# Efficient Reinforcement Learning with Relocatable Action Models

**Bethany R. Leffler** and **Michael L. Littman** and **Timothy Edmunds**
{bleffler,mlittman,tedmunds}@cs.rutgers.edu
Department of Computer Science
Rutgers University
NJ, USA

## Abstract

Realistic domains for learning possess regularities that make it possible to generalize experience across related states. This paper explores an environment-modeling framework that represents transitions as state-independent outcomes that are common to all states that share the same type. We analyze a set of novel learning problems that arise in this framework, providing lower and upper bounds. We single out one particular variant of practical interest and provide an efficient algorithm and experimental results in both simulated and robotic environments.

## Introduction

Early work in reinforcement learning focused on learning value functions that generalize across states (Sutton 1988; Tesauro 1995). More recent work has sought to illuminate foundational issues by proving bounds on the resources needed to learn near optimal policies (Fiechter 1994; Kearns & Singh 2002; Brafman & Tennenholtz 2002). Unfortunately, these later papers treat states as being completely independent. As a result, learning times tend to scale badly with the size of the state space—experience gathered in one state is not reused to learn about any other state. The generality of these results makes them too weak for use in real-life problems in robotics and other problem domains.

The work reported in this paper builds on advances that retain the formal guarantees of recent algorithms while moving toward algorithms that generalize across states. These results rely critically on assumptions and the best assumptions are those that are both satisfied by relevant applications and provide measurable computational leverage. Examples of assumptions for which formal results are known include the availability of a small dynamic Bayes net representation of state transitions (Kearns & Koller 1999), local modeling accuracy (Kakade, Kearns, & Langford 2003), and clearly divided clusters of transitions (Leffler *et al.* 2005).

The main assumption adopted in the current work is that states belong to a relatively small set of *types* that determine their transition behavior. In the remainder of the paper, we define our assumption, demonstrate that it leads to provable improvements in learning efficiency, and finally show that it is suitable for improving performance in a standard grid-world simulation and a robot-navigation task with distinct terrains. Because it provides measurable benefits in learning efficiency and is also a natural match for a real-life problem, we conclude that our assumption has value.



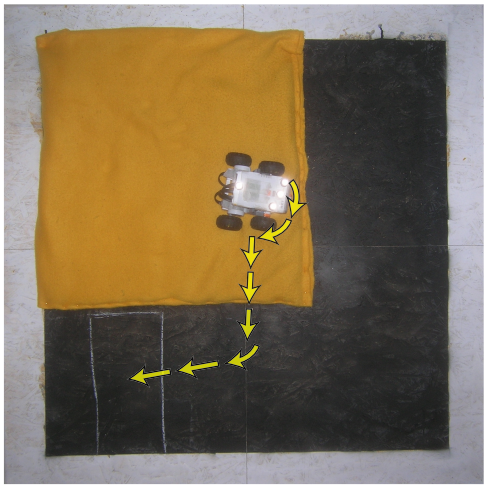

Figure 1: A robot navigates in the real world using our RAM-Rmax algorithm. Also shown is the path followed by the agent to reach the goal from the starting position.

## Background

A Markov decision process (MDP) is defined by a set of states $S$, actions $A$, transition function $T(s, a, s')$ (the probability of a transition from state $s \in S$ to $s' \in S$ when action $a \in A$ is taken), discount factor $0 \leq \gamma \leq 1$ and reward function $R(s, a)$ (expected immediate reward for taking action $a \in A$ from state $s \in S$).

An MDP defines a formal model of an environment that an agent can interact with and learn about. In the reinforcement-learning setting (Sutton & Barto 1998), the agent begins knowing the state and action spaces, but does not have knowledge of the transition and reward functions.

### An MDP Decomposition

Sherstov & Stone (2005) presented a formalism for MDPs

that we call the RAM (relocatable action model) representation. The formalism provides a decomposition, or factorization, of the transition function $T$ into three other functions:

- $\kappa : S \to C$ is the *type function*. It maps each state to a type (or cluster or class) $c \in C$.
- $t : C \times A \to \Pr(O)$ is the *relocatable action model*. It captures the outcomes of different actions in a state-independent way by mapping a type and action to a probability distribution over possible outcomes.
- $\eta : S \times O \to S$ is the *next-state function*. It takes a state and an outcome and provides the next state that results.

Additionally, $r : C \to \Re$ is a version of the reward function that depends only on state types. Thus, a RAM representation is defined by a set of states $S$, actions $A$, types $C$, outcomes $O$, type function $\kappa$, relocatable action model $t$, next-state function $\eta$, and reward function $r$.

To connect these quantities to standard MDP definitions, we describe how MDPs specified in either format can be captured by the other. First, if we have a RAM representation $\langle S, A, C, O, \kappa, t, \eta, r\rangle$, an equivalent MDP can be written as $\langle S, A, T', R'\rangle$ as follows. First, $R'(s, a) = r(\kappa(s), a)$. That is, the reward for action $a$ in state $s$ is found by checking the type of $s$ ($\kappa(s)$), then looking up the reward value for that type. Similarly, the transition probability is

$$T'(s, a, s') = \sum_{o \text{ s.t. } \eta(s,o)=s'} t(\kappa(s), a, o).$$

That is, the probability of transitioning to state $s'$ is found by considering all possible outcomes $o \in O$ for which the next-state function $\eta(s, o)$ takes us to $s'$. We then sum up, for each such outcome, the probability that $s$'s type results in that outcome.

Given an MDP $\langle S, A, T', R'\rangle$, we can construct a RAM representation $\langle S, A, C, O, \kappa, t, \eta, r\rangle$. Specifically, we can take $O = S$ and $C = S$. Then, the type function and next-state function are essentially identity functions $\kappa(s) = s$ and $\eta(s, s') = s'$. The relocatable action model is then just the transition function itself $t(s, a, s') = T'(s, a, s')$ and the reward function remains the same $r(s, a) = R'(s, a)$.

However, the RAM representation of transitions can sometimes be much smaller than the standard MDP representation. Let's take $n = |S|$ to be the size of the state space and $m = |A|$ to be the size of the action space. In addition, let $B$ be the number of bits needed to write down a transition probability. The standard MDP representation has size $mn^2B$ since there is an $n \times n$ transition matrix for each action. If the transition function is sparse with $h$ non-zero outcomes per transition, each state–action pair would need to list $h$ next states (each requiring at least $\log n$ bits), plus their probabilities, for a size of $nmh(\log n + B)$.

For the RAM representation, let $k = |C|$ be the number of types and $l = |O|$ be the number of outcomes. This representation has size $n \log k + kmlB + nl \log n$. Considering the representation size as a function of the number states and actions, we have $\Theta(n \log n + m)$ for the RAM representation and the much larger $\Theta(mn^2)$ for the standard representation.

For concreteness, let's consider a well-known example of grid-world dynamics (Russell & Norvig 1994). From each grid position, the agent can choose any of four directions. Transitions will take the agent in the intended direction with probability 0.8, perpendicularly left of the intended direction with probability 0.1, and perpendicularly right of the intended direction with probability 0.1. However, if motion in the resulting direction is blocked by a wall, the action will not change the state.

In the standard representation, each state has four possible actions. For each of these choices, there are at most four next states with non-zero probability. Thus, a sparse representation has at most $16n$ next states along with associated probabilities, for a size of $16n \log n + 16nB$.

In the RAM representation, we can assign the states to types in a number of different ways. One natural approach is to define the set of types as the $k = 16$ possible surrounding wall patterns and the $l = 5$ possible directional outcomes (including no movement). Outcomes then have the probabilities defined above, with no movement occurring in state types that would result in collisions with walls. For example, taking $NE$ to be the state type in which walls are to the north and east, n to be the action of attempting to go north, and $x$ to be the outcome of not moving, $t(NE,\text{n}, x) = 0.9$ since northward (probability 0.8) and eastward (probability 0.1) movement is blocked, resulting in no movement in these cases. Finally, the next-state function has an extremely simple structure derived from the relative locations of the grid squares. Based on these definitions, the total representation size is $4n + 320B + 5n \log n$[1]. For large $n$, this representation is about 1/3 the size of the standard representation.

In this example, the type function captures the contents of the grid cells—where the walls are. The relocatable action model captures the local dynamics—how movement works. The next-state function captures the topology of the state space—the neighborhood relations in the grid. The representation size is on par with the information needed in an informal description of the domain. Smaller representations can often mean simpler learning problems because there are fewer values that need to be estimated from experience. Next, we describe several RAM-representation-based learning problems and compare their relative difficulty.

## Learning Problems and Analysis

To define algorithms that learn with the RAM representation, we assume that the number of types $k$ and outcomes $l$ are small relative to the number of states. Without this constraint, as we saw earlier, the RAM representation is really no different from the standard MDP representation and the same algorithms and complexity bounds apply.

We focus on learning transition functions in this paper—learning rewards presents a similar, though simpler, set of issues. To model transition behavior, the learner has three functions to estimate: $\kappa$, $t$, and $\eta$. Different learning problems result from assuming that different functions are provided to the learner as background knowledge.

The relocatable action model $t$ is the central structure to be learned, since it captures how different actions behave. In the remainder of this section, we show that learning $t$ when

[1] Corrections to the originally published paper are noted in red.

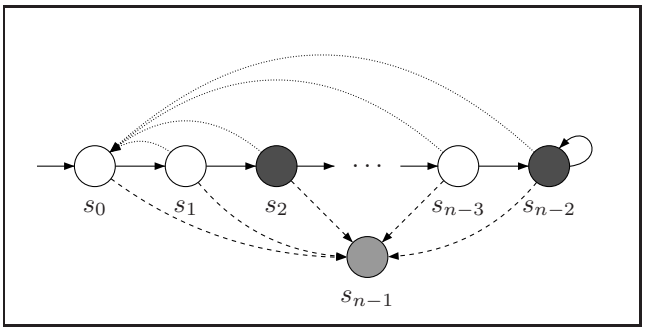


Figure 2: A RAM combination-lock environment showing 3 types (black $B$, white $W$, and gray/goal $G$) and 3 outcomes (solid/forward $F$, dashed/end $E$, and dotted/reset $R$).

only one of $\kappa$ and $\eta$ is known results in a learning problem that is no easier than learning in the general MDP representation. However, learning $t$ alone is a substantially easier problem. Our experimental sections provide justification for how $\kappa$ and $\eta$ can sometimes be derived in advance.

To provide intuition about the difficulty of the learning problem, we consider the question *How many steps does a learner need to take in an environment with initially unknown deterministic dynamics before it reaches an unknown goal state?* Answering this question provides a lower bound for the more general setting since it constitutes a special case in which rewards capture uniform step costs and a goal state, dynamics are deterministic, and near optimal behavior cannot be achieved until the goal is reached.

**Example Dynamics.** Our results can all be stated using a family of related problems, generalized from the results of Koenig & Simmons (1993) and illustrated in Figure 2. We call these problems "RAM combination-lock environments" (the combination is the sequence of actions that move the agent from the start state to the goal) and they have $n$ states and $m$ actions. State $s_0$ is a start state, state $s_{n-1}$ is the goal state, and states $s_1, \ldots, s_{n-2}$ are arranged as a sequential chain. There are $k = 3$ types of states: white states $W$, black states $B$, and one gray goal state $G$. There are $l = 3$ outcomes: move forward $F$, move to goal end $E$, and reset to start state $R$.

**Next-state Function Known.** Consider any learner in a RAM combination-lock environment where the next-state function $\eta$ is known. That is, for every state $s_i$, the learner knows that the $E$ outcome results in a transition to the goal $s_{n-1}$, the $F$ outcome results in a transition to $s_{i+1}$ (when $i = n-2$, we assume $F$ is a self transition), and the $R$ outcome results in a reset to state $s_0$. It also knows that each state $s_0, \ldots, s_{n-1}$ has one of three types, but doesn't know which is which ($\kappa$ is unknown). Finally, it knows that actions consistently map types to outcomes.

We adversarially construct a RAM combination-lock environment in which one arbitrarily chosen state $s_{i^*}$ is white, and all other non-goal states are black: $\kappa(s_{i^*}) = W$, $\kappa(s_{n-1}) = G$, and $\kappa(s_i) = B$, for all $0 \leq i < n$ where $i \neq i^*$. The relocatable action model for black states is for one action to move forward and all other actions to reset. For the white state, one action moves forward, one other arbitrary action $a_{j^*}$ goes to the goal state and all others reset. Symbolically, $t(B, a_0) = F$, $t(B, a_j) = R$ for $0 < j < m$, $t(W, a_0) = F$, $t(W, a_{j^*}) = E$, and $t(W, a_j) = R$ for $0 < j < m$ and $j \neq j^*$.

Since the only difference between white and black states is the outcome of one of the actions ($a_{j^*}$) in one of the states ($s_{i^*}$), the learner cannot identify the white state $s_{i^*}$ or the action that reaches the goal $a_{j^*}$ without trying at least $m-1$ actions in at least $n-1$ states. In addition, each time an action is taken in a state $s_i$ that does not result in reaching the goal, the state resets to $s_0$ and $i$ steps are needed to return to $s_i$ to continue the search. Since half of the values $i$ are greater than $n/2$, no learner in such an environment can be guaranteed to reach the goal in fewer than $\Omega(mn^2)$ steps. This bound matches that of Koenig & Simmons (1993), who also provide matching upper bounds using variants of Q-learning.

**Type Function Known.** Next, consider any learner in a RAM combination-lock environment where the type function $\kappa$ is known. That is, for every state $s_i$, the learner can "see" whether the state is white or black (or gray). It knows that all states of the same color produce the same outcomes in response to the same actions. However, the next-state function $\eta$ is unknown, so for each state $s$, the state that results from an outcome $o$ is not known.

We adversarially construct a RAM combination-lock environment in which roughly half of the states are white and half are black. The goal is gray, as before. Once again, action $a_0$ produces the forward outcome: $t(W, a_0) = t(B, a_0) = F$. For some arbitrarily chosen action $a_{j^*}$, $t(W, a_{j^*}) = t(B, a_{j^*}) = E$. All other actions reset: $t(W, a_j) = t(B, a_j) = R$ for $0 < j \leq m-1$ and $j \neq j^*$. The "secret" of the environment is that only one state has the $E$ outcome resulting in a transition to the goal state. We define the next-state function as:

$$\begin{aligned}
\eta(s, R) &= s_0, \text{ for all } s, \\
\eta(s_i, F) &= s_{i+1}, \text{ for } i < n-2, \\
\eta(s_i, F) &= s_i, \text{ for } i = n-2, \\
\eta(s_i, E) &= s_0, \text{ for } 0 < i < n-1 \text{ and } i \neq i^*, \\
\eta(s_{i^*}, E) &= s_{n-1}.
\end{aligned}$$

All states are essentially identical—action $a_0$ moves forward and all other actions reset to $s_0$. Only one action for one state produces an outcome that reaches the goal. Since outcomes $E$ and $R$ are only distinguishable in this one case, the learner has no choice but to try at least $m-1$ actions in at least $n-1$ states. Once again, $\Omega(mn^2)$ steps are required.

**Type and Next-state Functions Known.** In contrast, if both the next-state and type functions are known in advance, learners can exploit the structure in the environment to reach the goal state significantly faster.

The algorithm is presented in its general form in the next section (Figure 1). For domains like the ones described in this section, it behaves as follows. It keeps track of which $(c, a)$ pairs have been seen. For those that have been seen,

**Algorithm 1**: The RAM-Rmax algorithm for efficient learning in the RAM representation.

**Global data structures**: a value table $Q$, a transition count table $t_C$
**Constants**: maximum reward $\mathrm{rmax}$, experience threshold $M$

**forall** *cluster* $c \in C$, *action* $a \in A$, *outcome* $o \in O$ **do**
  $t_C(c, a, o) \leftarrow 0$;
**forall** *state* $s \in S$, *action* $a \in A$ **do**
  $Q(s, a) \leftarrow \mathrm{rmax}$;
$s_{cur} \leftarrow s_{start}$;
**while** $s \notin S_{terminal}$ **do**
  $s' \leftarrow$ `TakeAction(`$\arg\max_{a \in A} Q(s_{cur}, a)$`)`;
  **forall** *outcome* $o \in O$ **do**
    **if** $\eta(s_{cur}, o) = s'$ **then**
      $t_C(\kappa(s_{cur}), a, o) \leftarrow t_C(\kappa(s_{cur}), a, o) + 1$;
  **repeat**
    **forall** *state* $s \in S$, *action* $a \in A$ **do**
      $z \leftarrow \sum_{o \in O} t_C(\kappa(s), a, o)$;
      **if** $z < M$ **then** $Q(s, a) \leftarrow \mathrm{rmax}$;
      **else**
        $Q(s, a) \leftarrow r(s, a) \ldots$
        $+\gamma \sum_{o \in O} [t_C(\kappa(s), a, o)/z \ldots$
        $\times \max_{a' \in A} Q(\eta(s, o), a')]$;
  **until** $Q$ *stops changing* ;
  $s_{cur} \leftarrow s'$;

it records the outcome $o$ that resulted. It uses its learned model of transitions, along with the known type and next-state function to build a graph of the known environment. It always acts to take a path to the nearest state $s$ for which $(\kappa(s), a)$ has not been seen for some $a \in A$. Each such path is no longer than $n$ steps. Since there are only $k \times m$ pairs to see, the algorithm must reach the goal in $O(nmk)$ steps. Since we're assuming the number of types $k$ is much smaller than the number of states $n$, this bound is a big improvement over what is possible in the general case.

## Algorithm

The algorithm we propose (listed in Algorithm 1) is a variation of Rmax (Brafman & Tennenholtz 2002), although other model-based algorithms could be used. It is very similar to factored-Rmax (Guestrin, Patrascu, & Schuurmans 2002; Kakade 2003), which is a learning algorithm for environments in which transitions can be modeled using a set of dynamic Bayes nets. It assumes the structure of these nets is known but conditional probability values need to be learned. In our *RAM-Rmax* algorithm, the RAM representation of the environment ($\kappa$ and $\eta$) is known and the learner must estimate the missing conditional probability values $t$ to act near optimally.

Concretely, the RAM-Rmax algorithm receives as input $\kappa$ (the type function), $\eta$ (the next-state function), $r$ (the reward function, which could be learned), $\mathrm{rmax}$ (the maximum reward), and $M$ (the experience threshold or minimum number of transition samples needed to estimate probabilities). At each decision point, the agent is told its current state $s$. After choosing an action and executing it in the environment, the agent is told the resulting state $s'$. It keeps a count $t_C(c, a, o)$ of the outcomes $o$ observed from each type–action pair $(c, a)$.[2] It uses these statistics to create an empirical probability distribution over all possible outcomes, but only for type–action pairs that have been seen at least $M$ times. The $Q(s, a)$ value for a state–action pair $(s, a)$ is set to the maximum reward $\mathrm{rmax}$ if the corresponding transition probabilities are based on fewer than $M$ samples. All other values are determined by solving the Bellman equations.

Like Rmax, RAM-Rmax chooses actions greedily with respect to these computed $Q(s, a)$ values because they include the necessary information for encouraging exploration. In fact, RAM-Rmax, when applied to a general MDP written in the RAM representation via the transformation described earlier is precisely the Rmax algorithm.

**Theorem 1** *In a RAM MDP with $n$ states, $m$ actions, $k$ types, and $l$ outcomes, there is a value of $M$, roughly $l/\epsilon^2(1-\gamma)^4$, so that RAM-Rmax will follow a $4\epsilon$-optimal policy from its current state on all but $O(kml/(\epsilon^3(1-\gamma)^6))$ timesteps (ignoring log factors), with probability at least $1 - 2\delta$.*

The result can be proven using the tools of Strehl, Li, & Littman (2006). The numerator of the bound, $kml$, is the size of the relocatable action model. The analogous result for Rmax in general MDPs is a numerator of $n^2m$, the size of the general MDP model, which is substantially larger when $k$ and $l$ are small.

## Grid-World Experiment

As a first evaluation of our algorithm, we created a $9 \times 9$ grid world with 69 states, 1 goal, and 11 "pits", shown in Figure 3. The problem is inspired by the well-known marble-maze puzzle, although the dynamics are quite different.

The transition dynamics of the problem were identical to those of the grid world described earlier. The pits appear as shaded positions in the figure. If the agent enters a pit, it receives a reward of $-1$ and the run terminates. Each run begins in the state in the upper right corner and ends with a reward of $+1$ if the goal is reached. Each step has a reward of $-0.001$. The 69 different states are each one of 16 different types, depending on the local arrangement of walls.

Our learning agent was informed of the location of the goal, the state-to-type mapping $\kappa$, the reward function $r$, and the outcome function $\eta$, which is derived directly from the geometry of the grid. To master the task, the agent has to learn the type-specific outcomes of its actions and use this knowledge to build an optimal policy.

[2] We assume that outcomes can be uniquely determined from an $s, a, s'$ triple. More sophisticated approaches can be applied to learn when outcomes are not "observable", but they are beyond the scope of this paper.

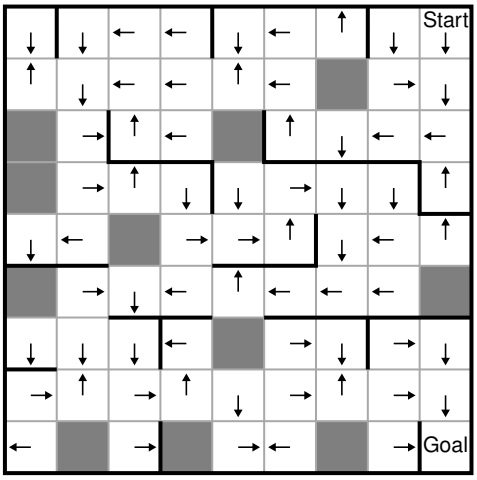


Figure 3: Grid world. Gray and "Goal" squares are absorbing states with $-1$ and $+1$ reward, respectively. The optimal policy for a per step reward of -0.001 is shown by arrows.

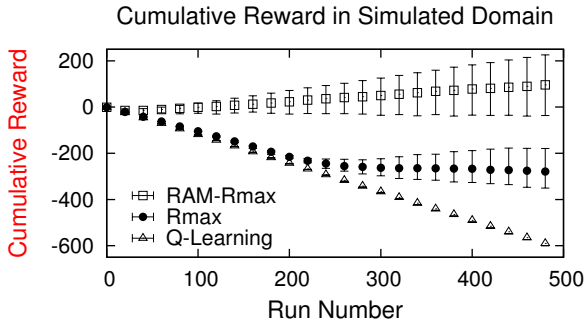


Figure 4: Comparing algorithms in a grid-world domain based on average cumulative reward. The error bars show the minimum and maximum reward over 10 experiments.

The optimal policy in this environment requires an average of approximately 453 steps to reach the goal. Because of the low step cost and the high cost of falling into a pit, the optimal choice when next to a pit is to attempt to move directly away from it. While this choice may lead the agent farther away from the goal, there is also a 10% chance that the agent will move in the desired direction and zero probability of falling into the pit.

Three learning algorithms were evaluated in this domain—RAM-Rmax, Rmax, and Q-learning. For both RAM-Rmax and Rmax, $M$, the experience threshold, was set to 5 based on informal experimentation. For Q-learning, exploration rate $\epsilon$ and learning rate $\alpha$ (Sutton & Barto 1998) were set equal to 0 and .1, respectively; several values were tested for these parameters and this combination was the first pair that resulted in convergence. The discount rate was set to $\gamma = 1$, for simplicity, since the task is episodic.

Figure 4 shows the cumulative reward each of these algorithms received over 500 runs (where each run begins when the agent is in the start state and ends when the agent enters a terminal state). The elbows of each of the curves indicate roughly where the learners begin to follow the optimal policy. Notice that for RAM-Rmax this event occurs between runs 50 and 60, for Rmax between runs 260 and 270, and for Q-learning around run 2000 (not shown).

As predicted, the additional structure provided to the RAM-Rmax learner allows it to identify an optimal policy more quickly than if this information were not available.

## Robotic Experiments

In this section, we describe the robotic experiments. These experiments demonstrate the utility of the RAM-Rmax algorithm in a real-life task and show that its assumptions can realistically be satisfied.

### The Environment

Our robot was a 4-wheeled robot constructed from the Lego Mindstorm NXT kit. Motors worked the front two tires independently. Control computations were performed on a laptop, which issued commands to the robot via Bluetooth.

The experimental environment was a $4 \times 4$-foot "room" with two different surfaces textures—wood and cloth. The surface types and configuration were chosen to assess the effectiveness of our learning algorithm. We found that that the robot traversed the cloth roughly 33% more slowly than it did the wood. Figure 1 provides an overview of the room.

An image taken from a camera placed above the empty room was sent as input into an image parser so the system could infer the mapping between surface types and the $x, y$ components of the state space. For this experiment, the surfaces were identified using a hand-tuned color-based classifier. Mounted around the room was a VICON motion-capture system, which we used to determine the robot state in terms of position and orientation.

### The Input and Output

At the beginning of each timestep of the experiment, the robot, or agent, is fed several pieces of information about its state. The localization system tells the robot its state and the image parser informs the robot of which surface type $c$ is associated with the current state. The robot knows from the beginning of the experiment which states are terminal (goal states with reward $rmax = 1$, and boundary states with reward $rmin = -1$), and the step reward $(-0.01)$ for each action that does not take it to a terminal state.

The actions that the robot can take are limited to turn left, turn right, and go forward. Each action is performed for 500 ms. After each action is taken, there is a 250 ms delay to allow the robot to come to a complete stop. Then, state information is once again sent to the agent and is re-evaluated to determine the next action. The cycle is repeated until the robot enters a terminal state.

### Results

We evaluated two learners, RAM-Rmax and Rmax. For both of these algorithms, we used $M = 4$ and set $\gamma = 1$. Figure 5 shows the cumulative reward that each of these learners received over 50 runs (where each run begins with the agent

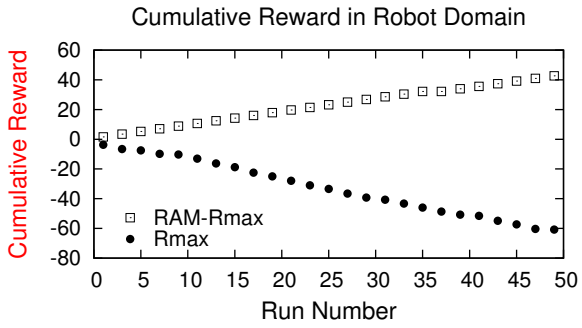


Figure 5: Comparing algorithms in a robot domain based on cumulative reward. Only one experiment is shown, due to the time necessary to run real-world experiments.

being placed in roughly the same place and ends when the robot enters a terminal state). The rapid rise of the RAM-Rmax curve shows that the learner almost immediately starts to follow an excellent policy. After the first run, the exploration phase of the learning was complete; all actions had been explored $M$ times in both state types. Over the next few runs, the model (and therefore the policy) was fine tuned. From run 5 on, the path taken seldom varied from that shown in Figure 1. Rmax, on the other hand, held the value of $\mathrm{rmax}$ for the majority of state–action pairs after 50 runs—it was still actively exploring. Setting $M = 1$ did not visibly speed the learning process.

The policy learned by Rmax and RAM-Rmax was somewhat more effective with a finely discretized state space, as its model was more accurate. Increasing the resolution in this way had a very negative impact on Rmax, since there were more states to explore. However, since RAM-Rmax's exploration time depends on the number of *types* of states, it was not affected apart from the increased computational cost of solving the model.

## Conclusion

In this work, we augmented a well-known reinforcement-learning algorithm, Rmax, with a particular kind of prior knowledge. The form of this knowledge was that each state is associated with a type and its type can be directly "seen" by the learner. In addition, states are related according to some known underlying "geometry" that allows state transitions to be predicted once a relocatable action model is learned. We illustrated these assumptions with two examples—a classic grid-world simulation and a robotic navigation task. We also provided formal results showing that these assumptions make it possible to learn more efficiently than is possible in general environments while some similar assumptions cannot be used to improve efficiency.

Future work will explore approaches for learning state types from perceptual experience with the goal of finding algorithms that are just as efficient but more autonomous than the algorithm presented here.

## Acknowledgements

This material is based upon work supported by NSF ITR-0325281, IIS-0329153, DGE-0549115, EIA-0215887, IIS-0308157, and DARPA HR0011-04-1-0050.

## References

Brafman, R. I., and Tennenholtz, M. 2002. R-MAX—a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3:213–231.

Fiechter, C.-N. 1994. Efficient reinforcement learning. In *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, 88–97. Association of Computing Machinery.

Guestrin, C.; Patrascu, R.; and Schuurmans, D. 2002. Algorithm-directed exploration for model-based reinforcement learning in factored MDPs. In *Proceedings of the International Conference on Machine Learning*, 235–242.

Kakade, S.; Kearns, M.; and Langford, J. 2003. Exploration in metric state spaces. In *Proceedings of the 20th International Conference on Machine Learning*.

Kakade, S. M. 2003. *On the Sample Complexity of Reinforcement Learning*. Ph.D. Dissertation, Gatsby Computational Neuroscience Unit, University College London.

Kearns, M. J., and Koller, D. 1999. Efficient reinforcement learning in factored MDPs. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI)*, 740–747.

Kearns, M. J., and Singh, S. P. 2002. Near-optimal reinforcement learning in polynomial time. *Machine Learning* 49(2–3):209–232.

Koenig, S., and Simmons, R. G. 1993. Complexity analysis of real-time reinforcement learning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 99–105. Menlo Park, CA: AAAI Press/MIT Press.

Leffler, B. R.; Littman, M. L.; Strehl, A. L.; and Walsh, T. 2005. Efficient exploration with latent structure. In *Proceedings of Robotics: Science and Systems*.

Russell, S. J., and Norvig, P. 1994. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: Prentice-Hall.

Sherstov, A. A., and Stone, P. 2005. Improving action selection in MDP's via knowledge transfer. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*.

Strehl, A. L.; Li, L.; and Littman, M. L. 2006. Incremental model-based learners with formal learning-time guarantees. In *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence (UAI 2006)*.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. The MIT Press.

Sutton, R. S. 1988. Learning to predict by the method of temporal differences. *Machine Learning* 3(1):9–44.

Tesauro, G. 1995. Temporal difference learning and TD-Gammon. *Communications of the ACM* 58–67.