Reinforcement Learning with Stochastic Reward Machines

Jan Corazza[1,2], **Ivan Gavran**[2], **Daniel Neider**[2]

[1] University of Zagreb
[2] Max Planck Institute for Software System

corazzajan@gmail.com, gavran@mpi-sws.org, neider@mpi-sws.org

Abstract

Reward machines are an established tool for dealing with reinforcement learning problems in which rewards are sparse and depend on complex sequences of actions. However, existing algorithms for learning reward machines assume an overly idealized setting where rewards have to be free of noise. To overcome this practical limitation, we introduce a novel type of reward machines, called stochastic reward machines, and an algorithm for learning them. Our algorithm, based on constraint solving, learns minimal stochastic reward machines from the explorations of a reinforcement learning agent. This algorithm can easily be paired with existing reinforcement learning algorithms for reward machines and guarantees to converge to an optimal policy in the limit. We demonstrate the effectiveness of our algorithm in two case studies and show that it outperforms both existing methods and a naive approach for handling noisy reward functions.

1 Introduction

The key assumption of a reinforcement learning (RL) model is that the reward function is *Markovian*: the received reward depends only on the agent's immediate state and action. For many practical RL tasks, however, the most natural conceptualization of the state-space is the one in which the reward function depends on the history of actions that the agent has performed. (Those are typically the tasks in which the agent is rewarded for complex behaviors over a longer period.)

An emerging tool used for reinforcement learning in environments with such non-Markovian rewards are *reward machines*. A reward machine is an automaton-like structure which augments the state space of the environment, capturing the temporal component of rewards. It has been demonstrated that Q-learning (Sutton and Barto 2018), a standard RL algorithm, can be adapted to use and benefit from reward machines (Toro Icarte et al. 2018).

Reward machines are either given by the user, or inferred by the agent on the fly (Gaon and Brafman 2020; Furelos-Blanco et al. 2020; Xu et al. 2020). The used learning methods ensure that the inferred machine is minimal, enabling quick optimal convergence. Besides a faster convergence, learning minimal reward machines contributes to the interpretability of problems with an unclear reward structure.

Reward machines only model deterministic rewards. When the machine is not known upfront, existing learning methods prove counterproductive in the presence of noisy rewards, as there is either no reward machine consistent with the agent's experience, or the learned reward machine explodes in size, overfitting the noise.

In this paper, we introduce the notion of a *stochastic reward machine*, which can capture noisy, non-Markovian reward functions, together with a novel algorithm for learning them. The algorithm is an extension of the constraint-based formulation of Xu et al. (2020). The extension relies on the parameters of the reward's distribution, making sure that experiential rewards respect the distribution. In every iteration, if the agent establishes that its current hypothesis about the machine is wrong, it updates the hypothesis (either by fixing the machine's parameters or by solving the constraint problem and learning a new machine).

While one could use the proposed algorithm to learn a suitable (non-stochastic) reward machine and use that machine to guide the reinforcement learning process, we recognize the value of modeling stochasticity explicitly. First, it reveals information about the distribution of rewards, improving interpretability of the problem at hand. Second, a direct correspondence between the stochastic reward function and the stochastic reward machine that models it makes the exposition clearer.

In our experimental evaluation, we demonstrate the successful working of our algorithm on two noisy, non-Markovian case studies. We compare our algorithm with existing methods (which do not deal explicitly with noise) on the same case studies: as expected, disregarding the noise by using existing inference algorithms for classical RMs performs substantially worse than our new approach. Finally, we compare our algorithm to a baseline method that tries to "average out" the noise.

To summarize, in this paper we 1) introduce stochastic reward machines, 2) present a novel algorithm for learning stochastic reward machines by a RL agent, and 3) experimentally demonstrate the efficacy of our algorithm.

1.1 Related Work

Using finite state machines to capture non-Markovian reward functions has been proposed already in the early work on the topic (Bacchus, Boutilier, and Grove 1996).

Toro Icarte et al. (2018; 2020) introduced reward machines (known as *Mealy machines* in other contexts) as a suitable formalism and an algorithm that takes advantage of the reward machine structure. Similar formalisms, including temporal logics, have been proposed by others, too (Jothimurugan, Alur, and Bastani 2019; Brafman, Giacomo, and Patrizi 2018; Camacho et al. 2019). This line of work assumes the reward machine to be given.

The assumption of the user-provided reward machine has been lifted in the follow-up works (Gaon and Brafman 2020; Xu et al. 2020; Furelos-Blanco et al. 2020). Learning temporal representations of the reward has been explored in different contexts: for multi-agents settings (Neary et al. 2020), for reward shaping (Velasquez et al. 2021b), or with user-provided advice (Neider et al. 2021). All these approaches are fragile in presence of noisy rewards. Other approaches focus on learning attainable sequences of labels (Hasanbeig et al. 2021; Icarte et al. 2019), disregarding reward values. If reward noise ranges over a finite set of values, Velasquez et al. (2021a) propose active learning of reward machine-like automata with a probabilistic transition function.

Outside of the non-Markovian context, many works studied noise in rewards. Everitt et al. (2017) give an overview of potential sources of noise/corruption and provide the impossibility result for learning under arbitrarily corrupted rewards. Romoff et al. (2018) propose learning a reward estimator function alongside the value function. Wang, Liu, and Li (2020) consider a problem of rewards being flipped according to a certain distribution. While all these works consider much richer noise models, they are not readily applicable to the non-Markovian rewards setting.

2 Preliminaries

This section introduces the necessary background on reinforcement learning and the formalism of reward machines for capturing non-Markovian rewards. We illustrate all notions on a running example called *Mining*. Mining, inspired by variations of Minecraft (e.g., (Andreas, Klein, and Levine 2017)), models the problem of finding and exploiting ore in an unknown environment. We use this example throughout the paper.

Fig. 1 shows the Mining world. An agent moves in a bounded grid-world intending to find valuable ore, gold (G) and platinum (P), and bring it to the marketplace (M). Furthermore, the agent's success depends on the purity of the ore and the market prices, which is stochastic and cannot be influenced by the agent (though the spread of the market prices can naturally be bounded). In order to do so successfully, the agent has to fulfill specific additional requirements, such as finding equipment (E) beforehand and not stepping into traps (T).

In reinforcement learning, an agent learns to solve such tasks through repeated, often episodic interactions with an environment. After each step, the agent receives feedback (a reward $r \in \mathcal{R} \subset \mathbb{R}$) based on its performance and acts to maximize a (discounted) sum of received rewards. This interaction forms a stochastic process: while the environment is in some state $s \in S$, the agent chooses an action $a \in A$ according to a policy $\pi(s, a)$ (a function mapping states to probability distributions over the action space), causing the environment to transition into the next state $s' \in S$ and giving the agent a reward r (where S is the state space and A is the action space of the environment). A realization of this process is a *trajectory* $s_0 a_1 s_1 \dots a_k s_k$ (optionally, rewards may be included in this sequence). The agent continually updates its policy (i.e., learns) based on received rewards.

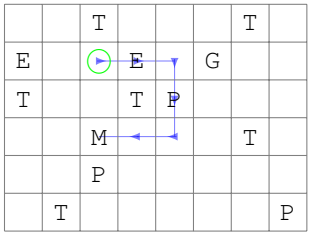


Figure 1: A simplified example of the Mining environment grid and a trajectory. The agent's initial state is shown as a circle. Cells display their state labels, or are blank if the label is $\emptyset$. The trajectory indicated by arrows shows the agent collecting the equipment (E), finding platinum (P) and bringing it to the marketplace (M).

A reinforcement learning environment is formalized as a *Markov decision process* (MDP). As is common in the context of reward machines, we equip our MDPs with a labeling function that maps transitions to labels. These labels correspond to high-level events that are relevant for solving the given task (e.g., finding gold, indicated by G).

Definition 1. A (labeled) **Markov decision process** is a tuple $\mathcal{M} = (S, s_I, A, p, P, L)$ consisting of a finite state space S, an agent's initial state $s_I \in S$, a finite set A of actions, and a probabilistic transition function $p\colon S \times A \times S \to [0, 1]$. Additionally, a finite set P of propositional variables, and a labeling function $L\colon S \times A \times S \to 2^P$ determine the set of relevant high-level events that the agent senses in the environment. We define the size of an MDP $\mathcal{M}$, denoted as $|\mathcal{M}|$, to be the cardinality of the set S (i.e., $|\mathcal{M}| = |S|$).

Let us briefly illustrate this definition. The agent always starts in state s_I. It then interacts with the environment by taking an action at each step. If the agent at state $s \in S$ takes the action $a \in A$, its next state will be $s' \in S$ with probability $p(s, a, s')$. For this transition (s, a, s'), a labeling function L emits a set of high-level labels. One can think of these labels as knowledge provided by the user. If the user can not provide any labeling function, L can simply return the current transition.

The Mining example can be modeled by the MDP in which states are fields of the grid world, and the agent's actions are moving in four cardinal directions. The transition function p models the possibility of the agent slipping when making a move. The propositional variables used for labeling are E (equipment found) P (platinum found), G (gold found), M (marketplace reached), T (fell into a trap), and $\emptyset$ (no relevant events).

To capture an RL problem fully, we need to equip an MDP with a reward function. Formally, a reward function R maps

trajectories from $(S \times A)^+ \times S$ to a cumulative distribution function over the set of possible rewards $\mathcal{R}$. For labeled MDPs, the reward function typically depends on finite sequences of observed labels rather than on the more low-level sequences of state-action pairs.

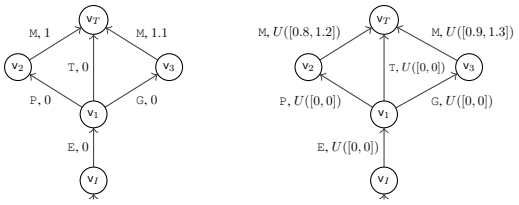
Let us emphasize the significance of defining the reward function R over finite sequences of states and actions. Using the entire history as the argument enables us to reward behaviors that respect temporal relations and persistence naturally. For instance, in the Mining example, the goal is to accomplish the following steps: (1) find equipment, (2) exploit a mine, (3) deliver the ore to the specified location, all while avoiding traps. Note that the order in which the agent performs these steps is crucial: finding ore and going to the marketplace without first picking up equipment is not rewarded; stepping into traps ends the episode with reward zero. Reward functions that make use of the agent's entire exploration (as opposed to only the current state and action) have first been studied by Bacchus, Boutilier, and Grove (1996) and are termed *non-Markovian* reward functions.

Toro Icarte et al. (2018) have shown that *reward machines (RMs)* are a powerful formalism for representing non-Markovian reward functions. Intuitively, one can view the role of reward machines as maintaining the sufficient amount of memory to turn the non-Markovian reward function back into a ordinary, Markovian one. This results in an important feature of RMs: they enable using standard RL algorithms (which would otherwise not be usable with non-Markovian reward functions). Furthermore, taking advantage of the structure present in RMs, the algorithms can be made more efficient.

On a technical level, RMs are finite-state automata that transduce a sequence $\ell_1 \ell_2 \dots \ell_k$ of labels into a sequence $r_1 r_2 \dots r_k$ of rewards. For the sake of brevity, we here omit a formal definition and introduce the concept of RMs using an example. To this end, let us consider the RM A in Fig. 2a, which *attempts* to capture the reward function of the Mining example and operates as follows.

Starting from the initial state v_I, the machine transitions to an intermediate state v_1 upon finding equipment (indicated by formula[1] $\mathtt{E}$). From there, A either moves to state v_2 (upon finding platinum) or to state v_3 (upon finding gold). The reward, however, is delayed until the agent reaches the marketplace (indicated by the label $\mathtt{M}$) and A transitions to the terminal state v_T. Once this happens, the machine outputs a reward of 1 (if the agent has previously collected gold) or a reward of 1.1 (if the agent has collected platinum). By contrast, violating the prescribed order, failing to reach the marketplace, or stepping onto a trap results in no reward for the agent. For the label sequence $(\emptyset, \{\mathtt{E}\}, \emptyset, \{\mathtt{P}\}, \emptyset, \emptyset, \{\mathtt{M}\})$ (from Fig. 1), the machine A will produce the reward sequence $(0, 0, 0, 0, 0, 0, 1)$.

Note, however, that the RM in Fig. 2a fails to capture the stochastic nature of rewards in the Mining example, which stems from varying purity of the ore and market fluctuations. This problem arises from an intrinsic property of reward machines: they only allow outputs to be real numbers and, hence, can only encode *deterministic* reward functions. This observation shows that reward machines, as currently defined and used in the literature, cannot capture the common phenomenon of noisy rewards! In the next section, we show how to generalize the model of reward machines in order to overcome this limitation.



(a) RM: rewards are real numbers.

(b) SRM: rewards are probability distributions.

Figure 2: Classical reward machine for the mining example (left) and stochastic reward machine (right). Transitions are labeled with input-output pairs, where labels are given by propositional formulas encoding subsets of 2^P. All states have a transition to v_T with output 0 on reading $\mathtt{T}$ (trap) (only depicted for v_1). All remaining, missing transitions are self-loops with output 0.

3 Stochastic Reward Machines

To capture stochastic, non-Markovian reward functions, we introduce the novel concept of stochastic reward machines.

Definition 2. A **stochastic reward machine** (SRM) $\mathsf{A} = (V, \mathsf{v}_I, 2^P, O, \delta, \sigma)$ is defined by a finite, nonempty set V of states, an initial state $\mathsf{v}_I \in V$, an input alphabet 2^P, a (deterministic) transition function $\delta : V \times 2^P \to V$, an output alphabet O which is a finite set of cumulative distribution functions (CDFs), and and output function $\sigma : V \times 2^P \to O$. We define the size of A, denoted as $|\mathsf{A}|$, to be $|V|$ (i.e., the cardinality of the set V).

To define the semantics of a SRM, let $s_0 a_1 s_1 \dots a_k s_k$ be a trajectory of an RL episode and $\ell_1 \dots \ell_k$ the corresponding label sequence with $\ell_{i+1} = L(s_i, a_{i+1}, s_{i+1})$ for each $i \in \{0, \dots, k-1\}$. The run of a SRM A on the label sequence $\ell_1 \dots \ell_n$ is then a sequence $\mathsf{v}_0 \ell_1 F_1 \mathsf{v}_1 \dots \mathsf{v}_{n-1} \ell_n F_n \mathsf{v}_n$ where $\mathsf{v}_{i+1} = \delta(\mathsf{v}_i, \ell_{i+1})$ and $F_i = \sigma(\mathsf{v}_i, \ell_{i+1})$ for all $i \in \{0, \dots, k-1\}$. The sequence $F_1 \dots F_k$ of CDFs is now used to determine the rewards that the agent receives: for each $i \in \{0, \dots, k-1\}$, the CDF F_i is sampled and the resulting reward $r_i \in \mathcal{R}$ is returned. This process results in a pair (λ, ρ) with $\lambda = \ell_1 \dots \ell_k$ and $\rho = r_1 \dots r_k$, which we call a *trace*.

We will often refer to SRM's output by the corresponding probability distribution. We focus on continuous but bounded probability distributions, $\{D_1([a_1, b_1]), \dots, D_n([a_n, b_n])\}$ where $n \in \mathbb{N}$ and $D_i([a, b])$ is a distribution over the interval $[a, b]$. It is not hard to see that the classical reward machines are a special

[1] We use propositional formulas to succinctly describe sets of labels. For instance, the formula $p \vee q$ over $\mathcal{P} = \{p, q\}$ corresponds to the set $\{\{p\}, \{q\}, \{p, q\}\}$.

case of Definition 2: one just has to set $a_i = b_i$ for each $i \in \{1, \ldots, n\}$, which will result in probability distributions that assign probability 1 to a single real number.

Fig. 2b shows an SRM B for the mining example. Note the difference to the classical RM of Fig. 2a: the transitions now output probability distributions instead of real values (non-trivial uniform distributions are on transitions from v_2 to v_T and from v_3 to v_T). This difference allows us to capture the noise in the rewards of the example. For example, the label sequence of our running example $(\emptyset, \{\mathtt{E}\}, \emptyset, \{\mathtt{P}\}, \emptyset, \emptyset, \{\mathtt{M}\})$ will be transduced to a sequence of distributions. Sampling these distributions produces different reward sequences (e.g., $(0, 0, 0, 0, 0, 0, 0.95)$ and $(0, 0, 0, 0, 0, 0, 1)$).

Toro Icarte et al. introduced a version of Q-learning for reward machines called QRM, which assumes that the reward machine outputs deterministic rewards. In the following, we examine if the guarantees of QRM can be retained when working with stochastic reward machines.

3.1 QRM With Stochastic Reward Machines

In addition to specifying a learning task, reward machines help with the learning process itself. QRM assumes knowledge of a reward machine representation of environment reward and splits the Q-function update over all RM states by using transition outputs in lieu of empirical rewards.

For an MDP transition (s, a, s') with label ℓ, QRM executes Q-function updates $Q^{\mathsf{v}}(s, a) \overset{\alpha}{\leftarrow} r + \gamma \max_{a'} Q^{\mathsf{v}'}(s', a')$ for each reward machine state v, with v' being the succeeding state, $\alpha \in \mathbb{R}$ the learning rate, and $r = \sigma(\mathsf{v}, \ell)$ the reward due to reading label ℓ in state v. These updates are equivalent to regular Q-learning updates in the cross-product of the original MDP and the reward machine. The reward function is Markovian with respect to the resulting cross-product decision process. As Q-learning also converges correctly for a *stochastic* Markovian reward, it is easy to see that QRM can find the optimal policy induced by an SRM by using samples $\hat{r} \sim \sigma(\mathsf{v}, \ell)$ in the update rule.

We also remark that SRMs allow for a relaxed notion of equivalence. As we will show in the following lemma, it is not necessary for two SRMs to be exactly equal in order to induce the same optimal policy. We generalize the notion of exact functional equivalence (that is necessary for RMs) into the equivalence in expectation.

Definition 3. SRMs A and B are **equivalent in expectation** ($\mathsf{A} \sim_{\mathbb{E}} \mathsf{B}$) if for every label sequence $\lambda = \ell_1 \ell_2 \ldots \ell_k$ we have $\mathbb{E}[\mathsf{A}(\lambda)_i] = \mathbb{E}[\mathsf{B}(\lambda)_i]$ for every $1 \leq i \leq k$, that is if they output sequences of CDFs with equal expected values (where $\mathsf{A}(\lambda)_i$ refers to the i-th CDF in the output sequence $\mathsf{A}(\lambda)$).

Lemma 1 can simplify representation and inference of SRMs, allowing the algorithm to rely only on the expected values of the transitions in the inferred SRM.

Lemma 1. If $\mathsf{A} = (V, \mathsf{v}_I, 2^P, O, \delta, \sigma)$ and $\mathsf{B} = (V', \mathsf{v}_I', 2^P, O', \delta', \sigma')$ are equivalent in expectation then they induce the same optimal policy over the same environment.

Now that it has been established that learning the optimal policy using stochastic reward machines is viable, the remaining question is whether one can drop the assumption that knowledge of the environment reward is accessible, and learn an SRM representation of it in conjunction with the policy (instead of assuming it to be given).

4 Inferring SRMs

In this section, we show how to infer SRMs from data obtained through the agent's exploration. In Section 4.1, we present a seemingly appealing baseline algorithm, and we explain its weaknesses. We follow it by proposing SRMI (SRM inference) as a better approach in Section 4.2.

Both algorithms intertwine RL and learning of SRMs by starting with an initial hypothesis SRM and

(1) running QRM which generates a sequence of traces, and
(2) if there are traces contradicting the current hypothesis, inferring a new one.

The steps repeat with the goal of recovering an SRM that captures the environment reward and using it to learn the optimal policy. QRM is performed in conjunction with the latest hypothesis. Traces which contradict the hypothesis are called *counterexamples*.

Due to Lemma 1, the task is simplified to finding an SRM that is merely equivalent in expectation with environment rewards (instead of agreeing on exact distributions). We assume that a bound on noise dispersion $\epsilon_c > 0$ is known (e.g., sensors come with pre-specified measurement error tolerance). Definition 4 uses the ϵ_c parameter to formalize the notion of consistency with a trace.

Definition 4. A trace $(\lambda, \rho) = (\ell_1 \ell_2 \ldots \ell_k, r_1 r_2 \ldots r_k)$ is ϵ_c**-consistent** with an SRM $\mathcal{H}$, which outputs a sequence of distributions $\mathcal{H}(\lambda) = d_1 d_2 \ldots d_k$ if for all $1 \leq i \leq k$ we have $|r_i - \mathbb{E}[d_i]| \leq \epsilon_c$, i.e. if all of the observed rewards r_i are plausible samples from $\mathcal{H}$.

SRMI can only recover an SRM representation of a noisy environment reward that meets an additional requirement, which we formalize in Assumption 1. Informally, the assumption requires the noise from one reward distribution not to fully conceal the signal of a different one (unless they share means). We are convinced this requirement is met in a large class of practical, real-world scenarios.

Assumption 1. Let $O = \{D_1([a_1, b_1]), \ldots, D_n([a_n, b_n])\}$ be the output alphabet of the environment SRM. Let $\epsilon_c = \max_i\{b_i - a_i\}/2$ be the noise dispersion bound known to the agent. We then assume that any two output distributions that can be covered with an ϵ_c-interval must have equal expectations: for all $1 \leq i, j \leq n$ and $\mu \in \mathbb{R}$ we have $[a_i, b_i] \cup [a_j, b_j] \subseteq [\mu - \epsilon_c, \mu + \epsilon_c] \implies \mathbb{E}[D([a_i, b_i])] = \mathbb{E}[D([a_j, b_j])]$.

This assumption is satisfied in the Mining example. The set $\{U([0.1, 0.2]), U([0, 1])\}$ breaks Assumption 1: ϵ_c-intervals cannot distinguish these distributions, and they differ in expected values so they must be distinguished. The set $\{U([0.1, 0.9]), U([0, 1])\}$ respects it, and distinguishing these distributions is unnecessary as they have equal expectations.

4.1 Baseline Algorithm

One may be tempted to repurpose existing techniques for inferring reward machines from a collection of traces. There are two important obstacles:

1. Traces may be prefix-inconsistent: during exploration, the agent may encounter traces $(\ell_1\ell_2\dots\ell_m, r_1r_2\dots r_m)$ and $(\ell'_1\ell'_2\dots\ell'_n, r'_1r'_2\dots r'_n)$ s.t. for some $1 \leq i \leq \min\{m,n\}$ we have $\ell_1\dots\ell_i = \ell'_1\dots\ell'_i$ but $r_1\dots r_i \neq r'_1\dots r'_i$. The consequence is that no reward machine can capture both traces.
2. Even if a collection of traces where noise is present is prefix-consistent, the (inferred) reward machine will tend to be impractically large because it will overfit noisy data.

The baseline algorithm solves these problems by obtaining multiple samples for each trace in the counterexample set, and producing estimates for transition means *before* inferring the structure of the reward machine. Starting with an initial hypothesis the following steps are repeated:

(1) run QRM which generates a sequence of traces

(2) when a counterexample is encountered, pause QRM and replay its trajectory until enough samples are collected

(3) preprocess the counterexample set so that multiple samples collected in (2) are collapsed into estimates for environment reward means

(4) use the deterministic RM inference method by Xu et al. (2020) to infer the new minimal consistent hypothesis

As knowledge of environment SRM structure is not assumed, it is necessary to sample traces (instead of individual transitions). The number of samples required in (2) is determined from ϵ_c and an additional parameter for the minimal distance between two different transition means in the environment SRM. The preprocessing in (3) ensures (up to a confidence level) that different estimates for the same means are aggregated into one, and the result respects ϵ_c-consistency with the original sample set.

This approach seems to eliminate the two issues presented by stochastic rewards: since every prefix is sampled many times and then averaged and aggregated, there can be no prefix inconsistencies. Furthermore, aggregation leaves little room for overfitting noise. However, the agent must be able to sample traces multiple times on demand. As we show in section 5, this is costly, and sometimes even impossible.

4.2 SRMI

In contrast to the baseline algorithm, SRMI uses counterexamples to improve hypotheses *immediately* by relying on a richer constraint solving method that is able to encode ϵ_c-consistency with the counterexample set directly. This removes the need for replaying trajectories.

As before, the task for the algorithm is to recover a minimal SRM that is equivalent in expectation to the true environment one, and use it to learn the optimal policy. Starting with an initial hypothesis SRM, the following steps are repeated:

(1) Run QRM and record all traces in a set A (Lines 5 to 6 in Algorithm 1).

Algorithm 1: SRMI

```
1  Initialize SRM H with a set of states V;
2  Initialize a set of q-functions Q = {q^v | v ∈ V};
3  Initialize X = ∅ and A = ∅;
4  for episode n = 1, 2, ... do
5      (λ, ρ, Q) ← QRM_episode(H, Q);
6      add (λ, ρ) to A;
7      if H not ε_c-consistent with (λ, ρ) then
8          add (λ, ρ) to X;
9          if found SRM Z isomorphic with H and
             ε_c-consistent with X then
10             H' ← Z;
11         else
12             infer H' from X;
13         end
14         H ← Estimates (H', A);
15         reinitialize Q;
16     end
17 end
```

(2) When a counterexample is encountered, add it to the set X and attempt to make the current hypothesis ϵ_c-consistent with X by shifting its outputs (Lines 7 to 10).

(3) If Step (2) failed, solve a constraint problem to infer the new hypothesis (Line 12).

(4) Compute the final mean estimates to correct outputs of the inferred hypothesis based on empirical rewards in A (Line 14).

The algorithm generates a sequence of hypothesis SRMs $\mathcal{H}_1\mathcal{H}_2\cdots$ and a sequence of counterexample sets $X_1X_2\cdots$ (with $X_i \subset X_j$ for all $i < j$) where $\mathcal{H}_j$ is consistent with X_{j-1} (and thus every X_i for $1 \leq i \leq j$). For simplicity, we assume noise distributions are symmetric, but SRMI can easily be extended to cover asymmetric ones (discussed in the appendix). Hypothesis outputs are in the form $D([\mu - \epsilon_c, \mu + \epsilon_c])$, where $D([a,b])$ is a symmetric distribution over an interval, and $\mu \in \mathbb{R}$ is the estimated mean of a particular transition. Two SRMs are structurally isomorphic (Line 9) if their underlying automata without the reward output are isomorphic in a graph-theoretic sense.

There are many ϵ_c-consistent SRMs that can be returned by the constraint solving method in Line 12, not necessarily having the best estimates for transition means. To correct for this, the function *Estimates* assigns sets of observed rewards to each hypothesis transition by simulating its runs on traces in A and uses them to compute the final estimates.

Our algorithm categorized every counterexample as either Type *1* or Type *2*. A counterexample (λ, ρ) is of Type *1* with respect to $\mathcal{H}_i$ if there exists a graph-isomorphic SRM $\mathcal{Z}$ that is consistent with (λ, ρ) and X_{i-1} (otherwise it is Type *2*). Then $\mathcal{H}_{i+1} = \mathcal{Z}$ and $X_i = X_{i-1} \cup \{(\lambda, \rho)\}$. Intuitively the current hypothesis can be "fixed" to become consistent with Type *1* counterexamples by shifting outputs without changing the structure of the SRM. We now discuss how our algorithm handles counterexamples of Type *2*.

Inference of Stochastic Reward Machines When a new Type *2* counterexample is encountered (i.e., outputs in the current hypothesis cannot be shifted to make it consistent), SRMI infers a new hypothesis from the counterexamples, effectively solving the following task.

Task 1. Given a set of traces X and dispersion ϵ_c, produce a minimal SRM that is ϵ_c-consistent with all traces in X.

We accomplish Task 1 by encoding it as a constraint problem in real arithmetic. Our encoding is an extension of the encoding used in JIRP algorithm (Xu et al. 2020). Minimality is ensured by starting from machines of size $n = 1$, increasing the size by 1 each time the constraint problem proves unsatisfiable, and returning the first successful result. For a given size, we use a collection of propositional and real variables from which one can extract an SRM, and constrain them so they (1) encode a valid SRM and (2) ensure that the SRM is consistent with the set X. More precisely, for size $n \in \mathbb{N} \setminus \{0\}$, parameter ϵ_c, and counterexample set X, we construct a formula Φ_n^{X,ϵ_c} with the following two properties:

(a) Φ_n^{X,ϵ_c} is satisfiable iff there exists an SRM $\mathcal{H}$ of size n that is ϵ_c-consistent with every trace in X.

(b) Every satisfying assignment for variables in Φ_n^{X,ϵ_c} contains sufficient information to construct a consistent SRM of size n.

The formula Φ_n^{X,ϵ_c} is built using the following variables:

- $d_{\mathsf{p},\ell,\mathsf{q}}$ are propositional variables, true iff $\delta(\mathsf{p},\ell) = \mathsf{q}$
- $o_{\mathsf{v},\ell}$ are real variables that match the value of $\mathbb{E}[\sigma(\mathsf{v},\ell)]$
- $x_{\lambda,\mathsf{v}}$ are propositional variables encoding machine runs, true if the SRM arrives in state v upon reading label sequence λ

We use these variables to define constraints (1) - (4). Formula 1 requires that at the beginning (after an empty sequence), the SRM is in the initial state. Formula 2 requires that the SRM transitions to exactly one state upon seeing a label.

The remaining two formulas connect the SRM to the set X (we use symbol $Pref(X)$ for the set of prefixes of traces in X). Formula 3 connects seen prefixes to the transition function captured by variables $d_{\mathsf{p},\ell,\mathsf{q}}$. Finally, Formula 4 ensures ϵ_c-consistency.

$$x_{\epsilon,\mathsf{v}_I} \wedge \bigwedge_{\mathsf{v}\in V\setminus\{\mathsf{v}_I\}} \neg x_{\epsilon,\mathsf{v}} \quad (1)$$

$$\bigwedge_{\mathsf{p}\in V}\bigwedge_{\ell\in 2^P}\left[\left[\bigvee_{\mathsf{q}\in V} d_{\mathsf{p},\ell,\mathsf{q}}\right] \wedge \left[\bigwedge_{\substack{\mathsf{q},\mathsf{q}'\in V\\ \mathsf{q}\neq\mathsf{q}'}} \neg d_{\mathsf{p},\ell,\mathsf{q}} \vee \neg d_{\mathsf{p},\ell,\mathsf{q}'}\right]\right] \quad (2)$$

$$\bigwedge_{(\lambda\ell,\rho r)\in Pref(X)}\ \bigwedge_{\mathsf{p},\mathsf{q}\in V} (x_{\lambda,\mathsf{p}} \wedge d_{\mathsf{p},\ell,\mathsf{q}}) \rightarrow x_{\lambda\ell,\mathsf{q}} \quad (3)$$

$$\bigwedge_{(\lambda\ell,\rho r)\in Pref(X)}\ \bigwedge_{\mathsf{v}\in V} x_{\lambda,\mathsf{v}} \rightarrow |o_{\mathsf{v},\ell} - r| \leq \epsilon_c \quad (4)$$

The formula Φ_n^{X,ϵ_c} is defined as a conjunction of formulas (1) - (4). One can easily see that properties (a), (b) hold for Φ_n^{X,ϵ_c} as there is a bijection between assignments for which Φ_n^{X,ϵ_c} is true and consistent SRMs (modulo distributions).

Let $\mathcal{H}'$ be the SRM constructed from a model that satisfies the above constraints, with $\sigma_{\mathcal{H}'}(\mathsf{v},\ell) = D([o_{\mathsf{v},\ell} - \epsilon_c, o_{\mathsf{v},\ell} + \epsilon_c])$. For every $(\lambda,\rho) \in X$ of length k and $1 \leq i \leq k$, due to (4), we have $|\,\mathbb{E}[\mathcal{H}'(\lambda)_i] - \rho_i| < \epsilon_c$ and so $\mathcal{H}'$ is the new ϵ_c-consistent hypothesis. When $\mathcal{H}'$ is constructed by correcting for a type *1* counterexample, it is consistent by definition.

Algorithm 2: Estimates

```
Initialize sets r(v, ℓ) for states v ∈ V and ℓ ∈ 2^P;
Initialize empty output function σ';
for (λ, ρ) ∈ A do
    Skip (λ, ρ) if not ε_c-consistent with H;
    Simulate a run of H on λ disregarding its outputs
      and record rewards from ρ in corresponding
      r(v, ℓ) sets;
    for v ∈ V, ℓ ∈ 2^P do
        μ' ← (max r(v, ℓ) + min r(v, ℓ))/2;
        Set σ'(v, ℓ) = U[μ' − ε_c, μ' + ε_c];
    end
end
Return H with σ' as the output function;
```

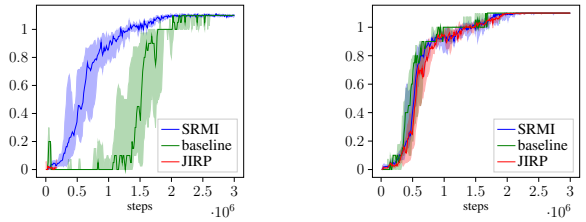
Correcting Output Distributions in Inferred SRMs As there can be many SRMs ϵ_c-consistent with X that are not equivalent in expectation, Task 1 need not have a unique solution. To illustrate how nonuniqueness can prohibit correct convergence of hypothesis SRMs, let X contain samples from a single-state SRM $\mathcal{T}$ with two possible outputs, $D([0,1])$ on ℓ_1 and $D([10,100])$ on ℓ_2 ($\epsilon_c = (100-10)/2 = 45$). For $\alpha \in \mathbb{R}$ let $\mathcal{H}_\alpha$ be a single-state SRM with two possible outputs, $D([\alpha - 45, \alpha + 45])$ on ℓ_1 and $D([10,100])$ on ℓ_2. Then for all $-44 \leq \alpha, \beta \leq 45$ ($\alpha \neq \beta$) we have $\mathcal{H}_\alpha \not\sim_{\mathbb{E}} \mathcal{H}_\beta$, yet both are ϵ_c-consistent with X. Thus SRMI must choose solutions to Task 1 so that any generated hypothesis sequence will converge to the same limit $\mathcal{T}$.

This is done in the *Estimates* step of Algorithm 1, which simulates runs of $\mathcal{H}'$ (the inferred solution to Task 1) on *consistent* traces in A, yielding final estimates of means based on empirical rewards (shown in Algorithm 2). The need to filter for consistency is due to the fact that the set of SRMs consistent with A is often strictly smaller than the set of those consistent with X. As X grows to capture more information about environment reward, the need for filtering recedes.

Using the midrange estimator for outputs in $\mathcal{H}$ ensures it remains consistent with the counterexamples. As new Type *1* counterexamples will always be found eventually, *Estimates* runs infinitely often which guarantees convergence to correct means.

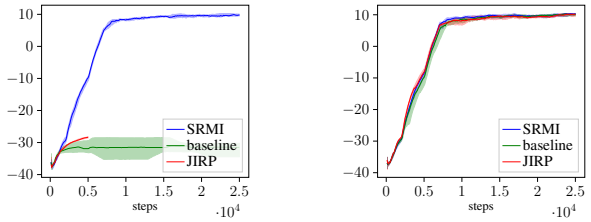
Convergence to an Optimal Policy

Theorem 1. Given Assumption 1 on the output alphabet of the environment SRM and an ϵ-greedy exploration strategy, SRMI converges in the limit to an SRM that is equivalent in expectation to the true environment one.



(a) Stochastic rewards (b) Non-stochastic rewards

Figure 3: Results on the mining environment



(a) Stochastic rewards (b) Non-stochastic rewards

Figure 5: Results on the harvest environment

The proof of Theorem 1 follows similar reasoning to the convergence proof for JIRP. We first establish that SRMI does not revisit structurally isomorphic SRMs. As there are only finitely many such structures for a fixed maximal size, SRMI "settles" in a final structure. Then Assumption 1 guarantees that *Estimates* will converge to the correct expectations.

Corollary 1. SRMI converges to an optimal policy in the limit.

Corollary 1 follows from Theorem 1 due to Lemma 1, which guarantees that two SRMs that are equivalent in expectation induce the same optimal policy, and finally due to the fact that QRM with stochastic reward machines converges to an optimal policy.

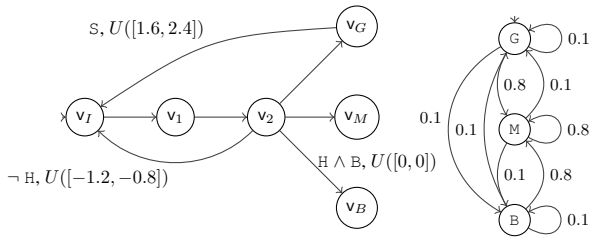
5 Results

To assess the performance of SRMI, we have implemented a Python 3 prototype based on code by Toro Icarte et al. (2018), which we will make publicly available (also see the supplementary material). To assess its performance, we compare SRMI to the baseline algorithm and the JIRP algorithm for classical reward machines on two case studies: the mining example from Section 2 and an example inspired by harvesting (which we describe shortly).

Our primary metric is the cumulative reward averaged over the last 100 episodes. We conducted 10 independent runs for each algorithm, using Z3 (de Moura and Bjørner 2008) as the constraint solver. All experiments were conducted on a 3 GHz machine with 1.5 TB RAM.

Mining Fig. 3a shows the comparison on the Mining environment. The interquartile ranges for the reward are drawn as shaded areas, while the medians are drawn as solid lines. For this case study, we have set the baseline algorithm to replay 20 traces per counterexample. As can be seen from the figure, SRMI converges faster to an optimal policy (reward 1) than both the baseline algorithm and JIRP. The latter times out because it is unable to deal the noise properly and tries to infer larger and larger RMs.

Fig. 3b compares SRMI, baseline, and JIRP on a non-stochastic version of the Mining environment using the reward machine of Fig. 2a to define the rewards. All algorithms perform equally well. Thus, SRMI does not incur a runtime penalty, even when used in non-stochastic settings.



(a) SRM: all missing transitions are self-loops with reward 0 (b) MDP

Figure 4: Harvest environment

Harvest The *Harvest* environment represents a crop-farming cycle. The agent is rewarded for performing a sequence of actions, `P`, `W`, `H`, `S` (plant, water, harvest, sell), and penalized for breaking it. MDP states `G`, `M`, `B` (good, medium, bad) transition as given by the dynamics in Fig. 4b. The labeling function $L(s, a, s')$ returns the transition, effectively making trajectories $s_0 a_1 s_1 \ldots a_k s_k$ their own label sequences. The reward mean depends on the MDP state during the harvest action as shown in Fig. 4a.

The Harvest example is well suited for showing the benefits of SRMI over the baseline, because the probability that the agent will repeatedly see a given trajectory is very low.

Fig. 5a shows the comparison on the Harvest environment. SRMI was successful in learning the optimal policy, while the baseline algorithm got stuck in collecting the required number of samples (5), and JIRP again timed out. In a modified Harvest environment, without noise, the algorithms do equally well (Fig. 5b).

6 Conclusion

In this work we introduced Stochastic reward machines as a general way of representing non-Markovian stochastic rewards in RL tasks, and the SRMI algorithm that is able to infer an SRM representation of the environment reward based on traces, and use it to learn the optimal policy. We have shown SRMI is an improvement over prior methods.

References

Andreas, J.; Klein, D.; and Levine, S. 2017. Modular multitask reinforcement learning with policy sketches. In *ICML'2017*, 166–175. JMLR. org.

Bacchus, F.; Boutilier, C.; and Grove, A. J. 1996. Rewarding

Behaviors. In *AAAI/IAAI, Vol. 2*, 1160–1167. AAAI Press / The MIT Press.

Brafman, R. I.; Giacomo, G. D.; and Patrizi, F. 2018. LTLf/LDLf Non-Markovian Rewards. In *AAAI*, 1771–1778. AAAI Press.

Camacho, A.; Icarte, R. T.; Klassen, T. Q.; Valenzano, R. A.; and McIlraith, S. A. 2019. LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning. In *IJCAI*, 6065–6073. ijcai.org.

de Moura, L. M.; and Bjørner, N. 2008. Z3: An Efficient SMT Solver. In *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, 337–340. Springer.

Everitt, T.; Krakovna, V.; Orseau, L.; and Legg, S. 2017. Reinforcement Learning with a Corrupted Reward Channel. In *IJCAI*, 4705–4713. ijcai.org.

Furelos-Blanco, D.; Law, M.; Russo, A.; Broda, K.; and Jonsson, A. 2020. Induction of Subgoal Automata for Reinforcement Learning. In *AAAI*, 3890–3897. AAAI Press.

Gaon, M.; and Brafman, R. I. 2020. Reinforcement Learning with Non-Markovian Rewards. In *AAAI*, 3980–3987. AAAI Press.

Hasanbeig, M.; Jeppu, N. Y.; Abate, A.; Melham, T.; and Kroening, D. 2021. DeepSynth: Automata Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning. In *AAAI*, 7647–7656. AAAI Press.

Icarte, R. A. T.; Waldie, E.; Klassen, T.; Valenzano, R.; Castro, M. P.; and McIlraith, S. A. 2019. Learning Reward Machines for Partially Observable Reinforcement Learning. In *NeurIPS*.

Jothimurugan, K.; Alur, R.; and Bastani, O. 2019. A Composable Specification Language for Reinforcement Learning Tasks. In *NeurIPS*, 13021–13030.

Neary, C.; Xu, Z.; Wu, B.; and Topcu, U. 2020. Reward machines for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2007.01962*.

Neider, D.; Gaglione, J.; Gavran, I.; Topcu, U.; Wu, B.; and Xu, Z. 2021. Advice-Guided Reinforcement Learning in a non-Markovian Environment. In *AAAI*, 9073–9080. AAAI Press.

Romoff, J.; Henderson, P.; Piché, A.; François-Lavet, V.; and Pineau, J. 2018. Reward Estimation for Variance Reduction in Deep Reinforcement Learning. In *CoRL*, volume 87 of *Proceedings of Machine Learning Research*, 674–699. PMLR.

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

Toro Icarte, R.; Klassen, T. Q.; Valenzano, R. A.; and McIlraith, S. A. 2018. Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, 2112–2121. PMLR.

Toro Icarte, R.; Klassen, T. Q.; Valenzano, R. A.; and McIlraith, S. A. 2020. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. *CoRR*, abs/2010.03950.

Velasquez, A.; Beckus, A.; Dohmen, T.; Trivedi, A.; Topper, N.; and Atia, G. 2021a. Learning Probabilistic Reward Machines from Non-Markovian Stochastic Reward Processes. arXiv:2107.04633.

Velasquez, A.; Bissey, B.; Barak, L.; Beckus, A.; Alkhouri, I.; Melcer, D.; and Atia, G. K. 2021b. Dynamic Automaton-Guided Reward Shaping for Monte Carlo Tree Search. In *AAAI*, 12015–12023. AAAI Press.

Wang, J.; Liu, Y.; and Li, B. 2020. Reinforcement Learning with Perturbed Rewards. In *AAAI*, 6202–6209. AAAI Press.

Xu, Z.; Gavran, I.; Ahmad, Y.; Majumdar, R.; Neider, D.; Topcu, U.; and Wu, B. 2020. Joint Inference of Reward Machines and Policies for Reinforcement Learning. In *ICAPS*, 590–598. AAAI Press.