An Object-Oriented Representation for Efficient Reinforcement Learning

Carlos Diuk CDIUK@CS.RUTGERS.EDU
Andre Cohen ACOHEN@CS.RUTGERS.EDU
Michael L. Littman MLITTMAN@CS.RUTGERS.EDU
RL^3 Laboratory, Department of Computer Science, Rutgers University, Piscataway, NJ USA

Abstract

Rich representations in reinforcement learning have been studied for the purpose of enabling generalization and making learning feasible in large state spaces. We introduce Object-Oriented MDPs (OO-MDPs), a representation based on objects and their interactions, which is a natural way of modeling environments and offers important generalization opportunities. We introduce a learning algorithm for deterministic OO-MDPs and prove a polynomial bound on its sample complexity. We illustrate the performance gains of our representation and algorithm in the well-known Taxi domain, plus a real-life videogame.

1. Introduction

In the standard Markov Decision Process (MDP) formalization of the *reinforcement-learning* (RL) problem (Sutton & Barto, 1998), a decision maker interacts with an environment consisting of finite state and action spaces. Algorithms for RL in MDP environments suffer from what is known as the *curse of dimensionality*: an exponential explosion in the total number of states as a function of the number of *state variables*. Learning in environments with extremely large state spaces is challenging if not infeasible without some form of generalization. Exploiting the underlying structure of a problem can enable generalization and has long been recognized as important in representing sequential decision tasks (Boutilier et al., 1999).

In this paper, we propose an extension to the standard MDP formalism, which we call *Object-Oriented MDPs (OO-MDPs)*, and present an efficient learning algorithm for deterministic OO-MDPs. We claim that this object-based approach is a natural way of viewing and describing many real-life domains that enables multiple opportunities for generalization. There are many ways of incorporating objects into models for learning and decision making—this paper explores one particular approach as a first attempt to understand the issues that arise.

Our representation has multiple connections with other formalisms proposed in the Relational Reinforcement Learning literature (van Otterlo, 2005), but emphasizes simplicity and tractability over expressive power. Our representation starts from attributes that can be directly perceived by the agent, rather than predicates or propositions introduced by the designer (although we allow the encoding of prior knowledge in propositional form). A similar formalism, *relational MDPs (RMDPs)*, was introduced by Guestrin et al. (2003) in the context of planning, and is based on the same insight. While our formalism has similarities to *RMDPs*, we introduce a number of changes, mainly in the way transition dynamics are described, to enable efficient learning and generalization.

To present and test our approach, we first provide benchmark experiments in the well-known Taxi domain (Dietterich, 2000). We further demonstrate its applicability by designing an agent that can solve an interesting problem in the real-life videogame *Pitfall*[1].

2. Notation

We use a standard Markov Decision Process (MDP) notation throughout this paper (Puterman, 1994). A finite MDP M is a five tuple $\langle \mathcal{S}, \mathsf{A}, T, \mathcal{R}, \gamma \rangle$. We use $T(s'|s,a)$ to denote the transition probability of state s' given state–action pair (s,a) and $R(s,a)$ to denote the expected reward value. A *deterministic MDP* is one in which there is a single next state s' for every given state s and action a; that is, $\forall s \in \mathcal{S}, a \in \mathsf{A}, \exists s' \in \mathcal{S} : T(s'|s,a) = 1$.

3. Object-oriented Representation

We will use the Taxi domain, defined by Dietterich (2000), as an example to introduce our formalism. Taxi is a grid-

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008. Copyright 2008 by the author(s)/owner(s).

[1] ©1982 Activision, Inc.

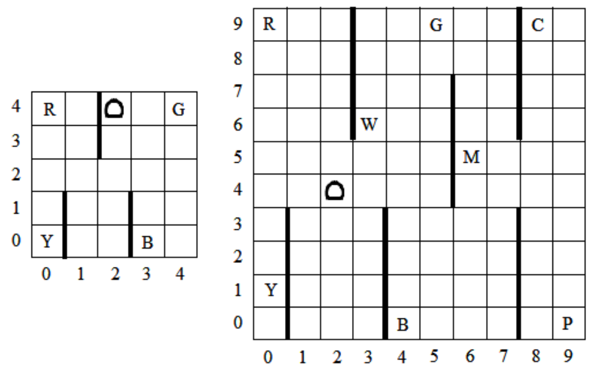

Figure 1. The taxi domain. (a) Original 5×5 Taxi problem. (b) Extended 10×10 version, with a different wall distribution and 8 possible passenger locations and destinations.

world domain (see Figure 1.a), where a *taxi* has the task of picking up a *passenger* in one of a pre-designated set of locations (identified in the figure by the letters *Y*, *G*, *R*, *B*) and dropping it off at a goal *destination*, also one of the pre-designed locations. The set of actions for the taxi are *North*, *South*, *East*, *West*, *PICKUP* and *DROPOFF*. Walls in the grid limit the taxi's movements.

A common factored-state representation for the Taxi problem uses Dynamic Bayesian Networks (DBNs) to indicate how state variables influence each other. For example, the location of the *taxi* after a *North* action only depends on its current location and is independent of the *passenger* or *destination* variables.

We depart from this representation and introduce one based on objects and their interactions. Many elements in our representation are similar to those of *relational MDPs* (Guestrin et al., 2003) with significant differences in the way we represent transition dynamics. Similar to *RMDPs*, we define a set of *classes* $\mathcal{C} = \{C_1, \ldots, C_c\}$. Each class includes a set of *attributes* $Att(C) = \{C.a_1, \ldots, C.a_a\}$, and each attribute has a *domain* $Dom(C.a)$. A particular environment will consist of a set of *objects* $O = \{o_1, \ldots, o_o\}$, where each object is an instance of one class: $o \in C_i$. The state of an object $o.state$ is a value assignment to all its attributes. The state of the underlying MDP is the union of the states of all its objects: $s = \bigcup_{i=1}^{o} o_i.state$.

An OO-MDP representation of Taxi has four object classes: *Taxi*, *Passenger*, *Destination* and *Wall*. *Taxi*, *Passenger* and *Destination* have attributes x and y, which define their location in the grid. *Passenger* also has a Boolean attribute *in-taxi*, which specifies whether the passenger is inside the taxi. *Walls* have an attribute that indicates their position in the grid. The Taxi domain, in its 5×5 version shown in Figure 1.a, has one object of each class *Taxi*, *Passenger*, and *Destination*, and multiple (26) objects of class *Wall*. This list of objects points out a significant feature of the OO-MDP representation. Whereas in the classical MDP model, the effect of encountering walls is felt as a property of specific locations in the grid, the OO-MDP view is that wall interactions are the same regardless of their location. As such, agents' experience can transfer gracefully throughout the state space.

When two objects interact in some way, they define a *relation* between them. A combination of the relation established, plus the internal states of the two objects, determines an *effect*—a change in value of one or multiple attributes in either or both interacting objects. This behavior is defined at the class level, meaning that different objects that are instances of the same class behave in the same way when interacting with other objects. Formally, a relation $r : C_i \times C_j \rightarrow \textit{Boolean}$ is a function, defined at the class level, over the combined attributes of objects of classes C_i and C_j. Its value gets defined when instantiated by two objects $o_1 \in C_i$ and $o_2 \in C_j$. For our Taxi representation, we will define 5 relations: $\textit{touch}_N(o_1, o_2)$, $\textit{touch}_S(o_1, o_2)$, $\textit{touch}_E(o_1, o_2)$, $\textit{touch}_W(o_1, o_2)$ and $\textit{on}(o_1, o_2)$, which define whether an object $o_2 \in C_j$ is exactly one cell North, South, East or West of an object $o_1 \in C_i$, or if both objects are overlapping (same x, y coordinates). Different domains require different relations.

When the object $\textit{taxi}_i \in \textit{Taxi}$ is on the northern edge of the grid and tries to perform a *North* action, it hits some object $\textit{wall}_j \in \textit{Wall}$ and the observed behavior is that it doesn't move. We say that a $\textit{touch}_N(\textit{taxi}_i, \textit{wall}_j)$ relation has been established and the effect of an action *North* under that condition is *no-change*. On the other hand, if $\neg\textit{touch}_N(\textit{taxi}_i, \textit{wall}_j)$ is true and the taxi performs the action *North*, the effect will be $\textit{taxi}_i.y \leftarrow \textit{taxi}_i.y + 1$. As stated before, these behaviors are defined at the class level, so we can refer in general to the relation $\textit{touch}_N(\textit{Taxi}, \textit{Wall})$ as producing the same kind of effects on any instance of $\textit{taxi}_i \in \textit{Taxi}$ and $\textit{wall}_j \in \textit{Wall}$.

We define some properties of these transition dynamics more formally in the next section.

3.1. Transition Dynamics

Every state s induces a certain value assignment to all attributes of all objects—and therefore all relations—in the domain. Transitions are determined by interactions between objects. Every pair of objects $o_1 \in C_i$ and $o_2 \in C_j$, their internal states $o_1.\textit{state}$ and $o_2.\textit{state}$, an action a, and the set of relations $r(o_1, o_2)$ that are true—or false—at the current state, determine an effect—a change of value in some of the objects' attributes.

Definition 1 *An* effect *is a single operation over a single attribute att in the OO-MDP. We will group effects into* types, *based on the kind of operation they perform. Ex-*

amples of types are arithmetic *(increment att by* 1*, subtract* 2 *from att), and* constant assignment *(set att to* 0*).*

Definition 2 *A* term t *is any Boolean function. In our OO-MDP representation, we will consider terms representing either a relation between two objects, a certain possible value of an attribute of any of the objects or, more generally, any Boolean function defined over the state space that encodes prior knowledge. All transition dynamics in an OO-MDP are determined by the different possible settings of a special set of terms called* T*.*

Definition 3 *A* condition *is a set* T_c *of terms and negations of terms from* T *that* must *be true in order to produce a particular effect* e *under a given action* a*.*

We can summarize an OO-MDP transition cycle as follows:

```
1: while agent is acting do
2:     Agent observes current state $s$ and returns action $a$.
3:     From state $s$, the environment extracts all relations that currently hold between objects and observes the value of all attributes of all objects, assigning a True/False value to all terms in $T$.
4:     For each (if any) fulfilled condition in $T_c$, there's an effect that will occur, determining a set of effects to be applied to $s$.
5:     If no conditions were fulfilled, no change takes place to $s$.
6:     Otherwise, the environment uses the set of effects to compute $s'$. New state $s \leftarrow s'$.
7:     The environment chooses a reward $r$ from $R(s, a)$.
8:     Agent is told $r$.
9: end while
```

4. *DOORMAX*: Learning and Solving Deterministic OO-MDPs

We introduce Deterministic Object-Oriented Rmax (*DOORMAX*), an algorithm for learning and solving deterministic OO-MDPs. *DOORMAX* is correct and, as we will show, provably efficient under the following assumptions.

Assumption 1 *For each action and each attribute, only effects of one type can occur.*

Assumption 2 *For every action* a*, attribute att and effect type* t*, there is a set* $\mathcal{CE}$ *of* condition–effect *pairs that determine changes to att given* a*. No effect can appear twice on this list, and there are at most* k *different pairs—*$|\mathcal{CE}| \leq k$*. Plus, no conditions* T_i *and* T_j *in the set* $\mathcal{CE}$ *contain each other:* $\neg(T_i \subset T_j \vee T_j \subset T_i)$*. The number of terms or negations of terms in any condition is bounded by a known constant* M*.*

Assumption 3 *Effects are* invertible*, that is, given states* s *and* s'*, for each attribute att and each effect type we can determine a unique effect that would transform att from its value in* s *to its value in* s'*.*

4.1. Definitions and Data Structures

We introduce some definitions, notation, and data structures that will be used to describe *DOORMAX*:

- T is the union of all terms t that will be involved in the conditions that determine the transition dynamics of the environment described by the OO-MDP, plus their negations $\neg t$, with $|T| = 2n$.
- For every state $s \in S$, the function *cond*(s) returns the subset of terms in T that are true in s.
- A condition $T_c \subseteq T$ is represented by a string c_S of length n, where $c_S^i = 1$ if $t_i \in T_c$, $c_S^i = 0$ if $\neg t_i \in T_c$ and $c_S^i = *$ if $t_i \notin T_c \wedge \neg t_i \notin T_c$.
- Given two conditions represented as strings c_1 and c_2, we define the commutative operator $\oplus : c \times c \rightarrow c$ as follows:

c_1	c_2	$c_1 \oplus c_2$
0	0	0
1	1	1
0	1	*
0|1	*	*

- A condition c_1 *matches* another condition c_2, noted $c_1 \models c_2$, if $\forall 1 \leq i \leq n : c_1^i = * \vee c_1^i = c_2^i$.
- For any states s and s' and attribute *att*, the function *eff*$_{att}(s, s')$ returns one effect of each type that would transform attribute *att* in s into its value in s'.
- A *prediction* p is a pair $(p.\textit{model}, p.\text{effect})$, where $p.\textit{model}$ is a condition that represents the set of terms that need to be true for p.effect to occur.
- For each action a, each attribute *att* and each effect type *type*, a set of predictions *pred*$(a, \textit{att}, \textit{type})$ is maintained. We refer to the set of *models* in a set of predictions as *pred*$(a, \textit{att}, \textit{type}).\textit{models}$.
- If an action a produces no effect from a given state s ($s' = s$), we call the induced condition *cond*(s) a *failure condition*. We define F_a to be a set of failure conditions for action a.
- Two effects are *incompatible* if, for any initial value of an attribute, applying these two effects would yield two different final values.

4.2. OO-MDP Representation of Taxi

To facilitate understanding of the notation and data structures, we present a full example of our representation in the Taxi domain.

The set of terms T, which determines the transition dynamics of the OO-MDP, includes the four $\textit{touch}_{N/S/E/W}$ relations between the taxi and the walls; the relevant relations between the taxi and the passenger and destination; the attribute value $\textit{passenger.in-taxi} = T$; and all their negations:

$$\begin{array}{lll} \{ & \textit{touch}_{N/S/E/W}(\textit{taxi}, \textit{wall}), & \textit{on}(\textit{taxi}, \textit{passenger}), \\ & \neg\textit{touch}_{N/S/E/W}(\textit{taxi}, \textit{wall}), & \neg\textit{on}(\textit{taxi}, \textit{passenger}), \\ & \textit{on}(\textit{taxi}, \textit{destination}), & \neg\textit{on}(\textit{taxi}, \textit{destination}), \\ & \textit{passenger.in-taxi} = T, & \textit{passenger.in-taxi} = F \ \} \end{array}$$

Consider the state s where the taxi is in position $(2, 4)$ (as in Figure 1.a), the passenger is inside the taxi, and the destination is G. For this state, the function $\textit{cond}(s)$ returns:

$$\begin{array}{lll} \{ & \textit{touch}_N(\textit{taxi}, \textit{wall}), & \neg\textit{touch}_S(\textit{taxi}, \textit{wall}), \\ & \neg\textit{touch}_E(\textit{taxi}, \textit{wall}), & \textit{touch}_W(\textit{taxi}, \textit{wall}), \\ & \neg\textit{on}(\textit{taxi}, \textit{passenger}), & \neg\textit{on}(\textit{taxi}, \textit{destination}), \\ & \textit{passenger.in-taxi} = T \ \}. & \end{array}$$

The corresponding 7-character string representation for this condition is 1001001, following the prior order for the terms.

Let's now assume that the agent tries to perform the action *East*, which takes it to state s' where the taxi is in location $(3, 4)$. The corresponding $\textit{cond}(s')$ is similar, except that now the taxi is not touching a wall to its West ($\neg\textit{touch}_W(\textit{taxi}, \textit{wall})$). The corresponding string representation of the new condition is: 1000001. The observed effect is that the taxi moved to location $(3, 4)$. In our representation, two effect types are allowed: arithmetic and constant assignment. Therefore, the function $\textit{eff}_{\textit{taxi}.x}(s, s')$ will return two values: increment(1) and set-to(3).

Now, the agent takes another *East* action, and gets to state s'', where location is $(4, 4)$, it's touching a wall to the East and standing on the destination. $\textit{cond}(s'')$ can now be represented as 1010011. The two observed effects to $\textit{taxi}.x$ are increment(1) and set-to(4). Note that the transition model for an OO-MDP need not predict the changes to the conditions, only to the attributes. The condition values are then derived separately using the knowledge of the relevant relations and their definitions.

Finally, we'll consider separately the actions that produce no effect. Let's assume the agent also attempted an action *North* from each of the previous states, which resulted in it hitting a wall and staying in the same state. We treat these cases differently: The corresponding conditions 1001001, 1000001 and 1010011 will be identified as failure conditions for action *North* and incorporated into the set $F_{\textit{North}}$. Whenever we observe a new condition c_i such that any existing condition in $F_{\textit{North}}$ matches it, we predict that performing a *North* action will have no effect.

4.3. Learning Algorithm

The *DOORMAX* algorithm (Algorithm 1) follows the general structure of most RL algorithms in the Rmax family, which work as follows. Using examples of transitions (s, a, s'), a learning algorithm constructs the transition model T. The learning algorithm must satisfy the KWIK (knows what it knows) conditions (Li et al., 2008), which say: (1) all predictions must be accurate (assuming a valid hypothesis class), and (2) however, the learning algorithm may also return $\perp$, which indicates that it cannot yet predict the output for this input. The sample complexity or KWIK bound of a learning algorithm is the maximum number of times it returns $\perp$. In the Rmax setting, any transition that cannot yet be predicted is assumed to lead to a fictious $s_{\max}$ state from which maximum reward can be obtained.

Algorithm 1 *DOORMAX*: `main()` method

```
1: // Set up data structures:
2: for all actions $a \in A$ do
3:     $F_a \leftarrow \emptyset$
4:     for all attributes $\textit{att} \in \bigcup_{c \in C} \textit{Att}(c)$ do
5:         for all effect types $\textit{type}$ do
6:             $\textit{pred}(a, \textit{att}, \textit{type}) \leftarrow \emptyset$
7:             Add $\textit{pred}(a, \textit{att}, \textit{type})$ to set of active predictions $\mathcal{P}$
8:         end for
9:     end for
10: end for
11: while $\neg$(Termination criterion) do
12:     Observe current state $s$.
13:     Choose action $a$ according to exploration policy, based on prediction for $T(s'|s, a)$ returned by predictTransition($s, a$).
14:     Observe new state $s'$.
15:     Update learned model using method addExperience($s, a, s', k$).
16: end while
```

The two main routines of the algorithm are `predictTransition` (Algorithm 2), which predicts the next state given a current state and action based on the current model, and `addExperience` (Algorithm 3), which learns a model of the OO-MDP. If `predictTransition` is not able to predict a next state with accuracy, it returns $s_{\max}$.

To help understand these routines, we present a couple of intuitions, based on the Taxi examples presented in the previous section. Notice that if we applied the $\oplus$ operator to *cond*(s) and *cond*(s'), the two conditions from which an *East* action produced an increment(1) effect, we would obtain: $1001001 \oplus 1000001 = 100$*001. The resulting condition indicates that the term *touch*$_W$(*wall*, *taxi*) is irrelevant with respect to action *East* and effect increment(1). If we also compare the two pairs of effects obtained, we observe that we consistently observed increment(1), whereas set-to(3) and set-to(4) are incompatible effects. These observations constitute the central ideas for the learning algorithm.

Algorithm 2 `predictTransition(s,a)` method

```
0: Inputs: state $s$ and action $a$.
0: Output: a predicted state $s' \in S \cup \{s_{\max}\}$.
1: if $\exists c \in F_a$ s.t. $c \models cond(s)$ then
2:    // The current condition is a known failure condition.
3:    Return $s$
4: else
5:    for all attributes $att \in \bigcup_{c \in C} Att(c)$ do
6:       $E \leftarrow \emptyset$
7:       for all effect types $type$ do
8:          if $\exists p \in pred(a, att, type)$ s.t. $p.model \models cond(s)_S$ then
9:             Add $p$.effect to $E$
10:         end if
11:      end for
12:      if $E = \emptyset \vee \exists e_i, e_j \in E$ s.t. $e_i$ and $e_j$ are incompatible then
13:         Return $s_{\max}$
14:      else
15:         // Set $E$ contains all the individual operations that need to be applied to attributes in $s$ in order to convert it to $s'$.
16:         $s' \leftarrow$ apply $E$ to $s$
17:         Return $s'$
18:      end if
19:   end for
20: end if
```

5. Analysis

Under the current assumptions, the effects of a given action on a given attribute assuming effects of a given type can be learned with a worst-case bound of $O(n^M)$, where $n = |T|$ is the number of terms and M is the maximum number of terms involved in any of the conditions. This worst-case bound can be guaranteed by a variant of *SLF-Rmax*, an algorithm introduced by Strehl et al. (2007).

The uniqness assumption, Assumption 2, is not needed for *SLF-Rmax* to achieve this worst-case bound. However, *DOORMAX*, by taking advantage of this assumption, is able to learn faster in many domains. Some empirical evidence to support this claim appears in Section 6.

If we assume M is a constant, SLF-Rmax can be used to provide guaranteed efficient results. However, for many domains DOORMAX will result much more efficient in practice. We conjecture that the two approaches can be run in parallel, to achieve the best of both.

Intuitively, the good empirical results of DOORMAX lie in the way condition-effects are learned each time they are observed. The worst-case occurs when the agent observes an exponential amount of failures before observing instances of the set of effects it needs to learn.

We now show that the problem of learning the transition dynamics of an OO-MDP has polynomial sample complexity in the KWIK setting, when by sample we only refer to the cases where an effect is observed (as opposed to failure samples where $s' = s$).

We split the proof in two parts. First, we show that learning the right (condition, effect) pairs for a single action and attribute is KWIK-learnable, and then we show that learning the right effect type for each action–attribute, given all the possible effect types, is also KWIK learnable.

Theorem 1 *The transition model for a given action a, attribute att and effect type type in a deterministic OO-MDP is KWIK-learnable with a bound of $O(nk+k+1)$, where n is the number of terms in a condition and k is the maximum number of effects per action–attribute.*

Proof:

Given state s and action a, the predictor for effect type *type* will return $\perp$ if *cond*(s) is not a known failure condition and there is no condition in *pred*$(a, att, type)$ that matches *cond*(s). In that case, it gets to observe s' and updates its model with *cond*(s) and the observed effect e. We show that the number of times the model can be updated until it always has a correct prediction is $O(nk + k + 1)$:

- if the effect e has never been observed before for this particular action, attribute and effect type, it gets added to *pred*$(a, att, type)$. This outcome happens at most k times, which is the maximum number of different effects allowed per action-attribute-type combination.

- if the effect e has never been observed, but $|pred(a, att, type)| = k$, the algorithm concludes that the current effect type is not the correct one for this action–attribute, and it removes all predictions of this type from its set $\mathcal{P}$. This event can only happen once.

- if the effect e is such that there already exists a prediction for it, $\perp$ is only returned if the existing condition in the model does not match *cond*(s). This case can only happen if a term in the model is a 0 or 1 and the observation is the opposite. Once it happens, that term becomes a *, so there will never be another mismatch for that term, as * matches either 0 or 1. In the worst case, with every $\perp$ returned, one term at a time gets converted into *. These updates can only happen n times for each effect in *pred*$(a, \mathit{att}, \mathit{type})$, for a total of nk times.

Therefore, there can be at most $nk + k + 1$ updates to the model for a particular action a, attribute *att* and effect type *type* before *pred*$(a, \mathit{att}, \mathit{type})$ either has a correct prediction or gets eliminated. □

Corollary 1 *The transition model for a given action and attribute in a deterministic OO-MDPs is KWIK-learnable with a bound of $O(h(nk + k + 1) + (h - 1))$, where n is the number of terms in a condition, k is the max number of effects per action–attribute, and h is the number of effect types.*

Proof: Whenever *DOORMAX* needs to predict s' given state s and action a, it will consult its current predictions for each attribute and effect type. It will return $\perp$ if:

- for any of the h effect types type_i, *pred*$(a, \mathit{att}, \mathit{type}_i)$ returns $\perp$. As shown in Theorem 1, *pred*$(a, \mathit{att}, \mathit{type}_i)$ can only return $\perp$ up to $nk + k + 1$ times. Therefore, this case can only happen $h(nk + k + 1)$ times.
- for some attribute *att*, there are two effect types type_1 and type_2 such that *pred*$(a, \mathit{att}, \mathit{type}_1) \neq$ *pred*$(a, \mathit{att}, \mathit{type}_2)$. When this happens, we get to observe the actual effect e, which will necessarily mismatch one of the predictions. The model will therefore be updated by removing either *pred*$(a, \mathit{att}, \mathit{type}_1)$ or *pred*$(a, \mathit{att}, \mathit{type}_2)$ from its set of predictions. This case can only occur $h - 1$ times for a given action and attribute.

We have shown that, in total, *DOORMAX* will only predict $\perp$ $O(h(nk + k + 1) + (h - 1))$ times before having an accurate model of the transition dynamics for an action and attribute in the OO-MDP. □

6. Experiments

First, we use the Taxi domain to demonstrate how *DOORMAX* makes use of the OO-MDP representation to outperform *Factored-Rmax*, an algorithm based on a factored-state MDP representation. Second, we show how *DOORMAX* and *Factored-Rmax* scale when the size of the state space increases, by comparing them on the 10×10 version of Taxi. Finally, we demonstrate how *DOORMAX* can be

Algorithm 3 `addExperience(s,a,s',k)` method

```
0:  Inputs: an observation $< s, a, s' >$; $k$, the maximum number of different effects possible for any action, attribute and effect type.
1:  if $s = s'$ then
2:    // Found a failure condition for action $a$, update $F_a$
3:    Remove all $c \in F_a$ s.t. $cond(s) \models c$.
4:    $F_a \leftarrow F_a \cup \{cond(s)\}$
5:  else
6:    for all attributes $att \in \bigcup_{c \in C} Att(c)$ do
7:      for all $e \in eff_{att}(s, s')$ do
8:        Find a prediction $p \in pred(a, att, e.type)$ such that $p$.effect $= e$
9:        if $\exists p$ then
10:         // We already have a (condition, effect) prediction for current $a$, att, and type. Update condition and verify that there are no overlaps.
11:         $p.model \leftarrow p.model \oplus cond(s)_S$.
12:         if $\exists c \in (pred(a, att, e.type) \setminus p).models$ s.t. $p.model \models c$ then
13:           // Conditions overlap, violating an assumption, meaning it is not the right type of effect for this action and attribute.
14:           Remove $pred(a, att, e.type)$ from $\mathcal{P}$
15:         end if
16:       else
17:         // We observed an effect for which we had no prediction. If its condition does not overlap an existing condition, then add this new prediction.
18:         if $\exists c \in pred(a, att, e.type).models$ s.t. $cond(s) \models c \vee c \models cond(s)$ then
19:           Remove $pred(a, att, e.type)$ from $\mathcal{P}$
20:         else
21:           Add $(cond(s), e)$ to $pred(a, att, e.type)$.
22:           // Verify that there aren't more than $k$ predictions for this action, attribute and type.
23:           if $|pred(a, att, e.type)| > k$ then
24:             Remove $pred(a, att, e.type)$ from $\mathcal{P}$
25:           end if
26:         end if
27:       end if
28:     end for
29:   end for
30: end if
```

applied to succesfuly model and solve a real-life problem, the *Pitfall* videogame.

6.1. Taxi

The first experiments we present are based on the Taxi domain previously introduced. We run experiments on two versions: the original 5×5-grid version presented by Dietterich (2000), which consists of 500 states, and an extended 10×10-grid version with 8 passenger locations and destinations, with 7200 states (see Figure 1). The purpose of the extended version is to demonstrate how *DOORMAX* scales by properly generalizing its knowledge about conditions and effects when more objects of the same known classes are introduced.

We compare *DOORMAX* against *Factored-Rmax*, an algorithm from the Rmax family that uses a factored-state MDP and models transitions using a DBN provided as input. Both algorithms are model based and use Rmax-style exploration, so we hope to be able to truly compare the underlying representations.

The representation used for *DOORMAX* was described in the previous sections. In the case of *Factored-Rmax*, we provide a DBN with some derived features that make learning faster. The state variables used are the *Taxi* x and y locations, plus two Boolean features: *in-taxi*, representing whether the *passenger* is in the *taxi*, and *at-destination*, representing whether the *taxi* is standing at the *passenger*'s *destination*.

The experiments for both algorithms and both versions of the Taxi problem were repeated 100 times, and the results averaged. For each experiment, we run a series of episodes, each starting from a random start state. We evaluate the agent's learned policy after each episode on a set of six "probe" combinations of ⟨taxi (x,y) location, passenger location, passenger destination⟩. The probe states used were: $\{(2,2),Y,R\}$, $\{(2,2),Y,G\}$, $\{(2,2),Y,B\}$, $\{(2,2),R,B\}$, $\{(0,4),Y,R\}$, $\{(0,3),B,G\}$. We report the number of steps taken before learning an optimal policy for these six start states.

The results are shown in the following table, with the last column showing the ratio between the results for the 10×10 version vs the 5×5 one:

	Taxi 5×5	Taxi 10×10	Ratio
Number of states	500	7200	14.40
Factored Rmax			
# steps	1676	19866	11.85
Time per step	43.59ms	306.71ms	7.03
OO-Rmax			
# steps	529	821	**1.55**
Time per step	13.88ms	293.72ms	21.16

We can see how *DOORMAX* not only learns with significantly less sample complexity, but also how well it scales to the larger problem. After increasing the number of states by more than 14 times, *DOORMAX* only requires 1.55 times the experience.

The main difference between *DOORMAX* and *Factored-Rmax* is their internal representation, and the kind of generalization it enables. After just a few examples in which $\neg touch_N(taxi, wall)$ is true, *DOORMAX* learns that the action *North* has the effect of incrementing $taxi.y$ by 1, whereas under $touch_N(taxi, wall)$ it fails. This knowledge, as well as its equivalent for $touch_{S/E/W}$, is generalized to all 25 (or 100) different locations. *Factored-Rmax* only knows that variable $taxi.y'$ in state s' depends on its value in state s, but still needs to learn the transition dynamics for each possible value of $taxi.y$ (5 or 10 different values). In the case of actions *East* and *West*, it's even worse, as walls make $taxi.x'$ depend on both $taxi.x$ and $taxi.y$, which are 25 (or 100) different values.

As *DOORMAX* is based on interactions between objects, it learns that the relation between taxi and wall is independent of the wall location. Each new wall is therefore the same as any known wall, rather than a new exception in the movement rules, the kind *Factored-Rmax* needs to learn.

6.2. Pitfall

Pitfall is a video game released in 1982 by Activision for the Atari game console. The goal is to have the main character (*Man*) traverse a series of screens while collecting as many points as possible while avoiding obstacles (such as holes, pits, logs, crocodiles and walls) and under the time constraint of 20 minutes. All transitions in Pitfall are deterministic. Our goal in this experiment was to have the *Man* cross the first screen from the left to the right with as few actions as possible. Figure 2 ilustrates this first screen.

Our experiments were run using a modified Atari emulator that ran the actual game and detected objects from the displayed image. We used a simple heuristic that identified objects by color clusters and sent joystick commands to the emulator to influence the play. For each frame of the game, a list of object locations was sent to an external learning module that analyzed the state of the game and returned an action to be executed before the emulator continued on to the next frame. If we consider that we start from screen pixels, the flat state representation for Pitfall is enormous: $16^{640x420}$. By breaking it down into basic objects, through an object recognition mechanism, the state space is in the order of the number of objects to the number of possible locations of each object: $6^{640x420}$. OO-MDPs allow for a very succint representation of the problem, that can be learned with only a few experience samples.

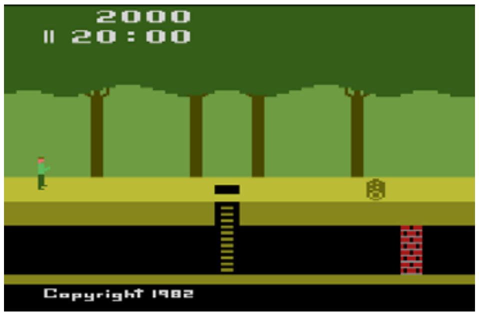

Figure 2. Initial screen of Pitfall.

The first screen contains six object types: *Man, Hole, Ladder, Log, Wall* and *Tree*. Objects have the attributes x, y, $width$ and $height$, which define their location on the screen and dimension. The *Man* also has a Boolean attribute of $direction$ that specifies which way he is facing. We extended the $touch_X$ relation from Taxi to describe diagonal relations between objects, including: $touch_{NE}(o_i, o_j)$, $touch_{NW}(o_i, o_j)$, $touch_{SW}(o_i, o_j)$ and $touch_{SE}(o_i, o_j)$. These relations were needed to properly capture the effects of moving on and off of ladders.

In our implementation of *DOORMAX*, we defined seven actions: *WalkRight, WalkLeft, JumpLeft, JumpRight, Up, Down* and *JumpUp*. For each of these actions, however, the emulator has to actually execute a set sequence of smaller frame-specific actions. For example, *WalkLeft* requires four frames: one to tell Pitfall to move the *Man* to the left, and three frames where no action is taken to allow for the animation of the *Man* to complete. Effects are represented as arithmetic increments or decrements to the attributes x, y, $width$, $height$, plus a constant assignment of either R or L to the attribute $direction$.

The starting state of Pitfall is fixed, and given that all transitions are deterministic, only one run of *DOORMAX* was necessary to learn the dynamics of the environment. *DOORMAX* learns an optimal policy after 494 actions, or 4810 game frames, exploring the area beneath the ground as well as the objects en route to the goal. Once the transition dynamics are learned, restarting the game results in the *Man* exiting the first screen through the right, after jumping the hole and the log, in 94 actions (905 real game frames), which is what the optimal policy requires.

A few examples of the (condition, effect) pairs learned by *DOORMAX* are shown below:

Action	Condition	Effects
WalkRight	direction $= L$	{direction $= R$, $\Delta x = +8$}
WalkRight	$touch_E(Man, Wall)$	no-effect
JumpRight	direction $= R$	$\Delta x = +214$
Up	$on(Man, Ladder)$	$\Delta y = +8$

7. Conclusions and Future Work

We introduced OO-MDPs, an object-oriented representation for reinforcement-learning problems that provides a natural way of modeling a broad set of domains, while enabling generalization. We presented *DOORMAX*, a learning algorithm for deterministic OO-MDPs that not only outperforms state-of-the-art algorithms for factored-state representations, but also scales very nicely with respect to the size of the state space, as long as transition dynamics between objects do not change. We presented bounds for learning transition dynamics of determinstic OO-MDPs in the KWIK framework.

One limitation of our work is that we do not yet have a provably efficient algorithm for stochastic domains, which is part of our future work. While OO-MDPs can model stochastic transitions, a more complex learning algorithm would be needed to learn transitions effectively in the face of noise.

The second component of our future research is the extension of the object-oriented model to be able to handle inheritance. We hope to be able to exploit knowledge about objects being part of a common super-class to learn their behaviors faster. Ideally, algorithms could also learn the object definitions and classes automatically, as well.

References

Boutilier, C., Dean, T., & Hanks, S. (1999). Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, *11*, 1–94.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, *13*, 227–303.

Guestrin, C., Koller, D., Gearhart, C., & Kanodia, N. (2003). Generalizing plans to new environments in relational mdps. *IJCAI* (pp. 1003–1010).

Li, L., Littman, M. L., & Walsh, T. J. (2008). Knows what it knows: A framework for self-aware learning. *Twenty-Fifth International Conference on Machine Learning*.

Puterman, M. L. (1994). *Markov decision processes—discrete stochastic dynamic programming*. New York, NY: John Wiley & Sons, Inc.

Strehl, A. L., Diuk, C., & Littman, M. L. (2007). Efficient structure learning in factored-state mdps. *AAAI* (pp. 645–650). AAAI Press.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. The MIT Press.

van Otterlo, M. (2005). *A survey of reinforcement learning in relational domains* (Technical Report TR-CTIT-05-31). CTIT Technical Report Series, ISSN 1381-3625.