Automatic Basis Function Construction for Approximate Dynamic Programming and Reinforcement Learning

Philipp W. Keller PKELLE@CS.MCGILL.CA
School of Computer Science, McGill University, Montreal, QC H3A 2A7, Canada

Shie Mannor SHIE@ECE.MCGILL.CA
Department of Electrical and Computer Engineering, McGill University, Montreal, QC H3A 2A7, Canada

Doina Precup DPRECUP@CS.MCGILL.CA
School of Computer Science, McGill University, Montreal, QC H3A 2A7, Canada

Abstract

We address the problem of automatically constructing basis functions for linear approximation of the value function of a Markov Decision Process (MDP). Our work builds on results by Bertsekas and Castañon (1989) who proposed a method for automatically aggregating states to speed up value iteration. We propose to use neighborhood component analysis (Goldberger et al., 2005), a dimensionality reduction technique created for supervised learning, in order to map a high-dimensional state space to a low-dimensional space, based on the Bellman error, or on the temporal difference (TD) error. We then place basis function in the lower-dimensional space. These are added as new features for the linear function approximator. This approach is applied to a high-dimensional inventory control problem.

1. Introduction

Markov Decision Processes (MDPs) are a standard framework for addressing the problem of planning and learning under uncertainty. Many methods for finding an optimal policy rely on computing value functions (see, e.g., Sutton &Barto, 1998), which associate long-term utilities to states of the process. In environments with large discrete state space, or with continuous state spaces, function approximation methods must be used to represent value functions. Much of the work currently carried out in reinforcement learning relies on linear function approximators (Tsitsiklis & Van Roy, 1997; Tadic, 2001). Such approximators, e.g., CMACs or radial basis functions, represent the value of a state as a linear combination of basis functions (see Sutton & Barto, Chapter 8). The parameters of the combination are usually learned from data, but in most of the work to date, the basis functions are specified by the designer of the system. However, the process of designing a function approximator for a given task can be quite difficult and time-consuming. Moreover, even with careful engineering, continuous problems with more than 10-12 dimensions cannot be handled by current techniques (Munos & Moore, 2002). Recently, a significant amount of work has been devoted to constructing bases for value functions automatically, e.g., (Mahadevan, 2005; Mannor et al., 2005; Ratitch & Precup, 2004; Smart, 2004; Ziv & Shimkin, 2005). We review this literature in Sec. 7. While these methods look very promising, there is no empirical or theoretical evidence to date of their efficiency in large problems.

In this paper, we propose a different approach for constructing basis functions automatically. Like Bertsekas and Castañon (1989), we repeatedly project the state space onto a lower dimensional space. We use neighborhood component analysis (NCA) (Goldberger et al., 2005) to produce a mapping to a low dimensional Euclidean space, and aggregate states in this space to define additional basis functions. Once the basis functions are constructed, the function approximator is trained as usual, e.g., using least-squares temporal difference learning (Boyan, 2002). We provide a motivation for this algorithm in terms of a reduction in the Bellman error, and show promising results on a high-dimensional inventory control problem.

2. Background and Notation

An MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ where $\mathcal{S}$ is the space of possible states of the environment, $\mathcal{A}$ is a set of actions available to the agent, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ defines a con-

Appearing in *Proceedings of the 23rd International Conference on Machine Learning*, Pittsburgh, PA, 2006.

ditional probability distribution over state transitions given an action, and $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function assigning immediate rewards to actions. A stationary policy $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$ defines the probability of selecting each action in each state. The objective is to find a policy maximizing the value function

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t) | s_0 = s\right],$$

where s_t and a_t are the state and action at time t respectively, $\gamma \in (0,1)$ is a discount factor, and actions are selected according to the policy π.

A standard approach to finding the optimal value function is policy iteration (Bertsekas, 2005). In policy iteration, a policy π is fixed and the corresponding value function V^{π} is computed; the resulting estimate is used to improve the policy, and the process is repeated. In this paper, we focus on the policy evaluation step.

We assume a large but finite state space $\mathcal{S}$. Hence, the expected immediate reward under policy π can be represented as a vector $R \in \mathbb{R}^{|\mathcal{S}|}$, and the transition probabilities under π can be represented as a matrix $P \in \mathbb{R}^{|\mathcal{S}| \times |\mathcal{S}|}$. Note that R and P depend on π, $\mathcal{R}$ and $\mathcal{P}$. Value functions can also be represented as vectors $V \in \mathbb{R}^{|\mathcal{S}|}$.

For any given policy π, the value function V^{π} is the unique solution to the Bellman equations:

$$V^{\pi} = R + \gamma P V^{\pi}. \tag{1}$$

Successive approximations of V^{π} can be obtained by repeatedly applying the Bellman backup operator

$$V^{k+1} = T(V^k) \stackrel{\text{def}}{=} R + \gamma P V^k. \tag{2}$$

Often the matrices R and P are not known but sample trajectories $\langle s_0, r_0, s_1, r_1, \ldots, s_T, r_T \rangle$ consisting of the sequence of states and rewards observed when following a policy are available. In this case one can estimate the model parameters R, P and apply the method just described.

For large or continuous state spaces, V^{π} cannot be represented explicitly, so instead it must be approximated. Theoretical results (Tsitsiklis & Van Roy, 1997; Tadic, 2001) show that some convergence guarantees can still be obtained for linear function approximators. In this case, the value function approximation has the form $\hat{V} = \Phi\theta$, where Φ is a $|\mathcal{S}| \times m$ matrix in which each row contains the feature vector for a particular state, and θ is a vector of m parameters. Typically it is desirable to have $m \ll |\mathcal{S}|$. Usually, only Φ is assumed to be given, and the learning algorithm adjusts θ. In this paper, we will be using the least-squares temporal difference learning algorithm (LSTD) introduced by Bradtke & Barto (1996) and extended by Boyan (2002).

The algorithm estimates approximations for the reward vector R and the matrix $I - \gamma P$ based on sample trajectories. In the version with no eligibility traces, used here, on every time step t, these approximations are incremented by $r_t \phi_t$ and $\phi_t (\phi_t - \gamma \phi_{t+1})^{\top}$ (outer product) respectively, where r_t is the reward at time step t and ϕ_t, ϕ_{t+1} are the feature vectors at time steps t and $t+1$. LSTD converges in the limit to the same parameters as the incremental TD algorithm (Sutton, 1988) but does not require a learning rate, and makes more efficient use of data samples at the expense of increased complexity per iteration.

The policy evaluation algorithm presented in this paper is comprised of the following generic stages:

1. compute an approximation of the value function;
2. estimate the Bellman error for a sample of the states;
3. learn a mapping from the states space to a low-dimensional Euclidean space based on the Bellman error estimates;
4. aggregate states in this low-dimensional space to construct additional basis functions.

We use LSTD for the first step. Step 2 can be accomplished using a model of the MDP or data samples. For step 3, we rely on NCA, which is discussed in Section 4. The aggregation step is accomplished using simple discretizations. Section 3 describes the motivation for our approach.

3. Automatically Selecting Basis Functions through Adaptive State Aggregation

We are interested in finding automatically a good feature matrix Φ. To do this, we will leverage ideas proposed by Bertsekas and Castañon (1989) in the context of state aggregation for the purpose of speeding up policy evaluation. In state aggregation, the state space is partitioned into m disjoint groups. Unlike us, Bertsekas and Castañon assume that the state space is small enough to afford a tabular representation of the value function.

The value function estimate V is stored as a vector of size $|\mathcal{S}|$, but applications of the Bellman backup operator (2) are interleaved with aggregation iterations of the form

$$V^{k+1} = V^k + \Psi y,$$

where Ψ is an $|\mathcal{S}| \times m$ aggregation matrix, such that $\Psi_{ij} = 1$ if state i is assigned to group j or 0 otherwise. The low-rank correction Ψy is computed as the solution to a small policy evaluation problem defined on an aggregate Markov chain with transition probabilities and rewards

$$P_A = \Psi^{\dagger} P \Psi,$$
$$R_A = \Psi^{\dagger}(T(V^k) - V^k),$$

where $\Psi^\dagger = (\Psi^\top\Psi)^{-1}\Psi^\top$ is the pseudoinverse of the aggregation matrix. In other words, $P_A \in \mathbb{R}^{m\times m}$ contains the group-to-group transition probabilities, and the rewards $R_A \in \mathbb{R}^m$ represent the average Bellman residuals for the states in each group. The m-dimensional vector y is obtained, analogously to (1), as the solution to $y = R_A + \gamma P_A y$:

$$y = (I - \gamma P_A)^{-1} R_A. \tag{3}$$

Bertsekas and Castañon show that the Bellman error after an aggregation iteration is described by a sum of two terms:

$$\begin{aligned} T(V^{k+1}) - V^{k+1} &= E_1 + E_2 \\ E_1 &= (I - \Pi)(T(V^k) - V^k) \\ E_2 &= \gamma(I - \Pi)P\Psi y \end{aligned} \tag{4}$$

where the matrix $\Pi = \Psi^\dagger\Psi^\top$ is an orthogonal projection onto the range of Ψ (it is symmetric and $\Pi^2 = \Pi$.) In fact, for any $x \in \mathbb{R}^{|\mathcal{S}|}$, $(\Pi x)_s$ is the mean value of the elements of x in state s's aggregate group. Then the first error term E_1 is the projection of the Bellman errors onto the nullspace of Ψ, and its s-th element is the Bellman error at state s minus the mean Bellman error in state s's aggregate group. A careful choice of Ψ can bound the contribution of E_1 in terms of the Bellman error $(T(V^k) - V^k)$ at iteration k.

Indeed, Bertsekas and Castañon propose to divide the range of Bellman errors at iteration k into m equally sized intervals of length $F(T(V^k) - V^k)/m$, with $F(x) = (\max_s x_s - \min_s x_s)$, and group the states with errors in each interval. This is shown to provide a reduction in the first term by a factor of at least $2/m$ in terms of the F pseudonorm defined above. The second term E_2 depends on how well the aggregation Ψ preserves the action of the transition matrix P, and is shown to be small for certain interesting classes of transition matrices. The algorithm just mentioned ensures convergence in general despite any error introduced by E_2 by interleaving applications of the Bellman operator with the aggregation steps.

Unfortunately, this approach cannot be directly applied when the value function is being approximated, because (a) the corrected value function may not be representable by the approximator; (b) it is not possible to efficiently represent the aggregation matrix Ψ in general; and (c) the Bellman error at each state cannot be computed efficiently.

Instead, we assume a linear value function approximation and aim to improve the feature matrix Φ. Suppose that we somehow got an aggregation matrix Ψ (for which the error term E_1 is small). The rows of this matrix provide features able to represent a significant portion of the Bellman error. We can update the approximation $\hat{V}$ as

$$\begin{aligned} \hat{V}^{k+1} &= \hat{V}^k + \Psi y = \Phi^k\theta^k + \Psi y \\ &= \begin{bmatrix}\Phi^k & \Psi\end{bmatrix}\begin{bmatrix}\theta^k \\ y\end{bmatrix} = \Phi^{k+1}\begin{bmatrix}\theta^k \\ y\end{bmatrix}. \end{aligned}$$

We have increased the size of the feature matrix Φ by adding m new state features. The new parameters could be set to $\theta^{k+1} = \begin{bmatrix}\theta^k & y\end{bmatrix}^\top$. However, since we will use LSTD (which converges only to an approximation of the optimal parameters, using sample trajectories) we will recompute θ^{k+1} in our experiments.

Unlike in adaptive state aggregation, we do *not* use the aggregation matrix directly to compute updates to our value function approximation. (If such updates could be represented by our current features, they would not be necessary.) Instead, we use an aggregation matrix to define new features for the linear approximator. The resulting approximator is *not* a state aggregator. In fact, the method can be applied regardless of the nature of the other features used.

As in dynamic programming, we focus on finding an aggregation matrix which minimizes the error term E_1. We propose a method similar in spirit to that of Bertsekas and Castañon, but which provides an aggregation matrix that is easy to represent in a compact form for large state spaces. To do this, we need to relax the strict bound on the within-group differences in Bellman errors, required in their algorithm. This is necessary in our case: using the F pseudonorm is inappropriate because we may only have a sampling of the Bellman residuals for a large state space, and so we cannot guarantee that all errors in a group will fall within a given interval. Moreover, the criterion used to judge the quality of an approximation $\hat{V}$ is not generally the sup-norm, but a squared-error type norm, or the errors encountered along trajectories. In this work we will attempt to choose Ψ such that $\|E_1\|_2/\|T(V^k) - V^k\|_2$ is small. Ideally, we would like the new basis functions to represent the Bellman error exactly, bringing this quantity to 0, but this would require a compact representation of the Bellman error over all states.

4. Neighborhood Component Analysis

We need a method for producing new basis functions. Suppose that states are actually represented as n-dimensional real-valued vectors, $\mathcal{S} \subset \mathbb{R}^n$. We will use an adaptation of NCA, a method originally intended for computing a distance metric to optimize nearest-neighbor classification performance (Goldberger et al., 2005). NCA is designed to learn weighted distance metrics between points $x, y \in \mathbb{R}^n$, $\mathcal{D}(x,y) = (x-y)^\top Q(x-y)$, where Q is a matrix giving the weights for every dimension. NCA searches for a matrix Q of the form $A^\top A$ where A is restricted to be in $\mathbb{R}^{d\times n}$ for $d \ll n$. In this case, the distance metric can be re-written

as:

$$\mathcal{D}(x,y) = (x-y)^\top A^\top A(x-y) = (Ax-Ay)^\top (Ax-Ay).$$

Hence, A is in fact a linear transformation from $\mathbb{R}^n$ to $\mathbb{R}^d$, and classification can take place in this lower dimensional space using the Euclidean norm. This reduces computational and storage requirements and provides an effective dimensionality reduction algorithm.

We adapt the technique to optimize nearest-neighbor regression performance, and compute a mapping from the state space $\mathcal{S}$ to $\mathbb{R}^d$ for a small d, using a sample of the state space with Bellman residuals estimates as labels. The learned transformation maps states with similar Bellman errors close together. We exploit this local property to produce a partition of the state space with small within-group differences in Bellman residuals.

4.1. Regression Objective

NCA assumes a stochastic nearest neighbor approximator (classifier, in the original context) where a query point $x_i \in \mathbb{R}^n$ is associated with a point x_j with probability

$$p_{ij} = \frac{\exp\left(-\left\|Ax_i - Ax_j\right\|^2\right)}{\sum_{k\neq i}\exp\left(-\left\|Ax_i - Ax_k\right\|^2\right)}, \qquad p_{ii} = 0. \tag{5}$$

At query time, a neighbor of the point x_i is chosen randomly according to the above probabilities and its label y_j is returned.

The expected squared error for a query point x_i is given by

$$\delta_i = \sum_j p_{ij}\left(y_j - y_i\right)^2,$$

and the leave-one-out crossvalidation error is given by

$$f(A) = \sum_i \delta_i = \sum_i \sum_j p_{ij}\left(y_j - y_i\right)^2.$$

We wish to minimize this quantity over the set of possible metrics $\left\{Q | Q = A^\top A, A \in \mathbb{R}^{d\times n}\right\}$. For an approximator, $f(A)$ is a measure of the training error.

The gradient of f with respect to A is given by (taking $x_{ij} = x_i - x_j$ and $y_{ij} = y_i - y_j$)

$$\begin{aligned}\frac{\partial f}{\partial A} &= -2A\sum_i\sum_j p_{ij}y_{ij}^2\left(x_{ij}x_{ij}^\top - \sum_k p_{ik}x_{ik}x_{ik}^\top\right)\\ &= 2A\sum_i\left(\delta_i\sum_k p_{ik}x_{ik}x_{ik}^\top - \sum_j p_{ij}y_{ij}^2x_{ij}x_{ij}^\top\right)\\ &= 2A\sum_i\sum_j\left(\delta_i - y_{ij}^2\right)\left(p_{ij}x_{ij}x_{ij}^\top\right).\end{aligned} \tag{6}$$

The last expression (6) affords the most efficient computation.

4.2. NCA Implementation

In our implementation, we use delta-bar-delta (Jacobs, 1988) with (6) to find A, though the conjugate-gradient method generally yielded similar results and required roughly the same number of gradient evaluations. We note that gradient evaluations are fairly costly.

The objective gives preference to matrices with large elements, since these increase the distances between points and thus strengthen the assignment to the single nearest neighbor for each point. To counter this effect and ensure convergence a regularization term, the squared Frobenius norm of A, is added to the objective, giving

$$\begin{aligned}g(A) &= f(A) + N\rho\left\|A\right\|_F,\\ \frac{\partial g}{\partial A} &= \frac{\partial f}{\partial A} + 2N\rho A,\end{aligned} \tag{7}$$

with ρ some small value (e.g., $\rho = 10^{-6}$) which can be used to control the norm of the resulting matrix, and N the number of sample points used to compute the gradient.

The computation of (6) takes time quadratic in the number of points used. Hence, in our computations we use only a sample of all the available points, selected uniformly. This could potentially be problematic if the points are not well distributed accross the state space, but we have not encountered such problems in our experiments.

To facilitate parameter selection and improve accuracy, the following refinements are made: all points and labels are normalized to the $[0,1]$ interval, the mean squared error is used instead of the sum squared error, and the distance to the nearest neighbor is subtracted from all distances used for the computation of p_{ij} in (5) to avoid underflow.

5. The Algorithm

Note that an aggregation matrix Ψ (as defined in Sec. 3) can be viewed as a projection from the state space to an m-dimensional feature space. We use NCA to find a transformation $A : \mathbb{R}^n \to \mathbb{R}^d$ from the state space to a low-dimensional Euclidean space. Here, we define basis functions mapping points in $\mathbb{R}^d$ to the feature space $\mathbb{R}^m$. Composing the two mappings yields Ψ.

The input points for NCA are states and the labels are Bellman error estimates. Hence, a discretization in $\mathbb{R}^d$ tends to minimize the difference in errors between aggregated states, since "nearby" states will fall in the same cell. This in turn tends to minimize the error term E_1 from (4).

The algorithm for approximate policy evaluation which constructs basis functions automatically is shown in Algorithm 1. The function NCA $(\{s_t\},\{e_t\},d)$ learns a transformation $A \in \mathbb{R}^{d\times n}$ using the states $\{s_t\}$ as data points and the Bellman error estimates $\{e_t\}$ as labels, as described in Section 4. The function SELECTFEATURES $(\{s_t\},\{e_t\},A,m)$

Algorithm 1 Policy evaluation

Given
the dimensionality d to be used in each projection
the number of features (basis functions) to be added in each iteration, m
the desired number of iterations K, and
a trajectory $\langle s_0, r_0, s_1, r_1, \ldots, s_T, r_T \rangle$. (or a set of trajectories)
$\Phi^0 \leftarrow \overrightarrow{1}$
$\theta^0 \leftarrow \text{LSTD}\left(\{s_t\}, \{r_t\}, \Phi^0\right)$
$\hat{V}^0 \leftarrow \Phi^0 \theta^0$
for $k = 1, 2, \ldots, K$ **do**
Estimate the Bellman residuals
$e_t \approx R(s_t) + \gamma \sum_{s \in \mathcal{S}} P_{s_t,s} \hat{V}_s^{k-1} - \hat{V}_{s_t}^{k-1}$
$A \leftarrow \text{NCA}\left(\{s_t\}, \{e_t\}, d\right)$
$\Psi \leftarrow \text{SELECTFEATURES}\left(\{s_t\}, \{e_t\}, A, m\right)$
$\Phi^k \leftarrow \left[\Phi^{k-1} \quad \Psi\right]$
$\theta^k \leftarrow \text{LSTD}\left(\{s_t\}, \{r_t\}, \Phi^k\right)$
$V^k \leftarrow \Phi^k \theta^k$
end for
return $\hat{V}^k$

returns a set of (at most) m features in the low-dimensional image space of A (possibly computed based on the state-reward trajectory), and $\text{LSTD}\left(\{s_t\}, \{r_t\}, \Phi^k\right)$ uses the LSTD(0) algorithm to estimate parameters for the linear function approximator with features Φ^k.

Initially, a single feature which is 1 everywhere is defined and LSTD is used to compute the parameters of the approximator. On every iteration the Bellman residuals $\{e_t\}$ are estimated for each state in the sample trajectory. In general the model parameters P and R are not be available so the residuals can only be approximated based on an approximate model, or on sampled data. The accuracy of the estimates is not crucial since NCA is robust to noise. In the results presented below, we use approximate Bellman errors, but using TD errors gives very similar results. The NCA algorithm is used to compute a transformation to a low-dimensional space in which new basis functions are defined. These are then appended to the feature matrix, and LSTD is repeated to obtain a new approximation $\hat{V}^k$.

The effectiveness of the algorithm results from the combination of the mapping A and the choice of features in the low-dimensional space. In our experiments we choose $d = 2$ and SELECTFEATURES simply discretizes the projection of the state space onto $\mathbb{R}^2$ into up to m aggregated states and returns the aggregation matrix (efficiently represented by the transformation A and a 2-dimensional grid). A heuristic is used to greedily construct a discretization with "square" cells. Repeated iterations allow a good approximation to the value function even if it cannot be represented by a single such aggregation matrix, since subsequent applications of NCA on the recomputed Bellman errors will yield different projections and discretizations. It is not necessary that the new features Ψ even be an aggregation. The analysis resulting in (4) holds for any Ψ such that y in (3) exists. For instance, radial basis functions could be defined in $\mathbb{R}^2$ and used instead of aggregation.

6. Experimental Results

In order to test our algorithm, we use an inventory control problem. The goal is to maintain an inventory of products such as to minimize a cost function. Typically, storage costs are associated with different items. If only one product is considered, the problem has an easy closed-form solution. However, for two or more goods, a closed-form optimal solution is not known. Even though the problem looks simple, correlations between the demands for different products can cause difficulties both for finding theoretical solutions, as well as for solving such problems in practice.

The version of the problem we use here is a direct extension of the problem described in (Bertsekas, 2005). We will use similar notation for consistency. We assume that there are l groups of products, each containing h goods. The goods in each group have correlated demand. Hence, one can think of l as being the "intrinsic" dimensionality of the problem. For each group i of products, there is a stochastic demand w_i at each time step, drawn independently from a multinomial distribution over $\{0, 1, 2, 3, 4\}$. The demand w_{ij} for product j of group i is then drawn from a normal distribution centered at w_i: $w_{ij} \sim \mathcal{N}(w_i, \sigma^2)$, where σ is a parameter controlling the correlation between products.

We denote by s_{ij} the inventory of the j-th good in group i. The state representation consists of these lh inventories along with lh additional dimensions consisting of pure noise $z_{ij} \sim \mathcal{N}(0, 1)$:

$$s = \left[s_{11}, s_{12}, \ldots, s_{1l} \ldots s_{l1}, s_{l2}, \ldots, s_{lh}, z_{11}, \ldots, z_{lh}\right]^\top.$$

The noise dimensions are intended to increase the difficulty of identifying the underlying dimensions of the problem, in addition to the within-group variance. We expect NCA to learn to ignore this part of the state representation. The actions are defined as a vector $u \in \mathbb{R}^{lh}$ specifying the amount ordered of each good.

We denote such a problem $\text{ICP}(l, h, \sigma)$. Note that it is easy to scale up the problem to many dimensions, and also to control the amount of noise. Let s_t, u_t and $w_t \in \mathbb{R}^{lh}$ denote the state, action and demand at time step t. The state is updated according to:

$$s_{t+1} = \begin{bmatrix} \bar{s}_t + u_t - w_t \\ z_{t+1} \end{bmatrix},$$

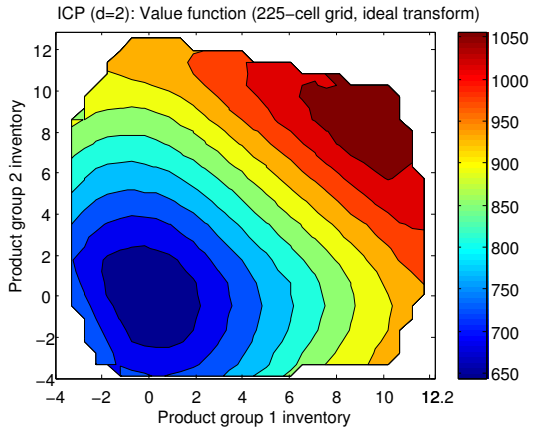

Figure 1. Value function estimate for the inventory control problem with two product groups, ICP(2,4,0).

where the ij-th element of $\bar{s}_t$ is the mean inventory of all the products in the i-th group at step t, and z_{t+1} is the vector containing the independently sampled noise variables. Replacing the inventories with the group means ensures that they remain correlated within a group regardless of the policy without requiring resets, and thus facilitates the choice of a policy to evaluate. Note that the demand w_t is not included in the state, as it is unknown to the agent; it is interpreted as the stochastic effect of the environment. The cost function is given by

$$\begin{aligned}\mathcal{R}(s_t,u_t,s_{t+1}) &= c_{aquire}^\top u_t + c_{short}^\top \max(0,-s_{t+1}) + \\ &\quad c_{store}^\top \max(0,s_{t+1}),\end{aligned}$$

where $c_{aquire} = [1,1,...,1]^\top$ is the cost of aquiring inventory, $c_{short} = [4,4,...,4]^\top$ is the cost associated with a unit shortage and $c_{store} = [2,2,...,2]^\top$ is the cost associated with storing one unit of each product. Of course, to make an interesting control problem, one would use different costs for different products. However, since here we are evaluating a fixed policy, this simple cost structure is sufficient. We use the policy:

$$u_{ij} = E[w_i] + 0.1(M_i - s_{ij}),$$

where M_i is the target inventory for the i-th set of products. We let $M_i = 2\ \forall i$.

In all the experiments we use one trajectory through the state space, of 5000 time steps. The discount factor is $\gamma = 0.9$. The regularization constant ρ for NCA is set to 10^{-5}. We use $N = 200$ sample points for each gradient evaluation and we run 800 iterations of delta-bar-delta, with parameters 0.1 for the linear increase of the learning rate, 0.75 for the averaging constant and 0.5 for the multiplicative decrease.

In the first set of experiments we use an ICP problem with two groups of products, each containing 4 products with correlated demand. When the standard deviation $\sigma = 0$, this amounts to a problem over two state variables. Hence,

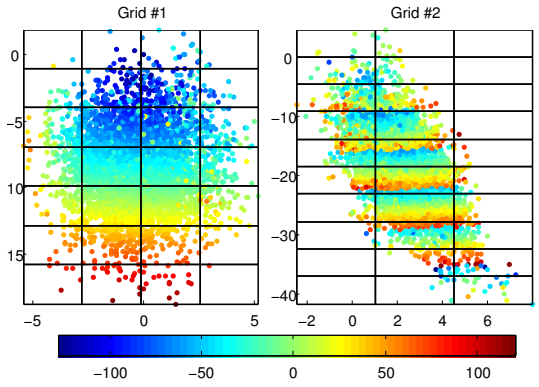

Figure 2. Bellman residual and discretizations obtained for ICP(2,4,0) on the first two iterations of the algorithm projecting to 2 dimensions and using ≤ 30 basis functions. The color indicates the magnitude of the Bellman residual and the cells indicate the discretization.

we can estimate a value function for comparison purposes using a fine discretization, and we can plot it along the two instrinsic dimensions for the purpose of visualizing what our algorithm does. The estimated value function is shown in Figure 1.

Figure 2 shows the projections obtained in the two first steps of the algorithm for this problem. On the first iteration, the goal of minimizing the variance in Bellman error within each group is clearly achieved. Since the intrinsic dimensionality of the problem is only two, essentially the same mapping is obtained on the second iteration. Note that the aspect ratio in the second iteration is $5:1$ versus $2:1$ on the first iteration. This is because NCA stretches the points along the axis of greatest local variations in Bellman errors. The regularization term of (7) prevents all points from being mapped to a line.

Figure 3 shows the successive value-function approximations when using much coarser 6-cell discretizations instead. Note that the shape of the value function mimics the smooth estimate of Figure 1. Roughly equivalent projections up to rotation are obtained on each iteration since there are only two intrinsic dimensions, which can be perfectly identified by a linear mapping. The accuracy improves in each iteration because the successive grids obtained are randomly rotated.

In order to assess numerically the quality of the solution obtained, we plotted the mean absolute Bellman error as a function of the number of basis functions created for two instances of the problems, ICP(2,4,0) and ICP(2,4,2) (which is noisier, since $\sigma = 2$). The results, depicted in Figure 4 are averaged over 20 independent runs. The solid lines represent the results of our algorithm, while the dotted lines are the results when we just use random projections onto a two-dimensional space. We note that random projections can actually be quite useful as a dimensionality reduction technique, and some recent papers (Fradkin &

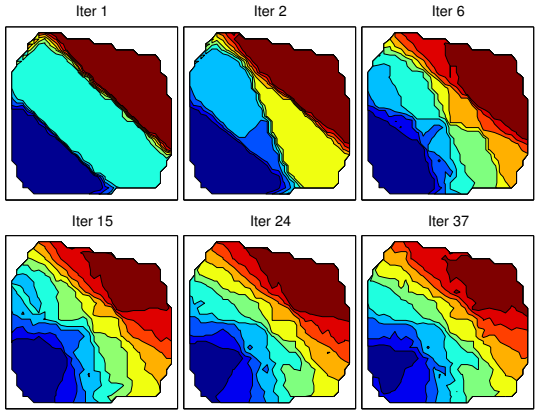

Figure 3. Successive value function approximations for ICP(2,4,0) with only 6 basis functions added per iteration. The estimate approaches the higher-resolution estimate from Figure 1.

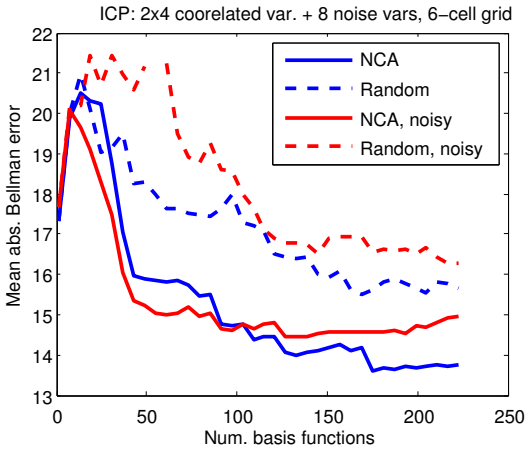

Figure 4. Bellman error after each iteration for ICP(2,4,0) and ICP(2,4,2) with 6-cell discretizations. For the dotted plots, random projections were used in place of NCA projections.

Madigan, 2003) have argued that they achieve good performance both for density estimation and for supervised learning, while being very cheap to compute. We observe indeed that, as the number of basis functions increases, the error obtained using random projections approaches the quality obtained with our algorithm. With few coarse basis functions both approaches are also comparable. However, our method shows an advantage when using a reasonable number of basis functions, regardless of the amount of noise.

In a second set of experiments, we looked at problems with 4 sets of goods, each containing 4 products with correlated demand. The 16 additional random noise variables bring the number of state variables to 32. However, the intrinsic dimensionality of the problem is 4. Figure 5 shows the average Bellman errors for three instances with $\sigma = 0$, 2 and 5 respectively. The dashed lines represent the mean error achieved using a fine discretization of the intrinsic state space on the same problem instances.

Note that in these experiments, we do not project onto the intrinsic dimensionality of the state space; instead, we still project to two dimensions, and use 6-cell or 30-cell uniform discretization to create basis functions. Thus, unlike in Figures 2 and 3, we obtain different projections on successive iterations. This is especially noticeable with the finer 30-cell grids. The first two projections (resulting in 60 basis functions) provide the largest improvement: they are essentially identifying the four intrinsic dimensions, two at a time. After two more projections (120 bases), little further improvement is seen. With the coarser 6-cell grids, the improvement due to each new projection is smaller, but this setting may be preferable if many different projections are needed to identify the underlying structure of the state space (due to the quality of the projections, or the complexity of the state space). We note that our approach is significantly better than random projections throughout. (The error bars are computed over 6 and 39 independent runs for the 6- and 30-cell algorithms respectively.) As expected, the higher the noise, the worse the performance of both algorithms. However, the NCA algorithm still obtains a good solution, whereas the random projections curve shoots off the graph. The degradation of the performance of random projections is due to the fact that the projections are not ignoring the pure noise dimensions, in addition to the suboptimal identification of the "real" dimensions. Note that while the algorithm using NCA projections stops improving after $\sim$ 100 basis functions, the random projection algorithm continues to achieve an improvement. Presumably both algorithms would achieve similar asymtoptic performance, but optimizing the projection significantly reduces the total number of basis functions needed.

7. Related Work

In the last couple of years, much work has been devoted to the topic of constructing basis functions automatically. The methods proposed by Smart (2004) and Ratich & Precup (2004) are essentially heuristics that attempt to exploit the intuition that trajectories taken by the system may lie along a low-dimensional manifold. Mannor et al. (2005) propose to adapt both the basis functions and the parameters vector simultaneously, using either gradient descent or the cross-entropy method. In this case, though, the resulting approximator cannot be considered linear anymore. Ziv & Shimkin (2005) adapt multigrid methods used in solving differential equations, in order to obtain a hierarchy of basis functions at different (but fixed) resolutions. Mahadevan (2005) analyzes the graph of connectivity between states in order to determine a set of basis functions sufficient to represent any value function (called proto-value functions). All of these approaches are quite different in flavor from what we propose here. However, the connection to proto-value functions will be investigated further in the future.

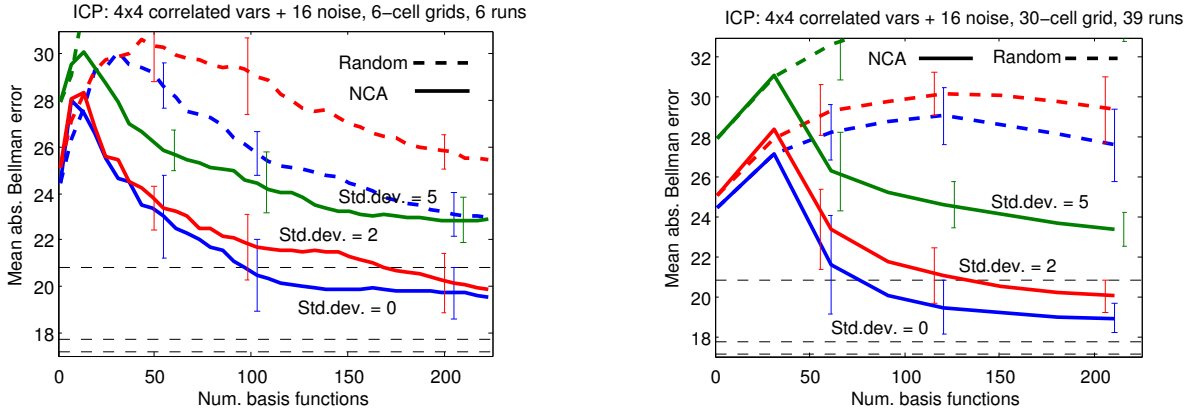

Figure 5. Bellman error after each iteration for ICP(4,4,σ) for different values of σ. Six and thirty basis functions are added for each projection on the left and right respectively. Results are averaged over 6 and 39 independent runs respectively.

8. Conclusions and Future Work

The discussion in Section 3 suggests the possiblity of finding *post hoc* bounds on the Bellman error, i.e., of bounding the Bellman error in terms of the sampled residuals in some norm. This could potentially provide a stopping criterion which ensures a desired accuracy in the value function approximation.

The effectiveness of the algorithm depends on the quality of the projection to the intermediate space, as suggested by the superiority of NCA over random projections, and on the basis functions placed in the low dimensional space. Other methods for finding (possibly non-linear) projections and for automatically placing basis functions in the low-dimensional space should be explored. The experiments show that the algorithm is capable of combining sets of features to identify the underlying dimensions of the problem. This shows promise for its application to problems with more complex structure, even though we are using only linear transformations.

Acknowledgements: We gratefully thank the three anonymous reviewers for their careful reviews. This research was supported in part by NSERC and CFI.

References

Bertsekas, D. (2005). *Dynamic programming and optimal control vol. 1 third edition*. Athena Scientific.

Bertsekas, D., & Castañon, D. (1989). Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE Transactions on Automatic Control*, *34*, 589–598.

Boyan, J. (2002). Technical update: Least-squares temporal difference learning. *Machine Learning*.

Bradtke, S., & Barto, A. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, *22*, 33–57.

Fradkin, D., & Madigan, D. (2003). Experiments with random projections for machine learning. In *Proc. of KDD*.

Goldberger , J., Roweis, S., Hinton, G., & Salakhutdinov, R. (2005). Neighbourhood components analysis. In *NIPS 17*, 513–520.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks*, *1*, 295–307.

Mahadevan, S. (2005). Samuel meets Amarel: Automating value function approximation using global state space analysis. In *Proceedings of AAAI*.

Mannor, S., Menache, I., & Shimkin, N. (2005). Basis function adaptation in temporal difference reinforcement learning. *Annals of Operations Research*, *134*, 215–238.

Munos, R., & Moore, A. (2002). Variable resolution discretization in optimal control. *Machine Learning*, *49*, 291–323.

Ratitch, B., & Precup, D. (2004). Sparse distributed memories for on-line value-based reinforcement learning. In *Proceedings of ECML*.

Smart, W. (2004). Explicit manifold representations for value-functions in reinforcement learning. In *Proceedings of the 8th int. symp. on ai and mathematics*.

Sutton, R. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*, 9–44.

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

Tadic, V. (2001). On the convergence of temporal-difference learning with linear function approximation. *Machine learning*, *42*, 241–267.

Tsitsiklis, J., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, *42*, 674–690.

Ziv, O., & Shimkin, N. (2005). Multigrid algorithms for temporal difference reinforcement learning. In *Proc.icml workshop on rich representations for rl*.