Geodesic Gaussian Kernels for Value Function Approximation*

Masashi Sugiyama
Department of Computer Science, Tokyo Institute of Technology
2-12-1, O-okayama, Meguro-ku, Tokyo, 152-8552, Japan
and
School of Informatics, University of Edinburgh
The King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, UK
sugi@cs.titech.ac.jp

Hirotaka Hachiya
Department of Computer Science, Tokyo Institute of Technology
2-12-1, O-okayama, Meguro-ku, Tokyo, 152-8552, Japan
hachiya@sg.cs.titech.ac.jp

Christopher Towell
School of Informatics, University of Edinburgh
The King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, UK
C.C.Towell@sms.ed.ac.uk

Sethu Vijayakumar
School of Informatics, University of Edinburgh
The King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, UK
sethu.vijayakumar@ed.ac.uk

*The current paper is a complete version of our earlier manuscript (Sugiyama et al., 2007). The major differences are that we included more technical details of the proposed method in Section 3, discussions on the relation to related methods in Section 4, and the application to map buliding in Section 6. A demo movie of the proposed method applied in simulated robot arm control and Khepera robot navigation is available from 'http://sugiyama-www.cs.titech.ac.jp/~sugi/2008/GGKvsOGK.wmv'.

Abstract

The least-squares policy iteration approach works efficiently in value function approximation, given appropriate basis functions. Because of its smoothness, the Gaussian kernel is a popular and useful choice as a basis function. However, it does not allow for discontinuity which typically arises in real-world reinforcement learning tasks. In this paper, we propose a new basis function based on *geodesic Gaussian kernels*, which exploits the non-linear manifold structure induced by the Markov decision processes. The usefulness of the proposed method is successfully demonstrated in simulated robot arm control and Khepera robot navigation.

Keywords

reinforcement learning; value function approximation; Markov decision process; least-squares policy iteration; Gaussian kernel;

1 Introduction

Designing a flexible controller of a complex robot is a difficult and time-consuming task for robot engineers. Reinforcement learning (RL) is an approach that tries to ease this difficulty (Sutton & Barto, 1998). Value function approximation is an essential ingredient in RL, especially in the context of solving Markov Decision Processes (MDPs) using policy iteration methods. Since real robots often involve large discrete state-spaces or continuous state-spaces, it becomes necessary to use function approximation methods to represent the value functions. A *least-squares* approach using a linear combination of predetermined *under-complete* basis functions has shown to be promising in this task (Lagoudakis & Parr, 2003; Osentoski & Mahadevan, 2007). For general function approximation, Fourier functions (trigonometric polynomials) Gaussian kernels (Girosi et al., 1995), and wavelets (Daubechies, 1992) are typical basis function choices and they have been employed in robotics domains (Lagoudakis & Parr, 2003; Kolter & Ng, 2007). Normalized-Gaussian bases (Morimoto & Doya, 2007) as well as local linear approximation are also used in humanoid robot control (Vijayakumar et al., 2002).

Fourier bases (global functions) and Gaussian kernels (localized functions) have certain smoothness properties that make them particularly useful for modeling inherently smooth, continuous functions. Wavelets provide basis functions at various different scales and may also be employed for approximating smooth functions with local discontinuity.

Typical value functions in RL tasks are predominantly smooth with some discontinuous parts (Mahadevan, 2005). To illustrate this, let us consider a toy RL task of guiding an agent to a goal in a grid world (see Figure 1(a)). In this task, a state corresponds to a two-dimensional Cartesian position of the agent. The agent cannot move over the wall, so the value function of this task is highly discontinuous *across* the wall. On the other hand, the value function is smooth *along* the maze since neighboring reachable states in the maze have similar values (see Figure 1(b)). Due to the discontinuity, simply employing

Fourier functions or Gaussian kernels as basis functions tend to produce undesired, non-optimal results around the discontinuity, affecting the overall performance significantly (see Figure 3(b)). Wavelets could be a viable alternative, but are *over-complete* bases—one has to appropriately choose a *subset* of basis functions, which is not a straightforward task in practice.

Recently, Mahadevan (2005) proposed considering value functions defined not on the Euclidean space, but on *graphs* induced by the MDPs (see Figure 1(c)). Value functions which usually contain discontinuity in the Euclidean domain (e.g., across the wall) could be smooth on graphs (e.g., along the maze) if the graph is built appropriately. Hence, approximating value functions on graphs can be expected to work better than approximating them in the Euclidean domain.

The spectral graph theory (Chung, 1997) showed that Fourier-like smooth bases on graphs are given as minor eigenvectors of the *graph-Laplacian* matrix (see Figure 2(c)). However, their global nature implies that the overall accuracy of this method tends to be degraded by local noise. Coifman and Maggioni (2006) defined *diffusion wavelets*, which posses natural multi-resolution structure on graphs (see Figure 2(d)). Mahadevan and Maggioni (2006) showed that diffusion wavelets could be employed in value function approximation, although the issue of choosing a suitable subset of basis functions from the over-complete set is not discussed—determining the resolution level as well as the smoothness within the level may not be straightforward in practice.

In the machine learning community, Gaussian kernels seem to be more popular than Fourier functions or wavelets because of their locality and smoothness (Girosi et al., 1995; Vapnik, 1998; Schölkopf & Smola, 2002). Furthermore, Gaussian kernels have 'centers', which alleviates the difficulty of basis subset choice, e.g., uniform allocation (Lagoudakis & Parr, 2003) or sample-dependent allocation (Engel et al., 2005). In this paper, we therefore define Gaussian kernels on graphs (which we call *geodesic Gaussian kernel*), and propose using them for value function approximation (see Figure 2(a)). Our definition of Gaussian kernels on graphs employs the *shortest paths* between states rather than the Euclidean distance, which can be computed efficiently using the *Dijkstra algorithm* (Dijkstra, 1959; Fredman & Tarjan, 1987). Moreover, an effective use of Gaussian kernels opens up the possibility to exploit the recent advances in using Gaussian processes for temporal-difference learning (Engel et al., 2005).

Basis functions defined on the state space can be used for approximating the state-action value function by extending them over the action space. This is typically done by simply copying the basis functions over the action space (Lagoudakis & Parr, 2003; Mahadevan, 2005). In this paper, we propose a new strategy for this extension, which takes into account the transition after taking actions. This new strategy is demonstrated to work very well when the transition is predominantly deterministic.

The paper describes the notations employed for the RL problem succinctly in Section 2. Section 3 formulates the Gaussian kernel based basis functions on graphs, including a method to generalize them to the continuous domains. Section 4 qualitatively discusses the characteristics of the proposed geodesic Gaussian kernel and the relation between the proposed and existing basis functions. Section 5 is devoted to extensive

experimental comparison between the proposed and existing basis functions. Section 6 shows applications of the proposed method in simulated kinematic robot arm control and mobile robot navigation. Section 7 provides concluding remarks and future directions.

2 Formulation of the Reinforcement Learning Problem

In this section, we briefly introduce the notation and reinforcement learning (RL) formulation that we will use across the manuscript.

2.1 Markov Decision Processes

Let us consider a Markov decision process (MDP) specified by

$$(\mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma), \tag{1}$$

where

- $\mathcal{S} = \{s^{(1)}, s^{(2)}, \ldots, s^{(n)}\}$ is a finite[1] set of states,
- $\mathcal{A} = \{a^{(1)}, a^{(2)}, \ldots, a^{(m)}\}$ is a finite set of actions,
- $\mathcal{P}(s'|s,a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is the conditional probability of making a transition to state s' if action a is taken in state s,
- $R(s,a,s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is an immediate reward for making a transition from s to s' by action a,
- $\gamma \in [0,1)$ is the discount factor for future rewards.

The expected reward $\mathcal{R}(s,a)$ for a state-action pair (s,a) is given as

$$\mathcal{R}(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) R(s,a,s'). \tag{2}$$

Let

$$\pi(s) : \mathcal{S} \to \mathcal{A} \tag{3}$$

be a deterministic policy which the agent follows. In this paper, we focus on deterministic policies since there always exists an optimal deterministic policy (Lagoudakis & Parr, 2003). Let

$$Q^{\pi}(s,a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R} \tag{4}$$

[1] For the moment, we focus on discrete state spaces. In Section 3.4, we extend the proposed method to continuous state spaces.

be a state-action value function for policy π, which indicates the expected long-term discounted sum of rewards the agent receives when the agent takes action a in state s and follows policy π thereafter. $Q^\pi(s,a)$ satisfies the following *Bellman equation*:

$$Q^\pi(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) Q^\pi(s', \pi(s')). \tag{5}$$

The goal of RL is to obtain a policy which produces the maximum amount of long-term rewards. The optimal policy $\pi^*(s)$ is defined as

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} Q^*(s,a), \tag{6}$$

where $Q^*(s,a)$ is the optimal state-action value function defined by

$$Q^*(s,a) = \max_\pi Q^\pi(s,a). \tag{7}$$

2.2 Least-Squares Policy Iteration

In practice, the optimal policy $\pi^*(s)$ cannot be directly obtained since $\mathcal{R}(s,a)$ and $\mathcal{P}(s'|s,a)$ are usually unknown; even when they are known, direct computation of $\pi^*(s)$ is often intractable.

To cope with this problem, Lagoudakis and Parr (2003) proposed approximating the state-action value function $Q^\pi(s,a)$ using a linear model:

$$\widehat{Q}^\pi(s,a;\boldsymbol{w}) = \sum_{i=1}^{k} w_i \phi_i(s,a), \tag{8}$$

where k is the number of basis functions which is usually chosen to be much smaller than $|\mathcal{S}| \times |\mathcal{A}|$. $\boldsymbol{w} = (w_1, w_2, \ldots, w_k)^\top$ are the parameters to be learned, $^\top$ denotes the transpose, and $\{\phi_i(s,a)\}_{i=1}^k$ are pre-determined basis functions. Note that k and $\{\phi_i(s,a)\}_{i=1}^k$ can depend on policy π, but we do not show the explicit dependence for the sake of simplicity. Assume we have roll-out samples from a sequence of actions:

$$\{(s_i, a_i, r_i, s_i')\}_{i=1}^t, \tag{9}$$

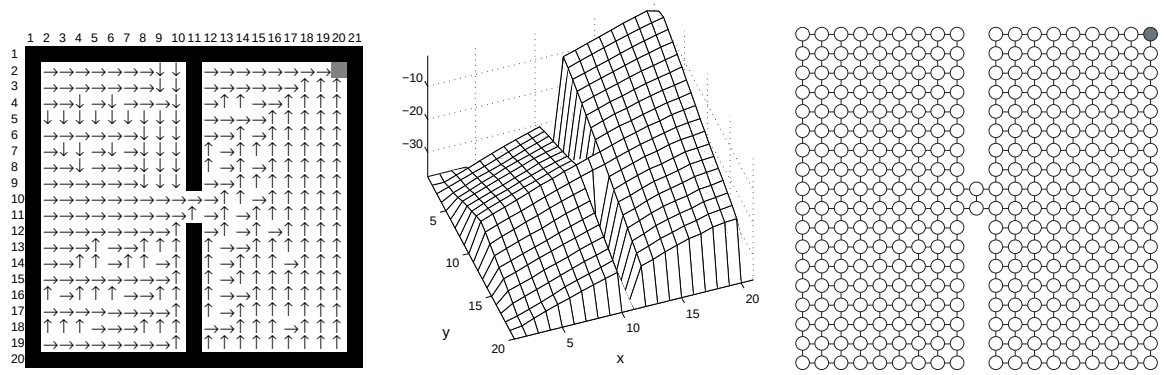
where each tuple denotes the agent experiencing a transition from s_i to s_i' on taking action a_i with immediate reward r_i. We suppose that we have an enough amount of samples to estimate the transition probability $\mathcal{P}(s'|s,a)$.

Under the Least-Squares Policy Iteration (LSPI) formulation (Lagoudakis & Parr, 2003), the parameter $\boldsymbol{w}$ is learned so that the Bellman equation (5) is optimally approximated in the least-squares sense[2]. Consequently, based on the approximated state-action value function with learned parameter $\widehat{\boldsymbol{w}}^\pi$, the policy is updated as

$$\pi(s) \longleftarrow \operatorname*{argmax}_{a \in \mathcal{A}} \widehat{Q}^\pi(s,a;\widehat{\boldsymbol{w}}^\pi). \tag{10}$$

Approximating the state-action value function and updating the policy is iteratively carried out until some convergence criterion is met.

[2]There are two alternative approaches: *Bellman residual minimization* and *fixed point approximation*. We take the latter approach following the suggestion in Lagoudakis and Parr (2003).



(a) Black areas are walls over which the agent cannot move while the goal is represented in gray. Arrows on the grids represent one of the optimal policies.

(b) Optimal state value function (in log-scale).

(c) Graph induced by the MDP and a random policy.

Figure 1: An illustrative example of an RL task of guiding an agent to a goal in the grid world.

3 Gaussian Kernels on Graphs

In the LSPI algorithm, the choice of basis functions $\{\phi_i(s,a)\}_{i=1}^k$ is an open design issue. Traditionally, Gaussian kernels have been a popular choice (Lagoudakis & Parr, 2003; Engel et al., 2005), but they cannot approximate discontinuous functions well. Recently, more sophisticated methods of constructing suitable basis functions have been proposed, which effectively make use of the *graph* structure induced by MDPs (Mahadevan, 2005). In this section, we introduce a novel way of constructing basis functions by incorporating the graph structure while the relation to the existing graph-based methods is discussed in the next section.

3.1 MDP-Induced Graph

Let G be a graph induced by an MDP, where states $\mathcal{S}$ are nodes of the graph and the transitions with non-zero transition probabilities from one node to another are edges. The edges may have weights determined, e.g., based on the transition probabilities or the distance between nodes. The graph structure corresponding to an example grid-world shown in Figure 1(a) is illustrated in Figure 1(c). In practice, such graph structure (including the connection weights) is estimated from samples of a finite length. We assume that the graph G is connected. Typically, the graph is *sparse* in RL tasks, i.e.,

$$\ell \ll n(n-1)/2, \tag{11}$$

where ℓ is the number of edges and n is the number of nodes.

3.2 Ordinary Gaussian Kernels

Ordinary Gaussian kernels (OGKs) on the Euclidean space are defined as

$$K(s,s') = \exp\left(-\frac{\mathrm{ED}(s,s')^2}{2\sigma^2}\right), \tag{12}$$

where $\mathrm{ED}(s,s')$ are the Euclidean distance between states s and s'; for example,

$$\mathrm{ED}(s,s') = \|\boldsymbol{x} - \boldsymbol{x}'\|, \tag{13}$$

when the Cartesian positions of s and s' in the state space are given by $\boldsymbol{x}$ and $\boldsymbol{x}'$, respectively. σ^2 is the variance parameter of the Gaussian kernel.

The above Gaussian function is defined on the state space $\mathcal{S}$, where s' is treated as a center of the kernel. In order to employ the Gaussian kernel in the LSPI algorithm, it needs to be extended over the state-action space $\mathcal{S} \times \mathcal{A}$. This is usually carried out by simply 'copying' the Gaussian function over the action space (Lagoudakis & Parr, 2003; Mahadevan, 2005). More precisely: let the total number k of basis functions be mp, where m is the number of possible actions and p is the number of Gaussian centers. For the i-th action $a^{(i)}$ $(\in \mathcal{A})$ $(i = 1, 2, \ldots, m)$ and for the j-th Gaussian center $c^{(j)}$ $(\in \mathcal{S})$ $(j = 1, 2, \ldots, p)$, the $(i + (j-1)m)$-th basis function is defined as

$$\phi_{i+(j-1)m}(s,a) = I(a = a^{(i)})K(s, c^{(j)}), \tag{14}$$

where $I(\cdot)$ is the indicator function:

$$I(a = a^{(i)}) = \begin{cases} 1 & \text{if } a = a^{(i)}, \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

3.3 Geodesic Gaussian Kernels

On graphs, a natural definition of the distance would be the *shortest path*. So we define Gaussian kernels on graphs based on the shortest path:

$$K(s,s') = \exp\left(-\frac{\mathrm{SP}(s,s')^2}{2\sigma^2}\right), \tag{16}$$

where $\mathrm{SP}(s,s')$ denotes the shortest path from state s to state s'. The shortest path on a graph can be interpreted as a discrete approximation to the *geodesic distance* on a non-linear manifold (Chung, 1997). For this reason, we call Eq.(16) a *geodesic Gaussian kernel* (GGK).

Shortest paths on graphs can be efficiently computed using the *Dijkstra algorithm* (Dijkstra, 1959). With its naive implementation, computational complexity for computing the shortest paths from a single node to all other nodes is $O(n^2)$, where n is the number of nodes. If the *Fibonacci heap* is employed, computational complexity can be reduced to $O(n \log n + \ell)$ (Fredman & Tarjan, 1987), where ℓ is the number of edges. Since the

graph in value function approximation problems is typically sparse (i.e., $\ell \ll n^2$), using the Fibonacci heap provides significant computational gains. Furthermore, there exist various approximation algorithms which are computationally very efficient (see Goldberg & Harrelson, 2005 and references therein).

Analogous to OGKs, we need to extend GGKs to the state-action space for using them in the LSPI method. A naive way is to just employ Eq.(14), but this can cause a 'shift' in the Gaussian centers since the state usually changes when some action is taken. To incorporate this transition, we propose defining the basis functions as the expectation of Gaussian functions after transition, i.e.,

$$\phi_{i+(j-1)m}(s,a) = I(a = a^{(i)}) \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a) K(s', c^{(j)}). \tag{17}$$

This shifting scheme is shown to work very well when the transition is predominantly deterministic (see Section 5 and Section 6.1 for experimental evaluation).

3.4 Extension to Continuous State Spaces

So far, we focused on discrete state spaces. However, the concept of GGKs can be naturally extended to continuous state spaces, which is explained here. First, the continuous state space is discretized, which gives a graph as a discrete approximation to the non-linear *manifold* structure of the continuous state space. Based on the graph, we construct GGKs in the *same* way as the discrete case. Finally, the discrete GGKs are interpolated, e.g., using a linear method to give *continuous* GGKs.

Although this procedure discretizes the continuous state space, it must be noted that the discretization is only for the purpose of obtaining the graph as a discrete approximation of the continuous non-linear manifold; the resulting basis functions themselves are continuously interpolated and hence, the state space is still treated as continuous, as opposed to other conventional discretization procedures.

4 Comparison to Related Basis Function Approaches

In this section, we discuss the characteristics of GGKs in comparison to existing basis functions using a toy RL task of guiding an agent to a goal in a deterministic grid-world (see Figure 1(a)). The agent can take 4 actions: up/down/left/right. Note that actions which make the agent collide with the wall are disallowed. A positive immediate reward of $+1$ is given if the agent reaches a goal state; otherwise it receives no immediate reward. The discount factor is set at $\gamma = 0.9$.

In this task, a state s corresponds to a two-dimensional Cartesian grid position $\boldsymbol{x}$ of the agent. For illustration purposes, let us display the state value function

$$V^{\pi}(s) : \mathcal{S} \to \mathbb{R}, \tag{18}$$

which is the expected long-term discounted sum of rewards the agent receives when the agent takes actions following policy π from state s. From the definition, it can be confirmed that $V^\pi(s)$ is expressed in terms of $Q^\pi(s,a)$ as

$$V^\pi(s) = Q^\pi(s, \pi(s)). \tag{19}$$

The optimal state value function $V^*(s)$ (in log-scale) is illustrated in Figure 1(b). An MDP-induced graph structure estimated from 20 series of random walk samples[3] of length 500 is illustrated in Figure 1(c). Here, the edge weights in the graph are set at 1 (which is equivalent to the Euclidean distance between two nodes).

4.1 Geodesic Gaussian Kernels

An example of GGKs for this graph is depicted in Figure 2(a), where the variance of the kernel is set at a large value ($\sigma^2 = 30$) for illustration purposes. The graph shows that GGKs have nice smooth surface *along* the maze, but not *across* the partition between two rooms. Since GGKs have 'centers', they are extremely useful for adaptively choosing a subset of bases, e.g., using a uniform allocation strategy, sample-dependent allocation strategy, or maze-dependent allocation strategy of the centers—a practical advantage over some non-ordered basis functions. Moreover, since GGKs are local by nature, the ill-effects of local noise is constrained locally—another property that is useful in practice.

The approximated value function obtained by 40 GGKs[4] are depicted in Figure 3(a), where we put one GGK center at the goal state and remaining 9 centers are chosen randomly. For GGKs, kernel functions are extended over the action space using the shifting scheme (see Eq.(17)) since the transition is deterministic (see Section 3.3). The proposed GGK-based method produces a nice smooth function along the maze while the discontinuity around the partition between two rooms is sharply maintained (cf. Figure 1(b)). As a result, for this particular case, GGKs give the optimal policy (see Figure 4(a)).

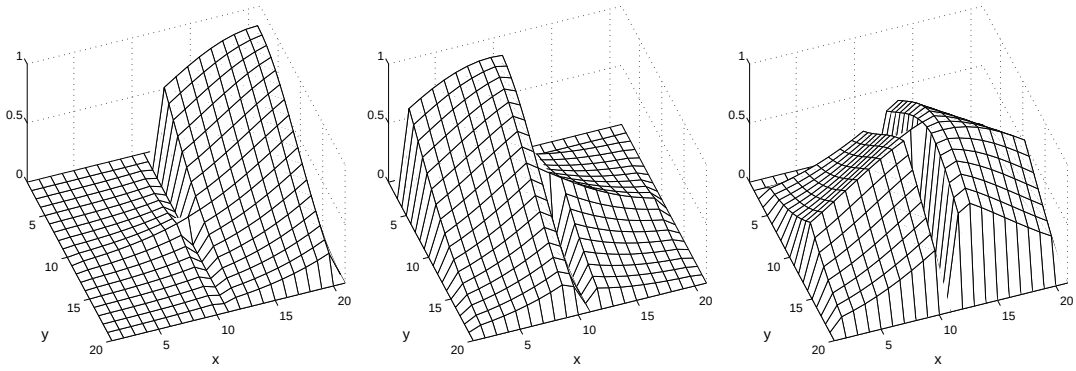
As discussed in Section 3.3, the sparsity of the state transition matrix allows efficient and fast computations of shortest paths on the graph. Therefore, the LSPI algorithm with GGK-based bases is still computationally attractive (see Section 5). GGKs includes an open parameter, i.e., *variance* σ^2 in Eq.(16). The effect of choice of the variance parameter will be discussed in Section 5.

4.2 Ordinary Gaussian Kernels

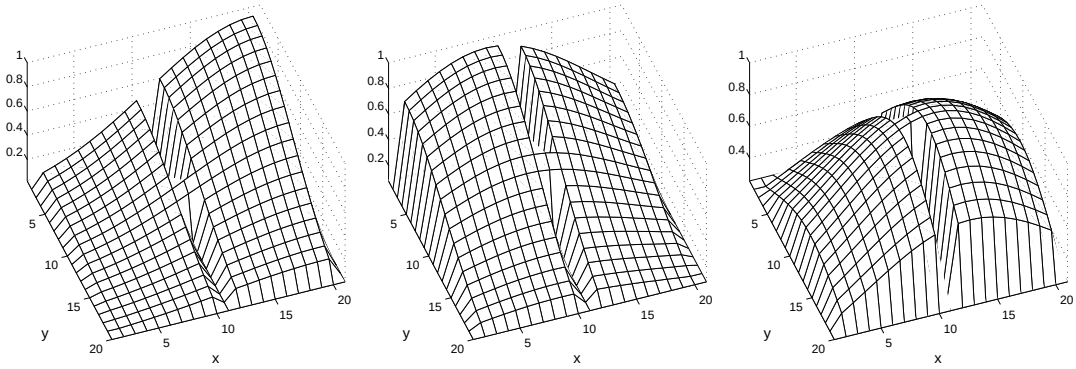
OGKs share some of the preferable properties of GGKs described above. However, as illustrated in Figure 2(b), the 'tail' of OGKs extends beyond the partition between two rooms. Therefore, OGKs tend to undesirably 'smooth' out the discontinuity of the value

[3]More precisely, in each random walk, we choose an initial state randomly. Then, an action is chosen randomly and transition is made; this is repeated 500 times. This entire procedure is independently repeated 20 times to generate the training set.

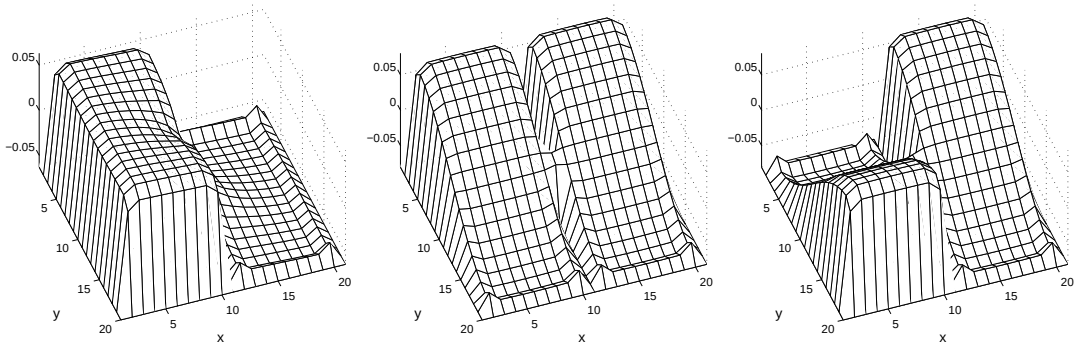
[4]Note that the total number k of basis functions is 160 since each GGK is copied over the action space as per Eq.(17).



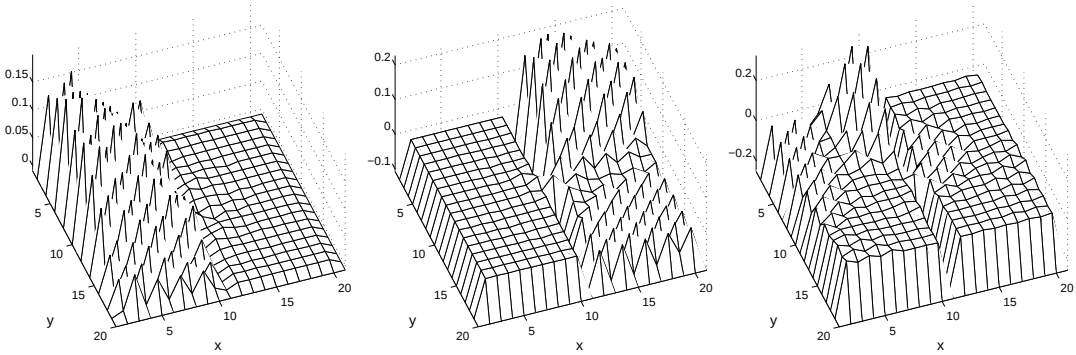
(a) Geodesic Gaussian kernels



(b) Ordinary Gaussian kernels

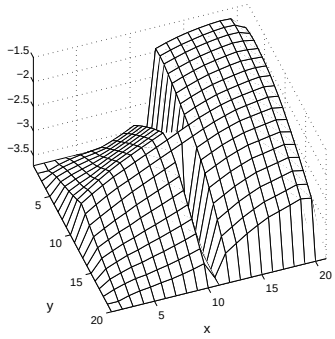


(c) Graph-Laplacian eigenbases

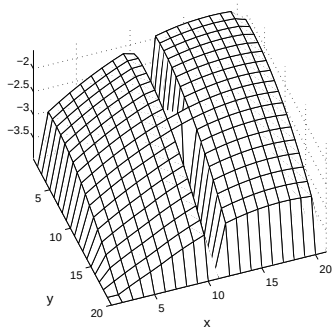


(d) Diffusion wavelets

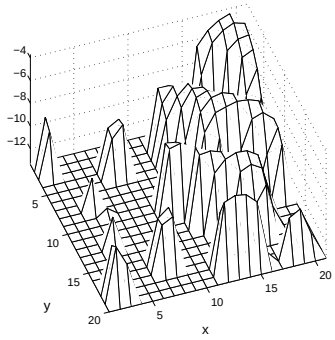
Figure 2: Examples of basis functions.



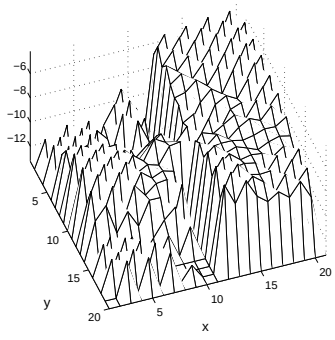
(a) Geodesic Gaussian kernels (MSE $= 1.03 \times 10^{-2}$)



(b) Ordinary Gaussian kernels (MSE $= 1.19 \times 10^{-2}$)

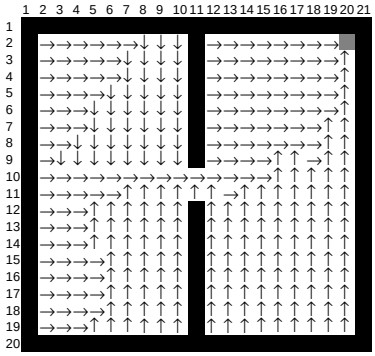


(c) Graph-Laplacian eigenbases (MSE $= 4.73 \times 10^{-4}$)

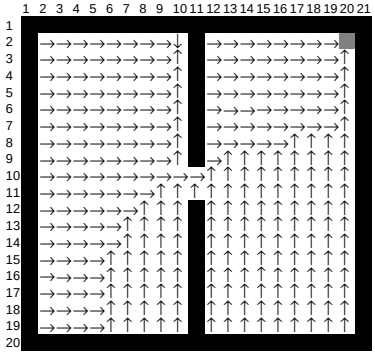


(d) Diffusion wavelets (MSE $= 5.00 \times 10^{-4}$)

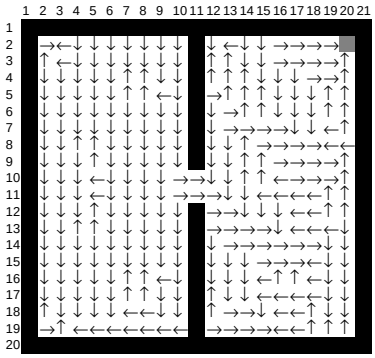
Figure 3: Approximated value functions in log-scale. The errors are computed with respect to the optimal value function illustrated in Figure 1(b)



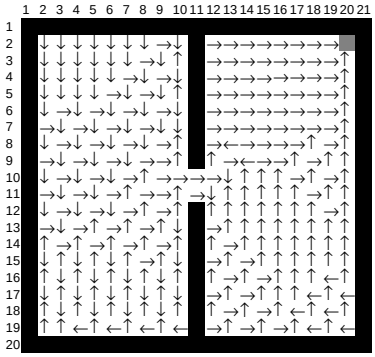
(a) Geodesic Gaussian kernels



(b) Ordinary Gaussian kernels



(c) Graph-Laplacian eigenbases



(d) Diffusion wavelets

Figure 4: Obtained policies.

function around the barrier wall (see Figure 3(b)). This causes an error in the policy around the partition (see $x = 10$, $y = 2, 3, \ldots, 9$ of Figure 4(b)).

4.3 Graph-Laplacian Eigenbases

Mahadevan (2005) proposed employing the *smoothest* vectors on graphs as bases in value function approximation. According to the spectral graph theory (Chung, 1997), such smooth bases are given by the minor eigenvectors of the *graph-Laplacian* matrix, which are called *graph-Laplacian eigenbases* (GLEs). GLEs may be regarded as a natural extension of Fourier bases to graphs.

Examples of GLEs are illustrated in Figure 2(c), showing that they have a nice Fourier-like structure on the graph. It should be noted that GLEs are rather global in nature, implying that noise in a local region can potentially degrade the global quality of approximation. An advantage of GLEs is that they have a natural ordering of the basis functions according to the smoothness. This is practically very helpful in choosing a subset of basis functions. Figure 3(c) depicts the approximated value function in log-scale, where top 40 smoothest GLEs out of 326 GLEs are used (note that the actual number of bases is 160 because of the duplication over the action space). It shows that GLEs globally give a very good approximation (although the small local fluctuation is significantly emphasized since the graph is in log-scale); indeed, the mean squared error (MSE) between the approximated and optimal value functions described in the captions of Figure 3 shows that GLEs give a much smaller MSE than GGKs and OGKs. However, the obtained value function contains systematic local fluctuation and this results in an inappropriate policy (see Figure 4(c)).

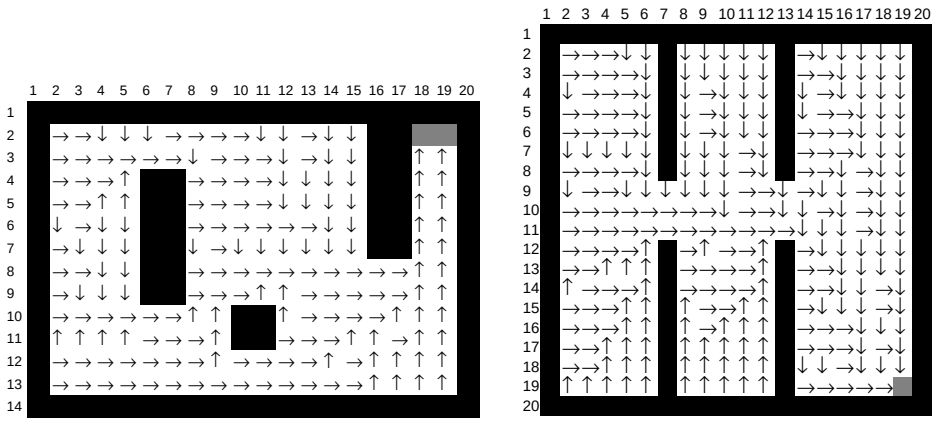
MDP-induced graphs are typically sparse. In such cases, the resultant graph-Laplacian matrix is also sparse and GLEs can be obtained just by solving a sparse eigenvalue problem—which is computationally efficient (see Section 5). However, finding minor eigenvectors could be numerically unstable.

4.4 Diffusion Wavelets

Coifman and Maggioni (2006) proposed *diffusion wavelets* (DWs), which are a natural extension of wavelets to the graph. The construction is based on a symmetrized random walk on a graph. It is diffused on the graph up to a desired level, resulting in a multi-resolution structure. A detailed algorithm for constructing DWs and mathematical properties are described in Coifman and Maggioni (2006), so we omit the detail here. We use the software provided by one of the authors of the paper as it is[5].

When constructing DWs, the maximum nest level of wavelets and tolerance used in the construction algorithm needs to be specified by users. We set the maximum nest level to 10 and the tolerance to 10^{-10}, which are the default values used in the sample code. Examples of DWs are illustrated in Figure 2(d), showing a nice multi-resolution structure on the graph. DWs are over-complete bases, so one has to appropriately choose a subset

[5] 'http://www.math.yale.edu/~mmm82/DWCode_.html'.



(a) Sutton's maze (b) Three-room maze

Figure 5: Two benchmark mazes used for simulation. In this experiment, we put GGKs at all the goal states and the remaining kernels are distributed uniformly over the maze; the 'shift' scheme described by Eq.(17) is used in GGKs.

of bases for better approximation. Figure 3(d) depicts the approximated value function obtained by DWs, where we chose the most global 40 DWs from 1626 over-complete DWs (note that the actual number of bases is 160 because of the duplication over the action space). The choice of the subset bases could possibly be enhanced using multiple heuristics; however, the current choice is reasonable since the Figure 3(d) shows that DWs give a much smaller MSE than Gaussian kernels. However, similar to GLEs, the obtained value function contains a lot of small fluctuations (see Figure 3(d)) and this results in an erroneous policy (see Figure 4(d)).

Thanks to the multi-resolution structure, computation of diffusion wavelets can be carried out recursively. However, due to the over-completeness, it is still rather demanding in computation time (see Section 5). Furthermore, appropriately determining the tuning parameters as well as choosing an appropriate basis subset is not a straightforward task in practice.

5 Experimental Comparison

In this section, we report the results of extensive and systematic experiments for illustrating the difference between GGKs and other basis function approaches.

We employ two deterministic grid-world problems illustrated in Figure 5, and evaluate the accuracy of approximated *value functions* by computing the mean squared error (MSE) with respect to the optimal value function and the performance of obtained *policies* by calculating the fraction of states from which the agent can get to the goal optimally (i.e., in the shortest number of steps). 20 series of random walk of length 300 are gathered as training samples, which are used for estimating the graph as well as the transition probability and expected reward. We set the edge weights in the graph at 1 (which is

equivalent to the Euclidean distance between two nodes).

This simulation is repeated 100 times for each maze and each method, randomly changing training samples in each run. The mean of the above scores as a function of the number of kernels is plotted in Figures 6–9. Note that the actual number of bases is four times more because of the extension of basis functions over the action space (see Eq.(14) and Eq.(17)).

First, we compare the performance of two kernel allocation strategies in GGKs:

(i) Kernels are put at all the goal states and the remaining kernels are distributed uniformly over the maze; the 'shift' strategy introduced in Section 3.3 is used.

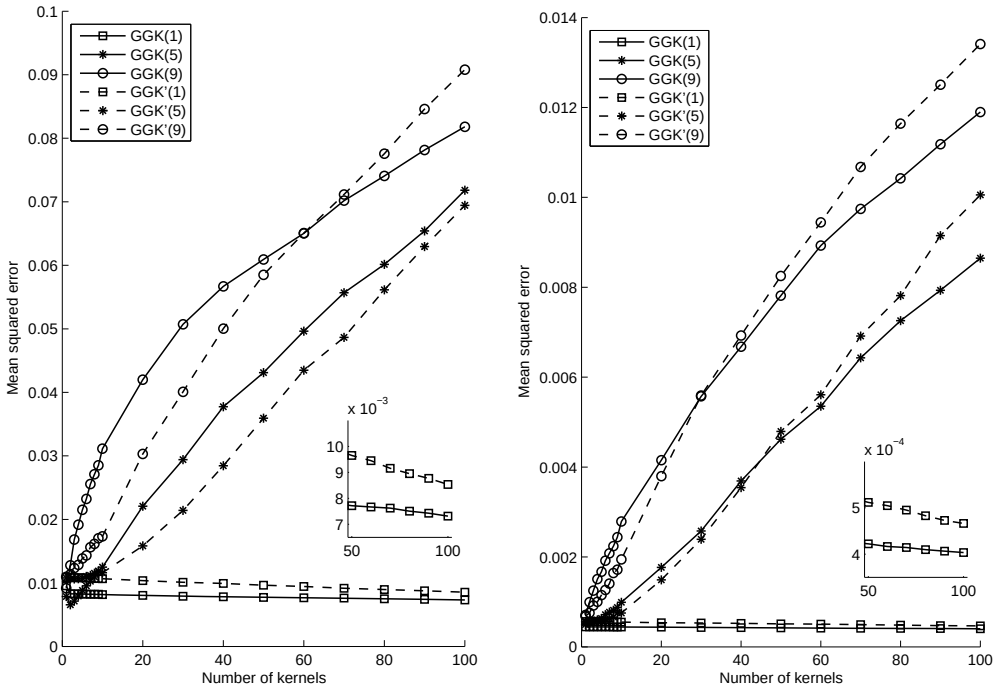
(ii) All kernels are just uniformly distributed over the maze and the 'shift' strategy is not used.

We test small ($\sigma = 1$), medium ($\sigma = 5$), and large ($\sigma = 9$) Gaussian widths. Figure 6 depicts MSEs of the approximated value functions, where the strategy (i) is denoted as GGK and the strategy (ii) is denoted as GGK'. The graphs show that the difference between the strategies (i) and (ii) is not so significant in terms of MSEs (dependence of the accuracy on the Gaussian width will be discussed below). Figure 7 depicts the fraction of optimal states in the obtained policy. The results show that when the number of kernels is small, the strategy (i) tends to perform significantly better than the strategy (ii) in terms of the quality of the obtained policy.

Next, we compare the performance of GGKs, OGKs, GLEs, and DWs. In OGKs, kernels are put at all the goal states and the remaining kernels are distributed uniformly over the maze; the 'shift' strategy is not used. Figure 8 depicts MSEs of the approximated value functions for each method. The graphs show that MSEs of GGKs with small width, OGKs with small width, GLEs, and DWs are very small and decrease as the number of kernels increases. On the other hand, MSEs of GGKs and OGKs with medium/large width are relatively large and counter-intuitively, increase as the number of kernels increases. Therefore, from the viewpoint of approximation quality of the value functions, GGKs and OGKs with smaller width seem to perform better.

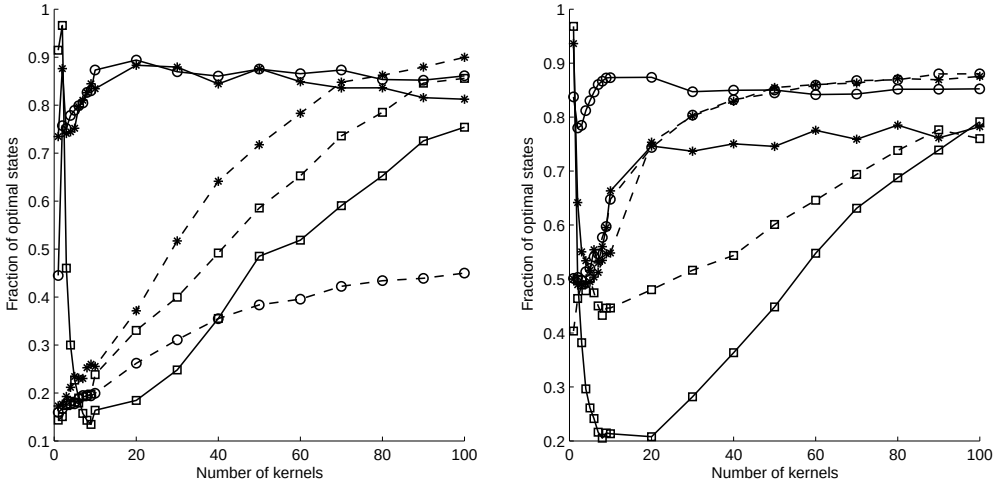
Figure 9 depicts the fraction of optimal states in the obtained policy. The graphs show that overall GGKs with medium/large width give much better policies than OGKs, GLEs, and DWs. An interesting finding from the graphs is that GGKs tend to work better if the Gaussian width is large, while OGKs show the opposite trend; this may be explained as follows. Tails of OGKs extend across the wall as illustrated in Figure 2(b). Therefore, OGKs with large width tend to produce undesired value functions and erroneous policies around the partitions. This tail effect can be alleviated if the Gaussian width is made small. However, this in turn makes the approximated value function non-smooth and fluctuating[6]; so the resulting policies are still erroneous. The fluctuation problem with a small Gaussian width seems to be improved if the number of bases is increased, while the tail effect with a large Gaussian width still remains even when the number of bases is

[6]This is a well-known drawback of Gaussian kernel based methods, see a standard textbook such as Bishop (1995).



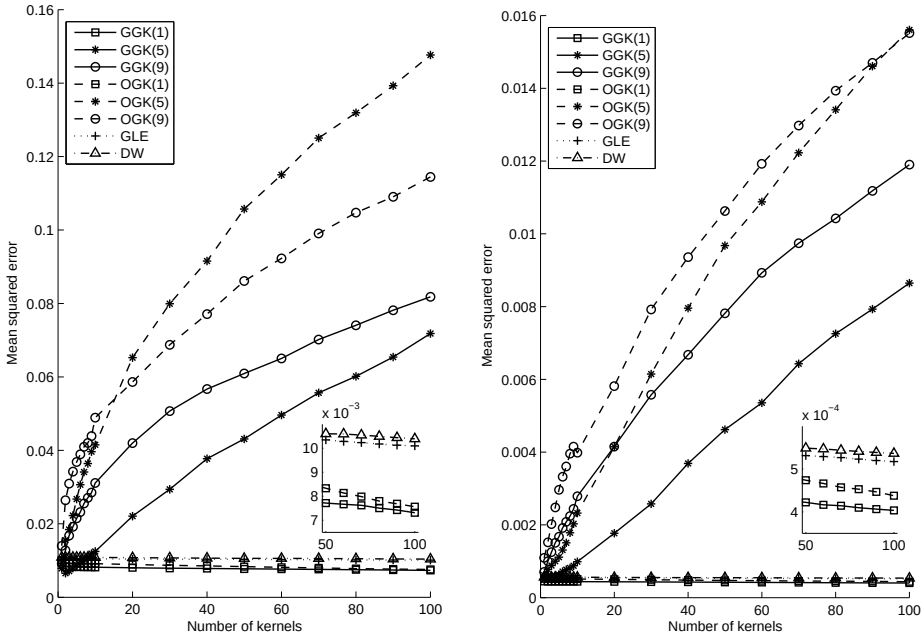
(a) Sutton's maze (b) Three-room maze

Figure 6: Mean squared error of approximated value functions averaged over 100 trials for the Sutton and three room mazes. In the legend, GGK denotes the GGK method with the kernel allocation strategy (i), i.e., kernels are put at all the goal states and the remaining kernels are distributed uniformly over the maze; the 'shift' strategy introduced in Section 3.3 is used. GGK' denotes the GGK method with the kernel allocation strategy (ii), i.e., all kernels are just uniformly distributed over the maze and the 'shift' strategy is not used. The standard deviation σ of GGK and GGK' is denoted in the bracket.



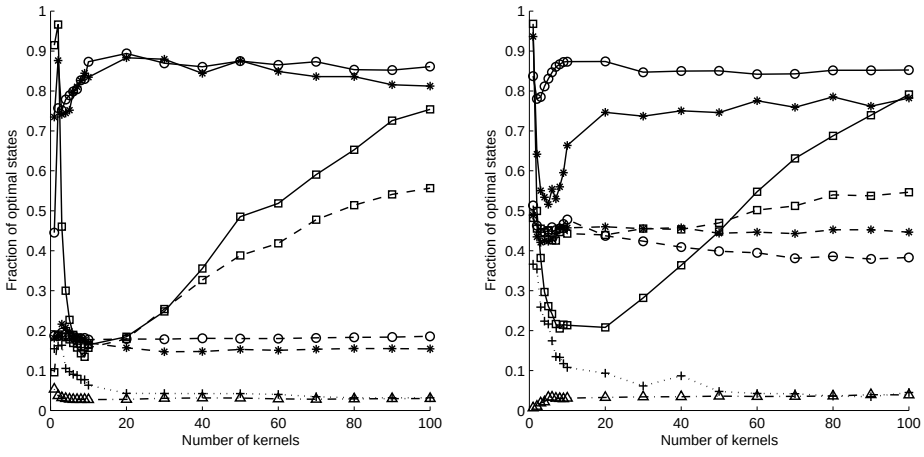
(a) Sutton's maze (b) Three-room maze

Figure 7: Fraction of optimal states averaged over 100 trials for the Sutton and three room mazes. The legends are the same as Figure 6.



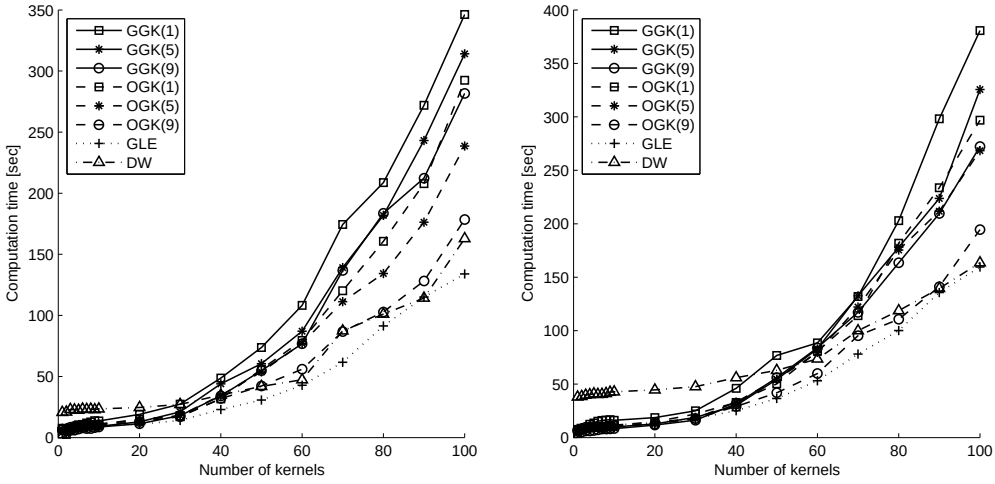
(a) Sutton's maze (b) Three-room maze

Figure 8: Mean squared error of approximated value functions averaged over 100 trials for the Sutton and three room mazes. In the legend, the standard deviation σ of GGKs and OGKs is denoted in the bracket.



(a) Sutton's maze (b) Three-room maze

Figure 9: Fraction of optimal states averaged over 100 trials for the Sutton and three room mazes. The legends are the same as Figure 8.



(a) Sutton's maze (b) Three-room maze

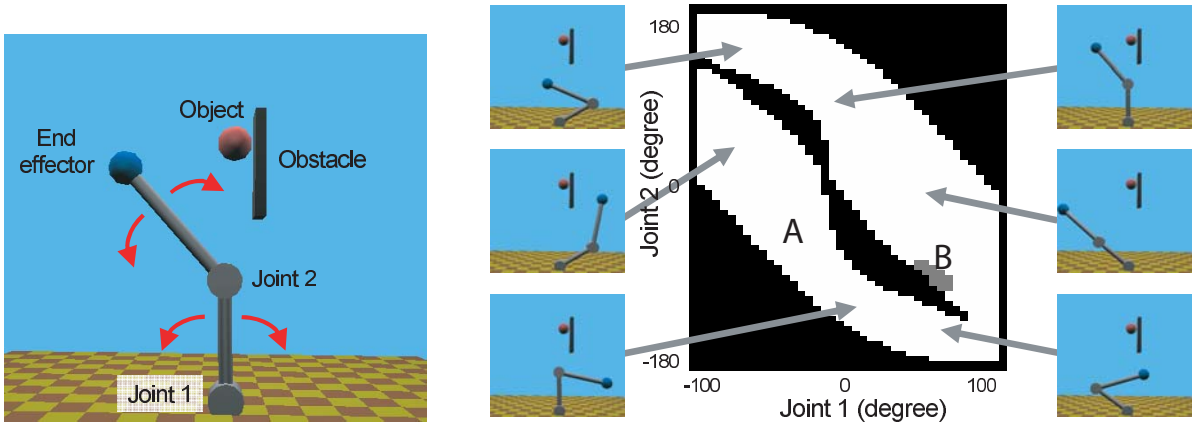
Figure 10: Computation time.

increased. On the other hand, GGKs do not suffer from the tail problem thanks to the geodesic construction. Therefore, GGKs allow us to make the width large without being affected by the discontinuity across the wall. Consequently, smooth value functions along the maze are produced and hence better policies can be obtained by GGKs with large widths. This result highlights a helpful property since it alleviates the practical issue of determining the values of the Gaussian width parameter.

The computation time of each method using our MATLAB implementation is summarized in Figure 10, showing that the proposed GGKs are slightly slower than other methods with the same number of bases. However, given that GGKs give much better policies with a small number of bases than others (see Figure 9), GGKs are computationally very efficient. Note that the running time of the Dijkstra algorithm was less than 0.1 second in the current simulations, which is negligibly small; the computation time was dominated by the LSPI iteration.

6 Applications

As discussed in the previous section, the proposed GGKs bring a number of preferable properties for making value function approximation effective. In this section, we investigate the application of the GGK-based method to the challenging problems of (simulated) robot arm control and mobile robot navigation and demonstrate its usefulness. Since GLEs and DWs appeared not to perform robustly in the pilot experiments carried out in the previous sections, we only test GGKs and OGKs here.



(a) A schematic (b) State space

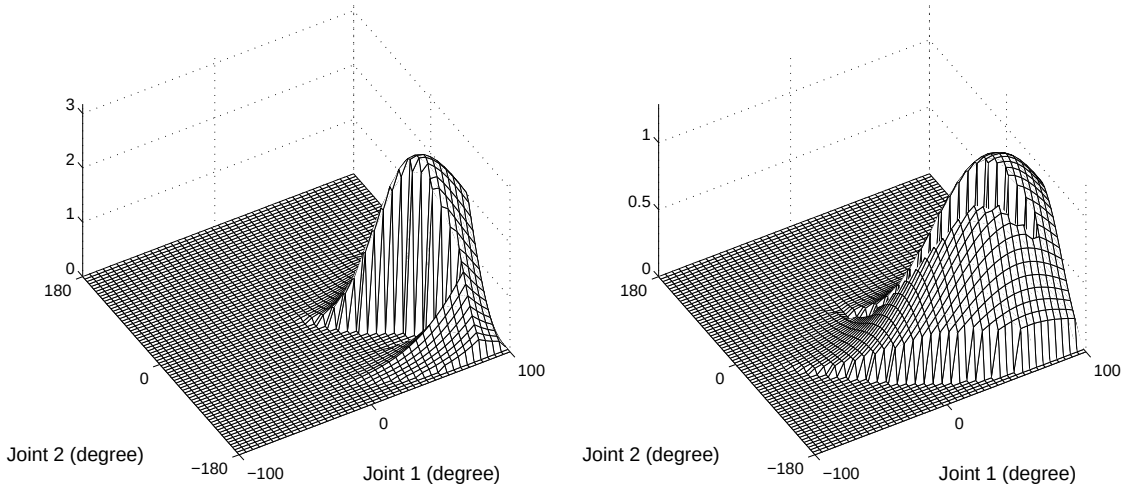
Figure 11: A two-joint robot arm. In this experiment, we put GGKs at all the goal states and the remaining kernels are distributed uniformly over the maze; the 'shift' scheme is used in GGKs.

6.1 Robot Arm Control

We use a simulator of a two-joint robot arm (moving in a plane) illustrated in Figure 11(a). The task is to lead the end-effector ('hand') of the arm to an object while avoiding the obstacles. Possible actions are to increase or decrease the angle of each joint ('shoulder' and 'elbow') by 5 degrees in the plane, simulating coarse stepper-motor joints. Thus the state space $\mathcal{S}$ is the 2-dimensional discrete space consisting of two joint-angles as illustrated in Figure 11(b). The black area in the middle corresponds to the obstacle in the joint-angle state space. The action space $\mathcal{A}$ involves 4 actions: increase or decrease one of the joint angles. We give a positive immediate reward $+1$ when the robot's end-effector touches the object; otherwise the robot receives no immediate reward. Note that actions which make the arm collide with obstacles are disallowed. The discount factor is set at $\gamma = 0.9$. In this environment, we can change the joint angle exactly by 5 degrees, so the environment is deterministic. However, because of the obstacles, it is difficult to explicitly compute an inverse kinematic model; furthermore, the obstacles introduce discontinuity in value functions. Therefore, this robot-arm control task is an interesting test bed for investigating the behavior of GGKs.

We collected training samples from 50 series of 1000 random arm movements, where the start state is chosen randomly in each trial. The graph induced by the above MDP consists of 1605 nodes and we assigned uniform weights to the edges. There are totally 16 goal states in this environment (see Figure 11(b)), so we put the first 16 Gaussian centers at the goals and the remaining centers are chosen randomly in the state space. For GGKs, kernel functions are extended over the action space using the shifting scheme (see Eq.(17)) since the transition is deterministic in this experiment.

Figure 12 illustrates the value functions approximated using GGKs and OGKs (similar to Section 4, we display *state* value functions although *state-action* value functions are



(a) Geodesic Gaussian kernels (b) Ordinary Gaussian kernels

Figure 12: Approximated value functions with 10 kernels (the actual number of bases is 40 because of the duplication over the action space).

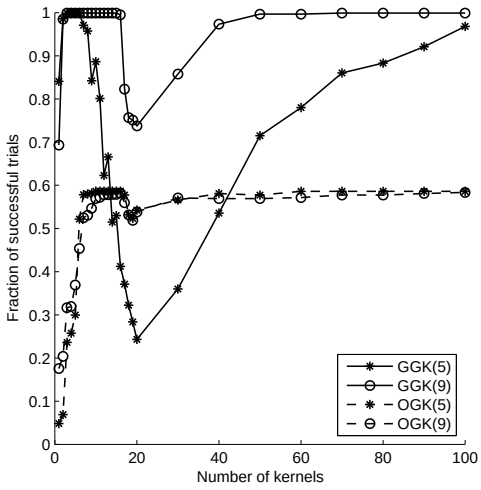


Figure 13: Number of successful trials.

approximated). The graphs show that GGKs give a nice smooth surface with obstacle-induced discontinuity sharply preserved, while OGKs tend to smooth out the discontinuity. This makes a significant difference in avoiding the obstacle: from 'A' to 'B' in Figure 11(b), the GGK-based value function results in a trajectory that avoids the obstacle (see Figure 12(a)). On the other hand, the OGK-based value function yields a trajectory that tries to move the arm *through* the obstacle by following the gradient upward (see Figure 12(b)), causing the arm to get stuck behind the obstacle[7].

[7]A demo movie is available from 'http://sugiyama-www.cs.titech.ac.jp/~sugi/2008/

Figure 13 summarizes the performance of GGKs and OGKs measured by the percentage of successful trials (i.e., the end-effector reaches the object) averaged over 30 independent runs. More precisely, in each run, totally 50000 training samples are collected using a different random seed, a policy is then computed by the GGK- or OGK-based LSPI method, and the obtained policy is tested. This graph shows that GGKs remarkably outperform OGKs since the arm can successfully avoid the obstacle. The performance of OGKs does not go beyond 0.6 even when the number of kernels is increased. This is caused by the 'tail effect' of OGKs; the OGK-based policy cannot lead the end-effector to the object if it starts from the bottom-left half of the state space

When the number of kernels is increased, the performance of both GGKs and OGKs once gets worse at around $k = 20$. This would be caused by our kernel allocation strategy: the first 16 kernels are put at the goal states and the remaining kernel centers are chosen randomly. When k is less than or equal to 16, the approximated value function tends to have a unimodal profile since all kernels are put at the goal states. However, when k is larger than 16, this unimodality is broken and the surface of the approximated value function has slight fluctuations, causing an error in policies and degrading performance at around $k = 20$. This performance degradation tends to recover as the number of kernels is further increased.

Overall, the above result shows that when GGKs are combined with our kernel-center allocation strategy, almost perfect policies can be obtained with a very small number of kernels. Therefore, the proposed method is computationally very advantageous.

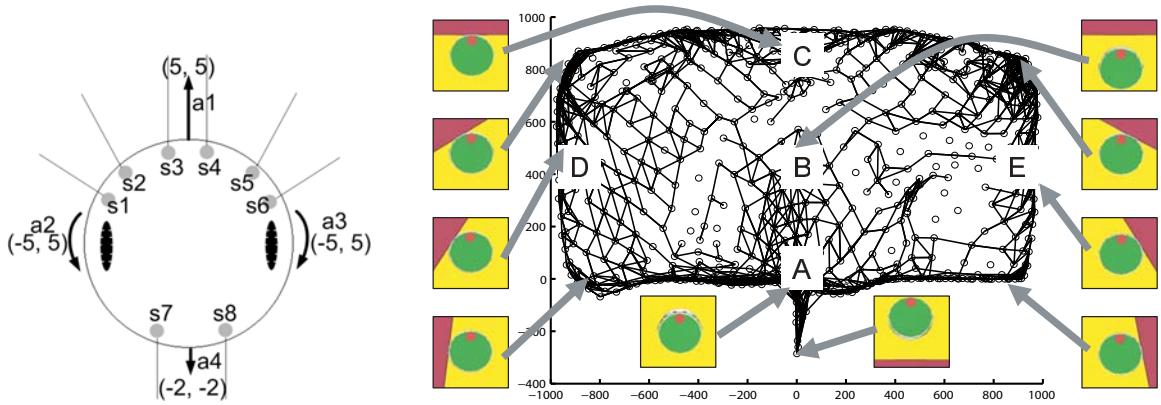
6.2 Robot Agent Navigation

The above simple robot-arm control simulation shows that the GGK method is promising. Here we apply GGKs to a more challenging task of mobile robot navigation, which involves a *high-dimensional, very large* state space.

We employ a *Khepera* robot illustrated in Figure 14(a) on the navigation task. A Khepera is equipped with 8 infra-red sensors ('s1' to 's8' in the figure), each of which gives a measure of the distance from the surrounding obstacles. Each sensor produces a scalar value between 0 and 1023: the sensor obtains the maximum value 1023 if an obstacle is just in front of the sensor and the value decreases as the obstacle gets farther till it reaches the minimum value 0. Therefore, the state space $\mathcal{S}$ is 8-dimensional. The Khepera has two wheels and takes the following 4 defined actions: forward, left-rotation, right-rotation, and backward (i.e., the action space $\mathcal{A}$ contains 4 actions). The speed of the left and right wheels for each action is described in Figure 14(a) in the bracket (the unit is pulse per 10 milliseconds). Note that the sensor values and the wheel speed are highly stochastic due to the cross talk, sensor noise, slip etc. Furthermore, perceptual aliasing occurs due to the limited range and resolution of sensors. Therefore, the state transition is also highly stochastic. We set the discount factor at $\gamma = 0.9$.

The goal of the navigation task is to make the Khepera explore the environment as much as possible. To this end, we give a positive reward +1 when the Khepera moves

`GGKvsOGK.wmv`'.



(a) A schematic

(b) State space projected onto a 2-dimensional subspace for visualization.

Figure 14: Khepera robot. In this experiment, GGKs are distributed uniformly over the maze without the 'shift' scheme.

forward and a negative reward -2 when the Khepera collides with an obstacle. We do not give any reward to the left/right rotation and backward actions. This reward design encourages the Khepera to go forward without hitting obstacles, through which extensive exploration in the environment could be achieved.

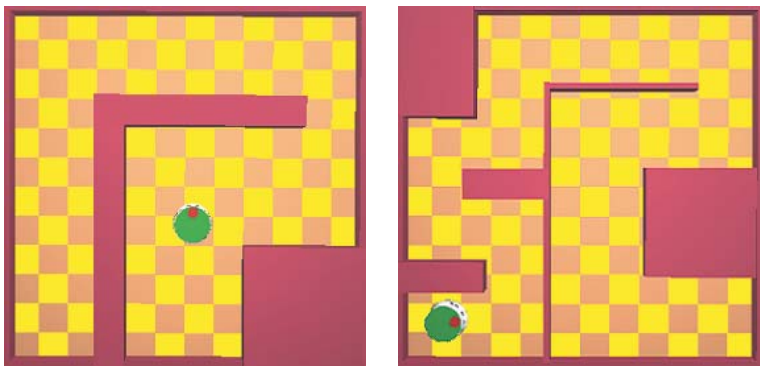
We collected training samples from 200 series of 100 random movements in a fixed environment with several obstacles (see Figure 15(a)). Then we constructed a graph from the gathered samples by discretizing the continuous state space using the *Self-Organizing Map* (SOM) (Kohonen, 1995). A SOM consists of neurons located on a regular grid. Each neuron corresponds to a cluster and neurons are connected to adjacent ones by neighborhood relation. The SOM is similar to the k-means clustering algorithm, but it is different in that topological structure of the entire map is taken into account; by that, the entire space tends to be covered. The number of nodes (states) in the graph is set at 696 (equivalent with the SOM map size of 24×29); this value is computed by the standard rule-of-thumb formula $5\sqrt{n}$ (Vesanto et al., 2000), where n is the number of samples. The connectivity of the graph is determined by state transitions occurred in the samples, i.e., if there is a state transition from one node to another in the samples, an edge is established between these two nodes and the edge weight is set according to the Euclidean distance between them.

Figure 14(b) illustrates an example of the obtained graph structure. For visualization purposes, we projected the 8-dimensional state space onto a 2-dimensional subspace[8] spanned by

$$\begin{array}{cccccccc} (-1 & -1 & 0 & 0 & 1 & 1 & 0 & 0), \\ (0 & 0 & 1 & 1 & 0 & 0 & -1 & -1). \end{array} \tag{20}$$

The i-th element in the above bases corresponds to the output of the i-th sensor (see

[8] We note that the projection is done *only* for the purpose of visualization; all the computations are carried out using the entire 8-dimensional data.



(a) Training (b) Test

Figure 15: Simulation environment

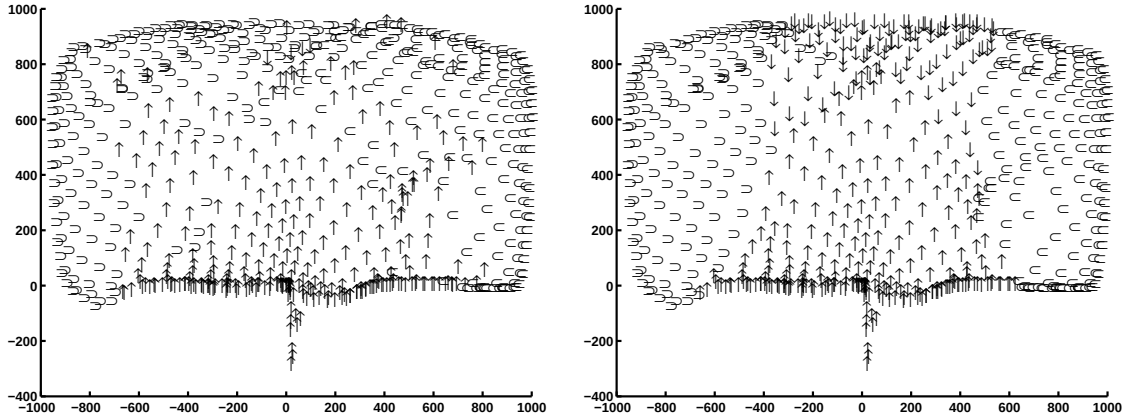
Figure 14(a)). The projection onto this subspace roughly means that the horizontal axis corresponds to the distance to the left/right obstacle, while the vertical axis corresponds to the distance to the front/back obstacle. For clear visibility, we only displayed the edges whose weight is less than 250. Representative local poses of the Khepera with respect to the obstacles are illustrated for salient nodes of the state-space MDP graph in Figure 14(b). This graph has a notable feature: the nodes around the region 'B' in the figure are directly connected to the nodes at 'A', but are very sparsely connected to the nodes at 'C', 'D', and 'E'. This implies that the geodesic distance from 'B' to 'C', 'B' to 'D', or 'B' to 'E' is typically larger than the Euclidean distance.

Since the transition from one state to another is highly stochastic in the current experiment, we decided to simply duplicate the GGK function over the action space (see Eq.(14)). For obtaining continuous GGKs, GGK functions need to be interpolated (see Section 3.4). We may employ a simple linear interpolation method in general. However, the current experiment has unique characteristics—at least one of the sensor values is always zero since the Khepera is never completely surrounded by obstacles. Therefore, samples are always on the surface of the 8-dimensional hypercube-shaped state space. On the other hand, the node centers determined by the SOM are not generally on the surface. This means that any sample is not included in the convex hull of its nearest nodes and we need to *extrapolate* the function value. Here, we simply add the Euclidean distance between the sample and its nearest node when computing kernel values; more precisely, for a state s that is not generally located on a node center, the GGK-based basis function is defined as

$$\phi_{i+(j-1)m}(s,a) = I(a = a^{(i)}) \exp\left(-\frac{(\mathrm{ED}(s,\tilde{s}) + \mathrm{SP}(\tilde{s}, c^{(j)}))^2}{2\sigma^2}\right), \tag{21}$$

where $\tilde{s}$ is the node closest to s in the Euclidean distance.

Figure 16 illustrates an example of actions selected at each node by the GGK-based and OGK-based policies. We used 100 kernels and set the width at 1000. The symbols '↑', '↓', '⊂', and '⊃' in the figure indicate forward, backward, left-rotation, and right-rotation



(a) Geodesic Gaussian kernels (b) Ordinary Gaussian kernels

Figure 16: Examples of obtained policies. The symbols '↑', '↓', '⊂', and '⊃' indicate forward, backward, left-rotation, and right-rotation actions.

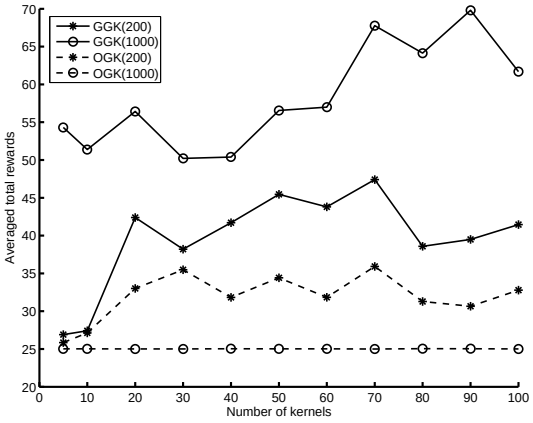


Figure 17: Average amount of exploration.

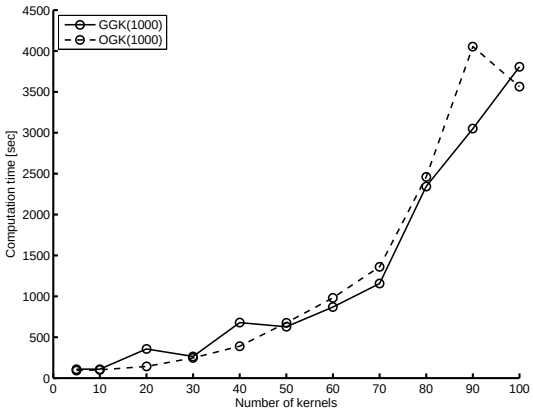
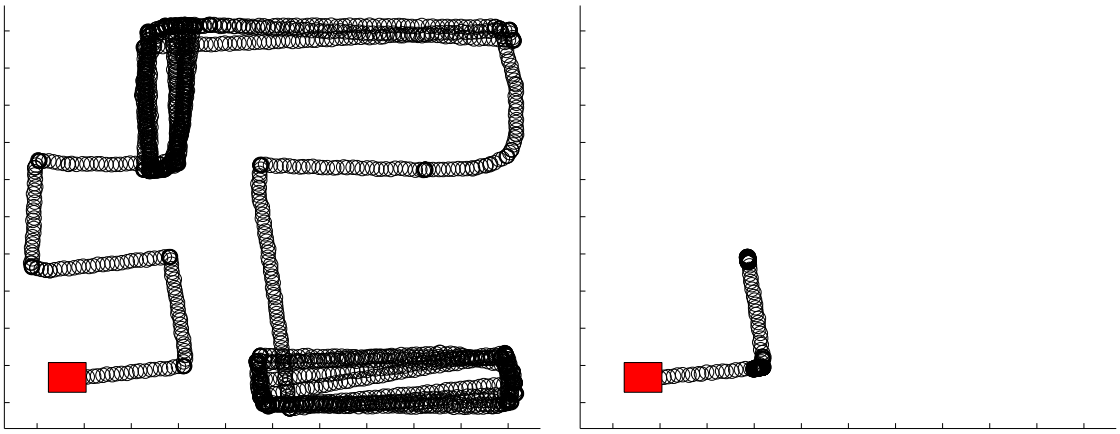


Figure 18: Computation time.

actions. This shows that there is a clear difference in the obtained policies at the state 'C'; the backward action is most likely to be taken by the OGK-based policy while the left/right rotation are most likely to be taken by the GGK-based policy. This causes a significant difference in the performance. To explain this, let us assume that the Khepera is at the state 'C', i.e., it faces a wall. The GGK-based policy guides the Khepera from 'C' to 'A' via 'D' or 'E' by taking left/right rotation actions and it can avoid the obstacle successfully. On the other hand, the OGK-based policy tries to plan a path from 'C' to 'A' via 'B' by activating the backward action; then, the forward action is taken at 'B'. Thus, the Khepera returns to 'C' again and ends up moving back and forth between 'C' and 'B'[9].

For the performance evaluation, we use a more complicated environment than the

[9]A demo movie is available from '`http://sugiyama-www.cs.titech.ac.jp/~sugi/2008/GGKvsOGK.wmv`'.



(a) Geodesic Gaussian kernels (b) Ordinary Gaussian kernels

Figure 19: Results of map building (cf. Figure 15(b)).

one used for gathering training samples (see Figure 15). Thus we are evaluating how well the obtained policies can be *generalized* to an unknown environment. In this test environment, we let the Khepera run from a fixed starting position (see Figure 15(b)) and take 150 steps following the obtained policy. We compute the sum of rewards, i.e., +1 for the forward action. If the Khepera collides with an obstacle before 150 steps, we stop the evaluation. The mean test performance over 30 independent runs is depicted in Figure 17 as a function of the number of kernels. More precisely, in each run, we construct a graph based on the training samples taken from the training environment and put the specified number of kernels randomly on the graph. Then, a policy is learned by the GGK or OGK-based LSPI method using the training samples. Note that the actual number of bases is four times more because of the extension of basis functions over the action space. The test performance is measured 5 times for each policy and the average is outputted. Figure 17 shows that GGKs significantly outperform OGKs, demonstrating that GGKs are promising even in the challenging setting with a high-dimensional huge state space.

Figure 18 depicts the computation time of each method as a function of the number of kernels. This shows that the computation time monotonically increases as the number of kernels increases and the GGK-based and OGK-based methods have comparable computation time. Given that the GGK-based method works much better than the OGK-based method with a smaller number of kernels (see Figure 17), the proposed method could be regarded as a computationally efficient alternative to the standard OGK-based method.

Finally, we apply the learned Khepera robot to map building. Starting from an initial position (indicated by a square in Figure 19), the Khepera robot takes an action 2000 times following the learned policy. We used 80 kernels with Gaussian width $\sigma = 1000$ in value function approximation. The results of GGKs and OGKs are depicted in Figure 19(a) and Figure 19(b). The graphs show that the GGK result gives a broader profile of the environment, while the OGK result only reveals a local area around the initial position.

7 Conclusions and Outlook

We proposed a new basis-construction method for value function approximation. The proposed *geodesic Gaussian kernels* (GGKs) have several preferable properties such as the smoothness along the graph and easy computability. We demonstrated the practical usefulness of the proposed method for challenging applications: both the robot-arm reaching experiments with obstacles and the Khepera exploration experiments showed quantitative improvements as well as intuitive, interpretable behavioral advantages evident from the experiments.

Experiments in Section 5 showed that GGKs with large width has larger MSEs than that with smaller width, but GGKs with large width gave better policies than that with smaller width. We conjecture that GGKs with large width give smoother value functions and hence, result in stable policies. Although this explanation would be intuitively reasonable, it needs to be elucidated in a more rigorous way.

It is shown that the policies obtained by GGKs are not so sensitive to the choice of the width of the Gaussian kernels, i.e., a reasonably large width works very well. This is a very useful property in practice. Also, the heuristics of putting Gaussian centers on goal states is shown to work quite well. Even so, it is an important future direction to develop a method for optimally tuning the width as well as the location parameters, e.g., based on the statistical machine learning theory (Vapnik, 1998; Hachiya et al., 2008).

When the transition is highly stochastic (i.e., the transition probability has a wide support), the graph constructed based on the transition samples could be noisy. When an erroneous transition results in a short-cut over obstacles, the graph-based approach may not work well since the topology of the state space changes significantly. Therefore, it is an important future work to evaluate the robustness of the proposed approach under very noisy environment and to develop a more robust method of building a graph from noisy transition samples.

In Section 3.4, we extended the proposed GGKs to continuous state space. A significant research direction will be to further explore the properties of the continuous GGKs and their application to real world, high-dimensional problems such as planning in anthropomorphic robots.

We defined the Gaussian kernels on the state space, and then extended them over the action space. If we define basis functions directly on the state-action space, the quality of value function approximation and the computational efficiency could be further improved. Our future research will focus on this topic.

In this paper, we have focused on a batch RL scenario where samples are gathered in the beginning. Another practical situation would be an online scenario where samples are gathered incrementally through the policy iteration process. Such an online scenario induces an *off-policy* situation, i.e., the policy used for data sampling and the policy used for evaluation are mismatched (Sutton & Barto, 1998). It is therefore essential to develop a method that can handle the off-policy situation efficiently, e.g., following the lines of Precup et al. (2000) and Hachiya et al. (2008).

Acknowledgements

The authors acknowledge financial support from MEXT (Grant-in-Aid for Young Scientists 17700142 and Grant-in-Aid for Scientific Research (B) 18300057), the Okawa Foundation, and EU Erasmus Mundus Scholarship.

References

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford: Clarendon Press.

Chung, F. R. K. (1997). *Spectral graph theory*. Providence, R.I.: American Mathematical Society.

Coifman, R., & Maggioni, M. (2006). Diffusion wavelets. *Applied and Computational Harmonic Analysis*, *21*, 53–94.

Daubechies, I. (1992). *Ten lectures on wavelets*. Philadelphia and Pennsylvania: Society for Industrial and Applied Mathematics.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*, 269–271.

Engel, Y., Mannor, S., & Meir, R. (2005). Reinforcement learning with Gaussian processes. *Proceedings of International Conference on Machine Learning*. Bonn, Germany.

Fredman, M. L., & Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, *34*, 569–615.

Girosi, F., Jones, M., & Poggio, T. (1995). Regularization theory and neural networks architectures. *Neural Computation*, *7*, 219–269.

Goldberg, A. V., & Harrelson, C. (2005). Computing the shortest path: A* search meets graph theory. *16th Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 156–165). Vancouver, Canada.

Hachiya, H., Akiyama, T., Sugiyama, M., & Peters, J. (2008). Adaptive importance sampling with automatic model selection in value function approximation. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08)*. Chicago, USA.

Kohonen, T. (1995). *Self-organizing maps*. Berlin: Springer.

Kolter, J. Z., & Ng, A. Y. (2007). Learning omnidirectional path following using dimensionality reduction. *In Proceedings of Robotics: Science and Systems*.

Lagoudakis, M. G., & Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research*, *4*, 1107–1149.

Mahadevan, S. (2005). Proto-value functions: Developmental reinforcement learning. *Proceedings of International Conference on Machine Learning*. Bonn, Germany.

Mahadevan, S., & Maggioni, M. (2006). Value function approximation with diffusion wavelets and Laplacian eigenfunctions. *Advances in Neural Information Processing Systems 18* (pp. 843–850). Cambridge, MA: MIT Press.

Morimoto, J., & Doya, K. (2007). Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning. *Robotics and Autonomous Systems*, *36*, 37–51.

Osentoski, S., & Mahadevan, S. (2007). Learning state-action basis functions for hierarchical MDPs. *Proceedings of the 24th International Conference on Machine Learning*.

Precup, D., Sutton, R. S., & Singh, S. (2000). Eligibility traces for off-policy policy evaluation. *Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 759–766). Morgan Kaufmann.

Schölkopf, B., & Smola, A. J. (2002). *Learning with kernels*. Cambridge, MA: MIT Press.

Sugiyama, M., Hachiya, H., Towell, C., & Vijayakumar, S. (2007). Value function approximation on non-linear manifolds for robot motor control. *Proceedings of 2007 IEEE International Conference on Robotics and Automation (ICRA2007)* (pp. 1733–1740).

Sutton, R. S., & Barto, G. A. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Vapnik, V. N. (1998). *Statistical learning theory*. New York: Wiley.

Vesanto, J., Himberg, J., Alhoniemi, E., & Parhankangas, J. (2000). *SOM toolbox for Matlab 5* (Technical Report A57). Helsinki University of Technology.

Vijayakumar, S., D'Souza, A., Shibata, T., Conradt, J., & Schaal, S. (2002). Statistical learning for humanoid robots. *Autonomous Robot*, *12*, 55–69.