SwiftVI: Time-Efficient Planning and Learning with MDPs

Kasper Overgaard Mortensen [1] **Konstantinos Skitsas** [1] **Emil Morre Christensen** [1] **Mohammad Sadegh Talebi** [2]
Andreas Pavlogiannis [1] **Davide Mottin** [1] **Panagiotis Karras** [1 2]

Abstract

Markov decision process (MDPs) find application wherever a decision-making agent acts and learns in an uncertain environment from facility management to healthcare and service provisioning. However, the tasks of learning model parameters and planning the optimal policy such an agent should follow raise a high computational cost, calling for solutions that scale to large numbers of actions and states. In this paper, we propose SwiftVI, a suite of algorithms that plan and learn with MDPs scalably by organizing the set of actions for each state in priority queues and deriving bounds for backup Q-values. Our championed solution prunes the set of actions at each state utilizing a tight upper bound and a single priority queue. A thorough experimental study confirms that SwiftVI algorithms achieve high efficiency gains robustly to model parameters.

1 Introduction

A *Markov decision process* (MDP) features an agent who transitions from one state to another by choosing, in each state, an action that determines a reward as well as a next-state sampled from probability distribution, as in a *Markov process* (Gagniuc, 2017), hence affects future rewards through an induced probability distribution. The decision-making capacity of the agent raises the problem of determining how to act in each state to maximize an objective as a function of the collected rewards.

We consider finite infinite-horizon discounted MDPs, in which the agent's objective is to find an action selection rule applicable at any state—i.e., a *policy*—that maximizes the sum of future *discounted* rewards in expectation (Puterman, 2014). Such MDPs capture decision-making scenarios arising in many real-life systems and serve as a mathematical model underlying many reinforcement learning scenarios (Szepesvári, 2010). While the set of possible policies could become arbitrarily complex, a seminal result due to Bellman (1957) ensures the existence of a *stationary deterministic* optimal policy fully characterized by Bellman's optimality equations. These can be solved by the *value iteration* (VI) algorithm, which improves an approximate solution across iterations considering the interdependence among states and asymptotically converges to an optimal policy (Bellman, 1957; Howard, 1960). Nevertheless, the task becomes overwhelming as the number of possible actions grows. Despite its well-established convergence, the computational efficiency and robustness of the VI algorithm has been scarcely examined.

Besides its application in *known* MDPs, VI is extensively used as a routine in several provably-efficient reinforcement learning algorithms under different learning performance metrics and in various MDP settings, including the average-reward setting (Jaksch et al., 2010; Filippi et al., 2010; Bourel et al., 2020; Burnetas & Katehakis, 1997; Saber et al., 2024), the discounted setting (Strehl & Littman, 2008; Lattimore & Hutter, 2014; Li et al., 2024), and the episodic setting (Dann & Brunskill, 2015). Furthermore, VI plays a key role in discrete *structured* MDPs and their corresponding RL problems. For instance, it is used in algorithms for factored MDPs (Talebi et al., 2021), robust MDPs (Hau et al., 2023), MDPs with reward machines (Bourel et al., 2023). These works *flatten* the structured (non-tabular) MDP and apply a VI routine to obtain a near-optimal policy.

Contributions In this paper, we propose scalable value iteration algorithms that solve discounted MDPs, exploiting their monotonicity properties and apt initial values to prune the search space. In view of the widespread use of value iteration in a variety of reinforcement learning algorithms, we believe that our proposals have a direct impact on the practicability of those methods, enabling their

[1]Department of Computer Science, Aarhus University, Denmark [2]Department of Computer Science, University of Copenhagen, Denmark. Correspondence to: Panagiotis Karras <piekarras@gmail.com>.

Proceedings of the 8th MLSys Conference, Santa Clara, CA, USA, 2025. Copyright 2025 by the author(s).

application in time-critical problem instances and in environments of constrained computational resources. Besides, our contributions apply to any process that iteratively selects actions maximizing a value function over an MDP, such as the *policy improvement* step of *policy iteration* (Puterman, 2014; Howard, 1960). For the sake of concreteness, we present them in the context of value iteration.

2 Background and Related Work

A finite *discounted infinite-horizon MDP* M is a tuple $M = (S, A, P, R, \gamma)$, where S is a finite set of states called the state-space; $A = \cup_{s\in S} A(s)$ is the action-space, with $A(s)$ specifying the finite set of possible actions in state $s \in S$; $P : S \times S \times A \to [0, 1]$ is a transition function, with $P(s'|s, a)$ denoting the probability of transiting to $s' \in S$ upon executing $a \in A(s)$ in $s \in S$; $R : S \times A \to \mathbb{R}$ is a reward function, with $R(s, a)$ specifying the distribution of (possibly random) rewards when $a \in A(s)$ is selected in $s \in S$; $\gamma \in (0, 1)$ is a factor that discounts future rewards (Puterman, 2014). We consider uniformly bounded rewards, that is, $R^{\text{abs}}_{\max} \overset{\text{def}}{=} \max_{s\in S, a\in A(s)} \{|R(s, a)|\} < \infty$. The interaction with an MDP M proceeds as follows. At each time step $t \geq 0$, the agent is in state $s_t \in S$—with s_0 decided by nature—and chooses an action $a_t \in A(s_t)$ by some rule. Then, M generates a reward $r_t \sim R(s_t, a_t)$ and a next state $s_{t+1} \sim P(\cdot|s_t, a_t)$. The agent receives the reward (at once or in phases) before the next time step $t + 1$ begins, in which M transits to s_{t+1}.

A mapping $\pi : S \to A$ is called a stationary deterministic policy. The value function of π is a mapping $V^\pi : S \to \mathbb{R}$ defined as: for all $s \in S$, $V^\pi(s) = \mathbb{E}^\pi\big[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\,\big|s_0 = s\big]$, where $\mathbb{E}^\pi$ signifies that the expectation is taken with respect to trajectories generated by following π, i.e., over all infinitely long sequences $(s_t, a_t)_{t\geq 0}$, where $a_t = \pi(s_t)$ and $s_{t+1} \sim P(\cdot|s_t, a_t)$ for all t, with $s_0 = s$. The action-value function of π, also called its Q-value, is defined as follows: for all $s \in S$ and $a \in A(s)$, $Q^\pi(s, a) = \mathbb{E}^\pi\big[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\,\big|s_0 = s, a_0 = 0\big]$.

The optimal value is defined as $V^*(s) \overset{\text{def}}{=} \sup_\pi V^\pi(s)$ for any $s \in S$, where 'sup' is taken over all possible policies. Any policy that attains V^* is an optimal policy, denoted as π^*; hence, $V^{\pi^*} = V^*$. Furthermore, given $\varepsilon \geq 0$, an ε-optimal policy is any policy π such that $V^\pi(s) \geq V^*(s) - \varepsilon$ for all $s \in S$. We call such a V^π an ε-approximation to V^*.

Problem 1. *Given $M = (S, A, R, P, \gamma)$, find a policy π^* that* maximizes *V^π at each state. Further, given an accuracy ε, find a policy π such that $V^\pi(s) \geq V^*(s) - \varepsilon$ at each state s.*

Even though stationary deterministic policies constitute the simplest class of MDP policies, by a fundamental result, for any finite MDPs there exists an optimal policy which is stationary deterministic (Puterman, 2014).

The optimal value V^* satisfies the following: for all $s \in S$,

$$V^*(s) = \max_{a\in A(s)} \left\{ R(s, a) + \gamma \sum_{s'\in S} P\left(s'|s, a\right) \cdot V^*(s') \right\}. \quad (1)$$

Alternatively, one has: for all $s \in S$ and $a \in A(s)$,

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'\in S} P(s'|s, a) \max_{b\in A(s')} Q^*(s', b).$$

Equation (1), called *optimal Bellman equation*, amounts to a system of $|S|$ *fixed-point* equations, which admits a unique solution V^* (Bellman, 1957) and can be solved numerically via, e.g., value iteration (cf. Section 2.1). An optimal policy π^* corresponds to V^* in choosing, in each state s, the action maximizing future reward: for all $s \in S$,

$$\pi^*(s) = \arg\max_{a\in A(s)} \left\{ R(s, a) + \gamma \sum_{s'\in S} P\left(s'|s, a\right) \cdot V^*(s') \right\}.$$

2.1 The Value Iteration Algorithm (VI)

We begin with recalling the definition of Optimal Bellman Operator (Puterman, 2014):

Definition 2 (Optimal Bellman Operator)**.** *Given $V \in \mathbb{R}^{|S|}$, we define the operator $B : \mathbb{R}^{|S|} \to \mathbb{R}^{|S|}$, for all $s \in S$, as:*

$$BV(s) \overset{\text{def}}{=} \max_{a\in A(s)} \left\{ R(s, a) + \gamma \sum_{s'\in S} P\left(s' \mid s, a\right) \cdot V(s') \right\}$$

Recursively, $B^n : \mathbb{R}^{|S|} \to \mathbb{R}^{|S|}$ with $B^1 = B$ and $B^{n+1} = B \circ B^n$, $n \geq 1$.

By Definition 2, Equation (1) becomes $V^* = BV^*$; that is, the optimal value V^* is the (unique) fixed point of the optimal Bellman operator B. Applying B to any vector $V \neq V^*$ brings V closer to the optimal value V^* in the infinity-norm; this fact motivates us to define the *greedy policy* with respect to V as follows: for all $s \in S$:

$$\pi_V(s) \overset{\text{def}}{=} \arg\max_{a\in A(s)} \left\{ R(s, a) + \gamma \sum\nolimits_{s'\in S} P(s' \mid s, a) \cdot V(s') \right\}$$

Consequently, this fact naturally gives rise to the *value iteration (VI) algorithm* (Bellman, 1957), presented in Algorithm 1, which iterates the computation $V^{i+1} = BV^i$ for $i \in \mathbb{N}$, starting from arbitrarily chosen *initial values* $V^0 \in \mathbb{R}^{|S|}$, preferably close to the unknown optimal values V^*. Thus, VI yields a sequence $(V^i)_{i\geq 0}$ such that $V^i = BV^{i-1} = \cdots = B^i V^0$, converging to V^* (Puterman, 2014) for any $V_0 \in \mathbb{R}^{|S|}$: $\lim_{i\to\infty} V^i = V^*$. More formally, for $i \in \mathbb{N}_0$ and $V^i = BV^{i-1}$,

$$\max_{s\in S} \left\{ \left| V^i(s) - V^*(s) \right| \right\} \leq \gamma^i \max_{s\in S} \left\{ \left| V^0(s) - V^*(s) \right| \right\}. \quad (2)$$

Hence, for any $V_0 \in \mathbb{R}^{|S|}$, $\lim_{i\to\infty} B^i V^0 = V^*$. Besides these asymptotic convergence guarantees that lead to π^*, VI can stop according to another criterion (Puterman, 2014): given $\varepsilon > 0$, when the error between two consecutive iterations is $\|V^i - V^{i-1}\|_\infty \leq \varepsilon\frac{1-\gamma}{\gamma}$, then the output policy $\pi_{V^{\text{VI}}}$ (with $V^{\text{VI}} = V^i$) is guaranteed to be ε-optimal, i.e., $\|V^{\text{VI}} - V^*\|_\infty < \varepsilon$. Thus, it yields an ε-approximation to Problem 1.

Algorithm 1: Value Iteration (VI)

```
Data: MDP $M = (S, A, P, R, \gamma)$, threshold $\varepsilon$, initial $V^0 \in \mathbb{R}^{|S|}$
Result: An $\varepsilon$-approximation $\tilde{V}^*$ to $V^*$
1 $i \leftarrow 0$
2 repeat
3     $i \leftarrow i + 1$
4     forall $s \in S$ do
5         $V^i(s) \leftarrow \max_{a\in A(s)} \left\{ R(s,a) + \gamma \sum_{s'\in S} P(s' \mid s,a) \cdot V^{i-1}(s') \right\}$
6 until $\max_{s\in S} \{|V^i(s) - V^{i-1}(s)|\} < \varepsilon\frac{1-\gamma}{\gamma}$
7 return $\{V^i(s)\}_{s\in S}$
```

Evidently, $-\frac{R^{\text{abs}}_{\max}}{1-\gamma} \leq V^\pi(s) \leq \frac{R^{\text{abs}}_{\max}}{1-\gamma}$ for any $s \in S$ and π. This result leads to an upper bound on the number of iterations of VI:

Theorem 3. *If $V^0(s) \in [-R^{abs}_{\max}/1-\gamma, R^{abs}_{\max}/1-\gamma]$ for all $s \in S$, then Algorithm 1 conducts at most $\left\lceil \frac{\log(\epsilon\cdot(1-\gamma)) - \log(2R^{abs}_{\max})}{\log\gamma} \right\rceil$ iterations.*

In Section 3, we design more efficient solutions to Problem 1 that share a common core with the VI algorithm, as defined below:

Definition 4 (Value Iteration Skeleton)**.** *An algorithm adheres to the* value iteration skeleton *if it:*

1. iterates *over its core, possibly after pre-processing;*
2. *treats states as* equals*, as Algorithm 1 does (Line 4);*
3. *finds the same V^i, for all i, when starting with V^0;*
4. *provably returns an ε-approximation of V^*.*

The time complexity of VI is $O(I|S|^2|A|)$, considering all states and all actions per state in I iterations. Our proposals aim to find the maximizing action per state (Line 5) without considering all actions; some algorithms (Haddad & Monmege, 2014) calculate *two* VI instances, V^i_L and V^i_U. While the initial vector V^0 should preferably be close to V^*, the literature scarcely considers it, given that VI converges from *any* initial value (see (2)). Contrariwise, we utilize the initial V^0 to ensure the *monotonicity* of the $(V^i)_{i\geq 0}$ series.

2.2 Bounded Value Iteration (BVI)

The *bounded value iteration* (BVI) algorithm applies *interval iteration* (Haddad & Monmege, 2014; Brázdil et al., 2014; Baier et al., 2017) in a discounted infinite horizon context. The core idea of interval iteration is to maintain lower and upper bounds to the value function via value iteration. One VI instance, V^i_L, approaches V^* from lower values and another instance, V^i_U, approaches V^* from higher values, hence $V^i_L \leq V^* \leq V^i_U$. Thereby, apart from the convergence guarantee of VI – i.e., $\|V^{\text{VI}} - V^*\|_\infty < \varepsilon$ –, which we omit from the pseudocode for clarity, we utilize a *simpler* criterion: if $\max_{s\in S} |V^i_U(s) - V^i_L(s)| < 2\epsilon$, then, as Theorem 5 states, the mean of the two bounds is provably an ε-approximation of V^*.

Theorem 5. *If $V^i_L \leq V^* \leq V^i_U$ and $\|V^i_U(s) - V^i_L(s)\|_\infty < 2\varepsilon$, then $\|V^* - \tilde{V}^*\|_\infty < \varepsilon$, with $\tilde{V}^* \overset{\text{def}}{=} \frac{1}{2}(V^i_L + V^i_U)$. Hence, $\tilde{V}^*$ is an ε-approximation of V^*.*

Further, BVI calls for starting bounds V^0_L and V^0_U that ensure the *monotonicity* as the following theorem, adapted from (Baier et al., 2017), specifies.

Theorem 6. *Consider $V^0_L, V^0_U \in \mathbb{R}^{|S|}$. (i) if $V^0_L \leq V^*$ and $V^0_L \leq BV^0_L$, then $(B^n V^0_L)_{n\in\mathbb{N}}$ converges monotonically to V^*; and (ii) if $V^0_U \geq V^*$ and $V^0_U \geq BV^0_U$, then $(B^n V^0_U)_{n\in\mathbb{N}}$ converges monotonically to V^*.*

This theorem is easily derived from the proposition:

Proposition 7. *For $V_1, V_2 \in \mathbb{R}^{|S|}$, $V_1 \leq V_2$ implies $BV_1 \leq BV_2$.*

Proof. Let $s \in S$. Since $V_1(s) \leq V_2(s)$, we have

$$\begin{aligned} BV_1(s) &\leq \max_{a\in A(s)} \left\{ R(s,a) + \gamma \sum_{s'\in S} P(s' \mid s,a) V_2(s') \right\} \\ &= BV_2(s), \end{aligned}$$

hence $BV_1 \leq BV_2$. □

The criterion of Theorem 5 may hold before two consecutive iterations in an instance differ by at most $\frac{\varepsilon(1-\gamma)}{\gamma}$, as the condition $\|V^{\text{VI}} - V^*\|_\infty < \varepsilon$ requires; yet, as BVI doubles the computational cost per iteration, it is worthwhile when the savings by earlier convergence outweigh that overhead. Still, BVI serves as a starting point for our *action elimination* variants exploiting the monotonicity of V^i_L and V^i_U.

2.3 Action Elimination

We first introduce the *backup* of an action $a \in A(s)$ in state $s \in S$ with respect to value functions $V_L \leq V^* \leq V_U$:

Definition 8 (Backup action)**.** *For any $s \in S, a \in A(s)$,*

$$Q_U(s,a) \overset{\text{def}}{=} R(s,a) + \gamma \textstyle\sum_{s'\in S} P(s' \mid s,a) \cdot V_U(s')$$

$$Q_L(s,a) \overset{\text{def}}{=} R(s,a) + \gamma \textstyle\sum_{s'\in S} P(s' \mid s,a) \cdot V_L(s')$$

We say that we *back up* an action when we calculate one of the above Q-values and *back up* a state when we find the maximum Q-value over all actions, i.e., apply the Bellman operator. *Action elimination* (MacQueen, 1967; Puterman, 2014) is based on the following theorem:

Theorem 9 (Suboptimal action)**.** *Consider value vectors V_L and V_U such that $V_L \leq V^* \leq V_U$. If an action $a \in A(s)$ at state $s \in S$ has $Q_U(s,a) < \max_{a' \in A(s)} \{Q_L(s,a')\}$, then a is never chosen in the maximization step, i.e., it is* suboptimal.

Proof. From the assumption and the bounds it follows that:

$$\begin{aligned}
&R(s,a) + \gamma \sum_{s' \in S} P\left(s' \mid s,a\right) \cdot V_U(s') \\
&< \max_{a' \in A(s)} \left\{ R(s,a') + \gamma \sum_{s' \in S} P\left(s' \mid s,a'\right) \cdot V_L(s') \right\} \\
&\leq \max_{a' \in A(s)} \left\{ R(s,a') + \gamma \sum_{s' \in S} P\left(s' \mid s,a'\right) \cdot V^*(s') \right\} \\
&\overset{\text{def}}{=} V^*(s)\,.
\end{aligned}$$

Thus, the value gained by $a \in A(s)$ at s is *suboptimal*. □

The above bounds are based on *arbitrary* V_L and V_U. We apply *action elimination* within the VI skeleton of Definition 4 to discover suboptimal actions.

Corollary 10 (Suboptimal action—iterative)**.** *In iteration $i \geq 1$ of a VI skeleton-based algorithm by Definition 4, assume we have value functions V_L^{i-1} and V_U^{i-1}, such that $V_L^{i-1} \leq V^* \leq V_U^{i-1}$. If an action $a \in A^i(s)$ at state $s \in S$ has $Q_U^i(s,a) < \max_{a' \in A^i(s)} \{Q_L^i(s,a')\}$, then a is never chosen in the maximization step from iteration i onward, hence may be* eliminated *and disregarded in subsequent iterations as* suboptimal.

Action elimination thus enables the convergence criterion (Puterman, 2014):

Remark 11 (Action elimination stopping criterion)**.** *Let $A^i(s)$ be the reduced action set remaining, due to* action elimination*, in state s at iteration i; we can terminate VI in the first iteration i with only one action remaining in each state, $\forall s \in S : |A^i(s)| = 1$.*

2.4 Best-Actions Only (BAO)

The *Best Actions Only* (BAO) algorithm (Grzes & Hoey, 2011; 2012; 2013) keeps updating the *best actions* of each state until the maximum Q-value difference between two successive iterations, known as *Bellman error*, is sufficiently small and utilizes Theorem 6 to perform the VI maximization step *without backing up* all actions in each iteration. We build upon this idea in Section 3.1.

2.5 Model-based Reinforcement Learning

In reinforcement learning (Sutton & Barto, 2018), an agent tries to find a near-optimal policy in an environment with initially unknown transition and reward functions via its collected experience. In the *model-based* design paradigm, where an approximate MDP is maintained, value iteration arises as a subroutine that finds a near-optimal policy on the approximate model of the environment. Here, we consider online reinforcement learning in discounted MDPs, where the agent has no prior knowledge on the underlying environment, beyond the state-action space, and its goal is to learn an ε-optimal policy. This gives rise to the exploration-exploitation tradeoff: the agent must balance between *exploration* of the environment to gain new information and *exploitation* of past knowledge.

To demonstrate the applicability of our proposals in this settings, we focus on two model-based algorithms, MBIE (Strehl & Littman, 2005) and UCRLγ (Lattimore & Hutter, 2014), designed for this setting. Both algorithms implement the celebrated principle of *optimism in the face of uncertainty* in a model-based fashion. More precisely, they maintain high probability confidence sets for unknown transition function P and reward function R of the underlying MDP built around their empirical estimates $\widehat{P}$ and $\widehat{R}$. This leads to a continuum of MDPs, $\mathcal{M}$, that are defined on the same state-action space as M, while trapping M with high probability. To implement the abovementioned optimism principle, one must perform VI over $\mathcal{M}$, which leads to finding an optimal policy over all plausible environment models collected in $\mathcal{M}$. MBIE and UCRLγ differ mainly in the choice of confidence sets they maintain and some algorithmic aspects including the frequency policy computation happens. Both algorithms are *PAC-MDP*, i.e., they admit high-probability sample complexity bounds that depends polynomially on $|S|, |A|, 1/(1-\gamma), \varepsilon$, and $\log(1/\delta)$, where δ denotes an input failure probability (Strehl & Littman, 2008). Specifically, instead of conducting updates in fixed intervals, UCRLγ operates in contiguous blocks of time-steps with unfixed length, each each block ending with an update of the estimators of the single last observed state-action pair, based on observations accumulated since its last update. Thereafter, it updates the policy and starts a new block. To end a block, the number of observations of the last selected state-action pair must have doubled since its last update, or, in the first update, reached a minimum. We finally mention that both algorithms are *PAC-MDP*, i.e., they admit high-probability sample complexity bounds that depend polynomially on $|S|, |A|, 1/(1-\gamma), \varepsilon$, and $\log(1/\delta)$, where δ denotes an input failure probability (Strehl & Littman, 2008).

2.6 Monotonic Tight Bounds

We now outline tight initial lower and upper bounds, V_L^0 and V_U^0, for a VI-based algorithm by Definition 4 and show that they satisfy the monotonicity property of Theorem 6.

Objective 12 (Initial values objective)**.** *We need initial values V_L^0 and V_U^0 that satisfy the following requirements:*

1. $V_L^0 \leq V^* \leq V_U^0$;
2. *Lower V_L^0 and Upper bounds V_U^0 are computed efficiently from the MDP instance $M = (S, A, P, R, \gamma)$;*
3. $V_L^0 \leq BV_L^0$ *and* $V_U^0 \geq BV_U^0$ *as Theorem 6 assumes;*
4. $\|V_U^0 - V^*\|_\infty$ *and* $\|V_L^0 - V^*\|_\infty$ *should be small.*

Puterman (2014) shows that monotonicity follows when $V^0 = 0$ and all rewards are either non-negative or non-positive, and proposes initial bounds. We show that these bounds adhere to Objective 12, using the following MDP reward matrix variables.

Definition 13 (R-matrix quantities)**.** *For all $s \in S$, define:*

$$r^*(s) \overset{\text{def}}{=} \max_{a\in A(s)} \{R(s,a)\}$$
$$r^*_{\min} \overset{\text{def}}{=} \min_{s\in S}\{r^*(s)\} = \min_{s\in S}\left\{\max_{a\in A(s)}\{R(s,a)\}\right\}$$
$$r^*_{\max} \overset{\text{def}}{=} \max_{s\in S}\{r^*(s)\} = \max_{s\in S}\left\{\max_{a\in A(s)}\{R(s,a)\}\right\}$$

We note the asymmetry: whereas $r^*_{\max}$ *is* the overall maximum reward, $r^*_{\min}$ *is not* the overall minimum reward. We obtain these values by scanning the R-matrix. We henceforward use $\kappa \overset{\text{def}}{=} \frac{\gamma}{1-\gamma}$. We obtain initial bound values from the above quantities (Puterman, 2014) and prove them based on the policy-based definition of the optimal value.

Theorem 14. *For all $s \in S$, it holds that*

$$\tfrac{r^*_{\min}}{1-\gamma} \leq r^*(s) + \kappa \cdot r^*_{\min} \leq V^*(s) \leq r^*(s) + \kappa \cdot r^*_{\max} \leq \tfrac{r^*_{\max}}{1-\gamma}$$

Proof. Consider a concrete *lower bound policy*, $\pi_{\min}(s) \overset{\text{def}}{=} \arg\max_{a\in A(s)}\{R(s,a)\}$, which simply chooses the action of highest reward in each state, with value bounded as:

$$\begin{aligned} V^{\pi_{\min}}(s) &= \mathbb{E}^{\pi_{\min}}\left[\sum_{t=0}^{\infty} \gamma^t R\left(s_t, \pi_{\min}(s_t)\right) \Big| s_0 = s\right] \\ &\overset{\text{def}}{=} \mathbb{E}^{\pi_{\min}}\left[\sum_{t=0}^{\infty} \gamma^t \cdot r^*(s_t) \Big| s_0 = s\right] \\ &\geq r^*(s) + \sum_{t=1}^{\infty} \gamma^t \cdot r^*_{\min} = r^*(s) + \kappa \cdot r^*_{\min} \end{aligned}$$

The lower bound thus follows. On the other hand, for an arbitrary π, we have:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R\left(s_t, \pi(s_t)\right) \Big| s_0 = s\right] \\ &\leq r^*(s) + \mathbb{E}^{\pi}\left[\sum_{t=1}^{\infty} \gamma^t \cdot r^*_{\max}\right] = r^*(s) + \kappa \cdot r^*_{\max} \end{aligned}$$

The upper bound on the optimal value follows. □

We now show that these bounds can be used as V_L^0 and V_U^0 to ensure monotonicity with minimal pre-processing.

Proposition 15. *Assume $V_L^0(s) = \kappa \cdot r^*_{\min} + r^*(s)$ and $V_U^0(s) = \kappa \cdot r^*_{\max} + r^*(s)$, for all $s \in S$. Then, $BV_L^0 \geq V_L^0$ and $BV_U^0 \leq V_U^0$.*

Proof. It follows that:

$$\begin{aligned} BV_L^0(s) &= \max_{a\in A(s)}\left\{R(s,a) + \gamma \sum_{s'\in S} P\left(s' \,|\, s,a\right) \cdot V_L^0(s')\right\} \\ &= \max_{a\in A(s)}\left\{R(s,a) + \gamma \sum_{s'\in S} P\left(s' \,|\, s,a\right) \cdot (\kappa r^*_{\min} + r^*(s))\right\} \\ &= \max_{a\in A(s)}\left\{R(s,a) + \gamma \sum_{s'\in S} P\left(s' \,|\, s,a\right) \cdot r^*(s)\right\} + \tfrac{\gamma^2}{1-\gamma} r^*_{\min} \\ &\geq \max_{a\in A(s)} \{R(s,a)\} + \gamma \cdot r^*_{\min} + \tfrac{\gamma^2}{1-\gamma} r^*_{\min} \\ &= r^*(s) + r^*_{\min}\kappa = V_L^0(s) \end{aligned}$$

where we use that for all s, $r^*_{\min} \leq r^*(s)$ and $r^*_{\max} \geq r^*(s)$, and for all $a \in A(s)$, $\sum_{s'\in S} P\left(s' \mid s,a\right) = 1$. The proof for $BV_U^0(s)$ is derived similarly. □

3 Proposed Algorithms

Here, we introduce our algorithms for efficient VI.

3.1 VI with Heap Maximization (VIH)

We first present our main algorithm, *upper VI with heap maximization* (VIH), and the associated backup technique by *heap maximization*. The correctness of VIH rests on the monotonicity of a well-chosen initial upper value presented in Section 2.6. We commence with showing a consequence of a monotonically decreasing V_U^i on the *backup* of each action $a \in A(s)$ in state $s \in S$.

Theorem 16. *Under the assumptions of Theorem 6,*

$$\forall s \in S,\ \forall a \in A(s),\ \forall i \in \mathbb{N}_0 : Q_U^{i+1}(s,a) \leq Q_U^i(s,a)\,.$$

Proof. Theorem 6 asserts that $V_U^i \leq V_U^{i-1}$ for all $i \in \mathbb{N}_0$. Then, for any $s \in S$ and $a \in A(s)$,

$$Q_U^{i+1}(s,a) \overset{\text{def}}{=} R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) \cdot V_U^i(s')$$
$$\leq R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) \cdot V_U^{i-1}(s') \overset{\text{def}}{=} Q_U^i(s,a)$$

hence the theorem follows. □

We now show the crucial property that VIH utilizes to efficiently perform the maximization step in Line 5 of Algorithm 1.

Theorem 17. *If the assumptions of Theorem 6 hold such that V_U^i converges, monotonically decreasing, to V^* and for $s \in S$, $i \in \mathbb{N}_0$, $\exists a \in A(s)$ such that $\forall a' \in A(s)$, $\exists j_{a'} \leq i : Q_U^{j_{a'}}(s,a') \leq Q_U^i(s,a)$, then $Q_U^i(s,a) = \max_{a' \in A(s)} \left\{Q_U^i(s,a')\right\}$, hence we can return this value in Line 5 of Algorithm 1.*

Proof. By Theorem 16, for $j \leq i$, $Q_U^i(s,a') \leq Q_U^j(s,a')$ and thus if $\forall a' \in A(s), \exists j_{a'} \leq i$ with $Q_U^{j_{a'}}(s,a') \leq Q_U^i(s,a)$ then

$$\forall a' \in A(s) : Q_U^i(s,a') \leq Q_U^{j_{a'}}(s,a') \leq Q_U^i(s,a)$$

hence $Q_U^i(s,a) = \max_{a' \in A(s)} \left\{Q_U^i(s,a')\right\}$. □

By Theorem 17, if the *backup* value for action a at state s in iteration i, $Q_U^i(s,a)$, is not less than the backup of each other action $a' \in A(s)$ in some previous iteration $j_{a'} \leq i$, then we may stop backing up actions at state s, and assign $Q_U^i(s,a)$ to $V_U^i(s)$ in Line 6 of Algorithm 3. Thus we get the following state backup technique.

Observation 18 (State backup by heap maximization)**.** *At state $s \in S$ and for each $a \in A(s)$, we maintain in a heap data structure a value $\tilde{Q}_U(s,a) = Q_U^{j_a}(s,a)$ for some previous iteration j_a. In iteration $i \geq j_a$, we do the following:*

1. *Find the maximum $\tilde{Q}_U(s,a)$ over all $a' \in A(s)$, $\tilde{Q}_U(s,a) = \max_{a' \in A(s)} \left\{\tilde{Q}_U(s,a')\right\}$ and its associated $a \in A(s)$.*
2. *Back up $\tilde{Q}_U(s,a)$ to $Q_U^i(s,a) \overset{\text{def}}{=} R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) \cdot V_U^{i-1}(s')$.*
3. *Repeat until $\tilde{Q}_U(s,a)$ remains unchanged in two consecutive iterations; by Theorem 17, this is the maximum value at state s in iteration i, hence we set $V_U^i(s) \leftarrow \tilde{Q}_U(s,a)$.*

VIH only manages upper bounds V_U^i and $Q_U^i(s,a)$, and not V_L^i and $Q_L^i(s,a)$, as there are no lower-bound equivalents of Theorems 16 and 17. A lower-bound version of Theorem 17 would yield the *minimum* of $Q_L^i(s,a)$, which bears no effect to our problem.

Algorithm 2: Upper VI with Heap Maximization

```
   Data: An MDP $M = (S, A, P, R, \gamma)$, threshold $\varepsilon$
   Result: An $\varepsilon$-approximation $\tilde{V}^*$ to $V^*$
1  $r^*_{\min}, r^*_{\max}, \{r^*(s)\}_{s \in S} \leftarrow$ get_r_values($R$)
2  forall $s \in S$ do
3      $V_U^0(s) \leftarrow \frac{\gamma}{1-\gamma} \cdot r^*_{\max} + r^*(s)$
4      Make max-heap Heap$^s_{\max}$
         of $a \in A(s) : \left(Q_U^0(s,a) \leftarrow V_U^0(s), a\right)$ entries
5  $i \leftarrow 0$
6  repeat
7      $i \leftarrow i + 1$
8      forall $s \in S$ do
9          repeat
10             $a^{old}_{max} \leftarrow$ Heap$^s_{max}$.get_top_action()
11             $Q_U^i(s, a^{old}_{max}) \leftarrow R(s, a^{old}_{max}) + \gamma \sum_{s' \in S} P(s' \mid$
                 $s, a^{old}_{max}) \cdot V_U^{i-1}(s')$
12             Heap$^s_{max}$.insert_lower_max_q_value($Q_U^i(s, a^{old}_{max})$)
13             $a^{new}_{max} \leftarrow$ Heap$^s_{max}$.get_top_action()
14         until $a^{old}_{max} = a^{new}_{max}$
15         $V_U^i(s) \leftarrow$ Heap$^s_{max}$.get_top_value()
16 until $\max_{s \in S} \left\{|V_U^i(s) - V_U^{i-1}(s)|\right\} < \varepsilon \frac{1-\gamma}{\gamma}$
17 return $\left\{V_U^i(s)\right\}_{s \in S}$
```

Algorithm 2 presents the VIH algorithm, which leverages the four properties in Objective 12, Theorem 17, and a max-heap structure to achieve efficient value iteration. In Line 14, it determines termination by comparing actions, eschewing unreliable value equality checks. To ensure that it re-selects the same action in consecutive calls in case of maximum Q-value ties, it orders tied actions by IDs. Yet VIH computes the same $V^i(s)$ values as standard VI starting from the same initial values, as it finds the same maximum value in each iteration. VIH organizes the $\tilde{Q}_U(s,a)$ values, $a \in A(s)$, in each state $s \in S$ in a max-heap data structure (Cormen et al., 2009) of value-action pairs, which allows for efficient retrieval of $\max_{a \in A(s)} \left\{\tilde{Q}_U(s,a)\right\}$ and an associated maximizing action $a_{\max} \in \arg\max_{a \in A(s)} \left\{\tilde{Q}_U(s,a)\right\}$.

It is noteworthy that the VIH algorithm performs *implicit* action eliminations. We say that an action $a \in A(s)$ in state s is *implicitly eliminated* in some iteration i before convergence if a does not contribute to any computation in any iteration after iteration i, i.e., in VIH terms, does not participate in any max-heap re-organization operation after it finds itself at the top of the heap and has its value decreased. As more actions are implicitly eliminated, the VIH algorithm performs progressively fewer and more lightweight re-organization operations of that kind, hence gaining efficiency. Starting from the following section, we propose algorithms that eliminate actions *explicitly*, aiming to avoid action backups while still finding correct maximum Q-values.

3.2 VI with Action Elimination (VIAE)

We now enrich the BVI algorithm with an action-elimination provision using bounds directly corresponding to VI instances. As Corollary 10 states, if an action $a \in A(s)$ is never chosen at state s in a future iteration's maximization step, we may prune a from $A(s)$. Such pruning invests computational effort to prune actions and hence save resources in later iterations, which consider smaller action sets, while retaining the number of iterations intact. Algorithm 4 in Appendix A.2 shows the ensuing algorithm, *value iteration with action elimination* (VIAE), which targets the case where the sets of actions are large. VIAE obtains bounds for *each* state value using *two* VI instances, hence differs from the regular use of action elimination (Puterman, 2014), which derives a pruning threshold applicable to *all* states using the change of state values in the current iteration of a *single* VI instance.

3.3 Delayed Action Elimination, VIAEH

We now augment VIAE (Algorithm 4) to a heap-based version, VIAEH, using a max-heap and a min-heap to organize $\forall s \in S : \left\{\tilde{Q}_U(s,a)\right\}_{a\in A(s)}$ values, similarly to VIH (Section 3.1). We thus lower the computational load of each iteration, leading to the concept of *delayed action elimination*; while action elimination becomes less drastic, we expect that the overall runtime to be improved vs. VIAE. These structures return, upon request, the following pairs:

$$(\mathtt{v}_{max}, \mathtt{a}_{max}) \leftarrow \left(\max_{a\in A^{i-1}(s)}\left\{\tilde{Q}_U(s,a)\right\}, \arg\max_{a\in A^{i-1}(s)}\left\{\tilde{Q}_U(s,a)\right\}\right)$$

$$(\mathtt{v}_{min}, \mathtt{a}_{min}) \leftarrow \left(\min_{a\in A^{i-1}(s)}\left\{\tilde{Q}_U(s,a)\right\}, \arg\min_{a\in A^{i-1}(s)}\left\{\tilde{Q}_U(s,a)\right\}\right)$$

Algorithm 5 in Appendix A.3 shows how VIAEH backs up each state; it first finds $V_L^i(s)$ by regular VI (Lines 1–6) and $V_U^i(s)$ as in VIH (Lines 7–13); then it performs *delayed* action elimination, where it backs up the lowest-value action $\mathtt{a}_{min}$ (Lines 16 and 21) and, as long as its new Q-value qualifies, prunes $\mathtt{a}_{min}$ (Lines 14–22). By Theorem 16, the Q-value is monotonically non-increasing, thus, unlike the highest-value action, $\mathtt{a}_{min}$ remains the same when backed up. Notably, $\tilde{Q}_U(s,a)$ values serve a dual purpose in Algorithm 5: they serve to calculate $V_U^i(s)$ in Lines 7–13, where we find their maximum per state using a max-heap, and to prune actions in Lines 14–22, where we find their minimum per state using a min-heap.

Unlike VIAE (Algorithm 4), VIAEH (Algorithm 5) does not necessarily eliminate an action in the *earliest* iteration possible. In iteration i of VIAEH, the $\tilde{Q}_U(s,a)$ value for each $a \in A^{i-1}(s)$ may represent a backup $Q_U^{j_a}(s,a)$ from a *different previous iteration* $j_a \leq i$, while we only backup an action when it acquires the largest $Q_U^{j_a}(s,a)$ value (Lines 7–12), until the same action stays on top after backed up. The action elimination process (Lines 18–22), ends when we backup the current smallest value in Line 21 and thereafter find that $Q_U^i(s,\mathtt{a}_{min}) \geq Q_L^i$. Thus we cannot eliminate any actions based on the values available in the end of iteration i. Still, there may be an action a' that never assumed the minimum Q-value in the action elimination stage (Lines 18–22) and was thus not backed up in the current iteration, yet, had it been backed up, it would have attained a value $Q_U^i(s,a') < Q_L^i \leq Q_U^i(s,\mathtt{a}_{min})$. Thus, VIAEH may *miss* a chance to prune a' as VIAE would do.

3.4 LB-mirroring and the VIAEHL Algorithm

While the VIAEH backup (Algorithm 5) serves to eliminate actions, albeit delayed, it is impeded by the need to calculate V_L^i lower bounds via backing up all actions in the reduced action set $A^{i-1}(s)$ (Lines 1–6), where the heap-based technique used for V_U^i is inapplicable. We develop a variant, VIAEHL, that eschews backing up all actions in $A^{i-1}(s)$ and caters to lower bounds differently.

We have hitherto used V_L^i and V_U^i as single VI instances that can ensure an ε-approximation either individually, in view of $\|V^{\text{VI}} - V^*\|_\infty < \varepsilon$, or in unison, by Theorem 5. Yet we may facilitate action elimination via a less tight lower bound $\tilde{V}_L^i$, not being a VI instance, such that $\tilde{V}_L^i \leq V_L^i \leq V^* \leq V_U^i$: we may check convergence individually for the VI instance V_U^i and also perform action elimination using $\tilde{V}_L^i$. Using a less tight lower bound shall delay action elimination, yet we may gain efficiency by calculating $\tilde{V}_L^i$ rather than maintaining V_L^i. To calculate $\tilde{V}_L^i$ in each iteration i, we define the following operator.

Definition 19 (Bellman subset-operator)**.** *Given a restricted action-space $\tilde{A} = \cup_{s\in S}\tilde{A}(s)$ with $\tilde{A}(s) \subseteq A(s)$. We define the associated optimal Bellman operator to $\tilde{A}$, $B_{\tilde{A}} : \mathbb{R}^{|S|} \to \mathbb{R}^{|S|}$, for any $V \in \mathbb{R}^{|S|}$ and $s \in S$, as:*

$$B_{\tilde{A}}V(s) \overset{\text{def}}{=} \max_{a\in\tilde{A}(s)}\left\{R(s,a) + \gamma\sum_{s'\in S} P\left(s' \mid s,a\right)\cdot V(s')\right\}$$

Given initial $\tilde{V}_L^0$, we then define $\tilde{V}_L^i \overset{\text{def}}{=} B_{\tilde{A}^i}\tilde{V}_L^{i-1}$, where $\tilde{A}^i$ the action space of actions sets reduced due to pruning till iteration i.

Theorem 20 (Bellman subset-operator lower-bound invariance)**.** *Let $\tilde{V}_L^0 \overset{\text{def}}{=} V_L^0$. Then $\tilde{V}_L^i \leq V_L^i$ for all $i \in \mathbb{N}_0$.*

Proof. We construct the proof by induction. Assume that $\tilde{V}_L^{i-1} \leq V_L^{i-1}$. Then, for all $s \in S$,

$$\tilde{V}_L^i(s) \overset{\text{def}}{=} \max_{a\in\tilde{A}^i(s)}\left\{R(s,a) + \gamma\sum_{s'\in S} P\left(s' \mid s,a\right)\cdot \tilde{V}_L^{i-1}(s')\right\}$$
$$\leq \max_{a\in A(s)}\left\{R(s,a) + \gamma\sum_{s'\in S} P\left(s' \mid s,a\right)\cdot V_L^{i-1}(s')\right\} \overset{\text{def}}{=} V_L^i(s)$$

Since $\tilde{V}_L^0 \overset{\text{def}}{=} V_L^0$, the theorem follows inductively. $\square$

Accordingly, in each iteration i, we aim to *dynamically* choose a subset $\tilde{A}^i(s) \subseteq A^i(s)$ such that $|\tilde{A}^i(s)| \ll |A^i(s)|$ and $|V^i_L(s) - \tilde{V}^i_L(s)|$ is small, and the delay caused in action elimination by the less tight bound is small, so that any additional backups due to delayed action elimination are compensated by backups saved by using $\tilde{V}^i_L(s)$ rather than $V^i_L(s)$. We choose $\tilde{A}^i(s)$ using the information we get by calculating $V^i_U(s)$. In particular, we follow the backups in the heap for the sake of $V^i_U(s)$ and back up the *same* actions to find $\tilde{V}^i_L(s)$, as actions backed up by being at the top of the max-heap are likely to also have high $\tilde{Q}^i_L(s,a) \overset{\text{def}}{=} R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) \cdot \tilde{V}^{i-1}_L(s')$.

3.5 Learning

We now show how our methods can accelerate algorithms for learning near-optimal policies in environments described by finite discounted MDPs. Multiple such methods use Value Iteration as a subroutine to find an ε-approximate optimal policy for an incompletely known environment (Strehl & Littman, 2005; 2008; Auer et al., 2008; Lattimore & Hutter, 2014), running VI optimistically for an estimate $\tilde{M} = \left(S, A, \tilde{P}, \tilde{R}, \gamma\right)$ of the real unknown model M, where $\tilde{R} = \hat{R} + \beta_{n(s,a)}$ and $\left\|\tilde{P}(\cdot|s,a) - \hat{P}(\cdot|s,a)\right\|_1 \leq \beta'_{n(s,a)}$, with $\hat{R}$ and $\hat{P}$ being respective empirical estimates of R and P, and $\beta_{n(s,a)}$ and $\beta'_{n(s,a)}$ being respective method-specific confidence radii that decay with the number $n(s,a)$ of observations for state-action pair (s,a). From Definition 13 and Theorem 14, assuming that the confidence bounds hold, it follows that:

$$V^*(s) \leq r^*(s) + \kappa \cdot r^*_{\max} \leq \max_{a \in A(s)} \{\tilde{R}(s,a)\} + \kappa \cdot \max_{s \in S} \left\{ \max_{a \in A(s)} \{\tilde{R}(s,a)\} \right\},$$

which enables the use of our upper bound-based VI variants in the learning setting, without prior knowledge of R.

Value Iteration is likely to be a bottleneck for such learning methods. Each time-step may update the reward and transition estimates and hence the policy estimate. In time-critical RL applications, MBIE and similar approaches may be impractical due to the cost of repeated VI, and this problem exacerbates in quality-critical scenarios with low ε tolerance for the ε-approximate policy, as a low ε tightens the VI termination criterion, demanding further runtime.

4 Experimental Study

We conduct experiments to assess our proposals among each other, vs. the baselines Value Iteration (VI), Interval Value Iteration (IVI), Value Iteration with Upper Bound (VIU), and vs. the state-of-the-art BAO solution (Grzes & Hoey, 2011; 2012; 2013) and AncVI (Lee & Ryu, 2023), a VI method that leverages an *anchor point* to smoothen convergence. We implemented[1] all algorithms in C++ 17 and ran planning experiments on a 378GB Linux server with Intel(R) Xeon(R) E5-2687W v3 @3.10GHz and learning experiments on a 128GB RAM Ubuntu 24.04 machine with Intel® Core™ 13700HX @3.70Ghz. We initialize algorithms based on the optimal monotonic bounds discussed in Section 2.6 as required.

4.1 Results on Random Models

We try out randomly generated MDPs. Every MDP is based on a triple (S, A, SS) featuring numbers of states S, actions A, supported states SS, and a reward distribution. Each state has the same number of actions and each action the same number of supported states. The rewards matrix follows a normal (Gaussian) distribution (ND) with parameters $\mu = 1000, \sigma^2 = 10$. We choose specific supported states and transition probabilities at random.

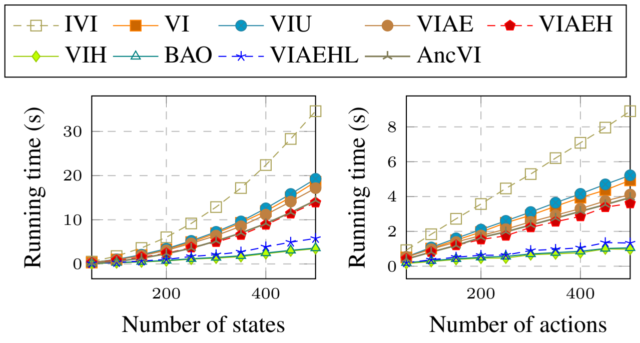

Figure 1. ND; time vs. states (10 A), actions (100 S), 10 SS.

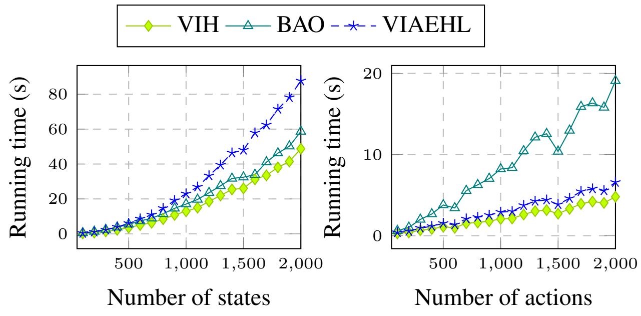

Figure 2. ND; time vs. states (100 actions) with $\frac{S}{10}$ SS, vs. actions (100 states) with 10 supported states.

Figure 1 plots the running time vs. number of states (left) and actions (right). Interval Value Iteration performs the worst, as its enhanced stopping criterion does not compensate for the double bound calculations. Plain VI and VIU are slower than action-elimination algorithms. VIAE improves over VI, and VIAEH, implemented with two heaps,improves over VIAE. Yet the top three top performers are VIAEHL, BAO, and VIH. VIH performs the best.

[1]https://github.com/constantinosskitsas/SwiftVI

Figure 2 presents the scalability of the three top-performers. BAO manages the growth of states relatively well, even under supported states per action growing proportionally to states. However, as the number of actions grows, VIAEHL and VIH outperform BAO.

4.2 Planning on Real-world Models

We next experiment on two real-world MDPs.

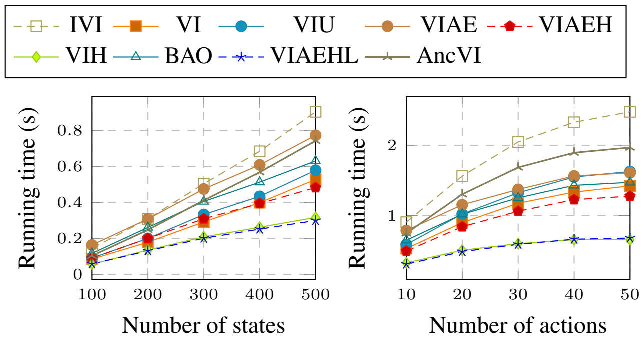

Figure 3. VM; time vs. states (10 actions), actions (500 states).

4.2.1 *Cloud management*

We first try an MDP for elastic cloud management (Lolos et al., 2017; Skitsas et al., 2022) described in Appendix A.4. Figure 3 presents our runtime results with increasing states and actions. Remarkably, VIAEHL and VIH are again the best-performing algorithms, even in this sparse-action setting, while VIAE without heap does not perform well. Besides, BAO is outperformed by regular VI, VIU, and all heap-based solutions. As the number of actions grows, BAO overtakes VIU only. VIAEHL and VIH stand out; VIAEHL has a slight advantage as the number of states grows, while usually the *implicit* action elimination of VIH dominated the *explicit* one of VIAEHL.

4.2.2 *Terrain-Maze MDP*

We try a Terrain-Maze MDP (Chen et al., 2021) in 2D and 3D. In 2D, we create a square grid with one terminal and up to 8 actions per state, one for each possible movement to a neighboring grid cell; we remove each action with probability 0.1, while ensuring each state has at least 2 actions. Given a selected action, the agent moves to the associated state with probability 0.8 and to one of up to 4 other neighboring locations with probability shared among them. We express terrain information via action costs, varying from -1 to -10 by a linearly decreasing distribution. In 3D, we use an additional axis z, with up to 26 actions per state and up to 10 supported states per action. Figure 4 shows our results. We exclude AncVI, which performs mediocrely in previous experiments. Simple action elimination falters, as there are too few actions with similar values to choose from and a few supported states per action. BAO is slow, as it keeps examining similarly-valued actions in each step. VIH achieves a tangible gain in 3D.

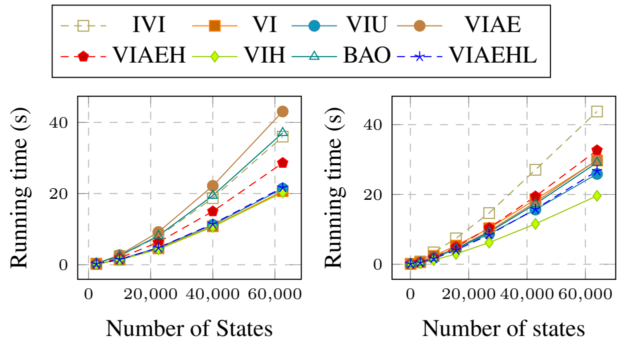

Figure 4. Terrain-Maze; left: 2D; right: 3D.

4.3 Learning on Incomplete Models

We apply VIH to the policy updates of MBIE and UCRLγ, to build algorithms SwiftMBIE and SwiftUCRLγ, respectively. For MBIE we further try a BAO-based variant to examine whether previously observed trends persevere. We set the initial upper bound value as in Section 3.5, and test the methods in incomplete MDP learning environments. We set discount factor $\gamma = 0.99$, failure probability $\delta = 0.01$ and report results for a range of ε choices.

4.3.1 *Results on incomplete random models*

We try learning incomplete random MDPs constructed as in Section 4.1 with the reward matrix following a Gaussian distribution (ND) with $\mu = 0.5$ and $\sigma^2 = 0.05$.

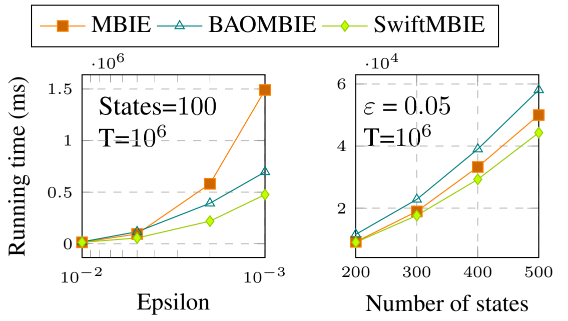

Figure 5. Runtime for MBIE variants vs. ε and number of states, Random MDPs (100 A, 50 SS); policy updates at each 10^4 steps.

MBIE. Figure 5 shows runtimes with MBIE variants on random MDPs. The first plot examines runtime vs. ε for 10^6 iterations on an MDP of 100 states. SwiftMBIE drastically outperforms MBIE at low ε and maintains a lead over BAOMBIE. Low ε calls for more value iterations, whereby confidence bounds tighten to multiple unique values, rendering heap-based updates more worthwhile. The second plot presents runtime vs. MDP size for 10^6 iterations with $\varepsilon = 0.05$. Here BAOMBIE trails the normal MBIE, while SwiftMBIE remains the fastest. As the algorithms are still exploring, heaps contain multiple similar values which must be all updated per iteration. We expect the SwiftMBIE addvantage to grow with more iterations.

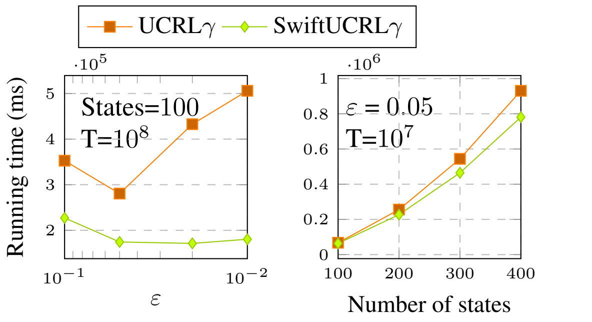

Figure 6. Runtime for UCRLγ variants vs. ε and number of states for Random MDPs (100 A and 50 SS).

UCRLγ. Figure 6 presents the runtime for UCRLγ variants on the same random MDPs, with increased time horizons to accommodate infrequent updates. The benefit of heap-based updates grows as ε diminishes, while a notable gap appears even with a shorter time-horizon.

4.3.2 Results on incomplete 4-Room Grids

We also try learning in a 4-room grid, where the agent starts from the top left corner and receives a reward of 1 at the bottom right corner. The agent chooses between actions, up, left, down, and right, with a probability of 0.3 to stay in the same state instead or slide perpendicularly to the chosen action. The agent stays in the same state if it hits a wall.

Figure 7 presents results for MBIE variants in grid world MDPs. For high ε, triggering fewer value iterations, BAOMBIE is slower than MBIE, yet SwiftMBIE stays comparable. As ε falls both outperform MBIE. We observed a similar trend for SwiftUCRLγ.

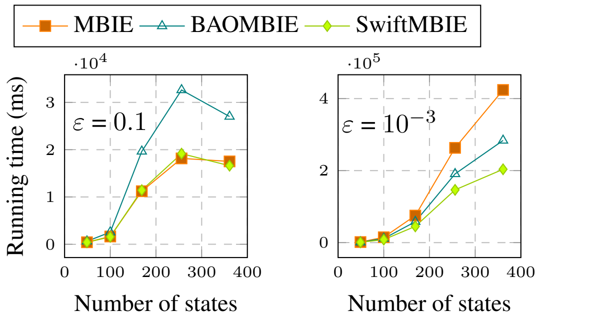

Figure 7. Runtime for MBIE variants in 4-Room Gridworld with different epsilon; $T = 10^6$; policy updates at each 10^4 steps.

4.3.3 Results on 4-Room Grids with known rewards

We also try 4-room grids with known rewards, using the upper bound of Section 2.6. Figure 8 presents results for MBIE variants with different update frequency. Interestingly, with more frequent (10^3) updates, SwiftMBIE performs worse at high $\varepsilon = 0.1$, as it updates often, before obtaining information for a sufficient amount of transitions, hence several values in the heap remain similar, as only one state provides a reward and must first be visited to impact other states. With less frequent updates (10^4), SwiftMBIE is the fastest. Figure 9 shows that SwiftUCRLγ remains faster than UCRLγ under known rewards. All variants achieve shorter runtimes, as known rewards tighten the initial bounds, leading to quicker updates.

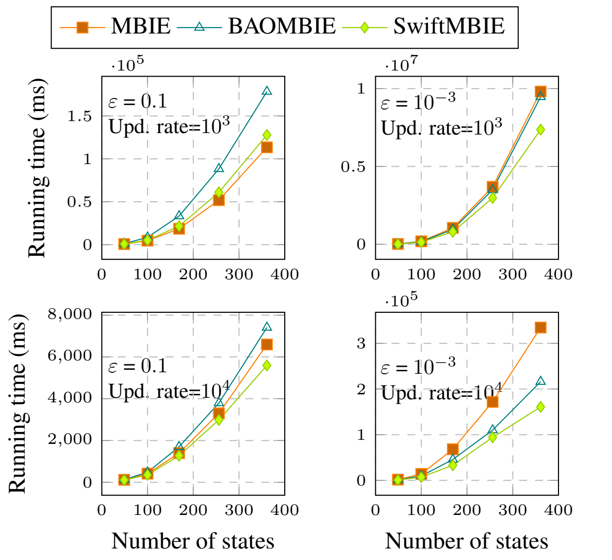

Figure 8. Runtime for MBIE variants for 4-Room Gridworlds with known reward; $T = 10^6$.

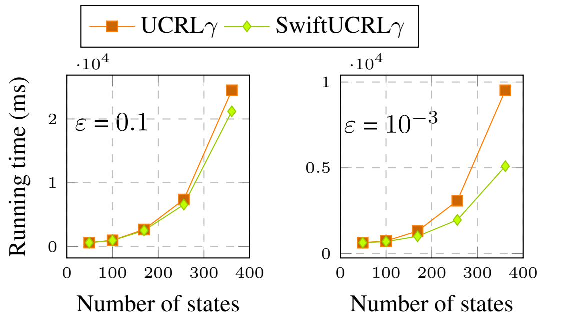

Figure 9. Runtime for UCRLγ variants for 4-Room Gridworlds with known reward; $T = 10^7$.

5 Conclusion

We proposed time-efficient algorithms for planning and learning with MDPs, which use heaps to avoid backing up all actions per iteration. We also crafted schemes that perform *action elimination* by deploying two VI instances and let lower-bound updates mirror upper-bound ones. In our experimental evaluation vs. baselines and the state-of-the-art solutions on diverse MDP instances for planning and learning, our proposals stand out in efficiency, while the mirroring variant has an advantage in larger state spaces.

Acknowledgements

Kasper Overgaard Mortensen and Konstantinos Skitsas are supported by the Independent Research Fund Denmark.

References

Auer, P., Jaksch, T., and Ortner, R. Near-optimal regret bounds for reinforcement learning. In *NeurIPS*, pp. 89–96, 2008.

Baier, C., Klein, J., Leuschner, L., Parker, D., and Wunderlich, S. Ensuring the reliability of your model checker: Interval iteration for markov decision processes. In *Computer Aided Verification (CAV)*, pp. 160–180, 2017.

Bellman, R. *Dynamic Programming*. Princeton University Press, 1st edition, 1957.

Bourel, H., Maillard, O., and Talebi, M. S. Tightening exploration in upper confidence reinforcement learning. In *ICML*, pp. 1056–1066, 2020.

Bourel, H., Jonsson, A., Maillard, O.-A., and Talebi, M. S. Exploration in reward machines with low regret. In *AISTATS*, pp. 4114–4146, 2023.

Brázdil, T., Chatterjee, K., Chmelik, M., Forejt, V., Kretínský, J., Kwiatkowska, M. Z., Parker, D., and Ujma, M. Verification of Markov decision processes using learning algorithms. In *ATVA*, pp. 98–114, 2014.

Burnetas, A. N. and Katehakis, M. N. Optimal adaptive policies for Markov decision processes. *Mathematics of Operations Research*, 22(1):222–255, 1997.

Chen, G., Gaebler, J. D., Peng, M., Sun, C., and Ye, Y. An adaptive state aggregation algorithm for markov decision processes. *CoRR*, abs/2107.11053, 2021.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms*. The MIT Press, 2009.

Dann, C. and Brunskill, E. Sample complexity of episodic fixed-horizon reinforcement learning. In *NeurIPS*, pp. 2818–2826, 2015.

Filippi, S., Cappé, O., and Garivier, A. Optimism in reinforcement learning and Kullback-Leibler divergence. In *48th Annual Allerton Conference*, pp. 115–122, 2010.

Gagniuc, P. A. *Markov chains : from theory to implementation and experimentation*. John Wiley & Sons, 2017.

Grzes, M. and Hoey, J. Efficient planning in R-max. In *AAMAS*, pp. 963–970, 2011.

Grzes, M. and Hoey, J. Analysis of methods for solving mdps. In *AAMAS*, pp. 1237–1238, 2012.

Grzes, M. and Hoey, J. On the convergence of techniques that improve value iteration. In *IJCNN*, pp. 1–8, 2013.

Haddad, S. and Monmege, B. Reachability in mdps: Refining convergence of value iteration. In *Reachability Problems - 8th International Workshop*, pp. 125–137, 2014.

Hau, J. L., Petrik, M., and Ghavamzadeh, M. Entropic risk optimization in discounted MDPs. In *AISTATS*, pp. 47–76, 2023.

Howard, R. A. *Dynamic Programming and Markov Processes*. MIT Press, 1960.

Jaksch, T., Ortner, R., and Auer, P. Near-optimal regret bounds for reinforcement learning. *The Journal of Machine Learning Research*, 11:1563–1600, 2010.

Lattimore, T. and Hutter, M. Near-optimal PAC bounds for discounted MDPs. *Theor. Comput. Sci.*, 558:125–143, 2014.

Lee, J. and Ryu, E. K. Accelerating value iteration with anchoring. In *NeurIPS*, pp. 53924–53963, 2023.

Li, G., Shi, L., Chen, Y., Chi, Y., and Wei, Y. Settling the sample complexity of model-based offline reinforcement learning. *The Annals of Statistics*, 52(1):233–260, 2024.

Lolos, K., Konstantinou, I., Kantere, V., and Koziris, N. Elastic management of cloud applications using adaptive reinforcement learning. In *IEEE BigData*, pp. 203–212, 2017.

MacQueen, J. A test for suboptimal actions in Markovian decision problems. *Operations Research*, 15(3): 559–561, 1967.

Puterman, M. L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Saber, H., Pesquerel, F., Maillard, O.-A., and Talebi, M. S. Logarithmic regret in communicating MDPs: Leveraging known dynamics with bandits. In *ACML*, pp. 1167–1182, 2024.

Skitsas, K., Papageorgiou, I. G., Talebi, M. S., Kantere, V., Katehakis, M. N., and Karras, P. SIFTER: space-efficient value iteration for finite-horizon mdps. *Proc. VLDB Endow.*, 16(1):90–98, 2022.

Strehl, A. L. and Littman, M. L. A theoretical analysis of model-based interval estimation. In *ICML*, pp. 856–863, 2005.

Strehl, A. L. and Littman, M. L. An analysis of model-based interval estimation for Markov decision processes. *J. Comput. Syst. Sci.*, 74(8):1309–1331, 2008.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Szepesvári, C. *Algorithms for reinforcement learning*. Springer Nature, 2010.

Talebi, M. S., Jonsson, A., and Maillard, O. Improved exploration in factored average-reward MDPs. In *AISTATS*, pp. 3988–3996, 2021.

A Appendix

A.1 Bounded Value Iteration

Algorithm 3 illustrates *bounded value iteration* (BVI).

Algorithm 3: Bounded Value Iteration (BVI)

```
Data: $M = (S, A, P, R, \gamma)$, threshold $\varepsilon$, $V_L^0(s), V_U^0(s)\, \forall s \in S$
Result: An $\varepsilon$-approximation $\tilde{V}^*$ to $V^*$
1  $i \leftarrow 0$
2  repeat
3      $i \leftarrow i+1$
4      forall $s \in S$ do
5          $V_L^i(s) \leftarrow \max_{a\in A(s)} \left\{ R(s,a) + \gamma \sum_{s'\in S} P(s' \mid s,a) \cdot V_L^{i-1}(s') \right\}$
6          $V_U^i(s) \leftarrow \max_{a\in A(s)} \left\{ R(s,a) + \gamma \sum_{s'\in S} P(s' \mid s,a) \cdot V_U^{i-1}(s') \right\}$
7  until $\max_{s\in S}\{V_U^i(s) - V_L^i(s)\} < 2\varepsilon$
8  return $\left\{\tilde{V}^*(s)\right\}_{s\in S} \leftarrow \left\{\frac{V_L^i(s)+V_U^i(s)}{2}\right\}_{s\in S}$
```

A.2 VI with Action Elimination

Algorithm 4 illustrates *value iteration with action elimination* (VIAE).

Algorithm 4: VI with Action Elimination

```
Data: An MDP $M = (S, A, P, R, \gamma)$, a precision threshold $\varepsilon$
Result: An $\varepsilon$-approximation $\tilde{V}^*$ to $V^*$
1  $r^*_{\min}, r^*_{\max}, \{r^*(s)\}_{s\in S} \leftarrow$ get_r_values($R$)
2  forall $s \in S$ do
3      $V_U^0(s) \leftarrow \frac{\gamma}{1-\gamma} \cdot r^*_{\max} + r^*(s)$; $V_L^0(s) \leftarrow \frac{\gamma}{1-\gamma} \cdot r^*_{\min} + r^*(s)$
4      $A^0(s) \leftarrow A(s)$
5  $i \leftarrow 0$
6  repeat
7      $i \leftarrow i+1$
8      forall $s \in S$ do
9          $Q_L^i \leftarrow -\infty$; $Q_U^i \leftarrow -\infty$            /* initialize as lowest possible value */
10         forall $a \in A^{i-1}(s)$ do
11             $Q_L^i(s,a) \leftarrow R(s,a) + \gamma \sum_{s'\in S} P(s,s',a) \cdot V_L^{i-1}(s')$
12             if $Q_L^i(s,a) > Q_L^i$ then
13                 $Q_L^i \leftarrow Q_L^i(s,a)$                         /* update new highest lower bound */
14             $Q_U^i(s,a) \leftarrow R(s,a) + \gamma \sum_{s'\in S} P(s,s',a) \cdot V_U^{i-1}(s')$
15             if $Q_U^i(s,a) > Q_U^i$ then
16                 $Q_U^i \leftarrow Q_U^i(s,a)$                         /* update new highest upper bound */
17         $A^i(s) \leftarrow A^{i-1}(s)$                           /* initialize new relevant set of actions */
18         forall $a \in A^{i-1}(s)$ do
19             if $Q_U^i(s,a) \leq Q_L^i$ then
20                 $A^i(s) \leftarrow A^i(s) \setminus \{a\}$                                /* prune actions */
21         $V_L^i(s) \leftarrow Q_L^i$; $V_U^i(s) \leftarrow Q_U^i$                             /* update bounds */
22 until $\max_{s\in S}\{V_U^i(s) - V_L^i(s)\} < 2\varepsilon$
23 return $\left\{\tilde{V}^*(s)\right\}_{s\in S} \leftarrow \left\{\frac{V_L^i(s)+V_U^i(s)}{2}\right\}_{s\in S}$
```

A.3 State Backup in VIAEH

Algorithm 5 shows how VIAEH backs up each state.

A.4 Cloud Management Platoform

The cloud management platform we study (Lolos et al., 2017; Skitsas et al., 2022) receives read requests. In each step, the coordinating agent may add or remove Virtual Machines (VMs) in the cluster to serve the incoming load. Model states express the current number of VMs and incoming load. We let the agent learn model parameters first and use them in evaluation period. We assume that the cluster size may vary from 1 to 50 Virtual Machines, while at any action the agent may add or remove VMs within capacity. The incoming load for training is a sinusoidal function, $load(t) = 50 + 50\sin\left(\frac{2\pi t}{250}\right)$; the frequency doubles in the evaluation period. The percentage of read request in the incoming load is given by the function $r(t) = 0.75 + 0.25\sin(\frac{2\pi t}{340})$. The RAM size is 1024 in the first 220 steps and 2048 for the next 220, and this pattern continues. The I/O operations per second are given by the function $io(t) = 0.6 + 0.4\sin\left(\frac{2\pi t}{195}\right)$. The capacity of the cluster at any time t is $capacity(t) = (10r(t) - io\_penalty - ram\_penalty)vms(t)$, where $vms(t)$ is the number of VMs in the cluster at time t; the parameter $io\_penalty$ is 0 when $io < 0.7$, $10io(t) - 0.7$ when $0.7 \leq io(t) \leq 0.9$ and 2 otherwise; $ram\_penalty = 0.3$ when the RAM size is 1024 and 0 otherwise. The reward for an action is:

$$reward(t) = \min(capacity(t+1), load(t+1)) - 2vms(t+1)$$

Thus, the agent is rewarded when the capacity of the system suffices to serve the load and penalized if it over- or under-delivers. After training, we treat the model as a fully observable MDP.

Algorithm 5: Backup of state s in VIAEH

```
    Data: State $s \in S$, $\left\{\tilde{Q}_U(s,a)\right\}_{a \in A^{i-1}(s)}$
 1  $Q^i_L \leftarrow Q^{i-1}_L$                                        /* initialize as lowest possible value */
 2  forall $a \in A^{i-1}(s)$ do
 3      $Q^i_L(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) \cdot V^{i-1}_L(s')$
 4      if $Q^i_L(s,a) > Q^i_L$ then
 5          $Q^i_L \leftarrow Q^i_L(s,a)$                               /* update new highest lower bound */
 6  $V^i_L(s) \leftarrow Q^i_L$                                         /* update lower bound */
 7  repeat
 8      $(\mathrm{v}^{old}_{max}, \mathrm{a}^{old}_{max}) \leftarrow \left(\max_{a \in A^{i-1}(s)} \left\{\tilde{Q}_U(s,a)\right\}, \arg\max_{a \in A^{i-1}(s)} \left\{\tilde{Q}_U(s,a)\right\}\right)$
 9      $Q^i_U(s, \mathrm{a}^{old}_{max}) \leftarrow R(s, \mathrm{a}^{old}_{max}) + \gamma \sum_{s' \in S} P(s' \mid s, \mathrm{a}^{old}_{max}) \cdot V^{i-1}_U(s')$
10      $\tilde{Q}_U(s, \mathrm{a}^{old}_{max}) \leftarrow Q^i_U(s, \mathrm{a}^{old}_{max})$      /* record upper bound */
11      $(\mathrm{v}^{new}_{max}, \mathrm{a}^{new}_{max}) \leftarrow \left(\max_{a \in A^{i-1}(s)} \left\{\tilde{Q}_U(s,a)\right\}, \arg\max_{a \in A^{i-1}(s)} \left\{\tilde{Q}_U(s,a)\right\}\right)$
12  until $\mathrm{a}^{old}_{max} = \mathrm{a}^{new}_{max}$
13  $V^i_U(s) \leftarrow \mathrm{v}^{new}_{max}$                       /* update upper bound */
14  $A^i(s) \leftarrow A^{i-1}(s)$                                      /* initialize new relevant set of actions */
15  $(\mathrm{v}_{min}, \mathrm{a}_{min}) \leftarrow \left(\min_{a \in A^i(s)} \left\{\tilde{Q}_U(s,a)\right\}, \arg\min_{a \in A^i(s)} \left\{\tilde{Q}_U(s,a)\right\}\right)$
16  $Q^i_U(s, \mathrm{a}_{min}) \leftarrow R(s, \mathrm{a}_{min}) + \gamma \sum_{s' \in S} P(s' \mid s, \mathrm{a}_{min}) \cdot V^{i-1}_U(s')$
17  $\tilde{Q}_U(s, \mathrm{a}_{min}) \leftarrow Q^i_U(s, \mathrm{a}_{min})$          /* record upper bound */
18  while $Q^i_U(s, \mathrm{a}_{min}) < Q^i_L$ do
19      $A^i(s) \leftarrow A^i(s) \setminus \{\mathrm{a}_{min}\}$                 /* prune away actions */
20      $(\mathrm{v}_{min}, \mathrm{a}_{min}) \leftarrow \left(\min_{a \in A^i(s)} \left\{\tilde{Q}_U(s,a)\right\}, \arg\min_{a \in A^i(s)} \left\{\tilde{Q}_U(s,a)\right\}\right)$
21      $Q^i_U(s, \mathrm{a}_{min}) \leftarrow R(s, \mathrm{a}_{min}) + \gamma \sum_{s' \in S} P(s' \mid s, \mathrm{a}_{min}) \cdot V^{i-1}_U(s')$
22      $\tilde{Q}_U(s, \mathrm{a}_{min}) \leftarrow Q^i_U(s, \mathrm{a}_{min})$      /* record upper bound */
```