluation framework in a continuous Markov reward process we make use of two quality measures: the Bellman Error (BE) and the mean absolute error with respect to the true value function. $BE(\tilde{V}(s)) = R(s) + \gamma \int_{s' \in S} P(s'|s,a) \int_{a \in A} \pi(a|s) \tilde{V}(s') da ds' - \tilde{V}(s)$ The Bellman error expresses the ability of the approximator to predict the value of the next state based on the current state's approximated value. It also indicates how well the function approximator has converged, and according to [10] it gives an upper bound on the approximation error: $\|V - \tilde{V}\|_\infty \leq \frac{\|BE(\tilde{V})\|_\infty}{1-\gamma}$ The true value function V used as a baseline in our experiments is calculated in all experimental settings using exhaustive Monte Carlo approximation and can be considered very close to the actual value function induced by the policy π. For testing we use the well-known benchmark problems: the mountain-car [13] where the state space is two dimensional (position, and velocity), continuous, and the actions are scalars from the continuous interval [-4 , 4]. To demonstrate the benefits of our Laplacian-based sparsification mechanism for the approximation accuracy we performed 15 experiments each having 150 episodes consisting of 350 steps ($\sim$45000 training-points) on the mountain-car problem. For action selection policy π we used a Gaussian policy with a fixed linear controller: $\pi(a|s) = \frac{1}{2\pi\sigma} \exp(-\frac{\|c(s)-a\|^2}{2\sigma 2})$ where σ is a small noise term and $c(s) = \theta^T s$ is the controller with fixed parameter vector θ. We also introduced stochasticity into the dynamics of the system by adding a small Gaussian noise term with 0.02 variance to the state transitions. To see the effects of the sparsification on the approximation accuracy, we performed the same experiments for a number of different maximum dictionary sizes. Figure 3(a) shows the obtained mean absolute errors with respect to the true value function. According to our experiments our Laplacian-based sparsification mechanism leads to a lower approximation error than standard ALD for all maximum dictionary sizes. The approximation accuracy even increases slightly when the dictionary size is drastically reduced as opposed to ALD where having fewer dictionary points raises the approximation error. This may be attributed to the better placement of data-points into regions of the state-space where the target function changes more rapidly. Figure 3(b) shows the evolution of the Bellman error from the same perspective. It can be seen that the Laplacian-based sparsification



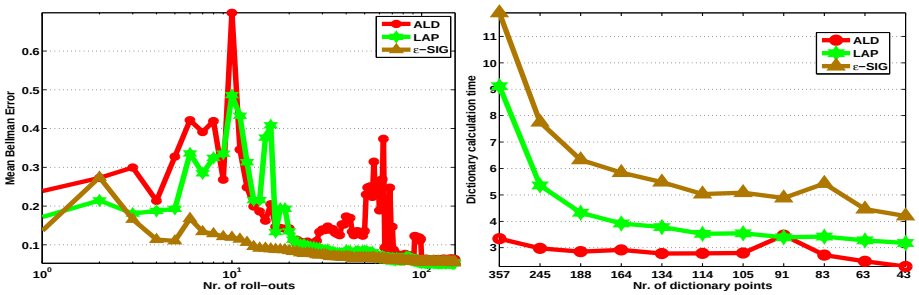
(a) (b)

Fig. 3. Evolution of the mean absolute error for different maximum dictionary sizes averaged over 15 experiments, in case of standard *ALD* sparsification, and our Laplacian based sparsification with knn and $\epsilon - SIG$ proximity graphs. The horizontal axis represents the maximum number of dictionary points, the vertical axis on figure (a) shows the mean squared absolute error while on figure (b) the mean squared Bellman error. Measurement of the errors was done on 3600 equally distributed points from the state space.

mechanism with knn or $\epsilon - SIG$ proximity graphs performs better at low dictionary sizes than ALD, the difference becoming more obvious as the dictionary size is further decreased. Figure 4(a) illustrates the evolution of the mean Bellman error as a function of the number of training points used for the estimation of the approximation parameters $\boldsymbol{w}$ from section 3. We calculated the dictionary beforehand based on a fixed training data set using each of the three sparsification methods. When small number of training data is used, the three dictionaries perform similarly. However when the training-data size increases the dictionary obtained by ALD leads to unstable value estimates, whereas using the dictionary obtained by our Laplacian based sparsification we obtain more accurate and stable values. The computational complexity of the presented approximation framework is influenced by the nearest neighbor search in case of graph expansion and the search for vertices with minimal score in case of the sparsification mechanism. The calculation of the individual scores of each graph vertex $v_i \in \mathcal{V}$ has a computational complexity of $O(0)$ since the scores can be stored and updated on-line for each vertex. Compared to the cost of approximate linear independence test our methods are slower as it can be seen from 4(b), but not by orders of magnitude, the difference becomes significant only by large dictionary sizes. The better approximation accuracy and faster convergence rates compensate for the higher computational requirements.

7 Conclusion

In this paper we presented an on-line algorithm with sparsification mechanism applicable to kernel-based value approximation. Our method can adjust the density of the dictionary set according to the characteristics of the target function, potentially leading to better approximation accuracy and faster convergence rates. The use of proximity graphs in the sparsification enables the extension of our methods with different distance-substitution kernels operating on graphs and opens up ways to different explo-



(a) (b)

Fig. 4. (a) Mean Bellman error as a function of training data set size. (b)Time required for the computation of the dictionary from approximately 45000 data-points using ALD and Laplacian based sparsification with knn and $\epsilon - SIG$ proximity graphs.

ration strategies like probabilistic road-maps or rapidly exploring random trees, directions that we plan to investigate in the future.

The authors acknowledge the support of the Romanian Ministry of Education and Research via grant PN-II-RU-TE-2011-3-0278.

References

1. Justin A. Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2-3):233–246, 2002.
2. Steven J. Bradtke, Andrew G. Barto, and Pack Kaelbling. Linear least-squares algorithms for temporal difference learning. In *Machine Learning*, pages 22–33, 1996.
3. Lehel Csató and Manfred Opper. Sparse On-Line Gaussian Processes In *Neural Computation*, 14/3, pages 641–668. The MIT Press, 2002
4. Yaakov Engel, Shie Mannor, and Ron Meir. The kernel recursive least squares algorithm. *IEEE Transactions on Signal Processing*, 52:2275–2285, 2003.
5. Bernard Haasdonk and Claus Bahlmann. Learning with distance substitution kernels. In *Pattern Recognition, 26th DAGM Symposium, Proceedings*, volume 3175 of *Lecture Notes in Computer Science*, pages 220–227. Springer, 2004.
6. Michail G. Lagoudakis and Ronald Parr. Least-squares policy iteration. *J. Mach. Learn. Res.*, 4:1107–1149, 2003.
7. Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.
8. Mauro R. Ruggeri and Dietmar Saupe. Isometry-invariant matching of point set surfaces. In *Eurographics Workshop on 3D Object Retrieval*, 2008.
9. Csaba Szepesvári. *Algorithms for Reinforcement Learning*. Morgan & Claypool, 2011.
10. Gavin Taylor and Ronald Parr. Kernelized value function approximation for reinforcement learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 1017–1024, New York, NY, USA, 2009. ACM.
11. Ulrike von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4), 2007.
12. X. Xu, D. Hu, and X Lu. Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, pages 973–992, 2007.
13. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.