Recursive Reinforcement Learning

Ernst Moritz Hahn* University of Twente

Mateo Perez* University of Colorado Boulder

Sven Schewe* University of Liverpool

Fabio Somenzi* University of Colorado Boulder

Ashutosh Trivedi* University of Colorado Boulder

Dominik Wojtczak* University of Liverpool

Abstract

Recursion is the fundamental paradigm to finitely describe potentially infinite objects. As state-of-the-art reinforcement learning (RL) algorithms cannot directly reason about recursion, they must rely on the practitioner's ingenuity in designing a suitable "flat" representation of the environment. The resulting manual feature constructions and approximations are cumbersome and error-prone; their lack of transparency hampers scalability. To overcome these challenges, we develop RL algorithms capable of computing optimal policies in environments described as a collection of Markov decision processes (MDPs) that can recursively invoke one another. Each constituent MDP is characterized by several entry and exit points that correspond to input and output values of these invocations. These recursive MDPs (or RMDPs) are expressively equivalent to probabilistic pushdown systems (with call-stack playing the role of the pushdown stack), and can model probabilistic programs with recursive procedural calls. We introduce *Recursive Q-learning*—a model-free RL algorithm for RMDPs—and prove that it converges for finite, single-exit and deterministic multi-exit RMDPs under mild assumptions.

1 Introduction

Reinforcement learning [36] (RL) is a stochastic approximation based approach to optimization, where learning agents rely on scalar reward signals from the environment to converge to an optimal behavior. Watkins's seminal approach [41] to RL, known as Q-learning, judiciously combines exploration/exploitation with dynamic programming to provide guaranteed convergence [40] to optimal behaviors in environments modeled as Markov decision processes (MDPs) with finite state and action spaces. RL has also been applied to MDPs with uncountable state and action spaces, although convergence guarantees for such environments require strong regularity assumptions. Modern variants of Q-learning (and other tabular RL algorithms) harness the universal approximability and ease-of-training rendered by deep neural networks [18] to discover creative solutions to problems traditionally considered beyond the reach of AI [30, 39, 33].

These RL algorithms are designed with a flat Markovian view of the environment in the form of a "state, action, reward, and next state" interface [9] in every interaction with the learning agent, where the states/actions may come from infinite sets. When such infinitude presents itself in the form of finitely represented recursive structures, the inability of the RL algorithms to handle structured environments means that the structure present in the environment is not available to the RL algorithm to generalize its learning upon. The work of [41] already provides a roadmap for hierarchically structured environments; since then, considerable progress has been made in developing algorithms for hierarchical RL [6, 13, 37, 31] with varying optimality guarantees. Still, the hierarchical MDPs

*Authors listed alphabetically.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

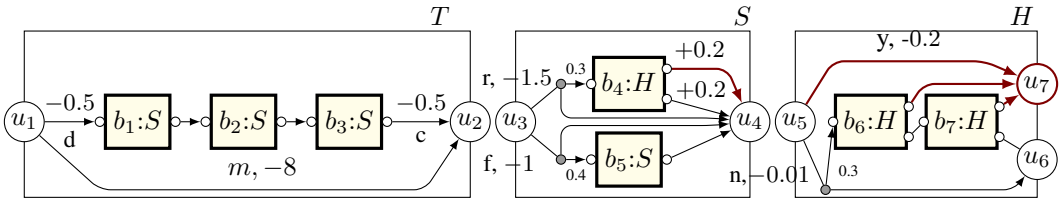


Figure 1: A recursive Markov decision process with three components T, S, and H.

are expressively equivalent to finite-state MDPs, although they may be exponentially more succinct (Lemma 1). Thus, hierarchical RL algorithms are inapplicable in the presence of unbounded recursion.

On the other hand, recursion occurs naturally in human reasoning [10], mathematics and computation [34, 35], and physical environments [29]. Recursion is a powerful cognitive tool in enabling a divide-and-conquer strategy [11] to problem solving (e.g., tower of Hanoi, depth-first search) and, consequently, recursive solutions enhance explainability in the form of intuitive inductive proofs of correctness. Unlike flat representations, the structure exposed by recursive definitions enables generalizability. Recursive concepts, such as recursive functions and data structures, provide scaffolding for efficient and transparent algorithms. Finally, the models of physical environments express the system evolution in the form of recursive equations. We posit that the lack of RL algorithms in handling recursion is an obstacle to their applicability, explainability, and generalizability. This paper aims to fill the gap by studying *recursive Markov decision processes* [16] as environment models in reinforcement learning. We dub this setting recursive reinforcement learning (RRL).

MDPs with Recursion. A recursive Markov Decision Process (RMDP) [16] is a finite collection of special MDPs, called component MDPs, with special entry and exit nodes that take the role of input parameter and return value, respectively. The states of component MDPs may either be the standard MDP states, or they may be "boxes" with input and output ports; these boxes are mapped to other component MDPs (possibly, the component itself) with some matching of the entry and exit nodes. An RMDP where every component has only one exit is called a 1-exit RMDP, otherwise we call it a general or multi-exit RMDP. Single-exit RMDPs are strictly less expressive than general RMDPs as they are equivalent to functions without any return value. Nonetheless, 1-exit RMDPs are more expressive than finite-state MDPs [32] and relate closely to controlled branching processes [17].

Example 1 (Cloud Computing)**.** *As an example of recursive MDP, consider the Boolean program shown in Figure 2. This example (inspired from [21]) models a cloud computing scenario to compute a task T depicted as the component T. Here, a decision maker is faced with two choices: either she can choose to execute the task monolithically (with a cost of 8 units) or chose to decompose the task into three S tasks. The process of decomposition and later combining the results cost 0.5 units. Each task S can either be executed on a fast, but unreliable server that costs 1 unit, but with probability 0.4 the server may crash, and require a recomputation of the task. When executed on a reliable server, the task S costs 1.5 units, however the task may be interrupted by a higher-priority task and the decision maker will be compensated for this delay by earning a 0.2 unit reward. During the interrupt service routine H, there is a choice to upgrade the priority of the task for a cost of 0.2 units. Otherwise, the interrupt service routing costs 0.01 unit (due to added delay) and the interrupt service routine itself can be interrupted, causing the service routine to be re-executed in addition to that of the new interrupt service routine. The goal of the RL agent is to compute an optimal policy that maximize the total reward to termination.*

This example is represented as a recursive MDP in Figure 1. This RMDP has three components T, S, and H. The component T has three boxes b_1, b_2, and b_3 all mapped to components S. The component T and S both have single entry and exit nodes, while the component H has two exit nodes. Removing the thick (maroon) transitions and the exit u_7 makes the RMDP 1-exit. The edges specify both, the name of the action and the corresponding reward. While the component T is non-stochastic, components S and H both have stochastic transitions depicted by the grey circle.

Recursive MDPs strictly generalize finite-state MDPs and hierarchical MDPs, and semantically encode countable state MDPs with the context encoding the stack frame of unbounded depth. RMDPs generalize several well-known stochastic models including stochastic context-free grammars [28, 26] and multi-type branching processes [22, 38]. Moreover, RMDPs are expressively equivalent to proba-

```
def T():
  a = ?({mono, divide});
  if a = mono:
    execute_mono() # −$8
  else :
    decompose() # −$0.5
    S()
    S()
    S()
    combine() # −$0.5
  return
```

(a) Task T

```
def S():
  a = ?({reliable, fast});
  if a = fast:
    try:
      execute_fast() # −$1
    except Crash: # prob = 0.4
      S()
  else:
    try:
      execute_reliable() #−$1.5
    except Interrupt: #prob = 0.3
      H() # +$0.2
  return
```

(b) Task S

```
def H():
  upgrade = ?({yes, no});
  if upgrade = no:
    try:
      ISR() # −$0.01
    except Interrupt: #prob = 0.3
      if (H()):
        return 1
      if (H())
        return 1
    return 0
  else : #−$0.2
    return 1
```

(c) Task H

Figure 2: A probabilistic Boolean program sketch where the choice of the hole (?) is to be filled by RL agent.

bilistic pushdown systems (MDPs with unbounded stacks), and can model probabilistic programs with unrestricted recursion. Given their expressive power, it is not surprising that reachability and termination problems for general RMDPs are undecidable. Single-exit RMDPs, on the other hand, are considerably more well-behaved with decidable termination [16] and total reward optimization under positive reward restriction [15]. Exploiting these properties, a convergent RL algorithm [21] has been proposed for 1-exit RMDPs with positive cost restriction. However, to the best of our knowledge, no RL algorithm exists for general RMDPs.

Applications of Recursive RL. Next, we present some paradigmatic applications of recursive RL.

- **Probabilistic Program Synthesis.** As shown in Example 1, RMDPs can model procedural probabilistic Boolean program. Hence, the recursive RL can be used for program synthesis in unknown, uncertain environments. Boolean abstractions of programs [4] are particularly suited to modeling as RMDPs. Potential applications include program verification [4, 5, 14] and program synthesis [19].
- **Context-Free Reward Machines.** Recently, reward machines [23] have been proposed to model non-Markovian reward signals in RL. In this setting, a regular language extended with the reward signals (Mealy machines) over the observation sequences of the MDP is used to encode reward signals. In this setting the RL algorithms operate on the finite MDP given by the product of the MDP with the reward machine. Following the Chomsky hierarchy, context-free grammars or pushdown automata can be used to provide more expressive reward schemes than regular languages. This results in context-free reward machines: reward machines with an unbounded stack. As an example of such a more expressive reward language, consider a grid-world with a reachability objective with some designated charging stations, where 1-unit dwell-time charges the unbounded capacity battery by 1-unit. If every action discharges the battery by 1-unit, the reward scheme to reach the target location without ever draining the battery cannot be captured by a regular reward machine. On the other hand, this reward signal can be captured with an RMDP, where charging by 1-unit amounts to calling a component and discharging amounts to returning from the component such that the length of the call stack encodes the battery level. More generally, any context-free requirement over finite-state MDPs can be captured using general RMDPs.
- **Stochastic Context-Free Grammars.** Stochastic CFGs and branching decision processes can capture a structured evolution of a system. These can be used for modeling disease spread, population dynamics, and natural languages. RRL can be used to learn optimal behavior in systems expressed using such stochastic grammars.

Overview. We begin the technical presentation by providing the formal definition of RMDPs and the total reward problem: which we show to be undecidable in general. We then develop PAC learning results under mild restrictions. In Section 3, we develop Recursive Q-learning, a model-free RL algorithm for RMDPs. In Section 4, we show that Recursive Q-learning converges to an optimal solution in the single-exit setting. Section 5 then demonstrates the empirical performance of Recursive Q-learning.

2 Recursive Markov Decision Processes

A Markov decision process $\mathcal{M}$ is a tuple (A, S, T, r) where A is the set of *actions*, S is the set of states, $T : S \times A \to \mathcal{D}(S)$ is the probabilistic transition function, and $r : S \times A \to \mathbb{R}$ is the reward function. We say that an MDP $\mathcal{M}$ is finite if both S and A are finite. For any state $s \in S$, $A(s)$ denotes the set of actions that may be selected in state s.

A recursive Markov decision process (RMDP) [16] is a tuple $M = (M_1, \ldots, M_k)$, where each *component* $M_i = (A_i, N_i, B_i, Y_i, \mathit{En}_i, \mathit{Ex}_i, \delta_i)$ consists of:

- A set A_i of actions;
- A set N_i of *nodes*, with a distinguished subset En_i of *entry* nodes and a (disjoint) subset Ex_i of *exit* nodes (we assume an arbitrary but fixed ordering on Ex_i and En_i);
- A set B_i of *boxes* along with a mapping $Y_i : B_i \mapsto \{1, \ldots, k\}$ that assigns to every box (the index of) a component. To each box $b \in B_i$, we associate a set of *call ports*, $\text{Call}_b = \{(b, en) \mid en \in \mathit{En}_{Y(b)}\}$, and a set of *return ports*, $\text{Ret}_b = \{(b, ex) \mid ex \in \mathit{Ex}_{Y(b)}\}$;
- we let $\text{Call}^i = \cup_{b \in B_i} \text{Call}_b$, $\text{Ret}^i = \cup_{b \in B_i} \text{Ret}_b$, and let $Q_i = N_i \cup \text{Call}^i \cup \text{Ret}^i$ be the set of all nodes, call ports and return ports; we refer to these as the *vertices* of component M_i.
- A transition function $\delta_i : Q_i \times A_i \to \mathcal{D}(Q_i)$, where, for each tuple $\delta_i(u, a)(v) = p$ is the transition probability of a transition from the source $u \in (N_i \setminus \mathit{Ex}_i) \cup \text{Ret}^i$ to the destination $v \in (N_i \setminus \mathit{En}_i) \cup \text{Call}^i$; we often write $p(v|u, a)$ for $\delta_i(u, a)(v)$.
- A reward function $r_i : Q_i \times A_i \to \mathbb{R}$ is the reward associated with transitions.

We assume that the set of boxes $B_1, \ldots, B_k$ and set of nodes $N_1, N_2, \ldots, N_k$ are mutually disjoint. We use symbols $N, B, A, Q, \mathit{En}, \mathit{Ex}, \delta$ to denote the union of the corresponding symbols over all components. We say that an RMDP is finite if k and all A_i, N_i and B_i are finite.

An execution of an RMDP begins at an entry node of some component and, depending upon the sequence of input actions, the state evolves naturally like an MDP according to the transition distributions. However, when the execution reaches an entry port of a box, this box is stored on a stack of pending calls, and the execution continues naturally from the corresponding entry node of the component mapped to that box. When an exit node of a component is encountered, and if the stack of pending calls is empty then the run terminates; otherwise, it pops the box from the top of the stack and jumps to the exit port of the just popped box corresponding to the just reached exit of the component. The semantics of an RMDP is an infinite state MDP, whose states are pairs consisting of a sequence of boxes, called the context, mimicking the stack of pending calls and the current vertex.

The height of the call stack is incremented (decremented) when a call (return) is made. A stack height of 0 refers to termination, while the empty stack has height 1.

The semantics of a recursive MDP $M = (M_1, \ldots, M_k)$ with $M_i = (A_i, N_i, B_i, Y_i, \mathit{En}_i, \mathit{Ex}_i, \delta_i, r_i)$ are given as a (infinite-state) MDP $[\![M]\!] = (A_M, S_M, T_M, r_M)$ where

- $A_M = \cup_{i=1}^{k} A_i$ is the set of actions;
- $S_M \subseteq B^* \times Q$ is the set of states, consisting of the stack context and the current node;
- $T_M : S_M \times A_M \to \mathcal{D}(S_M)$ is the transition function such that for $s = (\langle \kappa \rangle, q) \in S_M$ and action $a \in A_M$, the distribution $\delta_M(s, a)$ is defined as:
  1. if the vertex q is a call port, i.e. $q = (b, en) \in \text{Call}$, then $\delta_M(s, a)(\langle \kappa, b \rangle, en) = 1$;
  2. if the vertex q is an exit node, i.e. $q = ex \in \mathit{Ex}$, then if $\kappa = \langle \emptyset \rangle$ then the process terminates and otherwise $\delta_M(s, a)(\langle \kappa' \rangle, (b, ex)) = 1$ where $(b, ex) \in \text{Ret}(b)$ and $\kappa = \langle \kappa', b \rangle$;
  3. otherwise, $\delta_M(s, a)(\langle \kappa \rangle, q') = \delta(q, a)(q')$.
- the reward function $r_M : S_M \times A_M \to \mathbb{R}$ is such that for $s = (\langle \kappa \rangle, q) \in S_M$ and action $a \in A_M$, the reward $r_M(s, a)$ is zero if q is either a call port or the exit node, and otherwise $r_M(s, a)(\langle \kappa \rangle, q') = r(q, a)(q')$. We call the maximum value of the absolute one-step reward the *diameter* of an RMDP and denote it by $r_{max} = \max_{s,a} |r(s, a)|$.

Given the semantics of an RMDP M as an (infinite) MDP $[\![M]\!]$, the concepts of strategies (also called policies) as well as positional strategies are well defined. We distinguish a special class of strategies—called *stackless strategies*—that are deterministic and do not depend on the history or the stack context at all.

We are interested in computing strategies σ that maximize the *expected total reward*. Given RMDP M, a strategy σ determines sequences X_i and Y_i of random variables denoting the ith state and action of the MDP $[\![M]\!]$. The total reward under strategy σ and its optimal value are respectively defined as

$$\mathrm{ETotal}_\sigma^M(s) = \lim_{N\to\infty} \mathbb{E}_\sigma^M(s)\Big\{\sum\nolimits_{1\le i\le N} r(X_{i-1}, Y_i)\Big\}, \qquad \mathrm{ETotal}^M(s) = \sup_\sigma \mathrm{ETotal}_\sigma^M(s).$$

For an RMDP M and a state s, a strategy σ is called *proper* if the expected number of steps taken by M before termination when starting at s is finite. To ensure that the limit above exists, as the sum of rewards can otherwise oscillate arbitrarily, we assume the following.

Assumption 1 (Proper Policy Assumption)**.** *All strategies are proper for all states.*

We call an RMDP that satisfies Assumption 1 a *proper RMDP*. This assumption is akin to proper policy assumptions [7] often posed on the stochastic shortest path problems, and ensures that the total expected reward is finite. The expected total reward optimization problem over proper RMDPs subsumes the discounted optimization problem over finite-state MDPs since discounting with a factor λ is analogous to terminating with probability $1-\lambda$ at every step [36]. The properness assumption on RMDPs can be enforced by introducing an appropriate discounting (see Appendix F).

Undecidability. Given an RMDP M, an initial node v, and a threshold D, the *strategy existence problem* is to decide whether there exists a strategy in $[\![M]\!]$ with value greater than or equal to D when starting at the initial state $(\langle\emptyset\rangle, q)$, i.e., at some entry node q with an empty context.

Theorem 1 (Undecidability of the Strategy Existence Problem)**.** *Given a proper RMDP and a threshold D, deciding whether there exists a strategy with expected value greater than D is undecidable.*

PAC-learnability. Although it is undecidable to determine whether or not a strategy can exceed some threshold in a proper RMDP, the problem of ε-approximating the optimal value is decidable when parameters c_o, λ and b (defined below) are known. Our approach to PAC-learnability [1] is to learn the distribution of the transition function δ well enough and then produce an approximate, but not necessarily efficient, evaluation of our learned model.

To allow PAC-learnability, we need a further nuanced notion of ε-proper policies. A policy is called ε-proper, if it terminates with a uniform bound on the expected number of steps for all $\mathcal{M}'$ that differ from $\mathcal{M}$ only in the transition function, where $\sum_{q\in S, a\in A, r\in S} |\delta_{\mathcal{M}}(q,a)(r) - \delta_{\mathcal{M}'}(q,a)(r)| \le \varepsilon$ (we then say that $\mathcal{M}'$ is ε-close to $\mathcal{M}$), and where the support of $\delta_{M'}(q,a)$ is a subset of the support of $\delta_M(q,a)$ for all $q \in S$ and $a \in A$. An RMDP is called ε-proper, if all strategies are ε-proper for M for all states of the RMDP.

Assumption 2 (PAC-learnability)**.** *We restrict our attention to ε-proper RMDPs. We further require that all policies have a falling expected stack height. Namely, we require for all $\mathcal{M}'$ ε-close to $\mathcal{M}$ and all policies σ that the expected stack height in step k is bounded by some function $c_o - \mu \cdot \sum_{i=1}^{k} p_{\mathsf{run}}^{\mathcal{M}'_\sigma}(k)$, where $c_o \ge 1$ is an offset, $\mu \in]0,1]$ is the decline per step, and $p_{\mathsf{run}}^{\mathcal{M}'_\sigma}(k)$ is the likelihood that the RMDP $\mathcal{M}'$ with strategy σ is still running after k steps. We finally require that the absolute expected value from every strategy is bounded: $\left|\mathrm{ETotal}_\sigma^{\mathcal{M}'}\big((\langle\emptyset\rangle, q)\big)\right| \le b$ for some b.*

Theorem 2. *For every ε-proper RMDP with parameters c_o, μ, and b, $\mathrm{ETotal}^{\mathcal{M}}(s)$ is PAC-learnable.*

These parameters can be replaced by discounting. Indeed, our proofs start with discounted rewards, and then relax the assumptions to allow for using undiscounted rewards. Using a discount factor λ translates to parameters $b = \frac{d}{1-\lambda}$, $c_o = 1 + \frac{1}{1-\lambda}$, and $\mu = 1-\lambda$.

3 Recursive Q-Learning for Multi-Exit RMDPs

While RMDPs with multiple exits come with undecidability results, they are the interesting cases as they represent systems with an arbitrary call stack. We suggest an abstraction that turns them into a fixed size data structure, which is well suited for neural networks.

Given a proper recursive MDP $M = (M_1, \ldots, M_k)$ with $M_i = (A_i, N_i, B_i, Y_i, En_i, Ex_i, \delta_i, r_i)$ with semantics $[\![M]\!] = (A_M, S_M, T_M, r_M)$, the optimal total expected reward can be captured by the following equations $\text{OPT}_{\texttt{recur}}(M)$. For every $\kappa \in B^*$ and $q \in Q$:

$$y(\langle\kappa\rangle, q) \quad = \quad \begin{cases} y(\langle\kappa, b\rangle, en) & q{=}(b, en) \in \text{Call} \\ 0 & q \in Ex, \kappa = \langle\emptyset\rangle \\ y(\langle\kappa'\rangle, (b, q)) & q \in Ex, (b, q) \in \text{Ret}(b), \kappa{=}\langle\kappa', b\rangle \\ \max\limits_{a \in A(q)} \Big\{ r(q, a) + \sum\limits_{q' \in Q} p(q'|q, a) y(\langle\kappa\rangle, q') \Big\} & \text{otherwise.} \end{cases}$$

These equations capture the optimality equations on the underlying infinite MDP $[\![M]\!]$. It is straightforward to see that, if these equations admit a solution, then the solution equals the optimal total expected reward [32]. Moreover, an optimal policy can be computed by choosing the actions that maximize the right-hand-side. However, since the state space is countably infinite and has an intricate structure, an algorithm to compute a solution to these equations is not immediate.

To make it accessible to learning, we *abstract* the call stack $\langle\kappa, b\rangle$ to its *exit value*, i.e. the total expected reward from the exit nodes of the box b, under the stack context $\langle\kappa\rangle$. Note that when a box is called, the value of each of its exits may still be unknown, but it is (for a given strategy) fixed. Naturally, if two stack contexts $\langle\kappa, b\rangle$ and $\langle\kappa', b\rangle$ achieve the same expected total reward from each exit of the block b, then both the optimal strategy and the expected total reward, are the same.

This simple but precise and effective abstraction of stacks with exit values allows us to consider the following optimality equations $\text{OPT}_{\texttt{cont}}(M)$. For every $1 \leq i \leq k$, $q \in Q_i$, $\mathbf{v} \in \mathbb{R}^{|Ex_i|}$:

$$x(\mathbf{v}, q) \quad = \quad \begin{cases} x(\mathbf{v}', en)[\mathbf{v}' \mapsto (x(\mathbf{v}, q'))_{q' \in \text{Ret}_b}] & q = (b, en) \in \text{Call} \\ \mathbf{v}(q) & q \in Ex \\ \max\limits_{a \in A(q)} \Big\{ r(q, a) + \sum\limits_{q' \in Q} p(q'|q, a) x(\mathbf{v}, q') \Big\} & \text{otherwise.} \end{cases}$$

Here $\mathbf{v}$ is a vector where $\mathbf{v}(ex)$ is the (expected) reward that we get once we reach exit ex of the current component. Informally when a box is called, this vector is being updated with the current estimates of the reward that we get once the box is exited. The ex entry of this vector $\mathbf{v}' = (x(\mathbf{v}, q'))_{q' \in \text{Ret}_b}$ is $x(\mathbf{v}, (b, ex))$, which is the value that we achieve from exit (b, ex).

This continuous-space abstraction of the countably infinite space of the stack contexts enables the application of deep feedforward neural networks [18] with a finite state encoding in RL. It also provides an elegant connection to the smoothness of differences to exit values: if all exit costs are changed by less than ε, then the cost of each state within a box changes by less than ε, too. The following theorem connects both versions of optimality equations.

Theorem 3 (Fixed Point)**.** *If y is a fixed point of $\text{OPT}_{\texttt{recur}}$ and x is a fixed point of $\text{OPT}_{\texttt{cont}}$, then $y(\langle\emptyset\rangle, q) = x(\mathbf{0}, q)$. Moreover, any policy optimal from $(\mathbf{0}, q)$ is also optimal from $(\langle\emptyset\rangle, q)$.*

We design a generalization of the Q-learning algorithm [40] for recursive MDPs based on the optimality equations $\text{OPT}_{\texttt{cont}}$ shown in Algorithm 1. We implement several optimizations in our algorithm. We assume implicit transitions from the entry and exit ports of the box to the corresponding entry and exit nodes of the components. A further optimization is achieved by applying a dimension reduction on the representation of the exit value vector $\mathbf{v}$ by normalizing these values in such a way that one of the exits has value 0. This normalization does not affect optimal strategies as, when two stacks incur similar costs in that they have the same offset between the cost of each exit, the optimal strategy is still the same, with the difference in cost being this offset.

While the convergence of Algorithm 1 is not guaranteed for the general multi-exit RMDPs, the algorithm converges for the special cases of deterministic proper RMDPs and 1-exit RMDPs (Section 4). For the deterministic multi-exit case, the observation is straightforward as the properness assumption reduces the semantics to be directed acyclic graph, and the correct values are eventually propagated from the leafs backwards.

Theorem 4. *Tabular Recursive Q-learning converges to the optimal values for deterministic proper multi-exit RMDPs with a learning rate of* 1 *when all state-action pairs are visited infinitely often.*

Algorithm 1: Recursive Q-learning

Initialize $Q(s, v, a)$ arbitrarily
while *not converged* **do**
  $v \leftarrow \mathbf{0}$
  stack $\leftarrow \emptyset$
  Sample trajectory $\tau \sim \{(s, a, r, s'), ...\}$
  for s, a, r, s' *in* τ **do**
    Update α_i according to learning rate schedule
    if *entered box* **then**
      $\{s_{\text{exit}_1}, \ldots, s_{\text{exit}_n}\} \leftarrow \text{getExits}(s')$
      $v' \leftarrow [\max_{a' \in A(s_{\text{exit}_1})} Q(s_{\text{exit}_1}, v, a'), \ldots, \max_{a' \in A(s_{\text{exit}_n})} Q(s_{\text{exit}_n}, v, a')]$
      $v'_{\min} \leftarrow \min(v')$
      $v' \leftarrow v' - v'_{\min}$
      $Q(s, v, a) \leftarrow (1 - \alpha_i)Q(s, v, a) + \alpha_i(r + \max_{a' \in A(S')} Q(s', v', a') + v'_{\min})$
      stack.push(v)
      $v \leftarrow v'$
    else if *exited box* **then**
      $\{s_{\text{exit}_1}, \ldots, s_{\text{exit}_n}\} \leftarrow \text{getExits}(s)$
      Set k such that $s' = s_{\text{exit}_k}$
      $Q(s, v, a) \leftarrow (1 - \alpha_i)Q(s, v, a) + \alpha_i(r + v(k))$
      $v \leftarrow$ stack.pop()
    else
      $Q(s, v, a) \leftarrow (1 - \alpha_i)Q(s, v, a) + \alpha_i(r + \max_{a \in A(S')} Q(s', v, a'))$
    end
  end
end
return Q

4 Convergence of Recursive Q-Learning for Proper 1-exit RMDPs

Recall that a proper 1-exit RMDP is a proper RMDP where, for each component M_i, the set of exits Ex_i is a singleton. For this special case, we show that the recursive Q-learning algorithm converges to the optimal strategy. The optimality equations $\text{OPT}_{\texttt{cont}}(M)$ (similar to [15]) can be simplified in the case of 1-exit RMDPs whose unique fixed point solution will give us the optimal values of the total reward objective. For every $q \in Q$:

$$x(q) \quad = \quad \begin{cases} x(en) + x(b, ex') & q = (b, en) \in \text{Call}, ex = Ex_{Y(b)} \\ \max_{a \in A(q)} \Big\{ r(q, a) + \sum\limits_{q' \in Q} p(q'|q, a) x(q') \Big\} & \text{otherwise.} \end{cases}$$

We now denote the system of all these equations in a vector form as $\bar{x} = F(\bar{x})$. Given a 1-exit RMDP, we can easily construct its associated equation system above in linear time.

Theorem 5 (Unique Fixed Point)**.** *The vector consisting of* $\text{ETotal}^M(q)$ *values is the unique fixed point of* F*. Moreover, a solution of these equations provide optimal stackless strategies.*

Note that for the 1-exit setting, Algorithm 1 simplifies to Algorithm 2 since v is always 0 and $v_{\min}$ is always the maximum Q-value for the exit. The convergence of the recursive Q-learning algorithm for 1-exit RMDPs follows from Theorem 5 and stochastic approximation [40, 8].

Theorem 6. *Algorithm 2 converges to the optimal values in* 1*-exit RMDP when the learning rates satisfy* $\sum_{i=0}^{\infty} \alpha_i = \infty$*,* $\sum_{i=0}^{\infty} \alpha_i^2 < \infty$*, and all state-action pairs are visited infinitely often.*

In order to show efficient PAC learnability for ϵ-proper 1-exit RMDP M, it suffices to know an upper bound on the expected number of steps taken by M when starting at any vertex with the empty stack content, which will be denoted by K.

Theorem 7 (Efficient PAC Learning for 1-Exit RMDPs)**.** *For every* ϵ*-proper* 1*-exit RMDP with diameter* r_{max} *and the expected time to terminate* $\leq K$*,* $\text{ETotal}^{\mathcal{M}}(s)$ *is efficiently PAC-learnable.*

Algorithm 2: Recursive Q-learning (1-exit special case)

Initialize $Q(s, a)$ arbitrarily
while *not converged* **do**
  Sample trajectory $\tau \sim \{(s, a, r, s'), ...\}$
  for s, a, r, s' *in* τ **do**
    Update α_i according to learning rate schedule
    if *entered box* **then**
      $s_{\text{exit}} \leftarrow \text{getExit}(s')$
      $Q(s, a) \leftarrow$
       $(1 - \alpha_i)Q(s, a) + \alpha_i(r + \max_{a' \in A(S')} Q(s', a') + \max_{a' \in A(s_{\text{exit}})} Q(s_{\text{exit}}, a'))$
    else if *exited box* **then**
      $Q(s, a) \leftarrow (1 - \alpha_i)Q(s, a) + \alpha_i(r)$
    else
      $Q(s, a) \leftarrow (1 - \alpha_i)Q(s, a) + \alpha_i(r + \max_{a \in A(S')} Q(s', a'))$
    end
  end
end
return Q

5 Experiments

We implemented Algorithm 1 in tabular form as well as with a neural network. For the tabular implementation, we quantized the vector v directly after its computation to ensure that the Q-table remains small and discrete. For the neural network implementation we used the techniques used in DQN [30], replay buffers and target networks, for additional stability. The details of this implementation can be found in the appendix.

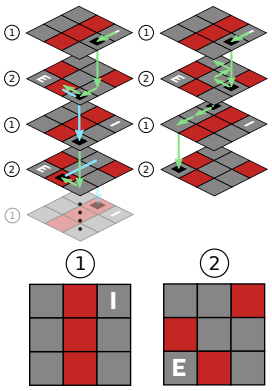
We consider three examples: one to demonstrate the application of Recursive Q-learning for synthesizing probabilistic programs, one to demonstrate convergence in the single-exit setting, and one to demonstrate the use of a context-free reward machine. We compare Recursive Q-learning to Q-learning where the RMDP is treated as an MDP by the agent, i.e., the agent treats stack calls and returns as if they were normal MDP transitions.

5.1 Cloud computing

The cloud computing example, introduced in Example 1, is a recursive probabilistic program with decision points for an RL agent to optimize over. The optimal strategy is to select the reliable server and to never upgrade. This strategy produces an expected total reward of -5.3425. Figure 3 shows that tabular Recursive Q-learning with discretization quickly converges to the optimal solution on this multi-exit RMDP while Q-learning oscillates around a suboptimal policy.

5.2 Infinite spelunking

Consider a single-exit RMDP gridworld with two box types, 1 and 2, shown at the bottom of the figure to the right. These box types are the two types of levels in an infinitely deep cave. When falling or descending to another level, the level type switches. Passing over a trap, shown in red, results in the agent teleporting to a random position and falling with probability 0.5.



The agent has fallen into the cave at the position denoted by I without climbing equipment. However, there is climbing equipment in one of the types of levels at a known location denoted by E. The agent has four move directions—north, east, south, west—as well as an additional action to descend further or ascend. Until the climbing equipment is obtained, the agent can only descend. Once the climbing equipment is obtained, the traps no longer affect the agent and the agent can ascend

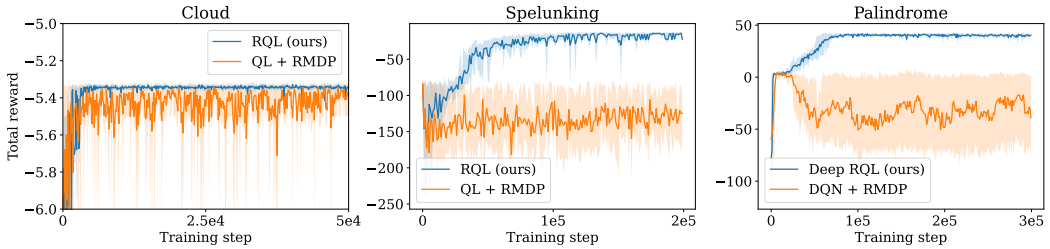


Figure 3: Learning curves. Ten runs were performed for each method. The mean is shown as a dark line. The region between the 10th and 90th percentile is shaded. RQL refers to tabular Recursive Q-learning, QL + RMDP refers to using tabular Q-learning on the RMDP, Deep RQL refers to Deep Recursive Q-learning, and DQN + RMDP refers to using Deep Q-networks on the RMDP.

only from the position where it fell down. With probability 0.01 the agent ascends from the current level with the climbing gear. This has the effect of box-wise discounting with discount factor 0.99. The agent's objective is to leave the cave from where it fell in as as soon as possible. The reward is -1 on every step.

There are two main strategies to consider. The first strategy tries to obtain the climbing gear by going over the traps. This strategy leads to an unbounded number of possible levels since the traps may repeatedly trigger. The second strategy avoids the traps entirely. The figure to the right shows partial descending trajectories from these strategies, with the actions shown in green, the trap teleportations shown in blue, and the locations the agent fell down from are shown as small black squares. Which strategy is better depends on the probability of the traps triggering. With a trap probability of 0.5, the optimal strategy is to try and reach the climbing equipment by going over the traps. Figure 1 shows the convergence of tabular Recursive Q-learning for 1-exit RMDPs to this optimal strategy while the strategy learned by Q-learning does not improve.

5.3 Palindrome gridworld

To demonstrate the ability to incorporate context-free objectives, consider a 3×3 gridworld with a goal cell in the center and a randomly selected initial state. The agent has four move actions—north, east, south, west—and a special control action. The objective of the agent is to reach the goal cell while forming an action sequence that is a palindrome of even length. What makes this possible is that when the agent performs an action that pushes against a wall of the gridworld, no movement occurs. To monitor the progress of the property, we compose this MDP with a nondeterministic pushdown automaton. The agent must use its special action to determine when to resolve the nondeterminism in the pushdown automaton. Additionally, the agent uses its special action to declare the end of its action sequence. To ensure properness, the agent's selected action is corrupted into the special action with probability 0.01. The agent is given a reward of 50 upon success, -5 when the agent selects an action that causes the pushdown automaton to enter a rejecting sink, and -1 on all other timesteps.

Figure 3 shows the convergence of Deep Recursive Q-learning to an optimal strategy on this example, while DQN fails to find a good strategy.

6 Related Work

Hierarchical RL is an approach for learning policies on MDPs that introduces a hierarchy in policy space. There are three prevalent approaches to specify this hierarchy. The options framework [37] represents these hierarchies as policies each with a starting and termination condition. The hierarchy of abstract machines (HAM) framework [31] represents the policy as as a hierarchical composition of nondeterministic finite-state machines. Finally, the MAXQ framework [13] represent the hierarchy using a programmatic representation with finite range variables and strict hierarchy among modules. Semi-Markov decision processes (SDMPs) and hierarchical MDPs are fundamental models that appear in the context of hierarchical RL. SMDPs generalize MDPs with timed actions. The RL algorithms for SMDPs are based on natural generalization of the Bellman equations to accommodate

timed actions. Hierarchical MDPs model bounded recursion and can be solved by flattening to a MDP, or by producing policies that are only optimal locally.

Recursive MDPs model *unbounded* recursion in the environment space. The orthogonality of recursion in environment space and in policy space means they are complementary—one can consider applying ideas in hierarchical RL to find a policy in an RMDP. The authors of [3] proposed using partially specified programs with recursion to constrain the policy space, but only considered bounded recursion. Hierarchical MDPs [37, 31, 13], and factored MDPs [12, 20] indeed offer compact representations of finite MDPs. These representations can be exponentially more succinct. However, note that finite instances of these formalism are not any more expressive than finite MDPs as instances of these formalism can always be rewritten as a finite MDP. On the other hand, recursive MDPs studied in this paper are strictly more expressive than finite MDPs. Even 1-exit RMDPs may not be expressible as finite MDPs, due to a potentially unbounded stack, but remarkably they can be solved exactly with a finite tabular model-free reinforcement learning algorithm (Theorem 6) without needing to resort to ad-hoc approximations of the unbounded stack configurations.

Context-free grammars in RL for optimization of molecules has been considered before by introducing a bound on the recursion depth to induce a finite MDP [25, 42]. Combining context-free grammars and reward machines was proposed as a future research direction in [24], where context-free reward machines have been first described in this paper. Recursive MDPs have been studied outside the RL setting [16], including results for 1-exit RMDP termination [16] and total reward optimization under positive reward restriction [15]. A convergent model-free RL algorithm for 1-exit RMDPs with positive cost restriction was proposed in [21]. The results on undecidability, on PAC-learnability, the introduction of the algorithm Recursive Q-learning, convergence results for Recursive Q-learning, and its deep learning extension are novel contributions of this paper.

7 Conclusion

Reinforcement learning so far has primarily considered Markov decision processes (MDPs). Although extremely expressive, this formalism may require "flattening" a more expressive representation that contains recursion. In this paper we examine the use of recursive MDPs (RMDPs) in reinforcement learning—a setting we call recursive reinforcement learning. A recursive MDP is a collection of MDP components, where each component has the ability to recursive call each other. This allows the introduction of an unbounded stack. We propose abstracting this discrete stack with a continuous abstraction in the form of the costs of the exits of a component. Using this abstraction, we introduce Recursive Q-learning—a model-free reinforcement learning algorithm for RMDPs. We prove that tabular Recursive Q-learning converges to an optimal solution on finite 1-exit RMDPs, even though the underlying MDP has an infinite state space. We demonstrate the potential of our approach on a set of examples that includes probabilistic program synthesis, a single-exit RMDP, and an MDP composed with a context-free property.

Acknowledgments. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements 864075 (CAESAR), and 956123 (FOCETA). This work is supported in part by the National Science Foundation (NSF) grant CCF-2009022 and by NSF CAREER award CCF-2146563. This work utilized the Summit supercomputer, which is supported by the National Science Foundation (awards ACI-1532235 and ACI-1532236), the University of Colorado Boulder, and Colorado State University. The Summit supercomputer is a joint effort of the University of Colorado Boulder and Colorado State University.

References

[1] Alekh Agarwal, Nan Jiang, Sham M Kakade, and Wen Sun. Reinforcement learning: Theory and algorithms. *CS Dept., UW Seattle, Seattle, WA, USA, Tech. Rep*, 2019.

[2] Rajeev Alur, Sampath Kannan, and Mihalis Yannakakis. Communicating hierarchical state machines. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Automata, Languages and Programming*, pages 169–178, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[3] David Andre and Stuart J. Russell. State abstraction for programmable reinforcement learning agents. In *Eighteenth National Conference on Artificial Intelligence*, page 119–125, USA, 2002. American Association for Artificial Intelligence.

[4] Thomas Ball and Sriram K Rajamani. Bebop: A symbolic model checker for boolean programs. In *International SPIN Workshop on Model Checking of Software*, pages 113–130. Springer, 2000.

[5] Thomas Ball and Sriram K Rajamani. The SLAM toolkit. In *Proceedings of CAV'2001 (13th Conference on Computer Aided Verification)*, volume 2102, pages 260–264, 2000.

[6] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13(1-2):41–77, 2003.

[7] Dimitri P Bertsekas and John N Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.

[8] Vivek S Borkar and Sean P Meyn. The ode method for convergence of stochastic approximation and reinforcement learning. *SIAM Journal on Control and Optimization*, 38(2):447–469, 2000.

[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *CoRR*, abs/1606.01540, 2016.

[10] Michael C. Corballis. *The Recursive Mind: The Origins of Human Language, Thought, and Civilization*. Princeton University Press, 2014.

[11] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[12] Thomas Degris and Olivier Sigaud. Factored markov decision processes. *Markov Decision Processes in Artificial Intelligence*, pages 99–126, 2013.

[13] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.

[14] Javier Esparza, David Hansel, Peter Rossmanith, and Stefan Schwoon. Efficient algorithms for model checking pushdown systems. In *International Conference on Computer Aided Verification*, pages 232–247. Springer, 2000.

[15] Kousha Etessami, Dominik Wojtczak, and Mihalis Yannakakis. Recursive stochastic games with positive rewards. *Theor. Comput. Sci.*, 777:308–328, 2019.

[16] Kousha Etessami and Mihalis Yannakakis. Model checking of recursive probabilistic systems. *ACM Trans. Comput. Log.*, 13(2):12:1–12:40, 2012.

[17] Kousha Etessami and Mihalis Yannakakis. Recursive markov decision processes and recursive stochastic games. *J. ACM*, 62(2):11:1–11:69, 2015.

[18] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

[19] Andreas Griesmayer, Roderick Bloem, and Byron Cook. Repair of boolean programs with an application to c. In *International Conference on Computer Aided Verification*, pages 358–371. Springer, 2006.

[20] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.

[21] Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Model-free reinforcement learning for branching markov decision processes. In Alexandra Silva and K. Rustan M. Leino, editors, *Computer Aided Verification - 33rd International Conference, CAV 2021, Virtual Event, July 20-23, 2021, Proceedings, Part II*, volume 12760 of *Lecture Notes in Computer Science*, pages 651–673. Springer, 2021.

[22] Theodore Edward Harris. The theory of branching process. Technical Report R-381-PR, Rand Corporation, May 1964.

[23] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2107–2116. PMLR, 10–15 Jul 2018.

[24] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. Reward machines: Exploiting reward function structure in reinforcement learning. *J. Artif. Intell. Res.*, 73:173–208, 2022.

[25] Egor Kraev. Grammars and reinforcement learning for molecule optimization. *CoRR*, abs/1811.11222, 2018.

[26] Karim Lari and Steve J Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer speech & language*, 4(1):35–56, 1990.

[27] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147(1):5–34, 2003. Planning with Uncertainty and Incomplete Information.

[28] Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

[29] James D Meiss. *Differential dynamical systems*. SIAM, 2007.

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

[31] Ronald Parr and Stuart J Russell. Reinforcement learning with hierarchies of machines. In *Advances in neural information processing systems*, pages 1043–1049, 1998.

[32] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[33] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, , M. Leach, K. Kavukcuoglu, , T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, January 2016.

[34] Thoralf Skolem. *Begründung der elementaren Arithmetik durch die rekurrierende Denkweise ohne Anwendung scheinbarer Veränderlichen mit unendlichem Ausdehnungsbereich*. Dybusach, 1923.

[35] Robert I Soare. *Turing computability: Theory and applications*. Springer, 2016.

[36] R. S. Sutton and A. G. Barto. *Reinforcement Learnging: An Introduction*. MIT Press, second edition, 2018.

[37] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[38] Vladimir A Vatutin and Andreı M Zubkov. Branching processes. ii. *Journal of Soviet Mathematics*, 67(6):3407–3485, 1993.

[39] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, and Petko Georgiev. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[40] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[41] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge United Kingdom, 1989.

[42] Zhenpeng Zhou, Steven Kearnes, Li Li, Richard N Zare, and Patrick Riley. Optimization of molecules via deep reinforcement learning. *Scientific Reports*, 9(1):1–10, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] We characterized the applicability of our method in terms of decidability and approximability. See Sections 3 and 4.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] RL for Recursive MDPs is too general for a meaningful discussion of societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] Mostly in the appendix.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See appendix.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] The inspiration for example 1 has been cited.
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Implementation details

Algorithm 3 shows the extension of tabular Recursive Q-learning to the neural network setting by using the additional techniques introduced by DQN. The algorithm performs two steps per sample: the first computes v and pushes the required values to the replay buffer, and the second samples the replay buffer and updates the neural network via the mean squared error.

For sampling trajectories, we sampled actions with the standard ε-greedy policy. We matched the hyperparameters for (deep) Recursive Q-learning and (deep) Q-learning on each example, with the discount factor set to $\lambda = 1$ for the latter. The experiments were run on a server with 12 CPU cores and no GPU.

Hyperparameters.

Cloud	
Parameter	Value
Test samples	100
Learning rate α	0.02
Exploration rate ε	0.1
Quantize amount	0.001

Spelunking	
Parameter	Value
Test samples	100
Learning rate α	0.2
Exploration rate ε	0.1

Palindrome	
Parameter	Value
Test samples	100
Initial ε	1
Final ε	0.1
Final ε timestep	$30k$
Buffer size	$20k$
Buffer warm up	$1k$
Batch size N	256
Update frequency C	500
Hidden layers	2
Hidden dimension	128
Activation function	tanh
Learning rate	0.0005
Optimizer	Adam

B Discussion on discounting

There are multiple choices for discounting in RMDPs. The most straightforward choice is that discounting by λ is equivalent to stopping the entire process with probability $1 - \lambda$. We call this step-wise discounting. One can transform this type of discounting into a total reward model satisfying Assumption 1 by adding a special exit (the *exit-lane*) which leads to the special exit in the box above with no reward. Unfortunately, step-wise discounting induces a multi-exit RMDP and policies may depend on the stack. This type of discounting is further discussed in Appendix F. An alternative choice for discounting in single-exit RMDPs is that discounting by λ is equivalent to stopping the current box—by leaving out of its only exit—with probability $1 - \lambda$. We call this box-wise discounting, which does not add any exits. With this type of discounting, the discount factor can be incorporated into Recursive Q-learning by simply multiplying the terms $\max_{a' \in A(S')} Q(s', a')$ and $\max_{a' \in A(s_{\text{exit}})} Q(s_{\text{exit}}, a')$ in Algorithm 2 by λ. However, box-wise discounting does not necessarily ensure properness for all single-exit RMDPs and discount factors $\lambda < 1$. Instead, the discount factor must be sufficiently small. Note that box-wise and step-wise discounting schemes are equivalent in MDPs (RMDPs with no recursive calls). Our results subsume these settings by considering total reward under the properness assumption.

C Exponential Succinctness of Hierchical MDPs (Proof from Section 1)

Lemma 1. *Hierarchical MDPs are exponentially more succinct than finite-state MDPs.*

Proof. This proof is adapted from a similar result on the (non-stochastic) recursive state machines from [2]. Consider a collection of hierarchical, deterministic MDPs $M_1, M_2, \ldots, M_n$. The MDP M_1 upon taking an action a gives a reward of 1 and terminates, while it gives a reward of 0 for any other action and moves to a sink. Each MDP M_i (for $i > 1$) upon an action a calls the MDP M_{i-1} twice in sequence and then accepts with a reward of 1, and for other actions it makes a transition to a sink

Algorithm 3: Deep Recursive Q-learning (DQN-style)

Buffer size N, update frequency C
Initialize network parameters θ
Set target network parameters $\theta^- \leftarrow \theta$
Initialize empty replay buffer
while *not converged* **do**
  $v \leftarrow \mathbf{0}$
  stack $\leftarrow \emptyset$
  Sample trajectory $\tau \sim \{(s, a, r, s'), ...\}$
  for s, a, r, s' *in* τ **do**
    `// Push update to replay buffer`
    if *entered box* **then**
      $\{s_{\text{exit}_1}, \ldots, s_{\text{exit}_n}\} \leftarrow \text{getExits}(s')$
      $v' \leftarrow [\max_{a' \in A(s_{\text{exit}_1})} Q(s_{\text{exit}_1}, v, a'; \theta), \ldots, \max_{a' \in A(s_{\text{exit}_n})} Q(s_{\text{exit}_n}, v, a'; \theta)]$
      $v'_{\min} \leftarrow \min(v')$
      $v' \leftarrow v' - v'_{\min}$
      buffer.add(*entered box*, (s, v, a, r, s', v'), $v'_{\min}$)
      stack.push(v)
      $v \leftarrow v'$
    else if *exited box* **then**
      $\{s_{\text{exit}_1}, \ldots, s_{\text{exit}_n}\} \leftarrow \text{getExits}(s)$
      Set k such that $s' = s_{\text{exit}_k}$
      buffer.add(*exited box*, (s, v, a, r, s', v), $v(k)$)
      $v \leftarrow$ stack.pop()
    else
      buffer.add(*normal*, (s, v a, r, s', v), $\perp$)
    end

    `// Update network`
    Sample minibatch $\{\textit{type}_j, (s_j, v_j, a_j, r_j, s'_j, v'_j), \textit{aux}_j\}_{j=1}^N$ of size N from replay buffer
    for $j = 1, \ldots, N$ **do**
      $$\text{targ}_j \leftarrow \begin{cases} r_j + \max_{a' \in A(s'_j)} Q(s'_j, v'_j, a'_j; \theta^-) + \textit{aux}_j & \textit{type}_j = \textit{entered box} \\ r_j + \textit{aux}_j & \textit{type}_j = \textit{exited box} \\ r_j + \max_{a \in A(s'_j)} Q(s'_j, v'_j, a'_j; \theta^-) & \textit{type}_j = \textit{normal} \end{cases}$$
    end
    $\mathcal{L} = \frac{1}{N} \sum_{j=1}^{N} (Q(s_j, v_j, a_j) - \text{targ}_j)^2$
    Update θ with respect to loss $\mathcal{L}$ with gradient descent
    Set $\theta^- \leftarrow \theta$ every C steps
  end
end
return θ

without providing any reward. This MDP M_n has the property that the optimal value is $2^n - 1$ and the optimal policy corresponds to choosing $2^n - 1$ a's in succession. This environment can only be expressed by a finite-state MDP with at least $2^n - 1$ states. □

D Proof of Theorem 1

Theorem 1 (Undecidability of the Strategy Existence Problem)**.** *Given a proper RMDP and a threshold D, deciding whether there exists a strategy with expected value greater than D is undecidable.*

Proof. We make use of the existing results for the *probabilistic finite automata* (PFAs) model. PFAs are essentially finite automata where nondeterminism is replaced by probabilistic transitions. Specially, when a letter is read, the next state is selected by chance with a fixed probability distribution

(that depends on the current state and letter only) instead of the controller selecting the new state. A word is accepted by a PFA if the probability of reaching the special accept state after reading this word is higher than a given fixed threshold $\lambda \in [0, 1]$. Madani, Hanks, and Condon [27] showed undecidability of checking the non-emptiness for $\lambda = 1/2$ and a *leaky PFA* that is a PFA in which at each step we stop the run (or, equivalently, move to a non-accepting state with a self loop) with a fixed probability γ.

As shown in [15], RMDPs can simulate probabilistic finite automata. We adapt this to show how proper RMDPs can simulate leaky PFAs. Such a RMDP has a single component consisting of a box corresponding to each input letter. The number of exits of this component is the number of states in the PFA plus one additional exit-lane exit. One of these exits corresponds to the special accepting state of the PFA. There is a single entry of this component which is the only non-trivial choice point for the controller. Each input letter has a corresponding action that leads to the box corresponding to this letter with probability $1 - \gamma$ and with probability γ to the exit-lane exit. There is a special start action that leads to the exit corresponding to the initial state of the PFA. Once a box corresponding to a letter, a, is exited the probabilistic transitions corresponding to the effect of reading a in the PFA takes place by transitioning to the exit of the component corresponding to the new state of the simulated PFA. An exit-lane exit of any box is connected directly to the exit-lane exit of the component it is in and the reward for such transitions is 0. Note that all that that happens once exit-lane is reached is that the whole content of the stack is popped which results in terminating without modifying the accumulated reward so far. Note that this RMDP is proper, because in each step there is a fixed γ chance of entering the exit-lane and terminating (once the whole stack is popped), so the expected number of steps taken by any strategy is finite.

We claim that the controller has a strategy that terminates at the accepting exit (with empty stack content) with probability $\geq \frac{1}{2}$ iff there exists a word accepted by a leaky PFA (and so its language is non-empty) with the same probability $\geq \frac{1}{2}$. The strategy would pick the letter of the input word in reverse by calling the corresponding box and once done select the special start action. It is easy to see that the behavior of the leaky PFA (including stopping with probability γ) is mimicked precisely.

Now, one can easily encode termination objective using rewards: assign reward of 1 to the final single transition just before the RMDP reaches the accepting exit with an empty stack content and reward of 0 to all other transitions. This shows that the the strategy existence problem for the expected total reward objectives is at least as hard as the strategy existence problem for termination objective which we showed to be as hard as non-emptiness of the language accepted by a leaky PFA; an undecidable problem. □

Note that the above proof carries over to a discounted setting, so the strategy existence problem in such a setting is also undecidable.

E Proof of Theorem 3

Theorem 3 (Fixed Point)**.** *If y is a fixed point of $OPT_{\texttt{recur}}$ and x is a fixed point of $OPT_{\texttt{cont}}$, then $y(\langle\emptyset\rangle, q) = x(\mathbf{0}, q)$. Moreover, any policy optimal from $(\mathbf{0}, q)$ is also optimal from $(\langle\emptyset\rangle, q)$.*

Proof. We first show that the fixed point y of $\text{OPT}_{\texttt{recur}}$ is unique and $y(\langle\kappa\rangle, q) = \text{ETotal}^M(\langle\kappa\rangle, q)$.

The values $\text{ETotal}^M(\langle\kappa\rangle, q)$ are indeed a fixed point of $\text{OPT}_{\texttt{recur}}$. Consider any state $\langle\kappa\rangle, q$. We proceed over all possible cases for the type of q.

If $q{=}(b, en) \in$ Call then $\text{ETotal}^M(\langle\kappa\rangle, q) = \text{ETotal}^M(\langle\kappa, b\rangle, en)$, and if $q \in Ex, (b, q) \in \text{Ret}(b), \kappa{=}\langle\kappa', b\rangle$ then $\text{ETotal}^M(\langle\kappa\rangle, q) = \text{ETotal}^M(\langle\kappa'\rangle, (b, q))$, both by definition of MDP $[\![M]\!]$. Similarly, if $q \in Ex$ then $\text{ETotal}^M(\langle\emptyset\rangle, q) = 0$ as we immediately terminate. Finally, if q is any other type of vertex, then

$$\text{ETotal}^M(\langle\kappa\rangle, q) = \max_{a \in A(q)} \Big\{ r(q, a) + \sum_{q' \in Q} p(q'|q, a) \text{ETotal}^M(\langle\kappa\rangle, q') \Big\},$$

because the best one can do while at $(\langle\kappa\rangle, q)$ is to pick an action that maximizes the one-step reward plus the weighted average of the expected total reward from the successor state.

On the other hand, any strategy that picks any such an action achieves $\text{ETotal}^M(\langle\kappa\rangle, q)$. Let us denote such a strategy by σ. This works because σ is guaranteed to terminate within a finite number of steps, K, when starting at any $(\langle\emptyset\rangle, q)$ due to Assumption 1. Note that the time to exit the current box when at any $(\langle\kappa\rangle, q)$ is also at most K. This is because when starting at $(\langle\kappa\rangle, q)$ and at $(\langle\emptyset\rangle, q)$ the transition structure of the model looks the same until an exit of the current box is reached. (Specifically, a mapping of $(\langle\kappa\kappa'\rangle, q)$ to $(\langle\kappa'\rangle, q)$ for every κ' and q is an isomorphism.) As a result, we get an upper bound of $K \cdot |\kappa|$ for the expected time to terminate from $(\langle\kappa\rangle, q)$ for any κ and q.

Note now that, for any $\epsilon > 0$, the probability of terminating when starting at $(\langle\kappa\rangle, q)$ after $K \cdot |\kappa|/\epsilon$ steps is at most ϵ, because otherwise the expected termination time would not $\leq K \cdot |\kappa|$. This gives us a bound of $K \cdot |\kappa| \cdot r_{max}$ for the expected total reward from $(\langle\kappa\rangle, q)$, because in each transition we can get at most r_{max}. In such a setting, the results from [32] imply that σ is optimal.

If there was another fixed point such that at least one coordinate is higher than ETotal^M, then a strategy constructed as above would obtain more than the supremum over all possible strategies of the expected total reward when starting at that state; a contradiction. On the other hand, if there was any other smaller fixed point than ETotal^M, then by using repeatedly using choices made by the strategy σ defined above we would converge at ETotal^M, but on the other we should never be able to improve based on the assumption that this is a fixed point; a contradiction. (For more details see the end of the proof of Theorem 5 in Appendix G).

Now suppose that there exists a fixed point point y of $\text{OPT}_{\texttt{recur}}$ and a fixed point x of $\text{OPT}_{\texttt{cont}}$ such that $y(\langle\emptyset\rangle, q) \neq x(\mathbf{0}, q)$. We now show how to reconstruct a fixed point $y'(\langle\kappa\rangle, q)$ of $\text{OPT}_{\texttt{recur}}$ that corresponds exactly to the fixed point $x(\mathbf{v}, q)$. This would lead to a contradiction as we just showed that $\text{OPT}_{\texttt{recur}}$ has a unique fixed point.

This will be done by a recursive parallel fixed point reconstruction process below. Let the current call stack be $\langle\kappa\rangle$ and valuation of the exits be $\mathbf{v}$. We start the process at $\kappa = \emptyset$ and $\mathbf{v} = \mathbf{0}$. We assign the values as follows.

$$y'(\langle\kappa\rangle, q) \quad = \quad \begin{cases} \text{make recursive call for } \kappa := \kappa b \text{ and } \mathbf{v} := (x(\mathbf{v}, q'))_{q' \in \text{Ret}_b} & q = (b, en) \in \text{Call} \\ \mathbf{v}(q) & q \in Ex \\ \max\limits_{a \in A(q)} \left\{ r(q,a) + \sum\limits_{q' \in Q} p(q'|q,a) x(\mathbf{v}, q') \right\} & \text{otherwise.} \end{cases}$$

It is clear that the value of all the states are the same in y' and x, so the optimal strategies would be the same as well. □

F PAC Learnability for RMDPs with Discounting and the Milder Restrictions from Section 2

In this section, we start with defining discounted rewards with the usual semantics. We then provide PAC learning results for RMDPs with discounted rewards (instead of the requirements on the expected decline of the stack size and the expected rewards) (Section F.2), and then extend these results to the undiscounted case (Section F.3).

F.1 Discounted Rewards

In the discounted setting, the objective in a RMDP M is to find a strategy $\sigma \in \Sigma_M$ that maximizes the *discounted reward* $\text{EDisct}(\lambda)^M_\sigma(s)$, which is defined as

$$\text{EDisct}(\lambda)^M_\sigma(s) = \lim_{N \to \infty} \mathbb{E}^M_\sigma(s) \left\{ \sum_{1 \leq i \leq N} \lambda^{i-1} r(X_{i-1}, Y_i) \right\},$$

for some discount factor $0 \leq \lambda < 1$. The optimal (discounted) value $\text{EDisct}(\lambda)^M_*(s)$ is then defined as

$$\text{EDisct}(\lambda)^M_*(s) = \sup_{\sigma \in \Sigma_M} \text{EDisct}(\lambda)^M_\sigma(s).$$

We say that σ is discounted optimal if $\text{EDisct}(\lambda)^M_*(s) = \text{EDisct}(\lambda)^M_\sigma(s)$ for every $s \in S$.

F.2 PAC Learning for RMDPs with Discounted Rewards

Lemma 2. *Let M be an RMDP with diameter d, M' be an RMDP that differs from M only by using a different transition function $\delta_{M'}$ ε-close to δ_M, and let σ an ε-proper strategy. Then* $|\text{EDisct}(\lambda)_\sigma^M(s) - \text{EDisct}(\lambda)_\sigma^{M'}(s)| \leq \frac{\varepsilon d}{(1-\lambda)^2}$.

Proof. We can simply estimate

$$\begin{aligned}|\text{EDisct}(\lambda)_\sigma^M(s) - \text{EDisct}(\lambda)_\sigma^{M'}(s)| &\leq \textstyle\sum_{i=1}^\infty \lambda^{i-1}|\mathbb{E}_\sigma^M(s)r(X_{i-1},Y_i) - \mathbb{E}_\sigma^{M'}(s)r(X_{i-1},Y_i)| \\ &\leq \textstyle\sum_{i=1}^\infty d\lambda^{i-1}(1-(1-\varepsilon)^i) = \frac{d}{1-\lambda} - \frac{(1-\varepsilon)d}{1-\lambda+\lambda\varepsilon} \\ &= \frac{\varepsilon d}{(1-\lambda)(1-\lambda+\lambda\varepsilon)} \leq \frac{\varepsilon d}{(1-\lambda)^2}\ ,\end{aligned}$$

where the first inequality is by triangulation, while the second is estimating the chance, that the difference between δ and δ' has been realized within the first i steps by $1-(1-\varepsilon)^i$. This bounds the sum of the different probabilities of taking a transition; the difference in the rewards is in this case bounded by the diameter d. □

As the estimation from the previous lemma survives a supremum operation over all ε-proper strategies we get the following corollary.

Corollary 1. *For an ε'-proper RMDP M with diameter d, and an M' that differs from M only by using a different transition function $\delta_{M'}$ ε-close to δ_M for $\varepsilon \leq \varepsilon'$, then* $|\text{EDisct}(\lambda)^M(s) - \text{EDisct}(\lambda)^{M'}(s)| \leq \frac{\varepsilon d}{(1-\lambda)^2}$.

Corollary 2. *Consequently, it suffices to learn the transition function δ_M of an ε'-proper RMDP*

- *with precision $\varepsilon'' = \min\left\{\varepsilon', \frac{\varepsilon(1-\lambda)^2}{d}\right\}$ to obtain (efficient) PAC learnability if we can (efficiently) evaluate the approximate RMDP M' obtained, or*
- *with precision $\varepsilon'' = \frac{1}{2}\min\left\{\varepsilon', \frac{\varepsilon(1-\lambda)^2}{d}\right\}$ if we can (efficiently) approximately evaluate M' with precision ε''*

with probability δ.

But this is a standard condition that does not differ much from MDPs and provides the following theorem.

Theorem 8. *Taking as input $\frac{1}{\varepsilon'}, \frac{1}{\varepsilon}, \frac{1}{\delta}, d, n$, we can construct with probability $\geq \delta$, for an ε'-proper RMDP M with n states and diameter d,* $\text{EDisct}(\lambda)^M(s)$ *with precision ε. If the evaluation (approximation) of an RMDP M' that differs from M only in using $\delta_{M'}$ instead of δ_M with precision* $\min\left\{\varepsilon', \frac{\varepsilon(1-\lambda)^2}{d}\right\}$ *(*$\frac{1}{2}\min\left\{\varepsilon', \frac{\varepsilon(1-\lambda)^2}{d}\right\}$*) is tractable in the parameters above, then we can efficiently PAC learn* $\frac{1}{2}\min\left\{\varepsilon', \frac{\varepsilon(1-\lambda)^2}{d}\right\}$.

Naturally, the discounted value for a given RMDP can be approximated by unraveling sufficiently deeply.

Corollary 3. *For every ε'-proper RMDP,* $\text{EDisct}(\lambda)^M(s)$ *is PAC-learnable.*

F.3 PAC Learning for RMDPs with Undiscounted Rewards

We now generalize this result to the undiscounted case we refer to in the paper. The first step is a reduction, which shows that the discounted case can be viewed as a special case of the undiscounted case by encoding the discounted payoffs by using an adjusted RMDP. We then continue to adjust the resulting RMDP further to add costs to "stopping" the RMDP in accordance with the level we are in.

This will then turn into a cost estimation, whereby the exit lane is entered to overestimate the effect of changing the strategy. For this to be estimated with a low cost, we use the knowledge that the expected number of steps till termination is finite.

F.3.1 From Discounting to Stopping

In MDPs, discounting can be simulated by stopping in each step with a chance of $1 - \lambda$, while continuing with a chance of λ. In RMDPs, there is no equivalent of stopping immediately, as we will first have to move down the stack. However, this can be simulated by adding a special exit to each component and block: from this exit of a block, we always continue with a reward of 0 to the special exit of the component, and thus have a cost-free direct way to termination, albeit it takes as many steps as the stack is high.

Once equipped with such an *exit lane* without rewards, we can use a reduction similar to the standard stopping reduction for MDPs: we simply go with probability $1-\lambda$ to the special exit of the component, and continue with a probability of λ, i.e., all former probabilities are multiplied by λ.

F.3.2 Adding Rewards to the Exit Lane

In order to connect the construction with the parameters from Section 2, we first consider the effect of changing the rewards on the exit lane.

If we change the rewards of all transitions that lead to the special exit from 0 to an upper bound $b \geq 0$ or a lower bound $-b \leq 0$ for the maximal or minimal expected reward for *any* strategy for *any* state with empty stack for *any* RMDP M' ε-close to M, then the collected reward when moving down the *exit lane* with rewards can serve as an over- or underrepresentation of the remaining reward for any strategy.

In turn the over- and underestimations for a stack of hight h are over- and underestimations for the remaining rewards after h steps.

F.3.3 Connecting our Parameters with Discounting

To see that the parameters we have provided generalize the case of discounting, note that, for an RMDP with discounting, one can choose $b = \frac{d}{1-\lambda}$, $c_o = 1 + \frac{1}{1-\lambda}$, and $\mu = 1 - \lambda$.

To justify the choice of $b = \frac{d}{1-\lambda}$, we observe that the expected sum of the weight-discounted number of steps is at most $\frac{1}{1-\lambda}$, such that the absolute value of the rewards collected is at most the diameter times this number.

To justify the choice for c_o and μ, we first define and estimate a different value for M^+ (and M^-): we define the value of a state as

1. the height of the stack plus $\frac{1}{1-\lambda}$ if the RMDP is still running and we are not on the exit lane,
2. the height of the stack if we are on the exit lane, and
3. 0 if the RMDP has terminated.

In the first case, the expected value of the state will drop by at least $1 - \lambda$: this is because it can only increase by 1 when the exit lane is not entered (which happens with a likelihood of λ), and goes down by $1 + \frac{1}{1-\lambda}$ if the exit lane is entered (with a likelihood of $1 - \lambda$), which leads to an expected value that is at least $2(1 - \lambda)$ smaller.

In the second case, the value is always reduced by 1.

Thus, whenever the RMDP has not terminated after k steps, the expected value of this sum is reduced by (at least) $\min\{1, 2(1 - \lambda)\}$, and therefore by at least $1 - \lambda$.

Consequently, the expected value of this sum is bounded by $c_o - \mu \cdot \sum_{i=1}^{k} p_{\mathsf{run}}^{M'_\sigma}(k)$, and it in turn bounds the stack height.

F.3.4 Wrapping up the Proof

Theorem 2. *For every ε-proper RMDP with parameters c_o, μ, and b,* $\mathrm{ETotal}^{\mathcal{M}}(s)$ *is PAC-learnable.*

To wrap this up to show Theorem 2, we can estimate the effect of moving to a different position due to a change of a strategy by moving to the exit lane with either reward b (for overestimation) or $-b$ (for underestimation).

Thus, if we refer to an (arbitrary but fixed) RMDP M' ε-close to M, and to the over- and under approximation of with exit lanes with rewards b and $-b$ and a likelihood to transition to the exit lane of ε as M^+ and M^-, respectively, we have

$$\text{ETotal}_\sigma^{M^+}\big((\langle\emptyset\rangle, q)\big) \geq \text{ETotal}_\sigma^{M'}\big((\langle\emptyset\rangle, q)\big) \geq \text{ETotal}_\sigma^{M^-}\big((\langle\emptyset\rangle, q)\big)$$

for all strategies σ, and thus

$$\text{ETotal}^{M^+}\big((\langle\emptyset\rangle, q)\big) \geq \text{ETotal}^{M'}\big((\langle\emptyset\rangle, q)\big) \geq \text{ETotal}^{M^-}\big((\langle\emptyset\rangle, q)\big)\ .$$

In order to estimate $\text{ETotal}_\sigma^{M^+}(s) - \text{ETotal}_\sigma^{M^-}(s)$, we first observe that the expected stack height must be converging to 0 for M_σ by our assumption that it is bounded by $c_o - \mu \sum_{i-1}^{k} p_{\mathsf{run}}^{M_\sigma}(k)$ by our assumption. (Note that, for convergence, it is enough to assume termination in expected finite time. The stronger assumption is used for to establish PAC learnability.)

For a given $\varepsilon' > 0$ we therefore pick a k such that, after $k' \geq k$ steps, the expected stack height is lower than $\frac{\varepsilon'}{b}$. E.g., we can choose k such that $k\mu\frac{\varepsilon'}{b} \geq c_o$ holds, such that the falling sequence $p_{\mathsf{run}}^{M_\sigma}(k)$ cannot be above $\frac{\varepsilon'}{b}$ for k steps. This is satisfied for $k = \left\lceil \frac{c_o b}{\mu\varepsilon'} \right\rceil$.

We then pick $\varepsilon = \frac{\varepsilon'}{k^2 b}$.

In this case, the likelihood of moving onto the exit lane before step k is lower than $\frac{\varepsilon'}{kc}$, and the difference in rewards along the exit lane in this case below kc, leaving a contribution to the difference of below ε'.

When entering the exit lane on or after step k, the expected stack height is below $\frac{\varepsilon'}{b}$, leading to a contribution to the expected distance between the rewards for M^+ and M^- below ε'.

Together, this provides

$$\text{ETotal}_\sigma^{M^+}\big((\langle\emptyset\rangle, q)\big) - \text{ETotal}_\sigma^{M^-}\big((\langle\emptyset\rangle, q)\big) \leq 2\varepsilon'\ .$$

This shows that the expected value converges. We still need to show that we can approximate it to up to $4\varepsilon'$ with a likelihood of at least $1 - \delta$.

For this, we can estimate $\text{ETotal}_\sigma^{M^+}\big((\langle\emptyset\rangle, q)\big)$ and $\text{ETotal}_\sigma^{M^-}\big((\langle\emptyset\rangle, q)\big)$ by cutting the runs after step k, then the error in the expected value is at most ε'.

Estimating $\text{ETotal}_\sigma^{M}\big((\langle\emptyset\rangle, q)\big)$ by cutting the runs after k steps leads to a value in between. Using triangulation, this entails that it estimates the value of $\text{ETotal}_\sigma^{M}\big((\langle\emptyset\rangle, q)\big)$.

But this can be unravelled into a finite MDP, and hence be PAC learned up to precision ε' with likelihood at least $1 - \delta$ with standard techniques.

G Reinforcement Learning for Single-Exit RMDPs

Theorem 5 (Unique Fixed Point)**.** *The vector consisting of* $\text{ETotal}^M(q)$ *values is the unique fixed point of* F*. Moreover, a solution of these equations provide optimal stackless strategies.*

Proof of Theorem 5. The proof proceeds over several lemmas.

Lemma 3. $\text{ETotal}^M(q)$ *is a fixed point of* F*.*

Proof. Let us consider any vertex v.

If v is an entry port, i.e., $q = (b, en) \in \text{Call}, ex = Ex_{Y(b)}$ then once the box b is entered, by Assumption 1, we will be almost surely reach the unique exit ex' of b no matter which action will be picked inside of b. Therefore, to maximize the total reward it suffices to focus solely first on maximizing the reward while inside b before ex' is reached no matter what the current call stack

is. The maximal reward possible to obtain is equal to $\text{ETotal}^M(en)$ by definition. Once the exit is reached, it suffices to maximize the reward from the exit port (b, ex'), which is at most equal to $\text{ETotal}^M((b, ex'))$. It is easy to see that these two strategies can be combined into a single strategy from q that can get arbitrarily close to the maximum possible value of $\text{ETotal}^M(en) + \text{ETotal}^M((b, ex'))$. So this all together shows that $\text{ETotal}^M(q) = \text{ETotal}^M(en) + \text{ETotal}^M((b, ex'))$ holds.

If v is any other non-exit vertex, then the best we can do is to pick any action available at v that maximizes expected reward from the successor vertex plus the one step reward for picking this action at v. Once the successor node is reached, we can then switch to the best possible strategy from that node. This shows that the value of $\max_{a \in A(q)} \Big\{ r(q,a) + \sum_{q' \in Q} p(q'|q,a)x(q') \Big\}$ is not only achievable but also the most one can expect to get when starting at v. □

Lemma 4. *For any fixed point $\bar{x}$ of F, there exists a stackless strategy $\sigma(\bar{x})$ such that* $\text{ETotal}^M_{\sigma(\bar{x})}(q) = \bar{x}(q)$ *for every q.*

Proof. This strategy $\sigma(x)$ will simply pick

$$\sigma(\bar{x})(\langle \kappa \rangle, q) = \arg\max_{a \in A(q)} \Big\{ r(q,a) + \sum_{q' \in Q} p(q'|q,a)x(q') \Big\}$$

from any vertex q and the call stack κ. This works, because as argued in Theorem 3 a mapping of $(\langle \kappa\kappa' \rangle, q)$ to $(\langle \kappa' \rangle, q)$ for all κ' and q gives us two isomorphic models until the exit of the top box in κ is reached and so the optimal strategy for $(\langle \emptyset \rangle, q)$ works for $(\langle \kappa \rangle, q)$ as well. In other words, the best thing to do at $(\langle \kappa \rangle, q)$ is to try to maximize the reward before exiting the current box which happens with probability 1 and the same holds for $(\langle \emptyset \rangle, q)$. In order to maximize this reward, the call stack being κ or $\emptyset$ makes no difference as the transitions only depend on the current component. □

Now, we can observe that $\bar{x}^*(q) = \text{ETotal}^M(q)$ is the largest fixed point of F. If there was $\bar{x}$ such that $\bar{x}^*(q) < \bar{x}(q)$ for some q then we could pick $\sigma(\bar{x})$ as our strategy and get the reward of $\bar{x}(q)$ from q due to Lemma 4. However, just by definition, $\bar{x}^*(q) = \text{ETotal}^M(q) = \sup_\sigma \text{ETotal}^M(q)$ is the largest possible reward obtainable from q as all strategies σ are considered; a contradiction.

Note that once a stackless strategy σ is fixed then $F(\bar{x})$ becomes equal to $A_\sigma \bar{x} + \bar{b}_\sigma$, where $\bar{b}_\sigma$ is the one-step transitions rewards obtained using σ and A_σ is a transition matrix derived from σ. Note that all entries of A_σ are non-negative. We will refer to F as F_σ in such a case and exploit this fact later in various proofs. Note that we can interpret $F_\sigma(\bar{x})$ as the expected total reward of using σ for each vertex once and then obtaining reward $\bar{x}$.

Lemma 5. *For any proper 1-exit RMDP and stackless strategy σ, we have that* $\text{ETotal}^M_\sigma = \sum_{i=1}^{\infty} A_\sigma^i b_\sigma$.

Proof. Notice that

$$\begin{aligned}
F_\sigma(\bar{x}) &= A_\sigma \bar{x} + b_\sigma \\
F_\sigma^2(\bar{x}) &= A_\sigma(A_\sigma \bar{x} + b_\sigma) + b_\sigma = A_\sigma^2 \bar{x} + (A_\sigma + I) b_\sigma \\
F_\sigma^3(\bar{x}) &= A_\sigma F_\sigma^2(\bar{x}) + b_\sigma = A_\sigma^3 \bar{x} + (A_\sigma^2 + A_\sigma + I) b_\sigma \\
&\vdots \\
F_\sigma^k(\bar{x}) &= A_\sigma^k \bar{x} + \Big(\sum_{i=0}^{k} A_\sigma^i \Big) b_\sigma .
\end{aligned}$$

Let $\mathbf{1}$ and $\mathbf{0}$ be vectors consisting only of 1s and 0s, respectively. As in [15], we can show that the expected number of steps taken before termination while using σ can be computed by iterating $\bar{y}_{i+1} = A_\sigma(\bar{y}_i) + \mathbf{1}$ when starting at $\bar{y}_0 = \mathbf{0}$. Intuitively, $\bar{y}_{i+1}$ will correspond to the expected number of steps taken before termination when we assume the call and return from a box to be executed in

parallel in a single step. The crucial assumption made in [15] that all rewards are positive (and equal to 1) in true in this case.

This shows that $\lim_{k\to\infty} F_\sigma^k(\mathbf{0}) = A_\sigma^k\mathbf{0} + \left(\sum_{i=0}^k A_\sigma^i\right)\mathbf{1}$ is finite. Therefore, we have that $\lim_{i\to\infty} A_\sigma^i$ is an 0 matrix, because otherwise that sum $\sum_{i=0}^k A_\sigma^i$ would not be finite as a sum of all non-negative matrices.

Now, one can see that $\lim_{k\to\infty} F_\sigma^k(\bar{x}) = \lim_{k\to\infty} A_\sigma^k\bar{x} + \left(\sum_{i=0}^k A_\sigma^i\right)b_\sigma = \lim_{k\to\infty}\left(\sum_{i=0}^k A_\sigma^i\right)b_\sigma$ converges absolutely, because $\left(\sum_{i=0}^k A_\sigma^i\right)|b_\sigma| \leq \left(\sum_{i=0}^k A_\sigma^i\right)r_{max} \leq Kr_{max}$, where K is the expected number of steps taken by any strategy as guaranteed by Assumption 1. It is clear that this limit $\lim_{k\to\infty} F_\sigma^k(\bar{x})$ is a fixed point of F_σ, because $F_\sigma(\lim_{k\to\infty} F_\sigma^k(\bar{x})) = \lim_{k\to\infty} F_\sigma^{k+1}(\bar{x}) = \lim_{k\to\infty} F_\sigma^k(\bar{x})$. As we already showed ETotal_σ^M is a fixed point of F_σ. It is clear that F_σ cannot have any other fixed point because, for arbitrary $\bar{x}$, we have $\lim_{k\to\infty} F_\sigma^k(\bar{x}) = A_\sigma^i b_\sigma$ which is fixed in terms of $\bar{x}$. This shows that $\text{ETotal}_\sigma^M = \sum_{i=1}^\infty A_\sigma^i b_\sigma$ has to hold. □

We are ready to show that no other than $\bar{x}^* = \text{ETotal}^M$ fixed point of F exists. Suppose there is one and let us denote it by $\bar{x}$. As we just showed $\bar{x} \leq \bar{x}^*$ and $\bar{x}(q) < \bar{x}^*(q)$ for some q (because otherwise $\bar{x}^* = \bar{x}$).

Let us denote $\sigma(\bar{x}^*)$ by $\sigma*$ and consider $F_{\sigma^*}(\bar{x})$. If any coordinates of $F_{\sigma^*}(\bar{x})$ is larger than $\bar{x}$ then we can improve the expected total reward by using σ^* once and then follow the strategy that gives us expected total reward of $\bar{x}$. A contradiction with the assumption that $\bar{x}$ is a fixed point of F.

So we have $F_{\sigma^*}(\bar{x}) \leq \bar{x}$ and by iterating this we get that $F_{\sigma^*}^k(\bar{x}) \leq \bar{x}$ for every k. In the previous Lemma we showed that $\text{ETotal}^M = \lim_{k\to\infty} F_{\sigma^*}^k(\bar{x}) \leq \bar{x} < \text{ETotal}^M$; a contraction. □

Theorem 7 (Efficient PAC Learning for 1-Exit RMDPs)**.** *For every ϵ-proper 1-exit RMDP with diameter r_{max} and the expected time to terminate $\leq K$,* $\text{ETotal}^{\mathcal{M}}(s)$ *is efficiently PAC-learnable.*

Proof. Let us denote the ϵ-proper 1-exit RMDP by M and its optimal total reward stackless strategy by σ^*. As we know that every strategy is ϵ-proper then so is σ^*. This means that for all M' that that are ε-close to $\mathcal{M}$ the expected time to terminate when using σ^* is also $\leq K$. Recall that ε-closeness means that $\sum_{q\in S, a\in A, r\in S} |\delta_{\mathcal{M}}(q,a)(r) - \delta_{\mathcal{M}'}(q,a)(r)| \leq \varepsilon$, and where the support of $\delta_{M'}(q,a)$ is a subset of the support of $\delta_M(q,a)$ for all $q \in S$ and $a \in A$.

Lemma 6. *If stackless σ is ε-proper, then* $\|\text{ETotal}_\sigma^M - \text{ETotal}_\sigma^{M'}\|_\infty \leq 2\epsilon K^2 r_{\max}$.

Proof. As shown in Lemma 5, the optimal total rewards in M satisfy $\bar{x} = A_\sigma\bar{x} + b_\sigma$ and in M' statisfy $\bar{x}' = A'_\sigma\bar{x}' + b_\sigma$. Moreover, $\bar{x}$ and $\bar{x}'$ are the sole fixed points of these equations. To avoid clutter, we will write A, A' and $\bar{b}$ instead of A_σ, A'_σ and $\bar{b}_\sigma$. We know that $\|A - A'\|_1 \leq 2\varepsilon$ and let $\mathcal{E} = A' - A$.

When we try to converge at $\bar{x}'$ by iterating $\bar{x}'_{i+1} = A'\bar{x}'_i + b'$ when starting at $\bar{x}'_0 = \bar{x}$, we obtain the following:

$$\begin{aligned}
\bar{x}'_1 &= A'\bar{x}_0 + \bar{b} = (A' - A)\bar{x} + A\bar{x} + \bar{b} = (A' - A)\bar{x} + \bar{x} = \mathcal{E}\bar{x} + \bar{x}_0 \\
\bar{x}'_2 &= A'\bar{x}_1 + \bar{b} = A'(\mathcal{E}\bar{x} + \bar{x}_0) + \bar{b} = A'\mathcal{E}\bar{x} + A'\bar{x}_0 + \bar{b} = A'\mathcal{E}\bar{x} + \bar{x}_1 \\
\bar{x}'_3 &= A'\bar{x}_2 + \bar{b} = A'(A'\mathcal{E}\bar{x} + \bar{x}_1) + \bar{b} = (A')^2\mathcal{E}\bar{x} + A'\bar{x}_1 + \bar{b} = (A')^2\mathcal{E}\bar{x} + \bar{x}_2 \\
&\vdots \\
\bar{x}'_{k+1} &= (A')^k\mathcal{E}\bar{x} + \bar{x}_k
\end{aligned}$$

and thus $\bar{x}' - \bar{x} = (\lim_{k\to\infty} \bar{x}'_{k+1}) - \bar{x}_0 = (\lim_{k\to\infty} \sum_{i=0}^k A'^i\mathcal{E}\bar{x} + \bar{x}_0) - \bar{x}_0 = (\sum_{i=0}^\infty A'^i)\mathcal{E}\bar{x}$.

Note that the time to terminate in M and M' from each vertex is equal to $(\sum_{i=0}^\infty A^i)\mathbf{1}$ and $(\sum_{i=0}^\infty A'^i)\mathbf{1}$, respectively. We know that all of these values are $\leq K$. We know that the absolute value of each entry of $\bar{x}$ is at most Kr_{max}, as the most we can get in each step is r_{max}

during the expected number of $\leq K$ steps. So $\|\mathcal{E}\bar{x}\|_\infty \leq 2\varepsilon K r_{max}\mathbf{1}$, and so $\|(\sum_{i=0}^{\infty} A'^i)\mathcal{E}\bar{x}\|_\infty \leq \|2\varepsilon K r_{max}(\sum_{i=0}^{\infty} A'^i)\mathbf{1}\|_\infty \leq 2\varepsilon K^2 r_{max}$ as required. □

We observe that the estimate in Lemma 6 holds when we take the supremum over all stackless ε-proper strategies. This is because for any two functions f and g such that $f(\sigma) - g(\sigma) \leq \varepsilon$ for every stackless σ and $\sigma' = \arg\max_\sigma f(\sigma)$ we get $\sup_\sigma f(\sigma) - \sup_\sigma g(\sigma) \leq f(\sigma') - g(\sigma') \leq \varepsilon$. In other words, Lemma 6 implies that $\|\text{ETotal}^M - \text{ETotal}^{M'}\|_\infty \leq 2\epsilon K^2 r_{\max}$.

We can now choose $\varepsilon' = \frac{\varepsilon}{2\epsilon K^2 r_{\max}}$, and learn δ_M up to precision ε' with probability $\geq \delta$ by the usual sampling techniques as done for finite MDPs.

We then obtain 1-exit RMDP M', which we can solve efficiently and exactly using linear programming similarly to what was done in [15]. First, let us create a variable t_q for every $q \in Q$. The linear program is then as follows.

$$\text{Minimize} \sum_q t_q \text{ subject to:}$$

$$t_q \geq t_{en} + t_{(b,ex')} \qquad \text{if } q = (b, en) \in \text{Call}, ex = Ex_{Y(b)}$$

$$t_q \geq r(q,a) + \sum_{q' \in Q} p(q'|q,a) t_{q'} \qquad \text{otherwise, for every possible } a \in A$$

One can easily show that any optimal solution gives us a fixed point of F which we know has to be equal to $\text{ETotal}^{M'}$. This is because if all inequalities for a variable t_q are non-strict then we can decrease the value of t_q to the maximum of their right hand sides and still get a valid solution. As linear programs can be solved in polynomial time, and $\text{ETotal}^{M'}$ is ϵ-close to ETotal^M, we obtain an efficient PAC learning algorithm. □