Structured Kernel-Based Reinforcement Learning

Branislav Kveton
Technicolor Labs
Palo Alto, CA
branislav.kveton@technicolor.com

Georgios Theocharous
Adobe
San Jose, CA
theochar@adobe.com

Abstract

Kernel-based reinforcement learning (KBRL) is a popular approach to learning non-parametric value function approximations. In this paper, we present structured KBRL, a paradigm for kernel-based RL that allows for modeling independencies in the transition and reward models of problems. Real-world problems often exhibit this structure and can be solved more efficiently when it is modeled. We make three contributions. First, we motivate our work, define a structured backup operator, and prove that it is a contraction. Second, we show how to evaluate our operator efficiently. Our analysis reveals that the fixed point of the operator is the optimal value function in a special factored MDP. Finally, we evaluate our method on a synthetic problem and compare it to two KBRL baselines. In most experiments, we learn better policies than the baselines from an order of magnitude less training data.

Introduction

Markov decision processes (MDPs) (Puterman 1994) are an established framework for sequential decision making under uncertainty. If a decision problem is Markovian, has a small number of states, and its model is known, it can be typically easily solved as an MDP. In practice, however, problems are large and their models are often unavailable. Such problems are typically hard to solve and have been studied in the field of reinforcement learning (RL) (Sutton and Barto 1998) for the past 30 years.

A popular approach to non-parametric RL is kernel-based RL (KBRL) (Ormoneit and Sen 2002). In kernel-based RL, the Bellman operator is approximated by a backup operator on a sample of the problem. The fixed point of this operator can be found by value iteration. Each step of value iteration takes $\theta(n^2)$ time, where n is the sample size. Hence, KBRL is not practical when the sample is large. Recently, Barreto *et al.* (2011), and Kveton and Theocharous (2012), proposed quantization of the KBRL operator on representative states. The time complexity of both methods is $O(n)$ and they perform well when the intrinsic dimension of the state space is small.

The state space of large real-world problems rarely lies on a low-dimensional manifold and structural assumptions may be necessary to solve such problems. In this paper, we show how to incorporate one specific assumption, *independencies* in the transition and reward models of solved problems, into KBRL. We make three contributions. First, we motivate our problem, introduce a structured KBRL backup operator, and prove that it is a contraction. Second, we show how to apply our backup operator computationally efficiently. Our analysis reveals that structured KBRL is equivalent to computing the optimal value function in a special factored MDP. To the best of our knowledge, this is the first such result in the context of KBRL and structured problems. Finally, we evaluate our approach on a synthetic problem and compare it to two KBRL baselines. Typically, we learn better policies than the baselines from an order of magnitude less training data.

In the rest of this paper, we assume that the state space is discrete, finite, and metric. The distance function is $d(\cdot, \cdot)$.

Background

Markov decision processes (MDPs) are a popular model for discrete-time stochastic control problems (Puterman 1994). Formally, an MDP is a tuple $\mathcal{M} = (S, \mathcal{A}, P, R)$, where S is a set of states, $\mathcal{A}$ is a set of actions, $P(s' \mid s, a)$ is a transition model that describes the dynamics of the MDP, and $R(s, a)$ is a reward model that assigns rewards to state-action pairs. An MDP *policy* is a mapping $\pi : S \to \mathcal{A}$. The quality of the policy is usually measured by the *infinite horizon discounted reward* $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where $\gamma \in (0, 1)$ is a *discount factor* and r_t is the immediate reward at time t. The *optimal policy* π^* can be computed from the *optimal value function*, which is the fixed point of the Bellman equation (Bellman 1957):

$$V^*(s) = \max_a \left[R(s, a) + \gamma \sum_{s'} P(s' \mid s, a) V^*(s') \right]. \quad (1)$$

The function V^* can be computed by value iteration, policy iteration, and linear programming (Puterman 1994).

Real-world problems often involve structure and therefore can be represented compactly. In this paper, we focus on the compact representation known as factored MDPs (Boutilier, Dearden, and Goldszmidt 1995). Formally, a *factored MDP* is a 4-tuple $\mathcal{M} = (\mathbf{X}, \mathcal{A}, P, R)$, where $\mathbf{X} = X_1 \times \cdots \times X_K$ is a state space factored into K state variables and $\mathcal{A}$ is a set of actions. For simplicity, we assume that all state variables X_k are binary. The *transition model* is factored as:

$$P(\mathbf{x}' \mid \mathbf{x}, a) = \prod_{k=1}^{K} P(\mathbf{x}'[k] \mid \mathbf{x}[\mathsf{pa}_k], a), \quad (2)$$

where $\mathbf{x}'[k] \in \{0, 1\}$ is the new state of the variable X_k, the *parent set* pa_k are the indices of the variables that affect this state, and $\mathbf{x}[\mathsf{pa}_k] \in \{0, 1\}^{|\mathsf{pa}_k|}$ is the state of these variables.

The parent set is typically small. We assume that the *reward function* is factored on the parent sets pa_k as:

$$R(\mathbf{x}, a) = \sum_{k=1}^{K} R_k(\mathbf{x}[\mathsf{pa}_k], a). \tag{3}$$

Similarly to MDPs, the optimal policy π^* in factored MDPs can be computed from the optimal value function V^*, which is the fixed point of the Bellman equation:

$$V^*(\mathbf{x}) = \max_a \left[R(\mathbf{x}, a) + \gamma \sum_{\mathbf{x}'} P(\mathbf{x}' \mid \mathbf{x}, a) V^*(\mathbf{x}') \right]. \tag{4}$$

In general, it is necessary to make structural assumptions on V^* to find it efficiently (Koller and Parr 1999; Guestrin et al. 2003; Kveton, Hauskrecht, and Guestrin 2006).

When the model is not available, the problem of learning the optimal policy is known as *reinforcement learning (RL)*. Sutton and Barto (1998) provide a comprehensive overview of existing RL algorithms. All of these methods learn from a sample of interactions with the environment. In this work, the *sample* is a set of n 4-tuples $\{(\mathbf{x}_t, a_t, \mathbf{r}_t, \mathbf{x}'_t)\}_{t=1}^n$, where $\mathbf{x}_t \in \{0,1\}^K$, a_t, $\mathbf{r}_t \in \mathbb{R}^K$, and $\mathbf{x}'_t \in \{0,1\}^K$ are the state, action, reward, and next state at time t, respectively. At time t, the next state of the variable X_k is $\mathbf{x}'_t[k] \in \{0,1\}$ and the state of the corresponding parent set is $\mathbf{x}_t[\mathsf{pa}_k] \in \{0,1\}^{|\mathsf{pa}_k|}$. The reward of the k-th reward function (Equation 3) at time t is $\mathbf{r}_t[k]$. When the reward model is not factored, we write r_t instead of $\mathbf{r}_t$.

Related work

Kernel-based RL (KBRL) (Ormoneit and Sen 2002) is a popular RL approach. In kernel-based RL, the Bellman operator is approximated by an operator on a sample:

$$\mathcal{T}_\lambda V(\mathbf{x}) = \max_a \sum_t \lambda^a(\mathbf{x}_t, \mathbf{x}) \left[r_t + \gamma V(\mathbf{x}'_t) \right]. \tag{5}$$

The kernel $\lambda^a(\mathbf{x}_t, \mathbf{x})$ is a function that measures the similarity of the states $\mathbf{x}$ and $\mathbf{x}_t$. This function must be normalized such that $\sum_t \lambda^a(\mathbf{x}_t, \mathbf{x}) = 1$ and be zero in all examples that are inconsistent with the action a, $a_t \neq a$.

KBRL has many favorable properties (Ormoneit and Sen 2002). First, the operator $\mathcal{T}_\lambda$ (Equation 5) has a unique fixed point. Second, the fixed point converges to the optimal value function V^* for the *Gaussian kernel*:

$$\lambda^a(\mathbf{x}_t, \mathbf{x}) \propto \exp\left[-\frac{d^2(\mathbf{x}_t, \mathbf{x})}{2\sigma^2} \right] \mathbb{1}\{a_t = a\} \tag{6}$$

when $n \to \infty$ and $\sigma \to 0$. Finally, note that the operator $\mathcal{T}_\lambda$ (Equation 5) depends on the value function V in n states $\mathbf{x}'_t$. Therefore, the backup of V by $\mathcal{T}_\lambda$ can be computed in $\theta(n^2)$ time because V needs to be updated in only n states and the cost of each update is $\theta(n)$.

The time complexity of kernel-based RL is $\Omega(n^2)$. Therefore, KBRL is not practical when the sample size n is large. This scalability issue has been addressed by several authors. Jong and Stone (2006) showed how to get a more informative sample of the problem using prioritized sweeping. Fitted Q iteration (FQI) (Ernst, Geurts, and Wehenkel 2005) is the first practical and general KBRL algorithm. The method is a variant of Q iteration, in which the Q function is approximated by a non-parametric regressor. The quality and computation time of FQI solutions depend on the chosen regressor. Recently, Kveton and Theocharous (2012), and Barreto *et al.* (2011), proposed quantization of the operator $\mathcal{T}_\lambda$ on k representative states. Both of these approaches perform well when the intrinsic dimension of the state space is small.

Structured KBRL can be viewed as aggregating synthetic examples based on structural assumptions on observed data. The idea of synthesizing a larger sample is similar to Fonteneau *et al.* (2013). The difference in our work is that we do not explicitly create new trajectories. Instead, we aggregate all of them, exponentially many in K, in a computationally efficient manner.

Structured kernel-based RL

Before we proceed to structured KBRL, we motivate it by a simple illustrative example. Our example is a Markov chain with two binary state variables, X_1 and X_2:

$$\begin{aligned} P(\mathbf{x}' \mid \mathbf{x}, a) &= \mathbb{1}\{\mathbf{x}[1] \neq \mathbf{x}'[1]\}\, \mathbb{1}\{\mathbf{x}[2] \neq \mathbf{x}'[2]\} \\ R(\mathbf{x}, a) &= 9\mathbf{x}[1] + \mathbf{x}[2]. \end{aligned} \tag{7}$$

The states of these variables alternate between 0 and 1, and change independently of each other. The reward function is additive; and the rewards for $X_1 = 1$ and $X_2 = 1$ are 9 and 1, respectively.

Our problem is a deterministic Markov chain. Therefore, the value of a state can be computed as a sum of discounted rewards in a single trajectory that starts in that state:

$$\begin{aligned} V^*((0,0)) &= \frac{10\gamma}{1-\gamma^2}, \quad V^*((0,1)) = \frac{1+9\gamma}{1-\gamma^2}, \\ V^*((1,0)) &= \frac{9+\gamma}{1-\gamma^2}, \quad V^*((1,1)) = \frac{10}{1-\gamma^2}. \end{aligned} \tag{8}$$

Suppose that we observe a sample $\{(\mathbf{x}_t, a_t, \mathbf{r}_t, \mathbf{x}'_t)\}_{t=1}^2$:

$$\begin{aligned} (\mathbf{x}_1, a_1, \mathbf{r}_1, \mathbf{x}'_1) &= ((0,1), a, (0,1), (1,0)) \\ (\mathbf{x}_2, a_2, \mathbf{r}_2, \mathbf{x}'_2) &= ((1,0), a, (9,0), (0,1)) \end{aligned} \tag{9}$$

and use KBRL to estimate V^*. Let V^*_λ be the fixed point of Equation 5 and the similarity of the states be measured by a nearest-neighbor kernel:

$$\lambda^a(\mathbf{x}_t, \mathbf{x}) \propto \mathbb{1}\left\{ d(\mathbf{x}_t, \mathbf{x}) = \min_i d(\mathbf{x}_i, \mathbf{x}) \right\}. \tag{10}$$

Then the value function V^*_λ is equal to V^* in both observed states, $(0,1)$ and $(1,0)$, but the unobserved states, $(0,0)$ and $(1,1)$, are mispredicted as:

$$\frac{1}{2} V^*_\lambda((0,1)) + \frac{1}{2} V^*_\lambda((1,0)) = \frac{5+5\gamma}{1-\gamma^2}. \tag{11}$$

A better estimator could be learned from more training data. Suppose that we cannot get a larger sample but we know the structure of the problem. Then we could synthetize a larger sample based on independence assumptions in its model. In our problem (Equation 7), the state variables X_1 and X_2 are independent, and so are their rewards. As a result, we could

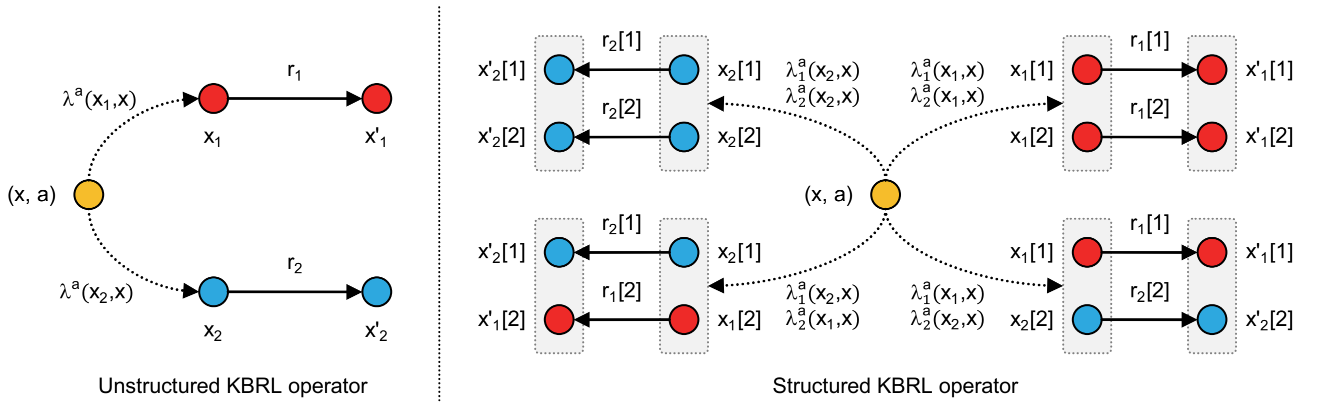

Figure 1: An illustration of unstructured (Equation 5) and structured (Equation 13) KBRL operators on our motivating example (Equation 7). The red and blue colors mark estimates corresponding to the first and second examples (Equation 9), respectively. The dotted and solid arrows mark stochastic and deterministic transitions, respectively. The kernel λ determines the probability of the stochastic transitions.

exchange the state and reward factors in the existing sample, and generate two new examples:

$$\begin{aligned}(\mathbf{x}_3, a_3, \mathbf{r}_3, \mathbf{x}'_3) &= ((0,0), a, (0,0), (1,1)) \\ (\mathbf{x}_4, a_4, \mathbf{r}_4, \mathbf{x}'_4) &= ((1,1), a, (9,1), (0,0)).\end{aligned} \tag{12}$$

The KBRL solution on the new sample $\{(\mathbf{x}_t, a_t, \mathbf{r}_t, \mathbf{x}'_t)\}_{t=1}^4$ is the optimal value function V^*.

Our example illustrates how *variable independence* helps in learning better value functions. In the rest of this section, we generalize our ideas to multiple state variables and more elaborate models.

Structured backup operator

Our solution is motivated by the following observation. The KBRL backup operator (Equation 5) can be viewed as being generated by a stochastic process. Given a state $\mathbf{x}$ and action a, a new state $\mathbf{x}_t$ is chosen proportionally to its similarity to $\mathbf{x}$, yields the reward of r_t, and then transitions into the next state $\mathbf{x}'_t$ (Jong and Stone 2006). Suppose that the next state of each state variable X_k depends only on its parent set pa_k. Then the following stochastic process would be appropriate for modeling this structure. For each variable X_k, a substate $\mathbf{x}_{t_k}[\mathsf{pa}_k]$ is generated based on its similarity to $\mathbf{x}[\mathsf{pa}_k]$, independently of all other substates; yields the reward of $\mathbf{r}_{t_k}[k]$; and then transitions into the next state $\mathbf{x}'_{t_k}[k]$.

We refer to the corresponding backup operator as a *structured backup operator*:

$$\bar{\mathcal{T}}_\lambda V(\mathbf{x}) = \max_a \sum_{t_1,\dots,t_K} \left(\prod_{k=1}^K \lambda_k^a(\mathbf{x}_{t_k}, \mathbf{x})\right) \left[\sum_{k=1}^K \mathbf{r}_{t_k}[k] + \gamma V(\tau(t_1,\dots,t_K))\right]. \tag{13}$$

The sum $\sum_{t_1,\dots,t_K}$ is over K indices t_k, where t_k is the index of the example corresponding to the k-th state variable X_k. All indices t_k take values from 1 to n. The *structured kernel*:

$$\prod_{k=1}^K \lambda_k^a(\mathbf{x}_{t_k}, \mathbf{x}) \tag{14}$$

is a product of K *kernel factors* $\lambda_k^a(\mathbf{x}_{t_k}, \mathbf{x})$. We assume that each kernel factor is a Gaussian kernel:

$$\lambda_k^a(\mathbf{x}_t, \mathbf{x}) \propto \exp\left[-\frac{d^2(\mathbf{x}_t[\mathsf{pa}_k], \mathbf{x}[\mathsf{pa}_k])}{2\sigma^2}\right] \mathbb{1}\{a_t = a\} \tag{15}$$

that is normalized such that $\sum_t \lambda_k^a(\mathbf{x}_t, \mathbf{x}) = 1$ for all states $\mathbf{x}$ and actions a. Note that in general, the kernel can be any function that measures the similarity of the substates $\mathbf{x}[\mathsf{pa}_k]$ and $\mathbf{x}_t[\mathsf{pa}_k]$. This function must be properly normalized and be zero in all examples that are inconsistent with the action a, $a_t \neq a$.

The *next state* $\tau(t_1,\dots,t_K) \in \{0,1\}^K$ is a vector whose k-th entry is the next state of the variable X_k at time t_k:

$$\tau(t_1,\dots,t_K)[k] = \mathbf{x}'_{t_k}[k] \quad \forall k \in \{1,\dots,K\}\,. \tag{16}$$

The *structured reward*:

$$\sum_{k=1}^K \mathbf{r}_{t_k}[k] \tag{17}$$

is a sum of K *reward factors* $\mathbf{r}_{t_k}[k]$. The k-th reward factor is the reward on the parent set of the variable X_k at time t_k. Our approach is illustrated on a simple example in Figure 1.

In the next section, we analyze the structured backup operator $\bar{\mathcal{T}}_\lambda$. In particular, we show that it is a contraction and identify its fixed point. Moreover, we show how to compute the backup $\bar{\mathcal{T}}_\lambda V(\mathbf{x})$ efficiently. Naively, it can be computed by summing all $\theta(n^K)$ terms in Equation 13. This is clearly infeasible even for relatively small n and K.

Theoretical analysis

First, we show that the operator $\bar{\mathcal{T}}_\lambda$ is a contraction mapping. Therefore, it has a unique fixed point.

Proposition 1. *The operator $\bar{\mathcal{T}}_\lambda$ is a contraction mapping.*

Proof: Let V and U be value functions on the state space $\mathbf{X}$. Then:

$$\begin{aligned}&\left\|\bar{\mathcal{T}}_\lambda V - \bar{\mathcal{T}}_\lambda U\right\|_\infty \\ &\quad\leq \gamma \max_{\mathbf{x},a} \sum_{t_1,\dots,t_K} \left(\prod_{k=1}^K \lambda_k^a(\mathbf{x}_{t_k}, \mathbf{x})\right) \\ &\qquad\qquad |V(\tau(t_1,\dots,t_K)) - U(\tau(t_1,\dots,t_K))| \\ &\quad\leq \gamma \left\|V - U\right\|_\infty . \end{aligned} \tag{18}$$

The first inequality follows from the definition of the operator $\bar{\mathcal{T}}_\lambda$ (Equation 13), an upper bound:

$$\left|\max_a f(a) - \max_a g(a)\right| \leq \max_a |f(a) - g(a)| \tag{19}$$

that holds for any two f and g, and the fact that each kernel factor $\lambda_k^a(\mathbf{x}_{t_k}, \mathbf{x})$ is non-negative for all inputs. The second inequality is due to $\sum_{t_1,\dots,t_K} \left(\prod_{k=1}^K \lambda_k^a(\mathbf{x}_{t_k}, \mathbf{x})\right) = 1$ for all $\mathbf{x}$ and a. This concludes our proof. ■

Second, we show how to rewrite the operator $\bar{\mathcal{T}}_\lambda$ compactly. As a result, the backup $\bar{\mathcal{T}}_\lambda V(\mathbf{x})$ can be computed more efficiently than by summing over $\theta(n^K)$ terms.

Proposition 2. *The operator $\bar{\mathcal{T}}_\lambda$ can be rewritten as:*

$$\bar{\mathcal{T}}_\lambda V(\mathbf{x}) = \max_a \left[\hat{r}(\mathbf{x}, a) + \gamma \textstyle\sum_{\mathbf{y}} \hat{p}(\mathbf{y} \mid \mathbf{x}, a) V(\mathbf{y})\right],$$

where $\mathbf{y}$ is a state from the set of perturbed next states $\mathbf{Y} \subseteq \mathbf{X}$.[1] *The* reward estimator $\hat{r}(\mathbf{x}, a)$ *is factored as:*

$$\hat{r}(\mathbf{x}, a) = \textstyle\sum_{k=1}^K \hat{r}_k(\mathbf{x}, a), \tag{20}$$

where $\hat{r}_k(\mathbf{x}, a)$ is estimated on a subspace $\mathbf{X}[\mathsf{pa}_k]$ as:

$$\hat{r}_k(\mathbf{x}, a) = \textstyle\sum_t \lambda_k^a(\mathbf{x}_t, \mathbf{x}) \mathbf{r}_t[k]. \tag{21}$$

The transition estimator $\hat{p}(\mathbf{y} \mid \mathbf{x}, a)$ *is factored as:*

$$\hat{p}(\mathbf{y} \mid \mathbf{x}, a) = \textstyle\prod_{k=1}^K \hat{p}_k(\mathbf{y}[k] \mid \mathbf{x}, a), \tag{22}$$

where $\hat{p}_k(\mathbf{y}[k] \mid \mathbf{x}, a)$ is estimated on a subspace $\mathbf{X}[\mathsf{pa}_k]$ as:

$$\hat{p}_k(y \mid \mathbf{x}, a) = \textstyle\sum_t \lambda_k^a(\mathbf{x}_t, \mathbf{x}) \mathbb{1}\{\mathbf{x}'_t[k] = y\}. \tag{23}$$

Proof: The product of the first reward factor $\mathbf{r}_{t_1}[1]$ with the kernel can be written as:

$$\begin{aligned}&\sum_{t_1,\dots,t_K} \left(\prod_{k=1}^K \lambda_k^a(\mathbf{x}_{t_k}, \mathbf{x})\right) \mathbf{r}_{t_1}[1] \\ &\quad= \sum_{t_1,\dots,t_K} \left(\prod_{k=2}^K \lambda_k^a(\mathbf{x}_{t_k}, \mathbf{x})\right) (\lambda_1^a(\mathbf{x}_{t_1}, \mathbf{x}) \mathbf{r}_{t_1}[1]) \\ &\quad= \underbrace{\left(\prod_{k=2}^K \sum_{t_k} \lambda_k^a(\mathbf{x}_{t_k}, \mathbf{x})\right)}_{1} \left(\sum_{t_1} \lambda_1^a(\mathbf{x}_{t_1}, \mathbf{x}) \mathbf{r}_{t_1}[1]\right) \\ &\quad= \sum_{t_1} \lambda_1^a(\mathbf{x}_{t_1}, \mathbf{x}) \mathbf{r}_{t_1}[1]. \end{aligned} \tag{24}$$

[1]We explicitly distinguish between $\mathbf{X}$ and $\mathbf{Y}$ because the set $\mathbf{Y}$ can be much smaller, depending on the sample and kernel.

The sum of products can be replaced by the product of sums because each term $\lambda_k^a(\mathbf{x}_{t_k}, \mathbf{x})$ and $\lambda_1^a(\mathbf{x}_{t_1}, \mathbf{x}) \mathbf{r}_{t_1}[1]$ depends only on one summation index. Similarly to Equation 24, the product of all reward factors $\mathbf{r}_{t_k}[k]$ with the kernel is:

$$\begin{aligned}&\sum_{t_1,\dots,t_K} \left(\prod_{k=1}^K \lambda_k^a(\mathbf{x}_{t_k}, \mathbf{x})\right) \left[\sum_{k=1}^K \mathbf{r}_{t_k}[k]\right] \\ &\quad= \sum_{k=1}^K \underbrace{\sum_{t_k} \lambda_k^a(\mathbf{x}_{t_k}, \mathbf{x}) \mathbf{r}_{t_k}[k]}_{\hat{r}_k(\mathbf{x}, a)} \\ &\quad= \hat{r}(\mathbf{x}, a). \end{aligned} \tag{25}$$

The product of the value function V with the kernel is:

$$\begin{aligned}&\sum_{t_1,\dots,t_K} \left(\prod_{k=1}^K \lambda_k^a(\mathbf{x}_{t_k}, \mathbf{x})\right) \gamma V(\tau(t_1,\dots,t_K)) \\ &\quad= \gamma \sum_{\mathbf{y}} V(\mathbf{y}) \\ &\qquad \sum_{t_1,\dots,t_K} \left(\prod_{k=1}^K \lambda_k^a(\mathbf{x}_{t_k}, \mathbf{x})\right) \mathbb{1}\{\tau(t_1,\dots,t_K) = \mathbf{y}\} \\ &\quad= \gamma \sum_{\mathbf{y}} V(\mathbf{y}) \sum_{t_1,\dots,t_K} \prod_{k=1}^K \lambda_k^a(\mathbf{x}_{t_k}, \mathbf{x}) \mathbb{1}\left\{\mathbf{x}'_{t_k}[k] = \mathbf{y}[k]\right\} \\ &\quad= \gamma \sum_{\mathbf{y}} V(\mathbf{y}) \prod_{k=1}^K \underbrace{\sum_{t_k} \lambda_k^a(\mathbf{x}_{t_k}, \mathbf{x}) \mathbb{1}\left\{\mathbf{x}'_{t_k}[k] = \mathbf{y}[k]\right\}}_{\hat{p}_k(\mathbf{y}[k] \mid \mathbf{x}, a)} \\ &\quad= \gamma \sum_{\mathbf{y}} \hat{p}(\mathbf{y} \mid \mathbf{x}, a) V(\mathbf{y}). \end{aligned} \tag{26}$$

In the first equality, we introduce a variable $\mathbf{y}$, which allows us to count all possible next states $\tau(t_1,\dots,t_K)$ efficiently. The sum of products can be replaced by the product of sums because all terms $\lambda_k^a(\mathbf{x}_{t_k}, \mathbf{x}) \mathbb{1}\left\{\mathbf{x}'_{t_k}[k] = \mathbf{y}[k]\right\}$ depend only on one summation index t_k.

Our main claim follows directly from substituting Equations 25 and 26 into Equation 13. ■

Proposition 2 suggests the following procedure for computing the backup $\bar{\mathcal{T}}_\lambda V(\mathbf{x})$. First, compute the *local* estimators $\hat{p}_k(y \mid \mathbf{x}, a)$ and $\hat{r}_k(\mathbf{x}, a)$, for all k, y, substates $\mathbf{x}[\mathsf{pa}_k]$, and actions a. Let the number of states in each subspace $\mathbf{X}[\mathsf{pa}_k]$ be $O(1)$. Then these estimators can be computed in $O(Kn)$ time. Second, compute the *global* estimators $\hat{p}(\mathbf{y} \mid \mathbf{x}, a)$ and $\hat{r}(\mathbf{x}, a)$ by aggregating the local ones, for all $\mathbf{y}$ and a. Since $\mathbf{y} \in \mathbf{X}$, the global estimators can be computed in $O(K\,|\mathbf{X}|)$ time. So overall, the time complexity of computing $\bar{\mathcal{T}}_\lambda V(\mathbf{x})$ is $O(K(|\mathbf{X}| + n))$. Note that $|\mathbf{X}| = O(2^K)$. Therefore, our new method is about $n^K / 2^K \approx (n/2)^K$ more efficient than summing over all terms in Equation 13, exponentially faster in K for all $n > 2$.

Proposition 2 also points out a novel connection between model-based and kernel-based RL. In particular, it says that the fixed point of the KBRL operator $\bar{\mathcal{T}}_\lambda$ (Equation 13) corresponds to the optimal value function in a factored Markov

Algorithm 1 `structKBRL`: Structured kernel-based RL.

```
Input:
    Sample $\{(\mathbf{x}_t, a_t, \mathbf{r}_t, \mathbf{x}'_t)\}_{t=1}^n$
    Approximation error $\varepsilon$

Estimate $\hat{r}(\mathbf{x}, a)$ and $\hat{p}(\mathbf{y} \mid \mathbf{x}, a)$ (Equations 20 and 22)
Initialize the value function $V^{(0)}$
$i \leftarrow 0$
repeat
    for all $\mathbf{x} \in \mathbf{X}$ do
        $V^{(i+1)}(\mathbf{x}) \leftarrow \bar{\mathcal{T}}_\lambda V^{(i)}(\mathbf{x})$ (Proposition 2)
    end for
    $i \leftarrow i + 1$
until $\left\| V^{(i)} - V^{(i-1)} \right\|_\infty < \varepsilon$

Output:
    Value function $V^{(i)}$
```

decision process $\mathcal{M}_\lambda = (\mathbf{X}, \mathcal{A}, \hat{p}, \hat{r})$. This is the first result of this kind for structured problems. Jong and Stone (2006) pointed out a similar connection in unstructured problems.

In reinforcement learning, the model of the problem is not available and model-based RL is often criticized for building it (Sutton and Barto 1998). Indeed, many RL algorithms can learn the optimal value function without building the model. Proposition 2 clearly says that learning of the optimal value function in a factored MDP, whose parameters are estimated using kernels, is equivalent to backups on the sample, where the model is not explicitly present. So in this setting, model-based and model-free RL are equivalent.

Algorithm

Based on Proposition 2, we propose the following algorithm. First, we collect a sample $\{(\mathbf{x}_t, a_t, \mathbf{r}_t, \mathbf{x}'_t)\}_{t=1}^n$. Second, we precompute the terms $\hat{r}(\mathbf{x}, a)$ and $\hat{p}(\mathbf{y} \mid \mathbf{x}, a)$ (Equations 20 and 22), for all $\mathbf{x}$, $\mathbf{y}$, and a. Finally, we iteratively apply the backup operator $\bar{\mathcal{T}}_\lambda$ until consecutive backups differ by less than ε. Our approach is outlined in Algorithm 1. We refer to it as Algorithm `structKBRL`.

The sample can be generated in $O(n)$ time. The statistics $\hat{r}(\mathbf{x}, a)$ and $\hat{p}(\mathbf{y} \mid \mathbf{x}, a)$ can be precomputed in $O(K(|\mathbf{X}|^2 + n))$ time, as discussed below Proposition 2. Finally, each $\bar{\mathcal{T}}_\lambda$ backup takes $O(|\mathbf{X}|^2)$ time. Therefore, the time complexity of Algorithm `structKBRL` is $O(K(|\mathbf{X}|^2 + n))$, linear in the sample size n and quadratic in the number of states $|\mathbf{X}|$.

Experiments

We perform two experiments. First, we compare Algorithm `structKBRL` to two KBRL baselines. Second, we study the sensitivity of our algorithm to the correctness of the model.

Network administration problem

All our experiments are performed on the network administration problem (Guestrin, Koller, and Parr 2001). The problem comprises a network of computers that crash randomly. When a computer crashes, the probability that its neighbors in the network crash increases. The network is controlled by a network administrator who can reboot crashed computers. This prevents spreading of their failures to other computers.

We study the ring network topology with K computers.[2] The state of each computer is modeled by a binary variable, where $X_k = 0$ and $X_k = 1$ mean that the k-th computer has crashed and works properly, respectively. The administrator can take K actions. The k-th action is rebooting of the k-th computer. The state of the variable X_k evolves according to the following model:

$$\begin{aligned} P(X'_k = 0 \mid a = k) &= 0 \\ P(X'_k = 0 \mid X_k = 0, a \neq k) &= 0.95 \\ P(X'_k = 0 \mid X_k = 1, X_{k \ominus 1} = 0, a \neq k) &= 0.33 \\ P(X'_k = 0 \mid X_k = 1, X_{k \ominus 1} = 1, a \neq k) &= 1/K, \end{aligned} \tag{27}$$

where $k \ominus \delta = ((k - \delta - 1) \bmod K) + 1$ and the parent set is $\mathsf{pa}_k = \{k, k \ominus 1\}$. The reward function is factored as:

$$R(\mathbf{x}, a) = 3\mathbf{x}[1] + \textstyle\sum_{k=2}^K \mathbf{x}[k]. \tag{28}$$

The discount factor γ is 0.9. Each problem is solved for 100 steps and its starting state is chosen at random. The number of training episodes is varied from one to one thousand. All policies are evaluated on 300 test episodes and their quality is measured by the expected cumulative discounted reward. All results are averaged over 20 randomly initialized runs.

State-of-the-art solutions

In the first experiment, we compare Algorithm `structKBRL` to two state-of-the-art KBRL baselines, `FQI` (Ernst, Geurts, and Wehenkel 2005) and kernel-based RL on representative states (Kveton and Theocharous 2012), which we refer to as `repKBRL`. We vary the number of training episodes from one to one thousand and observe how many episodes are needed to learn good policies. In `FQI`, we use 50 totally randomized trees and the minimum number of examples in the leaves of these trees is set to 16. The representative states in `repKBRL` are all 2^K possible network states. The heat parameter is set as $\sigma = 1/3$. Our results are shown in Figure 2. We observe two major trends.

First, Algorithm `structKBRL` learns nearly optimal policies as the number of training episodes increases. The quality of the policies improves as the sample size increases. In all problems, we learn 95% optimal policies from 10 or less training episodes.

Second, Algorithm `structKBRL` is more computationally and sample efficient than `FQI` and `repKBRL`. In most cases, we learn better solutions than the baselines from an order of magnitude smaller sample. The reason is that we model the structure of the problem. Algorithm `structKBRL` is usually significantly faster than `repKBRL`. Both methods are sample averagers and their computation times increase linearly with the sample size. However, each method averages the sample a different number of times, $\theta(K)$ and $\theta(2^K)$, and this is the reason for the observed speedup.

[2]We also experimented with other network topologies, such as the ring-of-rings and star. The trends on these problems are similar to those reported in this paper.

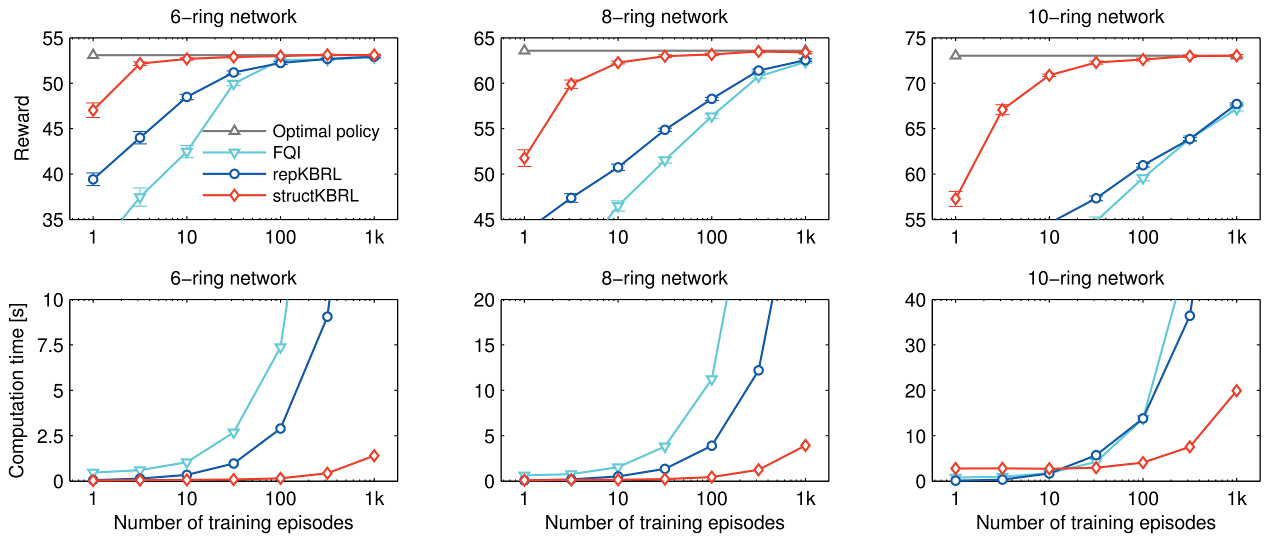

Figure 2: Comparison of three KBRL methods on three K-ring network administration problems. For each solution, we report the cumulative discounted reward and computation time.

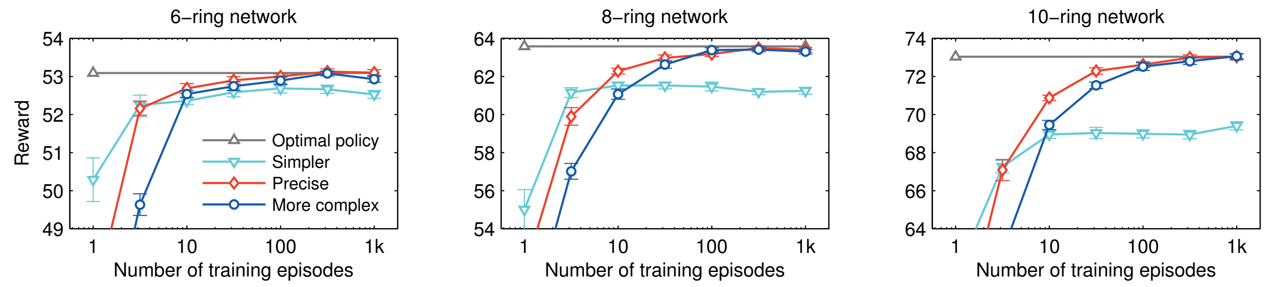

Figure 3: Comparison of three variants of Algorithm `structKBRL`; when the model of the world is accurate, simpler, and more complex; on three K-ring network administration problems. For each solution, we report the cumulative discounted reward.

Sensitivity analysis

In the second experiment, we evaluate the sensitivity of our approach to the correctness of the model. We compare three variants of Algorithm `structKBRL`. The first variant knows the underlying model $\mathsf{pa}_k = \{k, k \ominus 1\}$. The second variant has a *simpler* model, $\mathsf{pa}_k = \{k\}$, and the third variant has a *more complex* model, $\mathsf{pa}_k = \{k, k \ominus 1, k \ominus 2\}$. Our results are reported in Figure 3. We observe two major trends.

When the model is simpler, our algorithm initially learns better policies but fails to converge to the optimal solutions. Nevertheless, note that we learn 90% optimal policies in all problems. When the model is more complex, our algorithm initially learns worse policies but at the end converges to the optimal solutions.

Conclusions

In this paper, we propose structured KBRL, a reinforcement learning paradigm that combines structural assumptions and kernelized value function approximations. We introduce this concept, define a new structured backup operator, analyze it, and show how to apply it computationally efficiently. Based on our analysis, we propose a novel algorithm `structKBRL`. We evaluate the algorithm on a synthetic problem and show that it performs better than state-of-the-art KBRL baselines. Our paper makes first steps in a new direction and naturally we leave many questions open.

First, our experiments show that when a problem is structured, structured KBRL learns nearly optimal solutions from an order of magnitude smaller sample than KBRL. It is only natural to ask what is the convergence rate of our algorithm. KBRL convergences at the rate of $O(n^{-\frac{1}{2(K+2)}})$ (Ormoneit and Sen 2002), where K is the number of dimensions in the state space $\mathbf{X}$. Let Δ be the largest of the dimensions of the subspaces $\mathbf{X}[\mathsf{pa}_k]$. Then we expect Algorithm `structKBRL` to converge at the rate of $O(n^{-\frac{1}{2(\Delta+2)}})$. This claim needs to be proved.

Second, we assume that the state space $\mathbf{X}$ is discrete. It is not obvious how to extend our results to continuous spaces.

Third, the time complexity of our method is $O(|\mathbf{X}|^2)$ and therefore it is not yet suitable for solving large problems. So more approximations are necessary. One seemingly suitable approximation is replacing the expectation over all possible next states $\mathbf{y}$ by m representative states. This would reduce the time complexity of our method to $O(K(m^2 + n))$. The quality of this approximation needs to be properly analyzed.

References

Barreto, A.; Precup, D.; and Pineau, J. 2011. Reinforcement learning using kernel-based stochastic factorization. In *Advances in Neural Information Processing Systems 24*, 720–728.

Bellman, R. 1957. *Dynamic Programming*. Princeton, NJ: Princeton University Press.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 1995. Exploiting structure in policy construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1104–1111.

Ernst, D.; Geurts, P.; and Wehenkel, L. 2005. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research* 6:503–556.

Fonteneau, R.; Murphy, S.; Wehenkel, L.; and Ernst, D. 2013. Batch mode reinforcement learning based on the synthesis of artificial trajectories. *Annals of Operations Research*.

Guestrin, C.; Koller, D.; Parr, R.; and Venkataraman, S. 2003. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research* 19:399–468.

Guestrin, C.; Koller, D.; and Parr, R. 2001. Max-norm projections for factored MDPs. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, 673–682.

Jong, N., and Stone, P. 2006. Kernel-based models for reinforcement learning. In *ICML 2006 Workshop on Kernel Methods and Reinforcement Learning*.

Koller, D., and Parr, R. 1999. Computing factored value functions for policies in structured MDPs. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 1332–1339.

Kveton, B., and Theocharous, G. 2012. Kernel-based reinforcement learning on representative states. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, 977–983.

Kveton, B.; Hauskrecht, M.; and Guestrin, C. 2006. Solving factored MDPs with hybrid state and action variables. *Journal of Artificial Intelligence Research* 27:153–201.

Ormoneit, D., and Sen, S. 2002. Kernel-based reinforcement learning. *Machine Learning* 49:161–178.

Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY: John Wiley & Sons.

Sutton, R., and Barto, A. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.