Truncated Approximate Dynamic Programming With Task-Dependent Terminal Value

Amir-massoud Farahmand and **Daniel N. Nikovski**
Mitsubishi Electric Research Laboratories
Cambridge, MA, USA

Yuji Igarashi and **Hiroki Konaka**
Mitsubishi Electric Corporation
Hyogo 661-8661, Japan

Abstract

We propose a new class of computationally fast algorithms to find close to optimal policy for Markov Decision Processes (MDP) with large finite horizon T. The main idea is that instead of planning until the time horizon T, we plan only up to a truncated horizon $H \ll T$ and use an estimate of the true optimal value function as the terminal value. Our approach of finding the terminal value function is to learn a mapping from an MDP to its value function by solving many similar MDPs during a training phase and fit a regression estimator. We analyze the method by providing an error propagation theorem that shows the effect of various sources of errors to the quality of the solution. We also empirically validate this approach in a real-world application of designing an energy management system for Hybrid Electric Vehicles with promising results.

1 Introduction

We consider the problem of finding a close to optimal policy for Markov Decision Processes (MDP) with a finite horizon T. The usual approach to solve the finite-horizon problems is by a Dynamic Programming (DP) procedure that starts from time $t = T$ and computes the optimal value function by backward induction towards time $t = 1$. This simple Value Iteration procedure faces two possible computational challenges. The first is that for a large state space $\mathcal{X}$, each iteration of DP becomes expensive or even infeasible. For finite $\mathcal{X}$, the computation time is $O(|\mathcal{X}|^2)$. A discretization-based approach to deal with continuous $\mathcal{X}$ leads to impractical computation times, e.g., if $\mathcal{X}$ is a compact subset of $\mathbb{R}^d$, discretization at the ε-resolution leads to $O(\varepsilon^{-d})$ states, which quickly becomes very large as d grows. To address this curse of dimensionality, we may use a function approximator to provide a more compact approximate representation of the value function. This idea, which is sometimes called Approximate Dynamic Programming (ADP), has been studied in the reinforcement learning (RL), control engineering, and operations research communities (Bertsekas and Tsitsiklis 1996; Buşoniu et al. 2010; Szepesvári 2010; Powell 2011). More closely related to this work, but in the context of discounted MDPs, the use of function approximation in the form of Approximate Value Iteration and its variants has been studied, both empirically and theoretically, by (Ernst, Geurts, and Wehenkel 2005; Riedmiller 2005; Munos and Szepesvári 2008; Farahmand et al. 2009; Farahmand and Precup 2012; Mnih et al. 2015). For finite horizon MDPs, this idea has been theoretically studied by (Murphy 2005).

The second computational challenge is due to the time horizon T. Even though the computational cost is linear in T, this might still be a bottleneck if the problem has a large T. Similarly in the discounted case, the appearance of discount factor γ in performance bounds can be seen as determining the effective time horizon, which grows fast as γ approaches 1 (Scherrer and Lesner 2012).

The computational consequence of the time horizon might be a concern for real-time applications of finite-horizon MDPs where the available computational power is limited. For instance, our motivating application is the design of an energy management system for Hybrid Electric Vehicles (HEV), for which one should decide when to use the electric motor and when to use the internal combustion engine based on the state of the car and the route ahead in order to minimize the total fuel and electricity cost. This problem is commonly formulated as a finite-horizon MDP (Sciarretta and Guzzella 2007), in which the length of the route and the resolution of the decision points determine the appropriate horizon. Nonetheless, due to the computational limit of the usual onboard computers, solving the problem with a time horizon in the order of hundreds or thousands might be impractical. We shall return to this application later in the paper.

To address the large time horizon T, one may perform DP (or ADP for large state spaces) only for $H \ll T$ time steps and use a certain terminal value for the truncated procedure. One possibility for the choice of the new terminal value is to simply use a zero function. This indeed reduces the computational cost from $O(T)$ to $O(H)$, but it may lead to poor solutions, e.g., Example 6.5.1 of (Bertsekas 2000). Another possibility is to use the value function of some base policy as in the rollout algorithm, see e.g., (Bertsekas 2005) or Section 6.4 of (Bertsekas 2000). In that case one can show that the obtained policy is not worse than the base policy. One may also have an access to a domain expert's handcrafted heuristics that provides a good estimate for the truncated ter-

minal value, or learn the terminal value from the behaviour of an expert solving the MDP (Maddison et al. 2015), or learn it through a separate reinforcement learning process. Refer to (Gelly and Silver 2011) for a survey on several of these approaches in the context of computer Go, especially when used alongside Monte Carlo tree search. Note that the goal of all these approaches is to provide a sufficiently accurate estimate of the optimal value function as the terminal value at H time steps ahead. If a good estimate is known, the truncation does not lead to bad policies. Providing an estimate of the optimal value function to be used as the terminal value function is the direction that we pursue in this work.

We consider the scenario that we want to solve a problem belonging to a parameterized class of MDPs, in which the parameter describes the transition probability kernel and/or the reward function of the MDP, i.e., the task. Parameterized MDPs are natural way of thinking about sequential decision-making problems for which an agent has to solve a variety of similar tasks. Some examples are a robot manipulator that is given different targets in its workspace at each run, a soccer playing agent that learns a kicking policy based on the target location for the ball, and an HVAC system that controls the temperature based on the building parameters. Our approach is that in the training phase, we solve several instances of these MDPs by applying the full-horizon (A)DP to each of them. We use these solutions to train a function approximator that maps the parameters of the MDP (the task descriptor) to the value function. Later when we want to solve a new MDP, this learned function provides the terminal value at horizon H. After the initial training phase, the saving in the computation time is proportional to $\frac{T}{H}$. The suggested algorithm acts as a receding horizon controller. Instead of following the obtained policies for all H time steps, we replan after each time step. This makes the system more robust to the model uncertainties as well as the uncertainty in the terminal value function. We call this algorithm Truncated Approximate Dynamic Programming with Task-Dependent Terminal Value, and we introduce it in Section 2. This setup has clearly some elements of multi-task learning. Its underlying working assumption is that the value function of the MDP as a function of the parameter describing the MDP has certain regularities that can be exploited by a learning algorithm, e.g., small changes in the parameter leads to small changes in the value function. This is the case for many problems including the aforementioned energy management system for HEV.

We study the theoretical properties of this algorithm in Section 3. We provide an error propagation theorem for the proposed Truncated ADP procedure. We consider three sources of errors: 1) The error in estimating the terminal value function, 2) The error in applying the Bellman operator at each iteration of the procedure, and 3) The error in the model used for planning. This theoretical result can be used beyond the context of Truncated ADP. By setting H equal to T, we obtain an error propagation result for an ADP procedure for solving finite-horizon MDPs with model uncertainty. The result also holds when we do not use a model, but directly have access to samples, i.e., batch reinforcement learning sampling scenario.

Algorithm 1 Truncated ADP Solver with Task-Dependent Terminal Value

Input: MDP Model $(\mathcal{X}, \mathcal{A}, \mathcal{R}_\theta, \hat{\mathcal{P}}_\theta, T)$ for task θ; Horizon H, Dataset $\mathcal{D}_n = \left\{\left((X_i, t_i, \theta_i), V^*_{\theta_i,t_i}(X_i)\right)\right\}_{i=1}^n$; Initial State X_1
$\bar{V} \leftarrow \text{REGRESS}(\mathcal{D}_n)$ {Learn a regression estimator.}
Set X_1.
for $t = 1$ **to** T **do**
  $\hat{V} \leftarrow \text{TRUNCATEDADP}(\mathcal{R}_\theta, \hat{\mathcal{P}}_\theta, t, H, \bar{V}_\theta)$
  $A_t \leftarrow \hat{\pi}_t(X_t; \mathcal{R}_\theta, \hat{\mathcal{P}}_\theta, \hat{V}) \triangleq \text{argmax}_{a\in\mathcal{A}}\{r_{\theta,t}(X_t, a) + \int \hat{\mathcal{P}}_\theta(\mathrm{d}x'|x, a)\hat{V}_{t,t+1}(x')\}$
  $X_{t+1} \sim \mathcal{P}_\theta(\cdot|X_t, A_t)$ {Perform the action in the environment}
end for

Algorithm 2 $\text{TRUNCATEDADP}(\mathcal{R}_\theta, \hat{\mathcal{P}}_\theta, t, H, \bar{V}_\theta)$

Input: MDP Model $(\mathcal{X}, \mathcal{A}, \mathcal{R}_\theta, \hat{\mathcal{P}}_\theta, T)$ for task θ; Time step t; Horizon H; Global value function estimator $\bar{V}$.
if $t + H < T$ **then**
  $h \leftarrow H$
else
  $h \leftarrow T - t$
end if
$\hat{V}_{t,t+h} \leftarrow \bar{V}_{\theta,t+h}$
for $\tau = t + h - 1$ **to** t **do**
  $\tilde{V}_{t,\tau}(x) \triangleq \max_a \left\{r_{\theta,\tau}(x, a) + \int \hat{\mathcal{P}}_\theta(\mathrm{d}x'|x, a)\hat{V}_{t,\tau+1}(x')\right\}$ for all $x \in \mathcal{X}$.
  $\hat{V}_{t,\tau} \leftarrow \text{APPROX}(\tilde{V}_{t,\tau})$
end for
return $\hat{V}$

2 Truncated Approximate Dynamic Programming Framework

Consider a finite-horizon MDP $(\mathcal{X}, \mathcal{A}, \mathcal{R}_\theta, \mathcal{P}_\theta, T)$ parameterized by $\theta \in \Theta$. Here $T \in \mathbb{N}$ is the time horizon, $\mathcal{X}$ is a (possibly continuous) state space, $\mathcal{A}$ is a finite action set, $\mathcal{R}_\theta : \mathcal{X} \times \mathcal{A} \times \{1, 2, \ldots, T\} \to \mathcal{M}(\mathbb{R})$ is the immediate reward distribution, and $\mathcal{P}_\theta : \mathcal{X} \times \mathcal{A} \to \mathcal{M}(\mathcal{X})$ is the transition probability kernel.[1] The expected immediate reward at time t is $r_{\theta,t}(x, a) = \mathbb{E}\left[\mathcal{R}_{\theta,t}(\cdot|x, a)\right]$. We denote $V^*_\theta : \mathcal{X} \times \{1, \ldots, T\} \to \mathbb{R}$ as the optimal value function and $\pi^*_\theta : \mathcal{X} \times \{1, \ldots, T\} \to \mathcal{A}$ as the optimal policy. We may use double subscripted $V^*_{\theta,t} : \mathcal{X} \to \mathbb{R}$ to refer to the optimal value function at time step t for parameter θ. If the parameter θ is clear from the context, or its particular choice is not important, we may drop it.

The goal of Truncated ADP is to solve a given MDP parameterized by θ by planning not for all T horizon steps, but for some $H \leq T$ steps, which in practice may be chosen to be $H \ll T$. We assume that in the pre-planning phase, we have access to $V^*_{\theta'}$ for several instances of $\theta' \in \Theta$. The computational time of this pre-planning phase is not a concern for us.

[1] $\mathcal{M}(S)$ denotes the set of all probability distributions defined over a set S with a σ-algebra σ_S.

The Truncated ADP can be used in either planning or batch RL scenarios. In the planning scenario we assume that an approximate MDP $(\mathcal{X}, \mathcal{A}, \mathcal{R}_\theta, \hat{\mathcal{P}}_\theta, T)$ is known. The known transition model $\hat{\mathcal{P}}_\theta$ might be different from $\mathcal{P}_\theta$, in which case we have a modeling error. In the RL scenario, we assume that we have a batch of data in the form of $\{(X_{t,i}, A_{t,i}, R_{t,i}, X'_{t,i})\}_{i=1,\dots,N;t=1,\dots,T-1}$ with $X_{t,i} \sim \nu \in \mathcal{M}(\mathcal{X})$, $R_{t,i} \sim \mathcal{R}_\theta(\cdot|X_{t,i})$, and $X'_{t,i} \sim \hat{\mathcal{P}}_\theta(\cdot|X_{t,i})$. To make the presentation simpler, we focus on the planning scenario, and only add remarks on how the method can be applied in the RL setting. We are ready to describe the Truncated Approximate Dynamic Programming (TADP) with Task-Dependent Terminal Value, or simply TADP.

The TADP algorithm, described in Algorithm 1, is based on two insights. The first is that whenever the optimal value function as a function of θ has some kind of regularity (e.g., various notions of smoothness with respect to (w.r.t.) θ), we can hope to learn an approximate mapping from θ to the optimal value function by solving a regression/function fitting problem. We denote this estimator as $\bar{V} : \mathcal{X} \times \{1, 2, \dots, T\} \times \Theta \to \mathbb{R}$.

The second insight is based on the recursive property of the value function. As $V_t^*(x) = \mathbb{E}\left[\sum_{t'=t}^{T} r_{t'}(X_{t'}, \pi_{t'}^*(X_{t'})) \mid X_t = x\right]$ for any $t \in \{1, \dots, T\}$, we have

$$
\begin{aligned}
V_t^*(x) &= \mathbb{E}\left[\sum_{t'=t}^{t+H-1} r_{t'}(X_{t'}, \pi_{t'}^*(X_{t'})) + V_{t+H}^*(X_{t+H}) \mid X_t = x\right] \\
&\approx \mathbb{E}\left[\sum_{t'=t}^{t+H-1} r_{t'}(X_{t'}, \pi_{t'}^*(X_{t'})) + \bar{V}_{t+H}(X_{t+H}) \mid X_t = x\right],
\end{aligned}
$$

provided that $\bar{V}_{t+H}$ is a good approximation of V_{t+H}^*. So if $\bar{V}_{t+H}$ is accurate, it is enough to plan only for $H \leq T$ steps to have a good estimate of V_t^*. This leads to huge computational saving when $H \ll T$. The truncated ADP solver acts in a receding horizon manner, so at each time step, we solve a new truncated ADP of length H. These insights lead to Truncated ADP solver with Task-Dependent Terminal Value (Algorithm 1).

We briefly describe a few key elements of the algorithm. In the pre-planning phase (aka training phase), Algorithm 1 solves a regression problem using the given dataset $\mathcal{D}_n = \{((X_i, t_i, \theta_i), V_{\theta_i, t_i}^*(X_i))\}_{i=1}^n$ to learn $\bar{V}$. Any powerful regression method, including random forests, regularized regression in a reproducing kernel Hilbert space (RKHS), or a deep neural network can be used. Our empirical evaluations as well as the recent work by (Schaul et al. 2015) show the practical feasibility of learning $\bar{V}$ as a function of both state and the task θ (or the goal in the aforementioned work).

We may also provide some theoretical guarantees on the quality of learning V^* by $\bar{V}$. As an example, consider that $\mathcal{X} \times \Theta$ is a compact subset of $\mathbb{R}^{d_{\mathcal{X}\times\Theta}}$. If $V_{\theta,t}^*(x)$ as a function of (x, θ) belongs to a finite-norm subset of the Sobolev space $\mathbb{W}^k(\mathcal{X} \times \Theta)$, the space of all functions whose all k-th mixed derivatives are in $L_2(\mathcal{X} \times \Theta)$, it roughly holds that $\|\bar{V}_{\theta,t}^* - V_{\theta,t}^*\|_{1/2} \leq c_1 n^{-\frac{k}{2k+d_{\mathcal{X}\times\Theta}}}$. So as n increases, the error decreases too with a rate of convergence that depends on the joint dimension $d_{\mathcal{X}\times\Theta}$ and the complexity of V_θ^* as measured by k.

A practical approach to generate $\mathcal{D}_n$ is to randomly choose several θ_j $(j = 1, \dots, K)$, calculate $V_{\theta_j}^*$ for each of them, collect $V_{\theta_j,t_j}^*(X_{j,i})$ for many $(t_{j,i}, X_{j,i}) \in \{1, \dots, T\} \times \mathcal{X}$, and then store all of them in $\mathcal{D}_n$. The distribution of θ_j should be close to the distribution of θ encountered during the planning phase to ensure proper generalization. Furthermore, note that if the state space is large, the exact computation of $V_{\theta_i}^*$ itself might be infeasible. In that case, we use approximation scheme similar to various fitted/approximate value iteration algorithms to provide an estimate $\hat{V}_{\theta_i}^*$ of $V_{\theta_i}^*$ (Riedmiller 2005; Ernst, Geurts, and Wehenkel 2005; Murphy 2005; Munos and Szepesvári 2008; Farahmand et al. 2009; Farahmand and Precup 2012). This dataset is obtained offline, so its computation time is not a concern. For example, in the HEV energy management system described earlier, this dataset is not obtained by a car's onboard computer, but by a powerful computer of the car manufacturer.

After this initialization, Algorithm 1 repeatedly calls TRUNCATEDADP (Algorithm 2), which performs ADP with a truncated horizon H. TRUNCATEDADP uses $\bar{V}$ to set the terminal value. As before, if the state space is large, the exact computation of the Bellman operator, and as such $\tilde{V}$, might be difficult or impossible, so we use fitted/approximation value iteration algorithms, as mentioned earlier. For example we can randomly choose $\{X_j\}_{j=1}^N$ with $X_j \sim \nu \in \mathcal{M}(\mathcal{X})$, and define $Y_j^a = r_{\theta,\tau}(X_j, a) + \hat{V}_{t,\tau+1}(X'_j)$ with $X'_j \sim \hat{\mathcal{P}}_\theta(\cdot|X_j, a)$ for all $a \in \mathcal{A}$. Afterwards, we solve $|\mathcal{A}|$ regression problems $\hat{Q}_{t,\tau}(\cdot, a) \leftarrow \text{REGRESS}\left(\{X_j, Y_j^a\}_{j=1}^N\right)$, and let $\hat{V}_{t,\tau}(x) = \max_{a\in\mathcal{A}} \hat{Q}_{t,\tau}(x, a)$.[2] If REGRESS is chosen properly, the difference between $\hat{V}_{t,\tau}$ and $\tilde{V}_{t,\tau}$ becomes smaller as N increases. The same procedure can be used when the model is not known, but we have a batch of data in the form of $\{(X_{\tau,j}, A_{\tau,j}, R_{\tau,i}, X'_{\tau,j})\}_{j=1}^N$ (with $X'_{\tau,j}$ defined as before) for the transition happening at time step τ according to $\mathcal{P}_\theta$ (or even $\hat{\mathcal{P}}_\theta$). This constitute the APPROX step of the algorithm.

Before providing theoretical guarantees for TADP, let us briefly mention that there are some other algorithms with some similarities to TADP. One class is the family of Model Predictive Controllers (MPC), and in particular the rollout algorithm, cf. e.g., (Bertsekas 2005). The rollout algorithm is similar to TADP with $H = 1$ with a crucial difference that it uses the value of the rollout policy instead of the estimate $\bar{V}$. Moreover, the rollout algorithm does not have any multi-task aspect. The multi-task aspect of TADP with its attempt to learn $\bar{V}_\theta$ has some remote similarities with some works that learn a policy that generalizes over multiple tasks, e.g.,

[2] The semantics of double subscripts in $\hat{V}_{t,\tau}$ and $\tilde{V}_{t,\tau}$ is different from that in $V_{\theta,t}^*$, which was described earlier. In the former cases, the first index refers to the t-th call by Algorithm 1 and the second index refers to a particular iteration of Algorithm 2.

(da Silva, Konidaris, and Barto 2012; Deisenroth et al. 2014; Kober et al. 2012). All these methods learn a multi-task policy instead of multi-task value function $\bar{V}$ as in this work. More importantly, the learned policy is not used as a part of any further planning when the agent has to solve a new task. TADP, on the other hand, goes through a new planning phase after facing a new task, and uses the learned value function to bootstrap the planning at the truncation horizon.

3 Theoretical Analysis

In this section we analyze Truncated Approximate Dynamic Programming (TADP) with Task-Dependent Terminal Value and provide error propagation results similar to those of (Munos 2007; Farahmand, Munos, and Szepesvári 2010; Scherrer et al. 2012; Huang et al. 2015). Theorem 1 provides an $L_1(\rho)$ error upper bound for the difference between the optimal value function and the value function of following the policy sequence obtained by Algorithm 1 w.r.t. a performance measuring distribution $\rho \in \mathcal{M}(\mathcal{X})$, i.e., $\int_{\mathcal{X}} \left(V_t^*(x) - V_t^{\pi_{t:T}}(x)\right) \rho(\mathrm{d}x)$. The distribution ρ determines our emphasis over different regions of the state space. In the results, another distribution $\nu \in \mathcal{M}(\mathcal{X})$ appears with respect to which the norm of the contributing errors is measured. One may think of it as the sampling distribution used at each step of TRUNCATEDADP (Algorithm 2), but its choice is not limited to that. The proofs and more detailed discussions will be presented in an extended version of this work.

For the sequence of $\hat{V}_{t,t'}$ generated by Algorithm 2, we define the corresponding sequence of greedy policies: $\pi_{t,t'}(x) = \hat{\pi}_{t'}(x; \mathcal{R}_\theta, \hat{\mathcal{P}}_\theta, \hat{V}_{t,\cdot}) = \operatorname{argmax}_{a\in\mathcal{A}} \left\{ r_{\theta,t'}(x,a) + \int \hat{\mathcal{P}}_\theta(\mathrm{d}x'|x,a) \hat{V}_{t,t'+1}(x') \right\}$. We notice that the policy that is executed at time t by Algorithm 1 is indeed $\pi_{t,t}$. The sequence of such policies defines a non-stationary policy $\bar{\pi} = (\pi_{1,1}, \ldots, \pi_{T,T})$. We denote the subsequence of $\bar{\pi}$ from time t to the end of horizon by $\bar{\pi}_t = (\pi_{t,t}, \ldots, \pi_{T,T})$. To simplify the notations we omit the dependence of these quantities on θ.

We have three sources of errors in Algorithms 1 and 2. The first is the error caused by using the global value function approximator $\bar{V}$ to set the terminal value $\hat{V}_{t+h(t)}$. For time $\tau = t + h(t)$, we denote $\delta_\tau \triangleq \hat{V}_{t,\tau} - V_\tau^*$. Therefore, we have a sequence of $\delta_{h(1)}, \ldots, \delta_T$. The second source of error is caused by the model mismatch between the true model $\mathcal{P}$ of the MDP and the model $\hat{\mathcal{P}}$ used for planning in Algorithm 2. We denote this error by $\Delta\mathcal{P} : \mathcal{X} \times \mathcal{A} \to \mathcal{M}(\mathcal{X}) \cup -\mathcal{M}(\mathcal{X})$ with $\Delta\mathcal{P} \triangleq \hat{\mathcal{P}} - \mathcal{P}$. The final source of error is caused by performing ADP instead of exact DP. This is the error between $\hat{V}_{t,\tau}$ and $\tilde{V}_{t,\tau}$ in Algorithm 2. We define $e_{t,t'} \triangleq \hat{V}_{t,t'} - \tilde{V}_{t,t'}$ for $t = 1, \ldots, T$ and $t' = t, \ldots, t + h(t)$. The analysis does not depend on how this error is occured. So this might be an error caused by approximating the expectation over the next-state distribution, or the regression error in a fitted ADP procedure.

We use $\mathcal{P}^\pi$ as the transition probability kernel of following π, and for a sequence of policies $\pi_{1:m} = (\pi_1, \ldots, \pi_m)$, $\mathcal{P}^{\pi_{1:m}}$ as the transition probability kernel of choosing actions according to these policies consecutively. $(\mathcal{P}^\pi V)(x)$ is the expected value of function V w.r.t. the distribution induced by following $\mathcal{P}^\pi$ from state x. For a probability distribution $\rho \in \mathcal{M}(\mathcal{X})$, $\rho\mathcal{P}^\pi \in \mathcal{M}(\mathcal{X})$ is the distribution induced by following $\mathcal{P}^\pi$ when the initial distribution is ρ. For $1 \le p, q < \infty$, we define $\|\Delta\mathcal{P}^\pi\|_{p,q(\nu)} = \left[\int \mathrm{d}\nu(x) \left[\int |\Delta\mathcal{P}^\pi(\mathrm{d}y|x)|^p \right]^{\frac{q}{p}} \right]^{1/q}$.

We define the following concentrability coefficients similar to those defined by (Munos 2003; Kakade and Langford 2002; Munos 2007; Farahmand, Munos, and Szepesvári 2010; Scherrer et al. 2012; Huang et al. 2015).

Definition 1 (Concentrability Coefficients)**.** *Given two probability distributions $\rho, \nu \in \mathcal{M}(\mathcal{X})$, an integer $k \ge 0$, and an arbitrary sequence of policies $\pi_{1:k}$, $\rho\mathcal{P}^{\pi_{1:k}} \in \mathcal{M}(\mathcal{X})$ denotes the future-state distribution after choosing the initial state distribution ρ and following $\pi_{1:k}$ afterwards. Let $1 \le p < \infty$. Define the following concentrability coefficients: $c_{\rho,\nu}(k) \triangleq \sup_{\pi_{1:k}} \| \frac{\mathrm{d}(\rho\mathcal{P}^{\pi_{1:k}})}{\mathrm{d}\nu} \|_\infty$ and $\bar{c}_{\rho,\nu,p}(k) \triangleq \sup_{\pi_{1:k}} [\int | \frac{\mathrm{d}(\rho\mathcal{P}^{\pi_{1:k}})}{\mathrm{d}\nu}(y)|^p \mathrm{d}\nu(y)]^{1/p}$. If $\rho\mathcal{P}^{\pi_{1:k}}$ is not absolutely continuous w.r.t. ν, we set $c_{\rho,\nu}(k) = \infty$ and $\bar{c}_{\rho,\nu,p}(k) = \infty$.*

Definition 2. *Consider two probability distributions $\rho, \nu \in \mathcal{M}(\mathcal{X})$. For any integer $k \ge 0$ and real number $\beta > 0$, define the followings: $C(k; \rho, \nu) \triangleq \sum_{s=0}^k c_{\rho,\nu}(s)$ and $\bar{C}(k; \rho, \nu) \triangleq \sum_{s=0}^k \bar{c}_{\rho,\nu,2}(s)$.*

The following theorem is the main theoretical result of this paper.

Theorem 1. *Consider the sequence of policies $\bar{\pi} = (\pi_{1,1}, \ldots, \pi_{T,T})$ generated by Algorithm 1. Assume that $\hat{V}_{\cdot,\cdot} \in B(\mathcal{X}, V_{max})$. We then have*

$$\left\|V_1^{\bar{\pi}_1} - V_1^*\right\|_{1,\rho} \le c_1 \, C(T;\rho,\nu) \times \left[H \left(\max_{t,t'} \|e_{t,t'}\|_{1,\nu} + V_{max} \sup_\pi \|\Delta\mathcal{P}^\pi\|_{1,1(\nu)} \right) + \max_t \|\delta_t\|_{1,\nu} \right],$$

in which $c_1 > 0$ is a universal constant. The same bound would hold if we replace C with $\bar{C}$ and all the L_1-norms with the L_2-norms. Moreover, assuming that $c_{\rho,\nu}(s) = O(1)$ and $\bar{c}_{\rho,\nu,2}(s) = O(1)$, we also have

$$\left\|V_1^{\bar{\pi}_1} - V_1^*\right\|_{1,\rho} \le c_2 \Big[TH \Big(\max_{t,t'} \|e_{t,t'}\|_{1/2,\nu} + V_{max} \sup_\pi \|\Delta\mathcal{P}^\pi\|_{1/2,1/2(\nu)} \Big) + T \max_t \|\delta_t\|_{1/2,\nu} \Big],$$

in which $c_2 > 0$ is a universal constant.

This result shows the effect of the error in the global function approximator $\|\delta_\cdot\|_{1/2,\nu}$, the error of performing ADP $\|e_{t,t'}\|_{1/2,\nu}$, and the model mismatch error $\|\Delta\mathcal{P}\|_{1/2,\nu}$ on the performance loss $\|V_t^{\bar{\pi}_t} - V_t^*\|_{1,\rho}$. The dependence of the upper bound on the error caused by truncation is $O(T)$, while the dependence on the error caused by the model mismatch and ADP is $O(TH)$. Intuitively, at each iteration of TRUNCATEDADP (Algorithm 2) we only make error in estimating the terminal value once, but we make the modeling

error and the ADP errors H times. This error in estimating V_t^* by $\hat{V}_{t,t}$ happens at each time step t, so for an agent that start from time $t = 1$ and uses $\hat{V}_{t,t}$ to choose its action until it gets to $t = T$, it makes errors T times.

The increase of the upper bound as a function of H is intuitive as increasing the planning horizon means more accumulation of the errors caused by $e_{t,t'}$ and $\Delta\mathcal{P}$. If the estimate of the terminal value function $\bar{V}_t$ is such that $\|\delta_t\|$ is uniformly small for all t, it is better to choose H as small as possible, e.g., $H = 1$. Nevertheless, it seems that for some classes of MDPs, the estimation of V_t^* for larger t is easier. If the MDP is such that as we get farther from the terminal time T, the value function becomes more "complex", e.g., in the smoothness sense or some other relevant measure of function complexity, one can then show that $\|\delta_t\|$ is a decreasing function of t. Studying the subtle behaviour of learning $\bar{V}$ as a function of the complexity of V_t^* is an interesting future research topic. The main point, however, is that there is a tradeoff between choosing smaller H (leading to smaller ADP and modeling errors) and choosing larger H (leading to possibly smaller global value function estimation error for certain MDPs). We note that the observation that choosing smaller horizons might be beneficial have been made by (Jiang et al. 2015) and (Petrik and Scherrer 2009). Their results are from perspectives different from the error propagation theory.

These error propagation results have the same general flavour as some previous work in other settings, e.g., Approximate Value Iteration (Munos 2007), Approximate Policy Iteration (Munos 2003), and Approximate Modified Policy Iteration (Scherrer et al. 2012), all for discounted MDPs, finite-horizon MDP (Murphy 2005), and a finite horizon MDP with softmax-based Bellman operator (Huang et al. 2015). All of them consider the effect of terms similar to $e.$, but because of different setup, they do not consider the effect of truncation. Also they do not consider the effect of modeling error. Moreover, the current result uses a refined definition of concentrability coefficients, which is similar to (Farahmand, Munos, and Szepesvári 2010; Scherrer et al. 2012).

We summarize a few key steps of the proof. First we prove an upper bound on the error in approximating the optimal value function, that is $|\hat{V}_{t,t'} - V_{t'}^*|$. The upper bound depends on $\Delta\mathcal{P}$, $e_{t,t'}$, $\delta_{t+h(t)}$, as well as $\mathcal{P}$ itself. We then relate $V_t^* - V_t^{\bar{\pi}_t}$, the pointwise performance loss, to $|\hat{V}_{t,t'} - V_{t'}^*|$ and some other quantities. Therefore we obtain a pointwise upper bound on $V_t^* - V_t^{\bar{\pi}_t}$ that takes into account all three sources of errors in TADP. What makes this part of analysis intricate is that we have to 1) consider three sources of errors (δ, e, and $\Delta\mathcal{P}$) as opposed to the more usual case of only considering terms like e (Munos 2007; Farahmand, Munos, and Szepesvári 2010), and 2) take extra care in dealing with a horizon that is not only finite, but also smaller than T and may actually change depending on how far we are from T (recall that TADP uses $h(t) \leq H \leq T$). Finally, we use a change of measure argument to related the $L_1(\rho)$ performance loss to the aforementioned pointwise error. The argument uses a general definition of the concentrability coefficient as well as the decomposition of the relevant terms to error-related and concentrability-related terms.

4 Experiments

We apply the proposed method to design an energy management system for a hybrid vehicle (Sciarretta and Guzzella 2007). Hybrid Electric Vehicles are among the most energy-efficient, cost-effective, and dependable classes of transportation systems. This is mainly because of their ability of regenerating energy during braking that would otherwise be lost, and storing that energy into relatively less expensive batteries than those installed in purely electrical vehicles, as well as enjoying the long range, reliability, and high energy density typical for internal combustion engines (ICE). When two engines are available, an electric one and a fossil fuel one, an important question is when to use each of them. It has been shown that the overall energy efficiency of HEV can significantly be improved if the operation of the engines is coordinated carefully over the duration of an entire trip by taking into consideration the nature of the terrain that the trip will cover, and the current state of the vehicle. One approach that has been used to calculate the optimal policy is dynamic programming, see e.g., (Larsson, Johannesson, and Egardt 2014) and references therein.

Although true optimality is a very favourable feature of exact dynamic programming, its computational complexity is not a good match to the current level of embedded computers available in mass-produced passenger vehicles. TADP with Task-Dependent Terminal value has been designed with exactly this objective: reducing the onboard computational cost and delegating the costly computation offline. In this section, we study the quality of TADP's policies compared to the optimal policy. We study the effects of the number of training data used to learn the task-dependent terminal value, planning horizon, and the uncertainty in the model used for planning.

The hybrid vehicle optimization problem can be formulated as a finite-horizon MDP T, which pertains to a single trip between given starting and destination points, with the state variable x_t being the State of Charge (SoC) of the battery at time step t.[3] In its simplest form, the action a_t determines whether the electric motor (EM) should be used at time t or the ICE should be deployed. The objective is to minimize the total cost of the energy consumption during the trip. The energy consumption depends on the amount of fuel spent and the total electrical energy required to recharge the battery at the end of the trip. The amount of fuel or electricity consumed at each time step is a complex function of the vehicle's velocity, acceleration, and the road's slope. For the experiments, we use a simulator that uses principles similar to (Larsson, Johannesson, and Egardt 2014). We set T to 115 with each time step corresponding to one minute of physical time, so we plan for an almost two hour trip. The parameter θ is a vector of dimension 230 describing the altitude and the speed profile of the run-curve.

[3]Other formulations of the problem, such as discounted or average reward MDP, are possible too, but we focus on this rather standard finite horizon formulation.

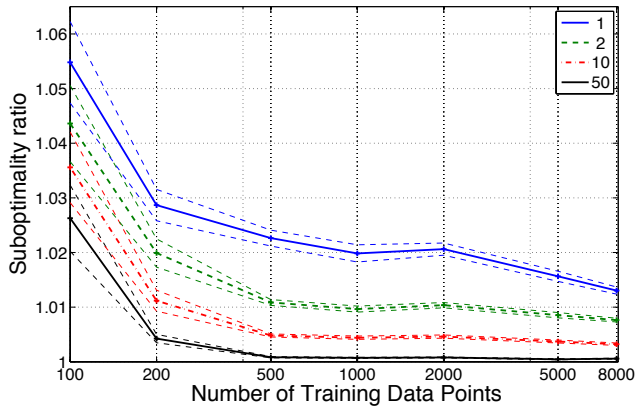

Figure 1: The suboptimality of TADP as a function of the number of training data points and the planning horizon H. The dashed line depicts one standard error.

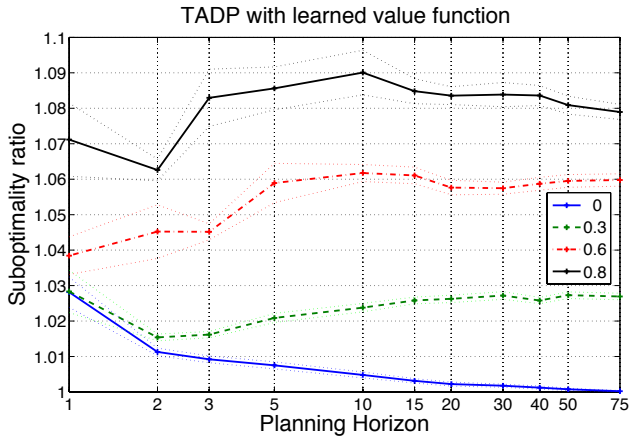

Figure 2: The suboptimality of TADP as a function of the planning horizon H and the uncertainty level Δ. The dashed line depicts one standard error.

The application of TADP requires us to learn $\bar{V}$, which is obtained as follows: We generate several random run-curves θ_i, find the "optimal" value function $V^*_{\theta_i}$ by dynamic programming, and select several random points in time and state from each random run-curve. These points define the training data. Instead of directly working with a $230 + 2$-dimensional vector (the vector of θ as well as x and t), we extract a 7-dimensional features that approximately summarize the route and the current state of the vehicle. We then use these features as an input to a reproducing kernel Hilbert space (RKHS)-based regularized least-squares regression estimator to find the estimate of $\bar{V}$.

After learning $\bar{V}$, we generate a new random run-curve θ from the same distribution which was used in generating training data and call Algorithm 1, which itself calls TRUNCATEDADP (Algorithm 2). We use DP (with 2000 discretization of the state variable x) and perform H iterations of the Bellman backup.

Figure 1 shows that when the number of training points n increases, the quality of TADP's policy increases too for all choices of the horizon. This indicates that the learned $\bar{V}$ is actually good enough for truncating the horizon, even though the parameter space Θ is high-dimensional (230). If the features were sufficiently rich to represent all possible run-curves, increasing n would result in having an ever improving $\bar{V}$ approximation of V^*_θ. If $\bar{V}(\theta) = V^*_\theta$ uniformly over θ, t, and x, even having $H = 1$ is sufficient to act optimally. This is of course not the case here (remember that our features are not necessarily optimal), so we expect that for $H = 1$, the suboptimality ratio does not eventually reach 1. Nevertheless, we see that it is continually improving for this range of training points, and it does not appear that the improvement is saturating. The quality of the policy with 8000 training points and $H = 1$ is within 1.5% of the optimal one, which is indeed remarkable. This might be compared with a "nominal" policy, which is defined as the policy that uses the battery whenever it is charged more than a certain threshold (5% in our experiments), and switches to ICE otherwise. This policy has the suboptimality ratio of ≈ 1.27.

We also observe that increasing H improves the quality of the policy with the same $\bar{V}$. Even a small $H \ll T$, for instance $H = 10$, leads to a high quality policy. This is an interesting tradeoff. We may increase the quality of the task-dependent terminal value $\bar{V}$ by using more data points and/or better function approximator architectures. This generally means that we spend more computation time offline. On the other hand, we may use a lower quality $\bar{V}$, but increase H. This results in an increase in the online computation. The right balance is problem specific and depends on several factors such as how the function approximator and each step of ADP are implemented, the onboard computational power, etc.

We now turn to studying the effect of uncertainty in the quality of resulting policy. Intuitively, if the model used for planning is uncertain, it does not make sense to plan long ahead. Here we fix the number of training points used to learn $\bar{V}$ to 2000. To generate modeling error, in the form of stochastic uncertainty, we multiply the change in the state of charge (i.e., the amount of battery that is used) and the fuel consumption by a random variable drawn from the uniform distribution $U(1 - \frac{\Delta}{2}, 1 + \frac{\Delta}{2})$ where Δ is the amount of uncertainty. We select Δ from the set $\{0, 0.3, 0.6, 0.8\}$. So for example, in the extreme case, the amount of fuel consumption used for planning can be between 60% to 140% of the actual value. This random sampling is done at each time step. So this can be seen as an unbiased model for the sensors measuring the state of charge and the fuel consumption. Here we report the results averaged over 20 independent runs.

Figure 2 shows that when there is a modeling error, increasing the horizon does not necessarily lead to improved performance. This is as opposed to the case when there is no modeling error. More interestingly, it appears that there is a sweet spot for the planning horizon H depending on the level of uncertainty. For high amount of uncertainty, however, the shortest planning horizon becomes $H = 1$. We also see that as the uncertainty level increases, the quality of the

policy degrades. One should not interpret the performance degradation as a downside of TADP per se, but an indication that whenever the model used for planning is very inaccurate and different from reality, one cannot and should not plan much ahead.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful suggestions.

References

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Bertsekas, D. P. 2000. *Dynamic Programming and Optimal Control*, volume I. Athena Scientific.

Bertsekas, D. P. 2005. Dynamic programming and suboptimal control: A survey from ADP to MPC. *European Journal of Control* 11(4-5):310–334.

Buşoniu, L.; Babuška, R.; De Schutter, B.; and Ernst, D. 2010. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press.

da Silva, B. C.; Konidaris, G.; and Barto, A. G. 2012. Learning parameterized skills. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*.

Deisenroth, M. P.; Englert, P.; Peters, J.; and Fox, D. 2014. Multi-task policy search for robotics. In *International Conference on Robotics and Automation (ICRA)*.

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

Farahmand, A.-m., and Precup, D. 2012. Value pursuit iteration. In *Advances in Neural Information Processing Systems (NIPS - 25)*, 1349–1357. Curran Associates, Inc.

Farahmand, A.-m.; Ghavamzadeh, M.; Szepesvári, Cs.; and Mannor, S. 2009. Regularized fitted Q-iteration for planning in continuous-space Markovian Decision Problems. In *Proceedings of American Control Conference (ACC)*, 725–730.

Farahmand, A.-m.; Munos, R.; and Szepesvári, Cs. 2010. Error propagation for approximate policy and value iteration. In *Advances in Neural Information Processing Systems (NIPS - 23)*. 568–576.

Gelly, S., and Silver, D. 2011. Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence* 175(11):1856–1875.

Huang, D.-A.; Farahmand, A.-m.; Kitani, K. M.; and Bagnell, J. A. 2015. Approximate MaxEnt inverse optimal control and its application for mental simulation of human interactions. In *AAAI Conference on Artificial Intelligence*.

Jiang, N.; Kulesza, A.; Singh, S.; and Lewis, R. 2015. The dependence of effective planning horizon on model accuracy. In *International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*.

Kakade, S., and Langford, J. 2002. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML)*, 267–274.

Kober, J.; Wilhelm, A.; Oztop, E.; and Peters, J. 2012. Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots* 33(4):361–379.

Larsson, V.; Johannesson, L.; and Egardt, B. 2014. Cubic spline approximations of the dynamic programming cost-to-go in HEV energy management problems. In *European Control Conference (ECC)*, 1699–1704.

Maddison, C. J.; Huang, A.; Sutskever, I.; and Silver, D. 2015. Move evaluation in Go using deep convolutional neural networks. In *International Conference on Learning Representations (ICLR)*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Munos, R., and Szepesvári, Cs. 2008. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research* 9:815–857.

Munos, R. 2003. Error bounds for approximate policy iteration. In *Proceedings of the 20th Annual International Conference on Machine Learning (ICML)*, 560–567.

Munos, R. 2007. Performance bounds in L_p norm for approximate value iteration. *SIAM Journal on Control and Optimization* 541–561.

Murphy, S. A. 2005. A generalization error for Q-learning. *Journal of Machine Learning Research (JMLR)* 6:1073–1097.

Petrik, M., and Scherrer, B. 2009. Biasing approximate dynamic programming with a lower discount factor. In *Advances in Neural Information Processing Systems (NIPS - 21)*. MIT Press. 1265–1272.

Powell, W. B. 2011. *Approximate Dynamic Programming: Solving the Curse of Dimensionality*. Wiley, 2nd edition.

Riedmiller, M. 2005. Neural fitted Q iteration – first experiences with a data efficient neural reinforcement learning method. In *16th European Conference on Machine Learning*, 317–328.

Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal value function approximators. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*.

Scherrer, B., and Lesner, B. 2012. On the use of non-stationary policies for stationary infinite-horizon Markov Decision Processes. In *Advances in Neural Information Processing Systems (NIPS - 25)*. 1835–1843.

Scherrer, B.; Ghavamzadeh, M.; Gabillon, V.; and Geist, M. 2012. Approximate modified policy iteration. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*.

Sciarretta, A., and Guzzella, L. 2007. Control of hybrid electric vehicles. *IEEE Control Systems Magazine* 27(2):60–70.

Szepesvári, Cs. 2010. *Algorithms for Reinforcement Learning*. Morgan Claypool Publishers.