Searching for Markovian Subproblems to Address Partially Observable Reinforcement Learning

Rodrigo Toro Icarte
Department of Computer Science
University of Toronto & Vector Institute
Toronto, Ontario, Canada
rntoro@cs.toronto.edu

Ethan Waldie
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
ethan.waldie@mail.utoronto.ca

Toryn Q. Klassen
Department of Computer Science
University of Toronto
Toronto, Ontario, Canada
toryn@cs.toronto.edu

Richard Valenzano
Element AI
Toronto, Ontario, Canada
rick.valenzano@elementai.com

Margarita P. Castro
Department of Mechanical and Industrial Engineering
University of Toronto
Toronto, Ontario, Canada
mpcastro@mie.utoronto.ca

Sheila A. McIlraith
Department of Computer Science
University of Toronto & Vector Institute
Toronto, Ontario, Canada
sheila@cs.toronto.edu

Abstract

In partially observable environments, an agent's policy should often be a function of the history of its interaction with the environment. This contradicts the Markovian assumption that underlies most reinforcement learning (RL) approaches. Recent efforts to address this issue have focused on training Recurrent Neural Networks using policy gradient methods. In this work, we propose an alternative – and possibly complementary – approach. We exploit the fact that in many cases a partially observable problem can be decomposed into a small set of individually Markovian subproblems that collectively preserve the optimal policy. Given such a decomposition, any RL method can be used to learn policies for the subproblems. We pose the task of learning the decomposition as a discrete optimization problem that learns a form of Finite State Machine from traces. In doing so, our method learns a high-level representation of a partially observable problem that summarizes the history of the agent's interaction with the environment, and then uses that representation to quickly learn a policy from low-level observations to actions. Our approach is shown to significantly outperform standard Deep RL approaches, including A3C, PPO, and ACER, on three partially observable grid domains.

Keywords: Partial Observability
Reinforcement Learning
Automata Learning
Reward Machines

Acknowledgements

We gratefully acknowledge funding from CONICYT (Becas Chile), the Natural Sciences and Engineering Research Council of Canada (NSERC), and Microsoft Research.

1 Introduction

Partially observable environments remain very challenging for RL agents because they break the Markovian assumption with respect to the agent's observations. As a result, agents in these environments require some form of memory to summarize past observations. Recent approaches either encode the observation history using recurrent neural networks [5, 10, 7] or use memory-augmented neural networks to provide the agent access to external memory [6]. We propose an alternative approach that searches for a decomposition of the task into a small set of individually Markovian subtasks.

For example, consider the 2-keys domain shown in Figure 1c. The agent (purple triangle) receives a reward of +1 when it reaches the coffee machine, which is always in the yellow room. To do so, it must open the two doors (shown in brown). Each door requires a different key to open it, and the agent can only carry one key at the time. At the beginning of each episode, the two keys are randomly located in either the blue room, the red room, or split between them. Since the agent can only see what is in the current room, this problem is partially observable.

This problem is quite difficult for current RL approaches: A2C, ACER, and PPO performed poorly on this task even after 5 million training steps (Section 4). However, it is decomposable into a small set of Markovian subproblems. The first involves searching for the keys. Notice that if the agent finds only one key in the red room, then (if it has learned enough about the domain) it can deduce that the second key is in the blue room. The next subproblem is to pick up a key. This is followed by a subproblem involving opening a door and retrieving the other key. Crucially, this last subproblem is Markovian because the agent already knows which room the key is in based on which key they previously picked up.

Main contributions: We propose a discrete optimization-based approach that finds a high-level decomposition of a partially observable RL problem. This decomposition splits the problem into a set of Markovian subproblems and takes the form of a *reward machine (RM)* [8]. We also extend an existing method for exploiting RMs to the partially observable case, so that we can use a found RM to quickly learn a policy from low-level observations to actions. Finally, we show that our approach significantly outperforms several well-known policy gradient methods on three challenging grid domains.

Related work includes some early attempts to tackle partially observability in RL based on automata learning, e.g. [3, 4]. Both works rely on learning finite state machines at a low-level (over the environment observations). In contrast, our approach relies on learning a decomposition of the problem at the abstract level given by a labelling function. This allows our approach to also work over problems with continuous (or very large) observation spaces.

2 Preliminaries

A *Markov Decision Process (MDP)* is a tuple $\mathcal{M} = \langle S, A, r, p, \gamma \rangle$, where S is a finite set of *states*, A is a finite set of *actions*, $r : S \times A \to \mathbb{R}$ is the *reward function*, $p(s, a, s')$ is the *transition probability distribution*, and γ is the *discount factor*. The objective of $\mathcal{M}$ is to find a policy $\pi^* : S \to \Pr(A)$ that maximizes the expected discounted reward for every state $s \in S$. When r or p are unknown but can be sampled, an optimal policy can be found using RL approaches like *q-learning*. This *off-policy* method uses sampled experience of the form (s, a, s', r) to update $\tilde{q}(s, a)$, an estimate of the optimal *q-function*.

A *Partially Observable Markov Decision Process (POMDP)* is a tuple $\mathcal{P}_{\mathcal{O}} = \langle S, O, A, r, p, \omega, \gamma \rangle$, where S, A, r, p, and γ are defined as in an MDP, O is a finite set of *observations*, and $\omega(s, o)$ is the *observation probability distribution*. At every time step t, the agent is in exactly one state $s_t \in S$, executes an action $a_t \in A$, receives an immediate reward $r_{t+1} = r(s_t, a_t)$, and moves to the next state s_{t+1} according to $p(s_t, a_t, s_{t+1})$. However, the agent does not observe s_{t+1}, and only receives an observation $o_{t+1} \in O$ via ω, where $\omega(s_{t+1}, o_{t+1})$ is the probability of observing o_{t+1} from state s_{t+1} [1]. As such, many RL methods cannot be immediately applied to POMDPs because the transition probabilities and reward function are not necessarily Markovian w.r.t. O.

3 Learning to Decompose Partially Observable Problems

Our approach to RL in a partially observable environment has two stages. In the first, the agent solves an optimization problem over a set of traces to find a "good" *reward machine (RM)*-based [8] decomposition of the environment. In particular, we look for an RM $\mathcal{R}$ that can be used to make accurate one-step Markovian predictions over the traces in the training set. In the second stage, the agent uses any standard RL algorithm to learn a policy directly from low-level observations to actions for each subtask identified in $\mathcal{R}$. If at some point $\mathcal{R}$ is found to make incorrect predictions, additional traces are added to the training set and a new RM is learned. This process continues for as long as is desired.

Reward Machines under Partial Observability

Let us begin by defining RMs and identifying how a given RM can be used by an RL agent in a partially observable environment. RMs are finite state machines that give reward on every transition, and were recently proposed as a way to expose the structure of a reward function to an RL agent [8]. In the case of partial observability, RMs are defined over a set of propositional symbols $\mathcal{P}$ that correspond to a set of high-level features the agent can detect using a *labelling function*

$L : O_\emptyset \times A_\emptyset \times O \to 2^{\mathcal{P}}$ where $X_\emptyset = X \cup \{\emptyset\}$. L assigns truth values to symbols in $\mathcal{P}$ given an environment experience $e = (o, a, o')$ where o' is the next observation after executing action a from observation o. We use $L(\emptyset, \emptyset, o)$ to assign truth values to the initial observation. We call a truth value assignment over $\mathcal{P}$ an *abstract observation* since it provides a high-level view of the low-level observations via L. We now formally define an RM as follows:

Definition 3.1 (reward machine)**.** Given a set of propositional symbols $\mathcal{P}$, a set of (environment) observations O, and a set of actions A, a Reward Machine is a tuple $\mathcal{R}_{\mathcal{P}OA} = \langle U, u_0, \delta_u, \delta_r \rangle$ where U is a finite set of states, $u_0 \in U$ is an initial state, δ_u is the state-transition function, $\delta_u : U \times 2^{\mathcal{P}} \to U$, and δ_r is the reward-transition function, $\delta_r : U \times 2^{\mathcal{P}} \to \mathbb{R}$.

RMs decompose problems into high-level states U and define transitions using conditions defined by δ_u. These conditions are over a set of binary properties $\mathcal{P}$ that the agent can detect using L. For example, consider the RM for the 2-keys domain shown in Figure 1d. We assume that the agent can use L to detect the room color (🟥, ☐, 🟦, and 🟨), the objects in the current room (🔑, ☕, and 🔒, where 🔒 represents a locked door), and whether it is carrying a key (▲). Each of these symbols is in $\mathcal{P}$. In the figure, we use "(🟥, ▲)" to denote that 🟥 and ▲ are true in the current state, and all other propositions (e.g. ☐) are false. We also use "(🟥, ▲); (🟦, ▲)" to say that the transition is taken if either of these sets of propositions is satisfied. Finally, we note the figure only shows propositions sets that induce RM state changes. For all other sets, the RM simply remains in the same state it was in the last step.

The agent starts at the initial RM state u_0 and stays there until it observes the red room with no keys (🟥), one key (🟥,🔑) or two keys (🟥,🔑,🔑), or similarly for the blue room. Each of these conditions is associated with a unique arrow indicating the state to which the RM transitions. If the agent enters the blue room and there is one key (🟦,🔑), then the RM state changes from u_0 to u_1. The transitions in the RM are also associated with a reward via δ_r.

When learning policies given an RM, one simple approach is to learn a policy $\pi(o, x)$ that considers the current observation $o \in O$ and the current RM state $x \in U$. While a partially observable problem might be non-Markovian over O, it can be Markovian over $O \times U$ for some RM $\mathcal{R}_{\mathcal{P}OA}$. We call such an RM a *perfect RM*. For example, Figure 1d shows a perfect RM for the 2-keys domain given a labelling function that detects events 🟥, 🟦, 🔑, and ▲. It is perfect because it can correctly keep track of the locations of the keys once this is determined, which is all that the agent needs to remember in order to decompose the problem in a Markovian way. Formally, we define a perfect RM for POMDP $\mathcal{P}_\mathcal{O}$ as follows:

Definition 3.2. An RM $\mathcal{R}_{\mathcal{P}OA} = \langle U, u_0, \delta_u, \delta_r \rangle$ is considered *perfect* for a POMDP $\mathcal{P}_\mathcal{O} = \langle S, O, A, r, p, \omega, \gamma \rangle$ with respect to a labelling function L if and only if for every trace $o_0, a_0, \ldots, o_t, a_t$ generated by any policy over $\mathcal{P}_\mathcal{O}$, the following holds: $\Pr(o_{t+1}, r_{t+1} | o_0, a_0, \ldots, o_t, a_t) = \Pr(o_{t+1}, r_{t+1} | o_t, x_t, a_t)$ where $x_0 = u_0$ and $x_t = \delta_u(x_{t-1}, L(o_{t-1}, a_{t-1}, o_t))$.

Interestingly, we can formally show that if the set of belief states [1] for the POMDP $\mathcal{P}_\mathcal{O}$ is finite, then there exists a perfect RM for $\mathcal{P}_\mathcal{O}$. In addition, we can show that the optimal policies for perfect RMs are also optimal for $\mathcal{P}_\mathcal{O}$.

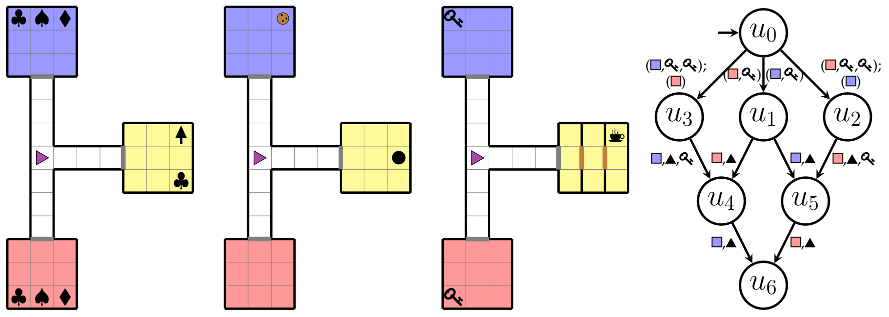
From Traces to Reward Machines

We now consider the problem of learning a perfect RM from traces, assuming one exists w.r.t. the given labelling function L. Since a perfect RM transforms the original problem into a Markovian problem over $O \times U$, we prefer RMs that accurately predict the next observation o' and the immediate reward r from the current observation o, the RM state x, and the action a. Instead of trying to predict the observations themselves, we propose a low-cost alternative which focuses on a necessary condition for a perfect RM: the RM must correctly predict what is *possible* and *impossible* at the abstract level given by L. It is impossible, for instance, to be at u_3 in the RM from Figure 1d and observe (🟥,🔑), because the RM is at u_3 iff the agent saw that the red room was empty or that both keys were in the blue room.

This idea is formalized in our optimization model (Figure 1e). Let $\mathcal{T} = \{\mathcal{T}_0, \ldots, \mathcal{T}_n\}$ be a set of traces, where each trace $\mathcal{T}_i$ is a sequence of observations, actions, and rewards: $\mathcal{T}_i = \{o_{i,0}, a_{i,0}, r_{i,0}, \ldots, o_{i,t_i}, a_{i,t_i}, r_{i,t_i}\}$. We now look for an RM $\langle U, u_0, \delta_u, \delta_r \rangle$ that can be used to predict $L(e_{i,t+1})$ from $L(e_{i,t})$ and the current RM state $x_{i,t}$, where $e_{i,t+1}$ is the experience $(o_{i,t}, a_{i,t}, o_{i,t+1})$ and $e_{i,0}$ is $(\emptyset, \emptyset, o_{i,0})$ by definition. The model parameters are the set of traces $\mathcal{T}$, the set of propositional symbols $\mathcal{P}$, the labelling function L, and a maximum number of states in the RM u_{max}. The model also uses the sets $I = \{0 \ldots n\}$ and $T_i = \{0 \ldots t_i - 1\}$, where I contains the index of the traces and T_i their time steps. The model has two auxiliary variables $x_{i,t}$ and $N_{u,l}$. Variable $x_{i,t} \in U$ represents the state of the RM after observing trace $\mathcal{T}_i$ up to time t. Variable $N_{u,l} \subseteq 2^{\mathcal{P}}$ is the set of all the next abstract observations seen from the RM state u and the abstract observations l at some point in $\mathcal{T}$. In other words, $l' \in N_{u,l}$ iff $u = x_{i,t}$, $l = L(e_{i,t})$, and $l' = L(e_{i,t+1})$ for some trace $\mathcal{T}_i$ and time t.

Constraints (2) and (3) ensure that we find a well-formed RM, while constraints (4) to (6) ensure that the found RM satisfies the current set of traces. Constraint (7) and (8) ensure that the $N_{u,l}$ sets contain at least every $L(e_{i,t+1})$ that has been seen right after l and u in $\mathcal{P}$. The objective function (1) comes from maximizing the log-likelihood for predicting $L(e_{i,t+1})$ using a uniform distribution over all the possible options given by $N_{u,l}$. A key property of this formulation is that any perfect RM is optimal with respect to the objective function in equation (1) when the number of traces (and their lengths) tends to infinity, if the traces are collected by a policy π such that $\pi(a|o) > \epsilon$ for all $o \in O$ and $a \in A$.

For solving this optimization problem, we found the local search algorithm *Tabu search* [2] to be effective. This method starts from an arbitrary feasible solution. It then iteratively examines all feasible "neighbouring" solutions, and moves



(a) Symbol domain. (b) Cookie domain. (c) 2-keys domain. (d) A perfect RM.

$$\begin{aligned}
&\underset{\langle U,u_0,\delta_u,\delta_r\rangle}{\text{minimize}} \sum_{i\in I}\sum_{t\in T_i} \log(|N_{x_{i,t},L(e_{i,t})}|) && (1)\\
&\text{s.t. } \langle U,u_0,\delta_u,\delta_r\rangle \in \mathcal{R}_{\mathcal{POA}} && (2)\\
&\quad |U| \leq u_{\text{max}} && (3)\\
&\quad x_{i,t} \in U \qquad \forall i \in I, t\in T_i && (4)\\
&\quad x_{i,0} = u_0 \qquad \forall i \in I && (5)\\
&\quad x_{i,t+1} = \delta_u(x_{i,t}, L(e_{i,t+1})) \;\; \forall i \in I, t\in T_i && (6)\\
&\quad N_{u,l} \subseteq 2^{\mathcal{P}} \qquad \forall u \in U, l \in 2^{\mathcal{P}} && (7)\\
&\quad L(e_{i,t+1}) \in N_{x_{i,t},L(e_{i,t})} \qquad \forall i \in I, t\in T_i && (8)
\end{aligned}$$

(e) Optimization model for learning RMs.

Figure 1: Our domains, a perfect RM for the keys domain, and our optimization model.

to the neighbour with the best evaluation according to the objective function. For us, neighbouring RMs are defined as those that differ by exactly one transition. When a time limit is hit, the best seen solution is returned. Tabu search also maintains a set of states, the *Tabu list*, and prunes them from the "neighbouring" solutions to avoid revisiting them.

Finding Policies For Learned Reward Machines

Once we have learned an RM, we can use any RL algorithm to learn a policy $\pi(o, u)$, by treating the combination of o and u as the current state. However, doing so does not exploit the problem structure that is exposed by the RM. To this end, an approach called *Q-Learning for RMs (QRM)* was proposed [8]. QRM learns one q-function $\tilde{q}_u$ (i.e., policy) per RM state $u \in U$. Then, given any sample transition, the RM can be used to emulate how much reward each q-value would receive from every RM state. Formally, experience $e = (o, a, o')$ is transformed into a valid experience $(\langle o, u\rangle, a, \langle o', u'\rangle, r)$ for training $\tilde{q}_u$ for each $u \in U$, where $u' = \delta_u(u, L(e))$ and $r = \delta_r(u, L(e))$. Hence, any off-policy learning method can take advantage of these "synthetically" generated experiences to train every q-function $\tilde{q}_u$. When q-learning is used to learn each policy, QRM is guaranteed to converge to an optimal policy when the problem is fully-observable.

To apply QRM on a learned RM in a partially observable environment, we must first learn values for the RM's reward function δ_r from the set of training traces $\mathcal{T}$. We do so by setting $\delta_r(u, l)$ as its empirical expectation over $\mathcal{T}$. In addition, we must handle an issue related to importance sampling. An experience (o, a, o') might be more or less likely depending on the RM state that the agent was in when the experience was collected. For example, experience (o, a, o') might be possible in one RM state u_i but not in u_j. Updating the policy for u_j using (o, a, o') would then introduce an unwanted bias to $\tilde{q}_{u_j}$. We handle this issue by only "transferring" an experience (o, a, o') from u_i to u_j, if the current RM indicates that experience is possible in u_j. For example, if some experience in Figure 1c consists of entering the red room and seeing only one key, then this experience will not be used to update the policies for states u_2, u_3, u_4, and u_6 of the perfect RM in Figure 1d. While this approach will not address the problem in all environments, we leave that as future work.

Simultaneously Learning a Reward Machine and a Policy

We now describe our overall approach for simultaneously finding an RM and exploiting that RM to learn a policy. Our approach starts by collecting a training set of traces $\mathcal{T}$ generated by a random policy during t_{w} "warmup" steps. This set of traces is used to find an initial RM $\mathcal{R}$ using Tabu search. The algorithm then initializes policy π, sets the RM state to the initial state u_0, and sets the current label l to the initial abstract observation $L(\emptyset, \emptyset, o)$. The standard RL learning loop is then followed: an action a is selected following $\pi(o, x)$, and the agent receives the next observation o' and the immediate reward r. The RM state is then updated to $x' = \delta_u(x, L(o, a, o'))$ and the policy π is improved using whatever RL method is being deployed using the last experience $(\langle o, x\rangle, a, r, \langle o', x'\rangle)$. Note that in an episodic task, the environment and RM are reset whenever a terminal state is reached.

If on any step, there is evidence that the current RM might not be the best one, our approach will attempt to find a new one. Recall that the RM $\mathcal{R}$ was selected using the cardinality of its prediction sets N (1). Hence, if the current abstract observation l' is not in $N_{x,l}$, adding the current trace to $\mathcal{T}$ will increase the size of $N_{x,l}$ for $\mathcal{R}$. As such, the cost of $\mathcal{R}$ will increase and it may no longer be the best RM. Thus, if $l' \notin N_{x,l}$, we add the current trace to $\mathcal{T}$ and search for a new RM. Recall that we use Tabu search, though any discrete optimization method could be applied. Our method only uses the new RM if its cost is lower than $\mathcal{R}$'s. If the RM is updated, a new policy is learned from scratch.

4 Evaluation and Discussion

We tested our approach on three partially observable grid domains, each with the same layout of three rooms with a connecting hallway. The agent can move in the four cardinal directions and can only see what is in the current room. These are stochastic domains where the outcome of an action randomly changes with a 5% probability. The first environment is

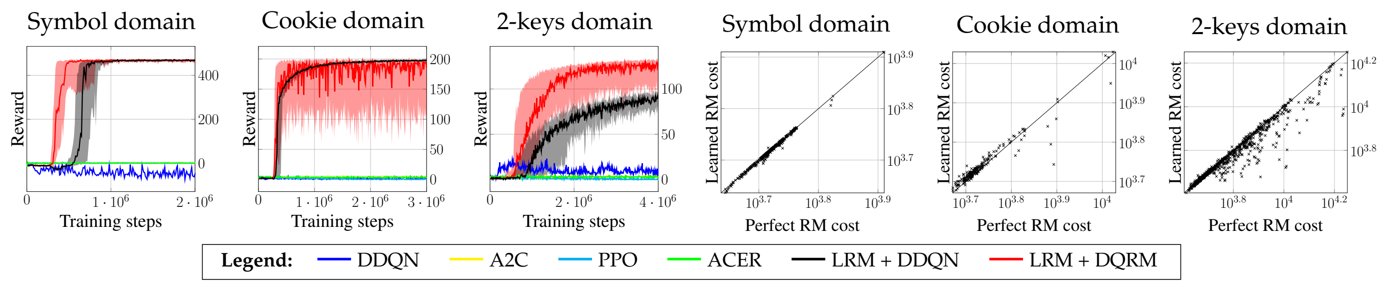


Figure 2: **Left**: Total reward collected every $10,000$ training steps. It shows the median performance over 30 runs and percentile 25 to 75 in the shadowed area for LRM approaches. The maximum performance is reported for the other approaches. **Right**: Comparison between the cost of the perfect RM and the cost of RMs found by Tabu search.

the *symbol domain* (Figure 1a). It has three symbols $\clubsuit$, $\spadesuit$, and $\blacklozenge$ in the red and blue rooms. One symbol from $\{\clubsuit, \spadesuit, \blacklozenge\}$ and possibly an up or down arrow are randomly placed at the yellow room. Intuitively, that symbol and arrow tell the agent where to go (e.g., $\clubsuit$ and $\uparrow$ tell the agent to go to $\clubsuit$ in the north room). If there is no arrow, the agent can go to the target symbol in either room. An episode ends when the agent reaches any symbol in the red or blue room, at which point they receive a reward of $+1$ if they reached the correct symbol and -1 for an incorrect symbol. The second environment is the *cookie domain* (Figure 1b). It has a button in the yellow room that, when pressed, makes a cookie randomly appear in the red or blue room. The agent receives reward $+1$ for reaching the cookie and may then go back to the button to make another one appear. Each episode is $5,000$ steps long, during which the agent should attempt to get as many cookies as possible. The final environment is the *2-keys domain* (Figure 1c) that was described in Section 1.

We tested two versions of our Learned Reward Machine (LRM) approach: LRM+DDQN and LRM+DQRM. Both learn an RM from experience as described in Section 3, but LRM+DDQN learns a policy using DDQN [9] while LRM+DQRM uses the modified version of QRM. In all domains, we used $u_{\max} = 10$, $t_{\mathrm{w}} = 200,000$, an epsilon greedy policy with $\epsilon = 0.1$, and a discount factor $\gamma = 0.9$. The size of the Tabu list and the number of steps that the Tabu search performs before returning the best RM found is 100. We compared against 4 baselines: DDQN [9], A2C [5], ACER [10], and PPO [7]. To provide DDQN some memory, its input is set as the concatenation of the last 10 observations, as commonly done by Atari playing agents. In contrast, A2C, ACER, and PPO already use an LSTM to summarize the observation history.

The left side of Figure 2 shows the total reward that each approach gets every $10,000$ training steps. The figure shows that the LRM approaches largely outperform all the baselines. We also note that LRM+DQRM learns faster than LRM+DDQN, but is more unstable. In particular, LRM+DQRM converged to a considerably better policy than LRM+DDQN in the 2-keys domain. We believe this is due to QRM's experience sharing mechanism that allows for propagating sparse reward backwards faster. In contrast, all the baselines outperformed a random policy, but none make much progress on any of the domains, even when run much longer ($5,000,000$ steps).

A key factor in the strong performance of the LRM approaches is that Tabu search finds high-quality RMs in less than 100 search steps (Figure 2, right side). In each plot, a point compares the cost of a handcrafted perfect RM with that of an RM $\mathcal{R}$ that was found by Tabu search while running our LRM approaches, where the costs are evaluated relative to the training set used to find $\mathcal{R}$. Being on or under the diagonal line (as in most of the points in the figure) means that Tabu search is finding RMs whose values are at least as good as the handcrafted RM. Hence, Tabu search is either finding perfect reward machines or discovering that our training set is incomplete and our agent will eventually fill those gaps.

For future work, we plan on exploring the use of recurrent neural networks for finding policies for each RM subtask. Doing so would mean that we might not have to find a perfectly Markovian high-level decomposition. We expect this will allow us to solve problems with less informative labelling functions, and using RMs with fewer states.

References

[1] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In *AAAI*, pages 1023–1028, 1994.

[2] F. Glover and M. Laguna. Tabu search. In *Handbook of combinatorial optimization*, pages 2093–2229. Springer, 1998.

[3] M. Mahmud. Constructing states for reinforcement learning. In *ICML*, pages 727–734, 2010.

[4] N. Meuleau, L. Peshkin, K.-E. Kim, and L. P. Kaelbling. Learning finite-state controllers for partially observable environments. In *UAI*, pages 427–436, 1999.

[5] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, pages 1928–1937, 2016.

[6] J. Oh, V. Chockalingam, S. Singh, and H. Lee. Control of memory, active perception, and action in minecraft. In *ICML*, pages 2790–2799, 2016.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[8] R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *ICML*, pages 2112–2121, 2018.

[9] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100, 2016.

[10] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.