Approximate Policy Iteration for Markov Decision Processes via Quantitative Adaptive Aggregations

Alessandro Abate[1], Milan Češka[1,2(✉)], and Marta Kwiatkowska[1]

[1] Department of Computer Science, University of Oxford, Oxford, UK
[2] Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic
ceskam@fit.vutbr.cz

Abstract. We consider the problem of finding an optimal policy in a Markov decision process that maximises the expected discounted sum of rewards over an infinite time horizon. Since the explicit iterative dynamical programming scheme does not scale when increasing the dimension of the state space, a number of approximate methods have been developed. These are typically based on value or policy iteration, enabling further speedups through lumped and distributed updates, or by employing succinct representations of the value functions. However, none of the existing approximate techniques provides general, explicit and tunable bounds on the approximation error, a problem particularly relevant when the level of accuracy affects the optimality of the policy. In this paper we propose a new approximate policy iteration scheme that mitigates the state-space explosion problem by adaptive state-space aggregation, at the same time providing rigorous and explicit error bounds that can be used to control the optimality level of the obtained policy. We evaluate the new approach on a case study, demonstrating evidence that the state-space reduction results in considerable acceleration of the policy iteration scheme, while being able to meet the required level of precision.

1 Introduction

Dynamic programming (DP) is one of the most celebrated algorithms in computer science, optimisation, control theory, and operations research [3]. Applied to reactive models with actions, it allows synthesising optimal policies that optimise a given reward function over the state space of the model. According to Bellman's principle of optimality, the DP algorithm is a recursive procedure over value functions. Value functions are defined over the whole state and action spaces and over the time horizon of the decision problem. They are updated backward-recursively by means of locally optimal policies and, evaluated at the

This work has been partially supported by the ERC Advanced Grant VERIWARE, the EPSRC Mobile Autonomy Programme Grant EP/M019918/1, the Czech Grant Agency grant No. GA16-17538S (M. Češka), and the John Fell Oxford University Press (OUP) Research Fund.

C. Artho et al. (Eds.): ATVA 2016, LNCS 9938, pp. 13–31, 2016.
DOI: 10.1007/978-3-319-46520-3_2

initial time point or in steady-state, yield the optimised global reward and the associated optimal policy. By construction, the DP scheme is prone to issues related to state-space explosion, otherwise known as the "curse of dimensionality". An active research area [5,6] has investigated approaches to mitigate this issue: we can broadly distinguish two classic approaches.

Sample-based schemes approximate the reward functions by sampling over the model's dynamics [6,14], either by regressing the associated value function over a given (parameterised) function class, or by synthesising upper and lower bounds for the reward function. As such, whilst in principle avoiding exhaustive exploration of the state space, they are associated to known limitations: they often require much tuning or selection of the function class; they are not always associated with quantitative convergence properties or strong asymptotic statistical guarantees; and they are prone to requiring naïve search of the action space, and hence scale badly over the non-determinism. In contrast to the state-space aggregation scheme presented in this paper, they compute the optimal policy only for the explored states, which can be in many cases insufficient.

Numerical schemes perform the recursion step for DP in a computationally enhanced manner. We distinguish two known alternatives. Value iteration updates backward-recursively value functions embedding the policy computation within each iteration. The iteration terminates once a non-linear equation (the familiar "Bellman equation") is verified. On the other hand, policy iteration schemes [4] distinguish two steps: policy update, where a new policy is computed; and policy evaluation, where the reward function associated to the given policy is evaluated (this boils down to an iteration up to convergence, or to the solution of a linear system of equations). Convergence proofs for both schemes are widely known and discussed in [5]. Both approaches can be further simplified by means of approximate schemes: for instance, the value iteration steps can be performed with distributed iterations attempting a modularisation, or via approximate value updates. Similarly to policy iteration, policy updates can be approximated and, for instance, run via prioritised sweeping over specific parts of the state space; furthermore, policy evaluations can be done optimistically (over a finite number of iterations), or by approximating the associated value functions.

In this work, we focus on the following modelling context: we deal with finite-state, discrete-time stochastic models, widely known as Markov decision processes (MDP) [16], and with γ-discounted, additive reward decision problems over an infinite time horizon. We set up an optimisation problem, seeking the optimal policy maximising the expected value of the given (provably bounded) reward function. We formulate the solution of this problem by means of a numerical approximate scheme.

Key Contributions. In this work we present a number of accomplishments:

– We put forward a modified policy iteration scheme which, while retaining the policy update step on the original MDP, performs an approximate policy evaluation by clustering the state space of the model.

- We derive explicit error bounds on the approximate policy evaluation step, which depend on the model dynamics, on the reward structure, and on the chosen state-space aggregation.
- We develop an automated policy iteration scheme, which adaptively updates the model aggregation at each policy evaluation step, according to the computed explicit error bounds.
- We argue that, unlike cognate literature, our quality certificates on the approximate policy evaluations only depend on manipulations of the abstract model.
- We argue that, whilst in the developed scheme the policy update is played out over the original (concrete) MDP, our approach can be extended to encompass an approximate policy update over the abstract model.
- With a case study, we show that the automated scheme does indeed improve the performance of the explicit policy iteration scheme, both in terms of state-space reduction and time.

Related Work. With emphasis on reward-based decision problems over MDP models, we can relate our contribution to the two alternative approaches discussed above, and quantitatively compare our scheme to existing numerical ones; note that sample-based approaches lack strong quantitative guarantees and therefore cannot be fairly compared.

Numerical and approximate schemes are discussed in [4] in detail. Specifically, with regards to policy iteration via approximate and optimistic policy updates, we argue that we provide certificates that only depend on manipulations of the abstract model and reward function, and that the approximation steps can be automatically embedded within the global policy iteration scheme.

Sample-based schemes are discussed in [6]; they differ from the numerical schemes in that they rarely provide guarantees. One exception is bounded real-time dynamical programming, for which precise bounds on approximation errors have been proposed, including policy synthesis for stochastic shortest path problems [14] and verification of quantitative specifications [7]. Further related work can be found in [9,15].

Finally, our work can be related to approaches which resort to uncertain (interval-based) MDPs as an abstraction framework and aim at providing lower and upper bounds on the probability of quantitative specifications. The work in [11] generates abstractions for MDPs using stochastic two-player games that can be further refined. The method computes lower and upper bounds on the minimum and maximum probability, which serve as a measure of the quality of the abstraction. Interval-based Markov chains have been used to obtain three-valued abstraction for discrete-space probabilistic systems [10], as well as to abstract continuous-space stochastic systems. In [2,8] the approximation error of the continuous dynamics is explicitly computed and can be tuned through different partitioning of the state space. In [13] bounded-parameter MDPs are used to abstract switched discrete-time stochastic systems.

2 Notations and Problem Setup

Model Syntax. We work with discrete-time Markov decision processes (MDP), with full state observations [3,16]. Formally, an MDP is defined as a triple (S, A, P), where

- $S = \{s_1, \ldots, s_n\}$ is the finite state space of size n;
- $A = \{a_1, \ldots, a_l\}$ is the finite action (input) space of size l;
- $P_{(\cdot)} : S \times A \times S \to [0, 1]$ is the transition probability matrix, which is such that $\forall i \in S, \forall a \in A : \sum_{j=1}^{n} P_a(i, j) = 1$.

We have assumed, for the sake of simplifying the notation, that all actions are available at any state s: this could be generalised by defining state-dependent sets $A(s), s \in S$, which are such that $A(s) \subseteq A$.

In order to characterise a run (a path) of the MDP, we consider finite or infinite strings of actions of the form $(a_0, a_1, a_2, \ldots), a_i \in A$. Of interest to this work, we structure actions as feedback functions from the model states S to the action space A, namely for any $k \geq 0, a_k$ takes the shape of a function $\mu_k : S \to A$. Further, we consider infinite strings of such feedback actions $\mu = (\mu_0, \mu_1, \mu_2, \ldots)$, which we denote as policies. We restrict to policies μ that are memoryless (Markovian) and deterministic (non-randomised), and denote with $\mu \in \mathcal{M}$ the set of all such admissible policies. For the problems of interest in this work, we seek to compute time-homogeneous policies, namely of the form $\mu = (\bar{\mu}, \bar{\mu}, \bar{\mu}, \ldots)$.

Model Semantics. Consider the model (S, A, P) and a given policy μ. The model is initialised via distribution $\pi_0 : S \to [0, 1]$, where $\sum_{s \in S} \pi_0(s) = 1$, and its transient probability distribution at time step $k \geq 0$ is

$$\pi_{k+1}(s) = \sum_{s' \in S} \pi_k(s') P_{\mu_k}(s', s) = P_{\mu_k}^T \pi_k, \tag{1}$$

or more concisely as $\pi_{k+1} = \pi_k P_{\mu_k}$ (where the π_k's are row vectors), and where of course $P_{\mu_k}(s', s) = P_{\mu_k(s')}(s', s)$.

The work in [1] has studied the derivation of a compact representation and an efficient computation of the vectors π_k for a Markov chain, which is an MDP under a time-homogeneous policy.

Decision Problem and Optimal Policy Synthesis. Consider a time-homogeneous reward function $g : S \times A \to \mathbb{R}_0^+$, which we assume to be bounded, and a discount factor $\gamma \in (0, 1)$. Consider the following decision problem

$$J^*(s) := \sup_{\mu \in \mathcal{M}} \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k g(s, \mu_k)\right],$$

for any $s \in S$, and where $\mathbb{E}$ denotes the expected value of a function of the process (as in the previous formula). Notice that in this setup the reward function unfolds

over an infinite time horizon; however, it is bounded in view of the presence of the discounting factor γ and the assumption on function g. We are also interested in deriving the optimal policy attaining the supremum, namely

$$\mu^*(s) := \arg\sup_{\mu\in\mathcal{M}} \mathbb{E}\left[\sum_{k=0}^{\infty}\gamma^k g(s,\mu_k)\right].$$

It is well known [3] that the class $\mathcal{M}$ of policies is sufficient to characterise the optimal policy given an MDP model and the additive optimisation setup above, namely we need not seek beyond this class (say, to randomised or non-Markovian policies). Further, the optimal policy is necessarily stationary (homogeneous in time).

Remark 1. It is likewise possible to consider decision problems where cost functions (similar in shape as those considered above) are infimised. Whilst in this work we focus on the first setup, our results are directly applicable to this second class of optimisation problems. □

Optimal Policy Synthesis: Characterisation via Dynamic Programming. Consider the class $\mathcal{F}$ of bounded functions $f : S \to \mathbb{R}_0^+$. In order to characterise the solution of the decision problem above as a recursive dynamic programming (DP) scheme, let us introduce operators (or mappings) $T, T_a : \mathcal{F} \to \mathcal{F}, a \in A$, such that

$$(T_a f)(s) = g(s,a) + \gamma \sum_{s'\in S} P_a(s,s')f(s'),$$

$$(Tf)(s) = \sup_{a\in A}\left\{g(s,a) + \gamma \sum_{s'\in S} P_a(s,s')f(s')\right\}.$$

In a more succinct vector form, we can express $T_a f = g_a + \gamma P_a f$, and further the condition $f_a = T_a f_a = g_a + \gamma P_a f_a$, so that $f_a = (I - \gamma P_a)^{-1} g_a$, which is a system of linear equations [3] that is relevant below (also in the form depending on the operator T). Further, the sequential application of this operator k times, where $k > 0$, is denoted as $(T_a^k f)(s) = (T_a(T_a^{k-1} f))(s), s \in S$ (and similarly for operator T).

Consider an initial value function $J_0 : S \to \mathbb{R}_0^+$. The DP algorithm hinges on the Bellman recursion which, for $s \in S$, operates as

$$(T^{k+1}J_0)(s) = \sup_{a\in A}\left\{g(s,a) + \gamma \sum_{s'\in S} P_a(s,s')(T^k J_0)(s')\right\}. \tag{2}$$

At the limit, the optimal value function satisfies the following fix-point equation:

$$J^*(s) = \sup_{a\in A}\left\{g(s,a) + \gamma \sum_{s'\in S} P_a(s,s')J^*(s')\right\}, \tag{3}$$

which is succinctly expressed as $J^* = TJ^*$ and known as the Bellman equation [3]. Given a non-trivial initial value function $J_0 : S \to \mathbb{R}_0^+$, the following convergence result holds: $J^*(s) = \lim_{k\to\infty}(T^k J_0)(s), s \in S$. The results above, focused on the characterisation of the optimal value function, also lead to the optimal policy μ^*.

The numerical solution of the discussed infinite-horizon decision problem hinges on the computation of the iterations in (2), or on the solution of the non-linear optimisation problem in (3), both of which can be computationally expensive when the cardinality of the state space $\mid S \mid$ is large. Several approaches have been developed to facilitate the numerical computation of optimal value functions and policies [5]. Two main schemes can be distinguished: value and policy iteration.

Value iteration boils down to iteratively computing applications of the optimal operator T, and exploiting monotonicity properties of the obtained value functions (in view of the operator's contractivity) to establish conditions for the convergence of the quantity $\lim_{k\to\infty} T^k J_0(s), s \in S$. Variants based on distributed and approximate iterations have also been developed [5]. We next focus on the alternative *policy iteration* scheme.

DP via Policy Iteration, Exact and Approximate. The policy iteration algorithm, which is proven to find the optimal policy in a finite number of steps, works as follows. Assume an initial (non-trivial) value function $J_0 : S \to \mathbb{R}_0^+$. Compute the corresponding optimal policy μ_0, which is such that $T_{\mu_0} J_0 = TJ_0$, namely compute

$$\mu_0(s) = \arg\sup_{a\in A} \left\{ g(s,a) + \gamma \sum_{s'\in S} P_a(s,s') J_0(s') \right\}.$$

This is known as the *policy update* step. The obtained policy μ_0 can be successively *evaluated* over a fully updated value function J_{μ_0}, which is such that $J_{\mu_0} = g_{\mu_0} + \gamma P_{\mu_0} J_{\mu_0}$ (as mentioned above). The scheme proceeds further by updating the policy as $\mu_1 : T_{\mu_1} J_{\mu_0} = TJ_{\mu_0}$; by later evaluating it via value function J_{μ_1}; and so forth until finite-step convergence.

We stress that the value update involved with the policy evaluation is in general quite expensive, and can be performed either as a recursive numerical scheme, or as a numerical solution of a linear system of equations. *Approximate policy iteration* schemes introduce approximations either in the policy update or in the policy evaluation steps, and are shown to attain suboptimal policies whilst ameliorating the otherwise computationally expensive exact scheme [4].

3 New Approximate Policy Iteration

We propose to speed up the above standard policy iteration scheme by accelerating the policy update step. The approach is inspired by recent work in [1], where a sequential and adaptive aggregation approach allows us to quantifiably

approximate the forward computation of the probability distribution in time (namely, the transient distribution) of a given Markov chain. In this work, we tailor this related work to the backward DP scheme, based on the fact that the policy evaluation steps work on a closed-loop model (indeed, a Markov chain obtained by selecting the currently optimal policy).

3.1 State-Space Aggregation for MDPs

We aggregate the MDP (S, A, P) into the abstract model $(\bar{S}, A, \bar{P})$ by a procedure that is inspired by our work in [1]. We partition $S = \cup_{i=1}^{m} S_i$, where the cardinality index m has been selected so that $m << n$, where again $n = |S|$. We denote the *abstract (aggregated) state space* as $\bar{S}$ and its elements (the *abstract states*) with $\phi_i, i = 1, \ldots, m$. Introduce the abstraction and refinement maps as $\alpha : S \to \bar{S}$ and $A : \bar{S} \to 2^S$, respectively – the first takes concrete points into abstract ones, whereas the latter relates abstract states to concrete partitioning sets. We argue that no abstraction of actions is needed at this stage, namely the aggregation of the MDP is performed for a given feedback function $\mu : S \to A$. For any pair of indices $i, j = 1, \ldots, m$, define the abstract transition probability matrix as

$$\bar{P}_{\mu}(\phi_i, \phi_j) \doteq \frac{\sum_{s \in A(\phi_i)} \sum_{s' \in A(\phi_j)} P_{\mu(s)}(s, s')}{|S_i|}.$$

This transition probability matrix can be de-aggregated piecewise constantly over the state space, as:

$$\forall s \in S_i, s' \in S_j, \quad \tilde{P}_{\mu}(s, s') = \frac{1}{|\ S_i\ |} \bar{P}_{\mu}(\phi_i, \phi_j).$$

Given an initial function $J_0(s), s \in S_i$, cluster it into $\bar{J}_0(\phi_i) = \frac{1}{|A(\phi_i)|} \sum_{s \in A(\phi_i)} J_0(s)$, where $\phi_i = \alpha(s)$, and de-cluster it into $\tilde{J}_0(s) = \bar{J}_0(\phi_i)$, for all $s \in A(\alpha(s))$. Similarly, given an initial policy μ_0, cluster the running reward function $g(s, \mu_0(s))$ (which is evaluated under a selected policy and thus only state dependent) into $\bar{g}(\phi_i) = \frac{1}{|A(\phi_i)|} \sum_{s \in A(\phi_i)} g(s, \mu_0(s))$, and later de-cluster it as $\tilde{g}(s) = \bar{g}(\phi_i)$. Given these definitions, the operators T_{μ}, T can then immediately aggregated as $\bar{T}_{\mu}, \bar{T}$.

Remark 2. The aggregation scheme described above can be alternatively implemented by selecting an arbitrary representative point within each partition $s^{\star} \in S_i$: $\bar{P}_{\mu}(\phi_i, \phi_j) \doteq \sum_{s' \in A(\phi_j)} P_{\mu(s^{\star})}(s^{\star}, s')$. This leads to formal connections with the notion of (forward) approximate probabilistic bisimulation [8]. □

3.2 Approximate Policy Iteration: Quantification and Use of Error Bounds

Approximate Policy Iteration. Algorithm 1 summarises the approximate policy iteration scheme. On line 2 the procedure performs an initial spatial aggregation based on an initial value function which, in the absence of alternatives,

Algorithm 1. Adaptive aggregation scheme for approximate policy iteration

Require: Finite MDP $M = (S, A, P)$, reward function g, initial policy μ_0, allowable error θ
Ensure: $\forall s \in S$ global error $\vec{E}(s) \leq \theta$
1: $policyTerm \leftarrow$ false; $valueTerm \leftarrow$ false; $\mu \leftarrow \mu_0$; $J \leftarrow g$
2: $(\bar{S}, \bar{P}_\mu, \bar{g}, \bar{J}, \vec{X}, \vec{Y}, Z) \leftarrow$ initAggregation (S, P_μ, g, J)
3: **while** $(!policyTerm)$ **do**
4: **while** $(!valueTerm)$ **do** ▷ Approximated policy evaluation
5: $(\bar{J}, \vec{E}, valueTerm) \leftarrow$ updateValues$(\bar{S}, \bar{P}_\mu, \bar{g}, \bar{J}, \vec{X}, \vec{Y}, Z)$
6: **if** $\vec{E} \geq \theta$ **then** ▷ Maximal error has been reached
7: $(\bar{S}, \bar{P}_\mu, \bar{g}, \vec{X}, \vec{Y}, Z) \leftarrow$ reAggregation $\left(S, P_\mu, g, \bar{J}, \vec{E}\right)$
8: $\bar{J} \leftarrow$ aggregate$(\bar{S}, J)$ ▷ Restart the policy iteration
9: $valueTerm \leftarrow$ false
10: $J \leftarrow$ deAggregation$(\bar{J})$; $valueTerm \leftarrow$ false
11: $(\mu, policyTerm) \leftarrow$ updatePolicy(P, J, μ) ▷ Policy update step
12: $(\bar{P}_\mu, \bar{g}, \vec{X}, \vec{Y}, Z) \leftarrow$ updateAggregatedSystem $\left(\bar{S}, P_\mu, g\right)$

is taken to be equal to the reward function, $J_0 = g$, and on an initial policy μ_0 (the choice of which is also quite arbitrary). The procedure builds the aggregated system comprising state space $\bar{S}$, transition matrix $\bar{P}_{\mu_0}$, value function $\bar{J}_0$, and reward function $\bar{g}$. The procedure also updates auxiliary data structures (for the quantities $\vec{X}, \vec{Y}$ and Z, to be introduced in Sect. 3.3) that are required for the computation of the error bounds $\vec{E}$. Further, the procedure named updateValues (line 5) performs policy evaluation by means of value function updates, namely it updates the aggregated value function based on the current aggregated policy. Note that this procedure introduces an approximation error (as further elaborated in the next section): as such, it also updates the vector of error bounds $\vec{E}$ and checks if the termination criterion for the value iteration is reached.

If the max allowable error bound θ has been exceeded before the termination criterion is met, the closed-loop MDP is re-aggregated (based on the error, as per line 7) and the policy evaluation step is restarted. Note that the adaptive re-aggregation step employs the current value function $\bar{J}$ and the current error $\vec{E}$, both of which reflect the model dynamics and the specific optimisation problem. In particular, the re-aggregation refines every cluster ϕ_i for which the current error $\vec{E}(\phi_i)$ is above the bound θ, and the new clustering takes $\bar{J}$ into consideration. The value function is reset using the values J corresponding to the last policy update.

If the value iterations terminate before the maximal error is reached, the final value function $\bar{J}$ is de-aggregated into J (line 9). Afterwards, the procedure updatePolicy updates the policy using the obtained J, and checks if a termination criterion over the policy update has been met. If not, before the next policy evaluation the aggregated system has to be updated: we retain the clustering $\bar{S}$ from the previous step, and thus only refresh (in view of the updated policy) the transition matrix, the reward function g, and the auxiliary data structures.

Approximate Policy Evaluation: Quantification of Error Bounds. Having obtained policy μ_k, its k-th step, policy evaluation performs the operation $\bar{J}_{k+1} := \bar{J}_k^{m_k}$, where $\bar{J}_k^{m_k} = \bar{T}_{\mu_k}^{m_k} \bar{J}_k$, and where m_k is a finite integer number accounting for the optimistic evaluation over the aggregated closed-loop operator $\bar{T}_{\mu_k}$. This update introduces two errors: the first is due to the aggregated computation; the second is due to the finite number of update steps (m_k). We then interpolate the obtained $\bar{J}_k^{m_k}(\bar{s})$ piecewise constantly over the concrete state space S, obtaining $\tilde{J}_k^{m_k}(s)$. We aim at comparing the following:

$$\left|\tilde{J}_k^{m_k}(s) - J_{\mu_k}(s)\right| \leq \left|\tilde{J}_k^{m_k}(s) - T_{\mu_k}^{m_k} J_k(s)\right| + \left|T_{\mu_k}^{m_k} J_k(s) - J_{\mu_k}(s)\right|. \tag{4}$$

Error bounds on the approximate evaluation of the current policy resort to the Bellman iteration. We introduce a number of terms $(\zeta_i^j(s), \xi_i(s), y_i(s)$, and the corresponding aggregated terms $Z^j, X, Y)$, which help in succinctly expressing parts of this iteration.

Definition 1. *Consider an MDP (S, A, P) with a fixed policy $\mu_k : S \to A$, and the aggregated MDP $(\bar{S}, A, \bar{P}_{\mu_k})$. Introduce the following quantities, $\forall s \in S_i, i \in \{1, \ldots, m\}$:*

$$\left|P_{\mu_k}(s, S_j) - \bar{P}_{\mu_k}(\phi_i, \phi_j)\right| = \zeta_i^j(s),$$

$$\left|J_k(s) - \tilde{J}_k(s)\right| = \left|J_k(s) - \frac{1}{\mid A(\alpha(s)) \mid} \sum_{s' \in A(\alpha(s))} J_k(s')\right| \leq \xi_i(s),$$

$$|g(s, \mu_k(s)) - \tilde{g}(s)| = \left|g(s, \mu_k(s)) - \frac{1}{A(\alpha(s))} \sum_{s' \in A(\alpha(s))} g(s', \mu_k(s'))\right| \leq y_i(s),$$

and further introduce

$$Z_i^j = \max_{s \in S_i} \zeta_i^j(s), \qquad Z^j = \max_{i=1,\ldots,m} Z_i^j,$$

$$X_i = \max_{s \in S_i} \xi_i(s), \qquad X = \max_{i=1,\ldots,m} X_i,$$

$$Y_i = \max_{s \in S_i} y_i(s), \qquad Y = \max_{i=1,\ldots,m} Y_i.$$

Theorem 1 (Error Bounds on Approximate Evaluation of a Given Policy). *A bound for Eq. (4) is the following:*

$$\left|\tilde{J}_k^{m_k}(s) - J_{\mu_k}(s)\right| \leq 2\mathcal{B}(m_k) + \mathcal{B}(m_k - 1) + (\tilde{J}_k^{m_k}(s) - \tilde{J}_k^{m_k - 1}(s)),$$

where

$$\mathcal{B}(m_k) = \sum_{i=0}^{m_k} \alpha^i Y + \alpha^{m_k} X + \sum_{i=0}^{m_k} \alpha^{m_k - i} \sum_{j=1}^{m} \bar{J}_k^i(\phi_j) Z^j.$$

Proof (Sketch). The desired upper bound on the error is obtained by first splitting it into two contributions:

$$\left|\tilde{J}_k^{m_k}(s) - J_{\mu_k}(s)\right| \leq \left|\tilde{J}_k^{m_k}(s) - T_{\mu_k}^{m_k} J_k(s)\right| + \left|T_{\mu_k}^{m_k} J_k(s) - J_{\mu_k}(s)\right|.$$

The first term results from performing the evaluation of policy μ_k over the aggregated model, and an upper bound is obtained from the three contributions, accounting respectively for the difference between concrete and aggregated running costs, initial value functions, and dynamics (namely transition probability matrices).

On the other hand, the second term results from an optimistic policy evaluation, iterating over value functions only a finite (m_k) number of times. The error can be obtained from [5, Chapter 1] and, importantly, fully computed over the abstract model. □

Remark 3. We comment on the asymptotics of the two contributions to the total error. The first contribution to the error in the previous proposition is bounded as the number of steps m_k grows, whereas the second term decreases exponentially. It might be meaningful to seek an empirical tradeoff, namely a parameter m_k minimising their sum. □

Within a single iteration of the policy iteration scheme, Theorem 1 has established an explicit bound on the approximate policy evaluation part. We are interested in assessing the sub-optimality of the policy obtained upon convergence of the approximate policy iteration scheme.

Theorem 2 (Bounds on Sub-optimality of Approximate Policy Iteration). *Assume that after a finite number of steps p a steady-state policy μ_p is obtained. Compute the upper bound δ on the error related to the approximate policy evaluation steps, namely $\delta = \max_{k=0,\ldots,p} \left|\tilde{J}_k^{m_k}(s) - J_{\mu_k}(s)\right|$. We obtain the following sub-optimality bound:*

$$\left|\tilde{J}_p^{m_p}(s) - J^*(s)\right| \leq \frac{2\gamma\delta}{1-\gamma},$$

where $\tilde{J}_p^{m_p}(s)$ is obtained from $\bar{J}_p^{m_p}$, and where $\bar{J}_p^{m_p} = \bar{T}_{\mu_p}^{m_p} \bar{J}_p$.

Proof. It follows from a straightforward adaptation of the results in [4, Section 3]. □

Remark 4. As a generalisation (relaxation of the assumptions) of the previous theorem, if no steady-state policy is attained, we obtain the following bound $\frac{2\gamma\delta}{(1-\gamma)^2}$. □

Remark 5. As a side remark notice that, within the iterative policy-update evaluation scheme, we do not need to account for a re-aggregation error (as in [1]), since this is already taken care by the initialisation of the policy evaluation scheme and the error terms X. Alternatively, we can avoid restarting the policy evaluation, which would reduce the re-aggregation overhead, and introduce a re-aggregation error as in [1]. □

Remark 6. Beyond policy evaluation, we can also perform an approximate version of policy update, and account for a global error, according to [5, Proposition 1.3.6]. □

3.3 Tighter and Computationally Faster Matrix Bounds

The error bounds given by Theorem 1 can be coarse, and thus not very useful in practice, since they may not adequately reflect the true empirical errors. The reason is that the error $\mathcal{B}(m_k)$ corresponding to the aggregation is state independent, and employs the global quantities X, Y and Z^j. In this section we will first improve the error bounds, then show how to approximate their computation to obtain a scheme that can speed up the overall DP algorithm.

As before we focus on the first iteration of policy evaluation. Define the matrix $Z \in \mathbb{R}^{m\times m}, Z_{ij} = Z_i^j$ and the column vector $\vec{X}(i) = X_i$. Then, of course, $\forall s \in S_i$ the third error (for the first value iteration) can be encompassed by $\gamma Z_{i\cdot}\bar{J}_0^0$, where $\bar{J}_0^0$ is a column vector and $Z_{i\cdot}$ is the i-th row of matrix Z. This leads to

$$\begin{aligned}\left|\tilde{J}_0^1(s) - J_0^1(s)\right| &\leq Y_i + \gamma P_{\mu_0}(s,\cdot)\vec{X} + \gamma Z_{i\cdot}\bar{J}_0^0\\ \text{(uniformly over } s) \quad &\leq Y_i + \gamma \bar{P}(\phi_i,\cdot)\vec{X} + \gamma Z_{i\cdot}\bar{J}_0^0,\end{aligned}$$

At the next (second) iteration, the error is

$$\begin{aligned}\left|\tilde{J}_0^2(s) - J_0^2(s)\right| &\leq (1+\alpha)Y_i + \gamma^2 \sum_{k=1}^{m} P_{\mu_0}^2(s,S_k)X_k + \gamma^2 \sum_{k=1}^{m} \bar{J}_0^0(\phi_k)Z^k + \gamma \sum_{j=1}^{m} \bar{J}_0^1(\phi_j)Z^j\\ &\leq (1+\alpha)Y_i + \gamma^2 P_{\mu_0}^2(s,\cdot)\vec{X} + \gamma^2 P_{\mu_0}(s,\cdot)Z\bar{J}_0^0 + \gamma Z_{i\cdot}\bar{J}_0^1\\ \text{(unif. over } s) \quad &\leq (1+\gamma)Y_i + \gamma^2 \bar{P}^2(\phi_i,\cdot)\vec{X} + \alpha^2 \bar{P}(\phi_i,\cdot)Z\bar{J}_0^0 + \gamma Z_{i\cdot}\bar{J}_0^1.\end{aligned}$$

Now, uniformising over $s \in S_j$ (i.e. over j-th cluster), we can directly write

$$\sup_{s\in S_j}\left|\tilde{J}_0^{m_0}(s) - J_0^{m_0}(s)\right| \leq \sum_{i=0}^{m_0} \gamma^i Y_j + \gamma^{m_0}\bar{P}^{m_0}(\phi_j,\cdot)\vec{X} + \sum_{i=0}^{m_0-1} \gamma^{m_0-i}\bar{P}^{m_0-i-1}(\phi_j,\cdot)Z\bar{J}_0^i, \tag{5}$$

where we have imposed that $\bar{P}^0(\phi_j,\cdot)Z = Z_{j\cdot}$.

Whilst providing tighter bounds, the computation of these formulas can be expensive due to the last term representing a number of matrix-matrix multiplications that is linear with the number of value function updates. We therefore introduce an approximate computation of the bounds, which combines the coarse and uniform bounds that can be easily computed, with the improved but expensive matrix bounds. The new computation attempts to make the approximate policy iteration practically useful, namely to provide considerable speedup of the computation whilst, at the same time, deriving informative bounds on the approximation error.

Define vectors $\vec{E}$ and $\vec{Y}$ such that $\vec{E}(j) = \sup_{s \in S_j} \left| \tilde{J}_0^{m_0}(s) - J_0^{m_0}(s) \right|$ and $\vec{Y}(j) = Y_j$. Based on Eq. (5) we obtain that

$$\vec{E} \leq \sum_{i=0}^{m_0} \gamma^i \vec{Y} + \gamma^{m_0} \bar{P}^{m_0} \vec{X} + \sum_{i=0}^{m_0-1} \gamma^{m_0-i} \bar{P}^{m_0-i-1} Z \bar{J}_0^i, \tag{6}$$

where $\leq$ is defined element-wise. These three terms can be approximated as follows:

$$\left(\sum_{i=0}^{m_0} \gamma^i \vec{Y} \right)(j) \leq \left(\sum_{i=0}^{B_1} \gamma^i \vec{Y} \right)(j) + \sum_{B_1+1}^{m_0} \gamma^i Y$$

$$\left(\gamma^{m_0} \bar{P}^{m_0} \vec{X} \right)(j) \leq \gamma^{m_0} X \text{ (if } m_0 \geq B_2)$$

$$\begin{aligned} \sum_{i=0}^{m_0-1} \gamma^{m_0-i} \bar{P}^{m_0-i-1} Z \bar{J}_0^i \leq & \sum_{i=m_0-B_3+1}^{m_0-1} \gamma^{m_0-i} \bar{P}^{m_0-i-1} Z \bar{J}_0^i \\ & + \sum_{i=0}^{m_0-B_3} \gamma^{m_0-i} \bar{P}^{B_3-1} Z \bar{J}_0^{m_0-B_3} \\ \leq & \sum_{i=m_0-B_3+1}^{m_0-1} \gamma^{m_0-i} \bar{P}^{m_0-i-1} Z \bar{J}_0^i \\ & + (m_0 - B_3 + 1) \gamma^{B_3} \bar{P}^{B_3-1} Z \bar{J}_0^{m_0-B_3}, \end{aligned}$$

where B_i for $i \in 1, 2, 3$ denotes three thresholds that affect the precision and time complexity of the computations. The approximation allows us to make the number of constant-vector, matrix-vector and matrix-matrix multiplications, required by the error computations, independent from the number of value function updates. Intuitively increasing these thresholds increases the precision, but also the time complexity. In our experimental evaluation we set $B_1 = B_2 = 10$ and $B_3 = 5$.

The first inequality holds, since $Y = \max_{i=1,\ldots,m} \vec{Y}(i)$. The second inequality holds, since $X = \max_{i=1,\ldots,m} \vec{X}(i)$ and P^{m_0} is a stochastic matrix. The third term in Eq. (6) is approximated as detailed next. The error related to the last, most significant, $B_3 - 1$ iterations is computed using the tighter matrix bounds (the first term in the right hand side of the last inequality). The error related to the first $k = m_0 - B_3 + 1$ iterations is approximated using a single vector obtained from the k-th iteration (the second term). The correctness follows from the monotonicity of J_0^k, i.e. $J_0^k \leq J_0^{k+1}$ (element-wise).

Finally, note that the computation of the bounds, as well as the policy evaluation itself, can be rewritten such that the expensive matrix-matrix multiplication can be replaced by matrix-vector multiplications.

4 Experimental Evaluation

We have developed a prototype implementation of the approximate policy iteration in PRISM [12]. Both the aggregated and the non-aggregated implementation use the explicit engine of PRISM for model construction and manipulation; however, the models are then translated into a sparse and fixed matrix representation, that is, a similar data structure as that used in the sparse engine, which is the fastest PRISM engine. We have run all experiments on a MackBook Pro™ with 2.9 GHz Intel Core i5 and 8 GB 1866 MHz RAM. Alongside memory usage, in the following experiments we report and compare runtimes for the policy iteration scheme, whereas the runtimes associated to the model construction, which are the same for aggregated and non-aggregated computations and hinge on the chosen engine in PRISM, are not included.

The practical performance of the proposed approximate policy iteration scheme depends on several related aspects. In our evaluation we attempt to dissect these aspects and identifying scenarios where our approach can achieve significant acceleration over the explicit algorithm, and, on the other hand, where it experiences practical performance limitations. We divide the experiments into two parts: (1) evaluation of the method for a fixed number of policy iterations (namely, policy updates/evaluations); and (2) evaluation of the convergence of the scheme.

We consider a case study from robot motion planning. The MDP model describes a finite two-dimensional discrete grid (say defined over integer variables $-D \leq x, y \leq D$), and deals with a robot moving over this map. The size of the state space thus is $|S| = (2D+1)^2$. The robot dynamics is affected via 5 actions (up, down, left, right, stay), which are not associated to fully deterministic moves, namely, there is a probability that performing a given action might result in an undesired output (e.g., for an action up the robot actually moves, say, to the right, as explained below).

We are interested in synthesising a policy that steers the robot to a specific point on the grid. We consider a reward function, which we seek to maximise over the infinite horizon over the available actions, that attains its maximum over the desired goal point. The reward function (which in this instance is independent of the actions) is embedded within a discounted, additive objective, of which we compute the expected value. As discussed earlier, the DP scheme will yield a memoryless, deterministic, and homogeneous policy as a function of the state space.

4.1 Fixed Number of Policy Updates and Evaluations

The required number of policy updates and evaluations (the latter obtained as value function updates) is key in the performance of the policy iteration scheme. This number depends on the structure of the system dynamics, which is affected by the aggregation procedure, and on parameters controlling the termination of the computations, which usually check the relative difference between consecutive updates. As such, we first assume that the number of the updates is the

same for both standard and approximate policy iterations, which allows us to assess the performance of the proposed aggregation scheme with respect to the following key aspects: (1) the number of value function updates, (2) the size of the model, (3) the discounting factor, and (4) the shape of the reward function. Later we evaluate how the state-space aggregation influences the convergence of the policy and value function updates.

We consider the following instance of the robotic case study (denoted as *Robotic 1*), where the actions are structured as follows: there is an 80 % chance of performing the intended action, i.e. moving to a position (x, y) and the remaining probability is uniformly distributed over four undesired local moves, namely $(x+1, y)$, $(x, y+1)$, $(x-1, y)$, and $(x, y-1)$. The reward function is defined as $g(x, y) = e^{-\frac{x^2+y^2}{\rho}}$, over a bounded range of integers x, y. Note that the parameter ρ affects the stiffness of the reward function. We have also added some obstacles to the map, and our experiments indicate that the map modification does not have a noticeable impact on the performance of the method, which demonstrates its robustness with respect to different motion planning scenarios.

Figures 1 and 2 illustrate the experimental outcomes. The curves display how the memory reduction factor and time speedup vary for different maximal error bounds. The maximal bound represents the threshold that controls the model re-aggregations, as per line 6 in Algorithm 1: whenever this threshold is reached, a model re-clustering is performed and the value function iterations for the policy evaluation are restarted.

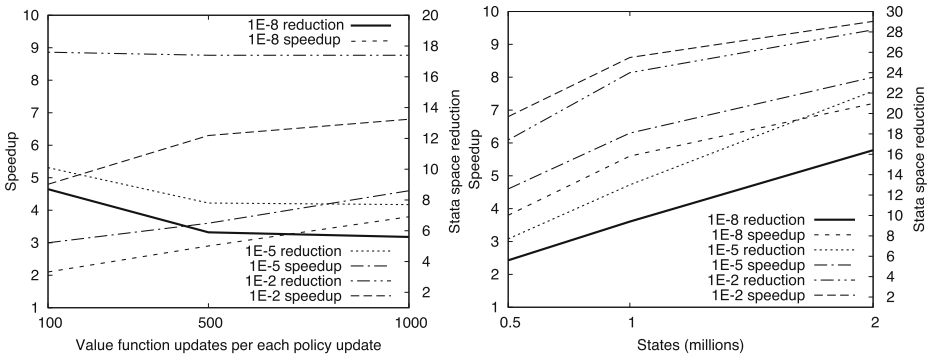

Fig. 1. Robotic 1 setup with the discounting factor $\gamma = 0.8$ and stiffness $\rho = 100$. **Left:** Fixed state space $|S| = 0.5$M and 50 policy updates. The figure shows results for different numbers of value function updates for a given policy update. The runtimes for non-aggregated computations over 100, 500 and 1000 value function updates are 137, 732 and 1174 s, respectively. **Right:** 50 policy updates and 1000 value function updates for a given policy update. The figure shows results for different sizes of the state space. The runtimes for non-aggregated computations over a model with $|S| = 0.5$M, 1M and 2M states are 1174, 2560 and 5571 s, respectively.

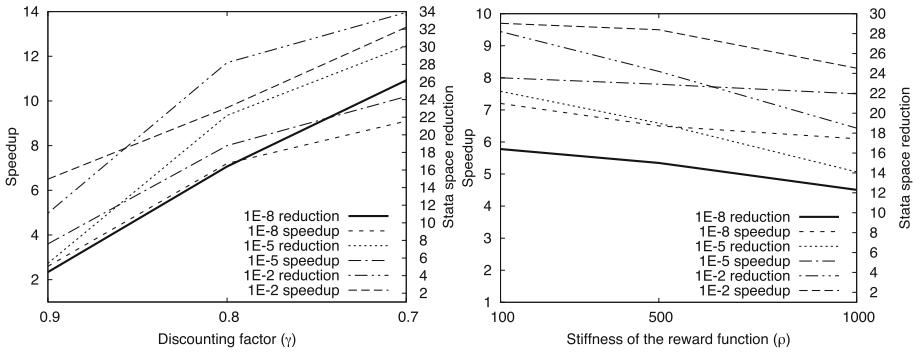

Fig. 2. Robotic 1 setup with $|S| = 2$M, 50 policy updates, 1000 value function updates per single policy update. **Left:** fixed stiffness $\rho = 100$: the figure shows results for different discounting factors γ. **Right:** fixed $\gamma = 0.8$: the figure shows results for different values ρ. Since the parameters γ and ρ have only a negligible impact on the runtimes of the non-aggregated computation, we report the runtime for the choice $\gamma = 0.8$, $\rho = 100$, which is 5571 s.

Figure 1 (left) shows how the average number of value function updates for a given policy update affect both the state-space reduction and the speedup. Since both the empirical errors and error bounds grow with increasing number of value function updates, we can observe a small decrease of the reduction factor. However, the trend becomes negligible later, likely in view of the convergence of the policy and value function updates. On the other hand, the speedup steadily increases, despite the decreasing state-space reduction: this is because the overhead related to every policy update, including updating the aggregated transition matrix and other data structures, becomes less relevant. Note that the number of re-aggregations decreases with number of policy updates, which also increases the speedup. For this case study the speedup saturates around 50 policy updates and 2000 value function updates per each policy update.

Figure 1 (right) confirms the scalability of our approach with respect to the state space size. Both the state reduction factor and the time speedup considerably grow with the increasing size of the model.

Figure 2 (left) illustrates the effect of the discounting factor γ on the policy iteration scheme. As expected, as the factor gets closer to the max value 1.0, both the empirical errors and the error bounds grow, and thus both the reduction factor and the speedup decrease. Factors above 0.9 limit the performance of our method, especially if a high precision is required: the current implementation of the aggregated scheme requires a high number of re-aggregations, which increases the overhead and results in a poor overall speedup. On the other hand, we do not consider factors below 0.7 since the model would converge too fast (faster than the 50/1000 policy/value function updates): still, the results indicate that better reduction factors would be achieved.

Finally, Fig. 2 (right) displays the robustness of our aggregation scheme against varying shapes of the reward function, where in our case study the stiffness of the reward is controlled by the parameter ρ. The current implementation of the aggregation strategy uses the average value of the reward function to adequately handle different shapes of the function. Note that for stiff functions we could additionally tune the aggregation strategy in order to provide better reduction and speedup, by states that are associated with small rewards (away from the maximum of the function) in large clusters.

4.2 Convergence of the Approximate Scheme

We consider a different variant of the case study (named *Robotic 2*), where for each action there is an 80 % probability that the robot does not move, a 15 % probability that the action has the intended effect, and the remaining probability is uniformly distributed over the four undesired outputs, similarly as in the previous variant of the model. The *Robotic 2* model displays slower dynamics and convergence to the optimum in the decision problem, and thus allows us to better evaluate how the state-space aggregation performs in time.

For both the aggregated and the non-aggregated computations, we use same termination criteria based on the difference between successive updates. In particular, the value function iteration (for policy evaluation) terminates if the values for all states in successive iterations differ by at most 1E-6, whereas the policy iteration terminates if there is no policy update in successive iterations, or if policy updates improve the value function by at most 1E-12. Note that these are the standard convergence thresholds used in PRISM for the numerical policy iteration scheme, and by decreasing them we slow down the overall convergence. This would improve the speedup of the adaptive scheme, due to a higher number of the value function updates and the policy updates: recall the result in Fig. 1 (left). The sub-optimality bounds for the non-aggregated computation are thus obtained as $\frac{2\gamma 1E{-}6}{1-\gamma}$, as per Theorem 2. Also note that there can be more than one optimal action over a state, so the difference in the policy does not necessarily correspond to an actual error.

Table 1 depicts the results for the discounting factor $\gamma = 0.85$ (top batch) and $\gamma = 0.95$ (bottom batch). The columns have the following meaning (from left to right): threshold on the maximal error bound $\mathcal{B}_{\max}$ for policy evaluation; maximal error bound $\mathcal{B}_{max}$; maximal empirical error $\mathcal{J}_{max}$ for policy evaluation; number of states that result in a different optimal action; global error bound $\mathcal{G}_{max}$ given by Theorem 2; reduction factor for memory usage; total number of policy updates and value function updates, respectively; and time speedup. We can see that, in all cases, decreasing the error bounds improves both the empirical errors and the optimality of the policy.

The top batch of the table demonstrates that, although the state-space reduction factor remains high for all three error thresholds, the overall time speedup is limited due to the low average number of value function updates (which can be run over the aggregated model) per policy update. As such, since in this case the convergence (overall number of iterations) is not affected by the reduction factor,

Table 1. Top: Robotic 2 setup with $|S| = 1$M, $\rho = 100$ and $\gamma = 0.85$: the non-aggregated computation has required 12/1.3K policy/value function updates, with a sub-optimality bound of 1.1E-5, and has taken 73 s. **Bottom:** Robotic 2 setup with $|S| = 1$M, $\rho = 100$ and $\gamma = 0.95$: the non-aggregated computation has required 55/11K policy/value function updates, with a sub-optimality bound of 3.8E-5, and has taken 674 s.

Errors					Aggregation		
Threshold	$\mathcal{B}_{max}$	$\mathcal{J}_{max}$	Policy	$\mathcal{G}_{max}$	Reduction	Iterations	Speedup
1E-2	2.6E-3	1.1E-5	10.5K	5.8E-2	33.7	13/1.3K	5.7
1E-5	7.9E-6	3.7E-8	4.5K	1.9E-4	30.1	10/1.1K	4.5
1E-8	6.1E-9	5.3E-11	0.5K	1.1E-5	27.4	15/1.6K	3.0
1E-2	9.5E-3	8.6E-6	36.5K	7.2E-1	22.2	22/8.6K	13.6
1E-5	7.2E-6	3.5E-8	11.9K	5.4E-4	13.2	44/21.5K	4.0
1E-8	7.9E-9	2.5E-11	1.3K	3.8E-5	8.1	55/31.0K	1.9

also the overall performance (i.e. the speedup with respect to the non-aggregated computation) is relatively stable.

The bottom part of the table shows that for a discounting factor closer to 1.0 the situation is different. In particular, both the reduction factor and the performance of the approximate policy iteration scheme downgrade with decreasing error bounds, whilst remaining faster than the iterations over the concrete model. In particular, for the error threshold 1E-2, the aggregation provides more than a 13-fold speedup, since the reduction factor is high and the approximate scheme converges faster (i.e. considerably fewer policy/value function updates are required). However, for lower error bounds both the reduction factor and the convergence speed decrease, which results in smaller speedups.

4.3 Discussion of the Experimental Results

Our experimental evaluation dissects important aspects of the DP algorithm that impact the performance metrics (i.e. reduction factors, convergence, precision, overall speedup) of the proposed approximate scheme. The experimental results clearly indicate that, for complex instances running over large state spaces and requiring a high number of policy and value function updates, our approximate scheme provides significant reduction of the computation time, while providing explicit bounds on the approximation errors. The maximal permissible error is specified by users and controls the tradeoff between the state-space reduction, which directly affects the speedup, and the precision of the computation in the form of maximal error of the value function.

The experiments further show that the overall performance of the method considerably depends on the aggregation strategy, namely, on a set of parameters and thresholds that control the aggregation. Intuitively, there is a tradeoff between the reduction vs. precision ratio and the overhead related to the re-aggregations. To provide a fair comparison we have run all experiments with the

same setting (except for the thresholds on the maximal error bounds). However, our observations show that a fine tuning of the parameters for a certain problem can lead to additional performance improvements.

The overhead related to updating the aggregated model (by means of its transition matrix), proved to have a significant impact on the overall performance. Therefore, a dynamic data structure implementing the model can improve the performance. Such a dynamic representation supports efficient local updates that are faster than global updates required by a static representation, and reduces the number of operations over the non-aggregated matrix. On the other hand, certain computations over dynamic structures (i.e. value function updates) might require additional overhead with respect to a static representation.

5 Conclusions and Future Work

In this article we have proposed a new approximate policy iteration scheme that mitigates the state-space explosion problem by adaptive state-space aggregation, at the same time providing rigorous and explicit error bounds that can be used to control the optimality level of the obtained policy.

The discussed approximate policy iteration scheme, and its associated error bounds, can be extended to approximate policy updates. This, on the one hand, would naturally incur an additional approximation error, but, on the other, would allow for a computational scheme completely based on aggregated (abstract) models.

References

1. Abate, A., Brim, L., Češka, M., Kwiatkowska, M.: Adaptive aggregation of Markov chains: quantitative analysis of chemical reaction networks. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 195–213. Springer, Heidelberg (2015). doi:10.1007/978-3-319-21690-4_12
2. Abate, A., D'Innocenzo, A., Benedetto, M.D.: Approximate abstractions of stochastic hybrid systems. IEEE Trans. Autom. Control **56**(11), 2688–2694 (2011)
3. Bertsekas, D.: Dynamic Programming and Optimal Control, vol. I. Athena Scientific, Belmont (1995)
4. Bertsekas, D.: Approximate policy iteration: a survey and some new methods. J. Control Theor. Appl. **9**(3), 310–335 (2011)
5. Bertsekas, D.: Dynamic Programming and Optimal Control, Vol. II: Approximate Dynamic Programming. Athena Scientific, Belmont (2012)
6. Bertsekas, D.: Tsitsiklis: Neuro-Dynamic Programming. Athena Scientific, Belmont (1996)
7. Brázdil, T., Chatterjee, K., Chmelík, M., Forejt, V., Křetínský, J., Kwiatkowska, M., Parker, D., Ujma, M.: Verification of Markov decision processes using learning algorithms. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 98–114. Springer, Heidelberg (2014). doi:10.1007/978-3-319-11936-6_8
8. D'Innocenzo, A., Abate, A., Katoen, J.-P.: Robust PCTL model checking. In: Proceedings of the HSCC 2012, pp. 275–285. ACM (2012)

9. Haesaert, S., Babuska, R., Abate, A.: Sampling-based approximations with quantitative performance for the probabilistic reach-avoid problem over general Markov processes. arXiv (2014). arXiv:1409.0553
10. Katoen, J.-P., Klink, D., Leucker, M., Wolf, V.: Three-valued abstraction for probabilistic systems. J. Logic Algebraic Program. **81**(4), 356–389 (2012)
11. Kattenbelt, M., Kwiatkowska, M., Norman, G., Parker, D.: A game-based abstraction-refinement framework for Markov decision processes. Formal Methods Syst. Des. **36**(3), 246–280 (2010)
12. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22110-1_47
13. Lahijanian, M., Andersson, S.B., Belta, C.: Formal verification and synthesis for discrete-time stochastic systems. IEEE Trans. Autom. Control **60**(8), 2031–2045 (2015)
14. McMahan, H.B., Likhachev, M., Gordon, G.J.: Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In: Proceedings of the ICML, pp. 569–576. ACM (2005)
15. Munos, R., Szepesvari, C.: Finite time bounds for fitted value iteration. J. Mach. Learn. Res. **9**, 815–857 (2008)
16. Puterman, M.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley and Sons, Hoboken (2005)

http://www.springer.com/978-3-319-46519-7

Automated Technology for Verification and Analysis
14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings
Artho, C.; Legay, A.; Peled, D. (Eds.)
2016, XI, 530 p. 102 illus., Softcover
ISBN: 978-3-319-46519-7