Learning Propositional Functions for Planning and Reinforcement Learning

D Ellis Hershkowitz and **James MacGlashan** and **Stefanie Tellex**
Brown University, Computer Science Dept.
115 Waterman Street, 4th floor
Providence, RI 02912-1910

Abstract

Massive state spaces are ubiquitous throughout planning and reinforcement learning (RL) domains: agents involved in furniture assembly, cooking automation and backgammon must grapple with problem formalisms that are much too expansive to solve by conventional tabular approaches. Modern tabular planning and RL techniques bypass this difficulty by using propositional functions to transfer knowledge across states – both within and across problem instances – to solve for near optimal behaviors in very large state spaces. We present a means by which useful propositional functions can be inferred from observations of transition dynamics. Our approach is based upon distilling salient relational values between pairs of objects. We then use these learned propositional functions to free the RL algorithm deterministic object-oriented RMAX (DOORMAX) of its dependence on expert-provided propositional functions. We also empirically demonstrate high correspondence between these learned propositional functions and expert-provided propositional functions. Our novel DOORMAX algorithm performs at a level near that of classic DOORMAX.

1 Introduction

Planning and reinforcement learning (RL) problems are riddled with large stochastic state spaces (Knepper et al. 2013; Bollini et al. 2013; Abel et al. 2015). For instance, the state space of a robotic pick-and-place task increases exponentially with the number of objects to be manipulated. Thus, even a task with few objects is characterized by an extremely large state space that is rendered stochastic by noisy robotic control. Similarly, a robot engaged in a cooking task can configure the layout of its ingredients in numerous ways with some failure probability, and so it too must grapple with a large stochastic state space (Abel et al. 2015; Bollini et al. 2013). There is, then, a need for methods that expediently solve planning problems with large stochastic state spaces.

Traditional tabular planning and RL methods – those that compute over every state in the state space – are ill-equipped to deal with such state spaces; large stochastic state spaces are often mired by irrelevant subspaces into which tabular planners needlessly sink computation. Often so much computation is wasted that the problem becomes prohibitively difficult for tabular planners (Abel et al. 2015).

More recent planning and RL approaches such as deterministic object-oriented RMAX (DOORMAX) (Diuk, Cohen, and Littman 2008), linear function approximation methods for value functions (Geramifard et al. 2013b) and goal-based action priors (Abel et al. 2015) perform state-to-state knowledge transfer by exploiting propositional functions. These methods leverage propositional functions to quantify the similarity of states and then transfer problem knowledge as appropriate. There also exist methods by which initial propositional functions are compounded into new propositional functions (Geramifard et al. 2013a). Thus, propositional functions are of critical importance in a number of planning and RL techniques for large state spaces.

To address the need for useful propositional functions in planning and RL problems, we derive a means of inferring propositional functions in a planning or RL setting. These propositional functions are derived by formulating predictions of transition dynamics in an object-oriented Markov Decision process (OO-MDP). Our predictions are based upon minimal relations between objects in the OO-MDP.

We begin with an explanation of OO-MDPs and the form of our predictions in Section 2. Section 3 details our algorithm, which consists of a relational featurization of an OO-MDP, creation of labeled datasets based on observations of transition dynamics, and conversion of these labeled datasets into propositional functions. We also leverage our methods to derive a novel version of DOORMAX that requires no propositional functions in Section 3.4. Section 4 details our experimental results in which we confirm alignment between learned propositional functions and those propositional functions normally specified by an expert. Also in Section 4, we demonstrate that our novel version of DOORMAX performs nearly as well as classic DOORMAX in terms of reward maximization over time but with substantially less expert knowledge required.

2 Background

We begin by explaining OO-MDPs, as well as vocabulary and OO-MDP-related data structures, that we use throughout this work. Note that much of our vernacular is a continuation of that used by Diuk, Cohen, and Littman.

2.1 OO-MDPs

Planning and RL problems are typically formalized as Markov Decision Processes (MDPs). An MDP is a five-tuple: $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$: $\mathcal{S}$ is a state space; $\mathcal{A}$ is an action set; $\mathcal{T}$ denotes $\mathcal{T}(s' \mid s, a)$, the probability of an agent applying action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$ and arriving in $s' \in \mathcal{S}$; $\mathcal{R}(s)$ denotes the reward received by the agent arriving in state s; $\gamma \in [0, 1]$ is a discount factor that defines how much the agent prefers immediate rewards over future rewards.

One of the notable shortcomings of tabular representations of MDPs is the lack of opportunities for knowledge transfer; every state is related to other states by the machinery of the MDP only in so far as one state might transition to the other given some action or series of actions. Consequently, the classic MDP cannot accommodate transfer of knowledge across similar states. Unlike the MDP, the OO-MDP (Diuk, Cohen, and Littman 2008) allows transfer of knowledge across similar states. It does so by exploiting the object-oriented structure of many planning and RL problems to create propositional functions.

Formally, an OO-MDP is identical to an MDP except in the added mechanisms it provides for state representation and propositional functions.

An OO-MDP defines a set of c *object classes*, $\mathcal{C} = \{C_1, \ldots, C_c\}$. Each $C_i \in C$ has a set of *attributes*, $Att(C_i) = \{C_i.a_1, \ldots, C_i.a_a\}$. Each attribute a_j for each object class C_i has some *domain*, $Dom_{C_i}(a_j)$, that defines the values that a_j can adopt. A single state in an OO-MDP consists of o instantiations of object classes, $\mathcal{O} = \{o_1 \ldots, o_o\}$, wherein each instantiation is an assignment to the attributes of the instantiated object class; that is, the OO-MDP state $s = \cup_{i=1}^{o} o_i$.

This state refactorization, in turn, allows one to define a space of highly generalized *propositional functions*, $\mathcal{P}$, that act on collections of objects and therefore OO-MDP states. That is, $p \in \mathcal{P} : s \in \mathcal{S} \rightarrow \{\text{true}, \text{false}\}$. These propositional functions are added to the OO-MDP and provide planners with a means of identifying similar states and transferring knowledge accordingly. As in an MDP, the goal of an agent in an OO-MDP is maximization of discounted reward.

Thus an OO-MDP is a seven-tuple: $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma, \mathcal{O}, \mathcal{P} \rangle$ where $s \in \mathcal{S}$ is notable for its object-oriented representation using $\mathcal{O}$, and $\mathcal{P}$ defines propositional functions, which provide planners with a means of relating similar states.

For example, the OO-MDP formalization of a grid world problem is the same as the MDP formalization except it defines object classes and propositional functions and uses these to define its state space:

$\mathcal{O}$: { agent, wall }, $Att(agent) = Att(wall) = \{\text{xLocation}, \text{yLocation}\}$, $Dom(\text{xLocation}) = Dom(\text{yLocation}) = [0, d) \in \mathcal{Z}^+$ where d is the dimension of the square grid world.

$\mathcal{P}$: wallToNorth, wallToSouth, wallToEast, wallToWest. Each propositional function is true iff there is a wall in the appropriate cell adjacent to the agent.

$\mathcal{S}$: $s \in \mathcal{S}$ is a set of collections of objects. The collection of objects consists of one instantiation of an agent and some number of walls where the xLocation and yLocation of the agent are free to vary over their domain for different states but the xLocation and yLocation of the walls stay fixed across states.

Note that the OO-MDP representation of grid world allows for a quantification of similar states unavailable to the MDP representation: for instance, states in which the agent is surrounded by walls on all sides but the north are identical as per the propositional functions defined by the OO-MDP representation of grid world.

2.2 Vocabulary and Data Structures

OO-MDP domain: In keeping with the vocabulary of Abel et al. (2015), OO-MDPs in a single domain share all aspects of the OO-MDP seven-tuple except for a state space $\mathcal{S}$. The propositional functions that we learn are applicable across the domain in which they are learned.

Effects: An effect is defined by a type, an attribute, an object class and a real number. The type of the effect dictates how the effect changes the attribute of the object class. The two types that we examine are assignment and arithmetic effects: *assignment effects* set attributes of the effect's object class to a fixed value and *arithmetic effects* add a value to the attribute of the object class of the effect. We let $\mathcal{Y} = \{arithmetic, assignment\}$ denote the set of effects used. The real number indicates what value is added (arithmetic effects) or assigned (assignment effects).

We let our space of effects be noted $\mathcal{E} = (\mathcal{Y} \times \cup_{O \in \mathcal{O}} Att(O) \times \mathcal{O} \times \mathbb{R})$

Effects are a means of hypothesizing how actions affect attributes of one object class. For this reason effects are treated as functions that input states in $\mathcal{S}$ and output states in $\mathcal{S}$. That is, effect $e \in \mathcal{E} : \mathcal{S} \rightarrow \mathcal{S}$.

Given a particular object class $oClass$, attribute att and states s and s', we let $\text{eff}_{oClass,att}(s, s') : (\mathcal{S} \times \mathcal{S}) \rightarrow (\text{effects} \subseteq \mathcal{E})$ return all effects that capture how $oClass$'s att changes from s to s' for all effect types in $\mathcal{Y}$.

Contradictory effects: Two effects are said to be contradictory for a particular state $s \in \mathcal{S}$ if they both act on the same attribute and object class but they would cause different values to be assigned to the attribute of the object class if applied in state s. That is, two effects, e_1 and e_2, are incompatible for state s iff $e_1(s) \neq e_2(s)$.

Predictions: Predictions are uniquely defined by an action and an effect. They are the unit of our algorithm that summarizes how an action changes the state of the problem. More formally, predictions are a tuple of an action and an effect: $pred \in (\mathcal{A} \times \mathcal{E})$.

Related Predictions: We define the set of predictions related to $pred \in (\mathcal{A} \times \mathcal{E})$ to be those predictions that have the same action as $pred$ and an effect with the same type, object class and attribute as $pred$.

3 Technical Approach

Our approach has three parts. First, we perform a generalized state featurization that is likely to inform transition dynamics. Next, we produce supervised learning datasets with this state featurization from an RL agent's observations. Last, we transform these datasets into propositional functions by building classifiers on the datasets.

3.1 Featurizing without Propositional Functions

We seek a state featurization that is likely to inform transition dynamics. Transition dynamics in most domains are determined by the *relative differences* of values of object attributes. For instance, an agent in grid world fails to move north because its *y* location is 1 less than a wall's *y* location. Moreover, transition dynamics are not just determined by relative differences but they are determined by *minimal* relative differences. For instance, the agent in grid world cannot go north because the *closest* wall directly north of the agent has a *y* position *1 greater* than the *y* position of the agent. We would like to produce functions that act on states and return real values that leverage these crucial insights. We define the set of these functions as $\mathcal{L}$:

$$\mathcal{L} = \{arith_{oClass_1,oClass_2,att_1,att_2}, geom_{oClass_1,oClass_2,att_1,att_2}\} \quad (1)$$

where $oClass_1, oClass_2 \in \mathcal{O}$ and $att_1 \in Att(oClass_1)$ and $att_2 \in Att(oClass_2)$:

$arith_{att_1,att_2,oClass_1,oClass_2}(s)$ is the minimal arithmetic difference between att_1 and att_2 for all object instances of $oClass_1$ and $oClass_2$ in s. Similarly, $geom_{att_1,att_2,oClass_1,oClass_2}(s)$ is the same but for a geometric difference:

$$arith_{att_1,att_2,oClass_1,oClass_2}(s) = \min_{o_1 \text{ of } oClass_1 \in s, o_2 \text{ of } oClass_2 \in s} |o_1.att_1 - o_2.att| \quad (2)$$

$$geom_{att_1,att_2,oClass_1,oClass_2}(s) = \min_{o_1 \text{ of } oClass_1 \in s, o_2 \text{ of } oClass_2 \in s} \frac{o_1.att_1}{o_2.att} \quad (3)$$

The following example illustrates the $arith$ function for an OO-MDP with object classes A and B both with attributes $attX$ and $attY$. Consider an individual state, s, with one object of class A, A_1, and two of class B, B_1 and B_2. The respective values of $attX$ and $attY$ are 5 and 10 for A_1, 4 and 2 for B_1 and 1 and 5 for B_2. We can evaluate $arith_{attX,attY,A,B}(s)$ by taking the minimum difference between the $attX$ value of all A instantiations and the and $attY$ value of all B instantiations:

$$arith_{attX,attY,A,B}(s) = min(|A_1.attX - B_1.attY|, |A_1.attX - B_2.attY|) = min(|5-4|, |5-5|) = 0 \quad (4)$$

Having formulated functions closely informed by transition dynamics, we now construct our full state featurization. The ith element of our featurized representation for a state corresponds to a unique parameterization of a function in $\mathcal{L}$. That is, each feature uniquely corresponds to a tuple of (a function in $\mathcal{L}$, a pair of object classes, a pair of object attributes). The ith feature's value is calculated by evaluating $l \in \mathcal{L}$ with a unique parameterization:

$$v(s)_i = l_{att_1,att_2,oClass_1,oClass_2}(s) \quad (5)$$

where att_1 and att_2 are attribute values of object classes $oClass_1$ and $oClass_2$ respectively.

3.2 Extracting Annotated Datasets

Having featurized our state space in a manner likely to inform transition dynamics, we now want to generate datasets to be used by a supervised learning classifier. More formally stated, given a set of observations of a RL agent, $O \subseteq (\mathcal{S} \times \mathcal{A} \times \mathcal{S})$ in an OO-MDP setting (with $\mathcal{P} = \emptyset$) and a function that featurizes our states, $v(s)$, we want to infer some set of annotated datasets, D_v. $d \in D_v$ is an annotated dataset of the form $d = ((v(s_0), b_0), \ldots, (v(s_{|d|}), b_{|d|}))$ where s_i corresponds to the initial state in the ith observation in O, and b_i is a boolean.

High level details of our algorithm are as follows. We maintain predictions as detailed in Section 2.2. Each prediction has an associated annotated dataset. This dataset consists of a set of states featurized according to v. Included states are those states that were an initial state in an $o \in O$ where the action of o is the prediction's action. The associated label of each state is true if the effect of the prediction was observed to take place between the initial state and resulting state in o. Otherwise the label is false. The end result is that each prediction has an associated dataset of states that notes those states under which the prediction's effect took place when its action was executed. Since effect occurrence is dictated by OO-MDP transition dynamics, generated datasets are closely informed by the transition dynamics of the OO-MDP. Also note that an integer k is specified; if there exist more than k related predictions for a particular prediction, those predictions are deemed unlikely to predict transition dynamics and so are ruled out. Our algorithm is largely inspired by DOORMAX's learn routine (Diuk, Cohen, and Littman 2008). Note that although we are using RL agent observations, our method is also applicable to planning problems since one can trivially "simulate" an RL agent if the transition dynamics are known. Pseudocode to generate D_v is show in Algorithm 1. ω stores contradictory predictions.

3.3 Inferring Propositional Functions from Annotated Datasets

Given a set of labeled datasets D_v where each dataset $d \in D_v$ consists of states in $\mathcal{S}$ featurized according to some featurization function v with labels of either true or false, we would like to generate some set of propositional functions.

We treat each dataset as a supervised learning dataset for learning a single propositional function. A separate binary classifier is trained on each dataset. This classifier's classify routine, in turn, acts as a propositional function: it classifies state s as true or false. We use a J48 decision tree implementation as detailed by (Quinlan 1993) implemented by the Weka machine learning library (Hall et al. 2009). Other machine learning methods could straightforwardly supplant the J48.

Thus the end result is $|D_v|$ classifiers whose classify routines define $|D_v|$ propositional functions. Having described how we generate propositional functions, we now explain how propositional functions of this style can replace those normally used by DOORMAX.

Algorithm 1 GenerateDataSets()

INPUT: a set of observations O, a state featurizer v, a k in the DOORMAX sense
OUTPUT: a set of labeled datasets D_v

$D_v \leftarrow \emptyset$
$\omega \leftarrow \emptyset$
//Loop over observations
for all $(s, a, s') \in O$ **do**
  for all $(oClass, att) \in (\mathcal{O} \times \text{Att}(oClass))$ **do**
    updatedDataSets$\leftarrow \emptyset$
    //Loop over hypothesized effects
    for all hypEffect $\in \text{eff}_{oClass,att}(s, s')$ **do**
      //Check for dataset with prediction for this effect
      if $\exists$ d $\in D_v$ s.t. d.pred.effect = hypEffect **then**
        d.add($(v(s), \text{true})$)
        updatedDataSets.add(d)
      else
        if (hypEffect.type, oClass, att, a) $\notin \omega$ **then**
          newPrediction $\leftarrow (a, null, \text{hypEffect})$
          newDataSet $\leftarrow \{(v(s), \text{true})\}$
          newDataSet.prediction = newPrediction
          D_v.add(newPrediction)
          //Rule out uninformative datasets
          relatedDataSets $\leftarrow \cup d \in D_v$ s.t. d.prediction is related to newPrediction
          if |relatedDataSets|$> k$ **then**
            ω.add((newPrediction.effect, oClass, att, a))
            D_v.removeAll(relatedDataSets)
          end if
          updatedDataSets.add(d)
        end if
      end if
    end for
    //Update all appropriate datasets that did not receive a true with a false
    for all d $\in D_v$ and $\notin$ updatedDataSets s.t. d.prediction.action = a **do**
      d.add($(v(s), \text{false})$)
    end for
  end for
end for
return D_v

3.4 DOORMAX with Learned Propositional Functions

DOORMAX is an RMax (Brafman and Tennenholtz 2003) implementation that exploits the OO-MDP state representation for efficient model learning in deterministic domains. DOORMAX's input consists of an OO-MDP without $\mathcal{T}$, a parameter k which dictates the maximum number of predictions in any set of related predictions, an initial state s_0 and any parameters for planners that it uses. DOORMAX maintains a set of failure conditions for each action's effect on each attribute of each class, F, a set of predictions, α, and a set of contradictory effect types, object class, attribute, action tuples, ω. Each time DOORMAX observes a state, action and resulting state, it performs a learn routine to update its predictions and then a prediction routine to produce a model that can be planned over.

DOORMAX normally requires propositional functions to determine the conditions under which an effect is thought to take place. These conditions are constructed out of the propositional functions supplied to the OO-MDP. The conditions can be thought of as propositional functions unto themselves which return "true" when they match a state's corresponding condition. We modify DOORMAX by replacing entire conditions with propositional functions learned in our framework.

We modify DOORMAX's learn routine as follows. All predictions now have an associated dataset of states featurized according to the featurizing function v. States included in a prediction's dataset are those states that served as an initial state in an observation whose action matches the prediction's action. A state in a prediction's dataset is labeled with "true" if the prediction's effect was observed to take place from that state. A state is labeled with "false" otherwise. The rest of the DOORMAX learn machinery runs as usual. Full pseudocode is shown in Algorithm 2.

We modify DOORMAX's predict routine as follows. Each prediction is now associated with a dataset rather than a condition. A classifier is trained on each prediction's dataset. When predicting a resulting state, a prediction's action is posited if the trained classifier returns "true" for the current state. As before, we use a J48 decision tree for our classifier. If contradictory effects are returned or there is no relevant prediction or failure condition, transition to an illusory state of maximum reward, RMAX, is predicted. Full pseudocode is shown in Algorithm 3.

Our modified version of DOORMAX consists of classic DOORMAX with its learn and predict routines replaced by Algorithms 2 and 3 respectively.

4 Experimental Results

We empirically confirm the correspondence between our learned propositional functions and those normally provided by experts. We also demonstrate the near DOORMAX-level performance of our novel DOORMAX algorithm in a taxi domain.

Algorithm 2 DOORMAXLearnWithoutPFs()

INPUT: a state s, an action a and the state s' that resulted from taking a in s and the variables from top-level DOORMAX
OUTPUT: none

for all $(oClass, att) \in (\mathcal{O} \times \text{Att}(oClass))$ **do**
  updatedPredictions$\leftarrow \emptyset$
  //Loop over hypothesized effects
  for all hypEffect $\in$ eff$_{oClass,att}(s, s')$ **do**
    //Check for prediction for this effect
    if $\exists$ pred $\in \omega$ s.t. pred.effect = hypEffect **then**
      pred.d.add($(v(s)$, true))
      updatedPredictions.add(pred)
    else
      if (hypEffect.type, oClass, att, a) $\notin \omega$ **then**
        newDataSet $\leftarrow \{(v(s)$, true$)\}$
        newPrediction $\leftarrow (a, null,$ hypEffect)
        newPrediction.dataSet = newDataSet
        ω.add(newPrediction)
        //Rule out predictions if more than k related
        relatedPredictions $\leftarrow \cup p \in \omega$ s.t. p is related to newPrediction
        if |relatedPredictions|$> k$ **then**
          ω.add((newPrediction.effect, oClass, att, a))
          ω.removeAll(relatedDataSets)
        end if
        updatedPredictions.add(newPrediction)
      end if
    end if
  end for
  //Update all appropriate datasets that did not receive a true with a false
  for all p $\in \omega$ and $\notin$ updatedPredictions s.t. p.action = a **do**
    p.d.add($(v(s)$, false))
  end for
end for

Algorithm 3 DOORMAXPredictWithoutPFs()

INPUT: The variables from top-level DOORMAX
OUTPUT: A model, $\mathcal{T}$
for all $(s, a) \in (\mathcal{S} \times \mathcal{A})$ **do**
  totalPool $\leftarrow \emptyset$
  for all $(objectClass) \in \mathcal{O}$ **do**
    currentPool $\leftarrow \emptyset$
    for all $eType \in \mathcal{Y}$ **do**
      for all $att \in Att(\mathcal{O})$ **do**
        for all pred $\in \alpha$ s.t. pred.action == a and pred.effect.effectType == $eType$ **do**
          classifier = J48(pred.d)
          if classifier.classify(s) **then**
            currentPool.add(pred.effect)
          end if
        end for
      end for
      if $\exists (e_1, e_2) \in$currentPool such that e_1 contradicts e_2 **then**
        $\mathcal{T}(s, a, RMAX) = 1$, continue looping over (s, a)
      end if
      if currentPool.isEmpty() and (eType, oClass, att, a) $\notin \omega$ **then**
        $\mathcal{T}(s, a, RMAX) = 1$, continue looping over (s, a)
      end if
    end for
  end for
  //Apply all predicted effects
  resultingState $\leftarrow s$
  for all effect $\in$ totalPool **do**
    resultingState $\leftarrow$ effect(resultingState)
  end for
  $\mathcal{T}(s, a, resultingState) = 1$
end for
return $\mathcal{T}$

4.1 Taxi Domain

Taxi problems (Dietterich 2000) consist of an agent that can move north, east, south and west in a two-dimensional grid world, wherein movement actions may be thwarted by walls. In taxi problems, walls occupy space between cells rather than cells themselves and there are also passengers, each of which occupy a single cell and which the taxi can pick up or drop off. When a passenger is picked up it moves with the taxi until dropped off. The problem is considered solved when all passengers are delivered to their goal cells.

Figure 1 visualizes what we term the classic taxi state. The grey circle corresponds to the taxi. Squares of various colors indicate goal locations for theoretical passengers. The red circle corresponds to the passenger who wishes to be delivered to the red square. Walls are black lines and there are implicit walls around the edges of the map.

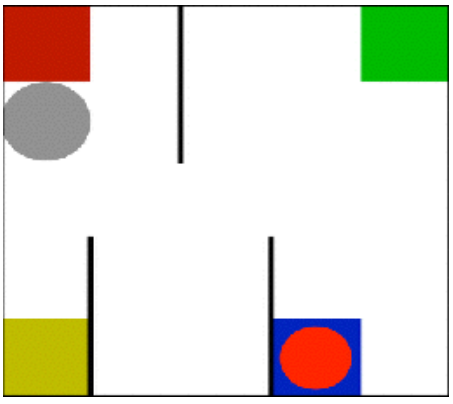

Figure 1: The classic taxi domain problem

The formal OO-MDP representation of the taxi problem used in our experiments is as follows:

$\mathcal{A}$: { $taxiMoveNorth$, $taxiMoveEast$, $taxiMoveSouth$, $taxiMoveWest$, $pickupPassenger$, $dropOffPassenger$}

$\mathcal{O}$: { goalLocation($xLocation$, $yLocation$, $goalType$), $taxi$($xLocation$, $yLocation$, $passengerInTaxi$), passenger($xLocation$, $yLocation$, $goalType$, $inTaxi$), verticalWall($wallOffSet$, $bottom$, top), horizontalWall($wallOffSet$, $leftStart$, $rightStart$) }

Values in parenthesis indicate attributes of each of the object classes. Note that $goalLocation$s are locations that passengers with matching $goalType$s are trying to reach. The offset of vertical walls indicates the horizontal offset from the origin whereas the $bottom$ and top attributes indicate the start and end y positions of the wall. Symmetric clarification applies to horizontal walls. All attributes range over $\mathcal{Z}$.

$\mathcal{R}$: uniform -1 for all states.

$\mathcal{T}$: Transitions are deterministic and self-evident by action names.

γ : .95.

$\mathcal{S}$: all those states reachable from the initial state using $\mathcal{A}$ according to $\mathcal{T}$.

Additionally, conventional taxi problems supply a set of expert-provided propositional functions, $\mathcal{P}$. The functions traditionally provided are:

$\mathcal{P}$: { $wallToNorthOfTaxi$, $wallToEastOfTaxi$, $wallToSouthOfTaxi$, $wallToWestTaxi$, $passengerInTaxi$, $taxiAtPassenger$ }

Our taxi domain is a set of taxi OO-MDPs where $\mathcal{S}$ is the only aspect of the OO-MDP that varies. The $\mathcal{S}$ for a particular OO-MDP is those states reachable by $\mathcal{A}$ from an initial state. The initial state is identical to the classic taxi OO-MDP initial state but with the three vertical walls, all passengers' initial positions, all goal locations' positions and the taxi's initial position uniformly and independently randomized within the boundaries of the map. The width and height of the map also are uniformly and independently randomized between 5 and 30. The map is not necessarily square.

4.2 Agreement with Existing Propositional Functions

We demonstrate correspondence between our learned propositional functions and those normally expert-provided for our taxi domain. Taxi domain is used because 4 of those propositional functions normally supplied for the OO-MDP directly correspond to four propositional functions that will be learned by our method: wallToEast, wallToWest, wallToSouth and wallToNorth. This correspondence exists because the conditions for four predictions of DOORMAX when run on a taxi domain are uniquely determined by one of these propositional functions. For instance, the prediction that north adds 1 to the agent's y position has a condition that only depends on the wallToNorth propositional function. This direct correspondence accommodates ease of evaluation of the efficacy with which we learn propositional functions, since we may simply compare a given learned proposition to its expert counterpart.

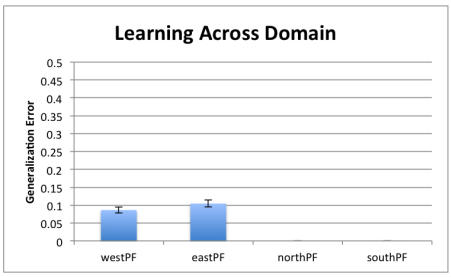

Figure 2: Percent error rates for learned propositional functions (PFs) compared to expert-provided PFs

Experimental Setup Our training set consists of 10,000 observations of the result of random actions on random states where random states are sampled uniformly from the union of all states of all OO-MDPs in the domain. $|D_v| = 6$ and we examine error rates for the 4 propositional functions that correspond to a wall being adjacent to the taxi for north, east, south and west. Error quantification involves 10 trials of sampling 1,000 states uniformly from the union of all states defined by all OO-MDPs in the domain (for 10,000 states total) and measuring the percentage of states for which learned propositional functions predict something different from their corresponding propositional function in $\mathcal{P}$. Corresponding functions were manually determined.

Experimental Results When propositional functions are learned across the taxi domain, we observe a strong correspondence between propositional functions in $\mathcal{P}$ and their corresponding learned propositional functions. The wallToNorth and wallToSouth propositional functions are learned nearly perfectly. We attribute this result to the fact that there are no horizontal walls in our taxi domain other than those at the top or bottom of the map, making learning adjacency to horizontal walls particularly easy. Full results are detailed in Figure 2.

4.3 DOORMAX with Learned Propositional Functions

We now test our modified DOORMAX, which uses learned rather than expert-provided propositional functions.

Experimental Setup We conduct experiments in a taxi domain problem with the classic taxi domain state as the initial state. We run tabular RMAX, classic DOORMAX, as well as our modified DOORMAX. $k = 2$ for both versions of DOORMAX. Each algorithm is allowed to run until a learning episode terminates by arriving in the terminal state. 10 learning episodes are run for all three algorithms. The number of cumulative steps for all previous learning episodes is reported for each learning episode. Since a uniform negative reward is used, the number of cumulative steps is equivalently interpreted as cumulative cost received. When the slope of this plot ceases to change, the algorithm has converged on a policy.

Experimental Results RMAX takes approximately 4,500 actions to converge on the (optimal) policy, which we

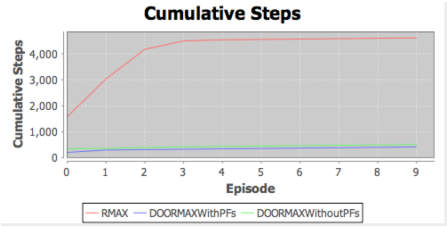

Figure 3: Performance of RMAX(RMAX), classic DOORMAX (DOORMAXWITHPFs), and DOORMAX with learned propositional functions(DOORMAXWithoutPFs)

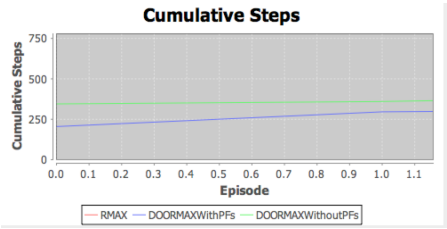

Figure 4: Expansion of the performance data in Figure 3

might reasonably expect given RMAX's slow, tabular nature. DOORMAX with expert-provided propositional functions takes approximately 260 actions, and our DOORMAX without any expert-provided propositional functions takes 350 actions. Our novel DOORMAX, then, performs nearly as well as classic DOORMAX but requires dramatically less expert knowledge in the form of propositional functions. Full results are shown in Figures 3 and 4.

5 Related Work

Jetchev, Lang, and Toussaint perform notably similar work in a robotics setting. They formalize symbol abstraction as an optimization problem. Symbols have associated propositional functions and a loss signal is received based on how well the symbols predict transition dynamics and received reward. Propositional functions are learned by training classifiers on a "geometric feature" representation of the problem. While their work and our work both approach learning propositional functions by exploiting transition dynamics, their work approaches it from a function optimization perspective while our method approaches the problem from an OO-MDP framework. We suspect that our OO-MDP approach will lend itself to greater generalization than Jetchev, Lang, and Toussaint's method because it is factored around objects. We hope to further compare our methodologies in future work.

Konidaris, Kaelbling, and Lozano-Perez present theoretical results for what sorts of symbolic representations are sufficient for evaluating plans in low-level continuous domains and present a means to automatically learn these representations (2014; 2015). In particular they demonstrate that an agent's environment and the actions available to an agent fully determine the symbolic representations needed for relational planning. While this work deals with semi-Markov decision processes (SMDPs) with action spaces that include options (Sutton, Precup, and Singh 1999) (unlike our OO-MDP domains), the intimate relationship it describes between symbolic representations and an agent's environment is encouraging for our work, which exploits the agent's environment in the form of the transition dynamics of the agent's domain. Furthermore, our work differs in that our goal is not to learn propositional functions that replace the state space for high-level relational planning, but augment it with useful information that can be used in various ways, such as for model learning.

Autoencoders (Hinton and Salakhutdinov 2006) can be used similarly to our methodology to learn high-level representations. An autoencoder is an artificial neural network. Given sample inputs, autoencoders are trained to learn the identity function by setting their target output layer to be identical to the input layer. The hidden nodes of the network act as a compression of the data that can then be used as a high-level representation. However, autoencoders do not learn object-parameterized binary functions that could be used as propositional functions for OO-MDPs and therefore may not generalize in the same ways.

Deep Q-Learning also compresses large state spaces by using a deep neural network for function approximation in Q-Learning (Mnih et al. 2013). However, Deep Q-Learning is focused on model-free learning, whereas our approach greatly benefits model-based RL algorithms, which can provide different opportunities for knowledge transfer and generalization.

Batch incremental feature dependency discovery (Batch-iFDD) (Geramifard et al. 2013a) also offers a means of deriving novel propositional functions in planning domains. In particular, it learns useful conjunctions of more atomic propositional functions in batch. However, this methodology requires an initial pool of atomic propositional functions, which are method does not, and so it does not fill the same niche as our approach. We are, nonetheless, interested in supplying our learned propositional functions as atomic propositions to Batch-iFDD.

6 Conclusion

We present a means of automatically learning propositional functions for planning and RL algorithms. Our method performs propositional function inference over RL agent observations. Given the generality of OO-MDP propositional functions, our learned propositional functions could play a critical role in knowledge transfer algorithms that transfer knowledge both within a single OO-MDP (e.g. DOORMAX or (Geramifard et al. 2013b)) and those that transfer knowledge across OO-MDPs with distinct state spaces (e.g. (Abel et al. 2015)). We demonstrate the former by supplanting the normally expert-provided propositional functions of DOORMAX with our learned propositional functions. Note that no aspect of our algorithm requires determinism, so our propositional functions are learnable in both deterministic and stochastic domains. Empirical results are gathered in taxi domain and include demonstration of correspondence between learned propositional functions and those normally expert-provided. We also demon-

strate the near DOORMAX-level performance of our modified DOORMAX, which runs without propositional functions.

In future work we hope to explore the extent to which these learned propositional functions can be utilized. In particular we are interested in applying these learned propositional functions to the many algorithms that exploit propositional functions for planning and RL (Abel et al. 2015; Geramifard et al. 2013b). We also want to explore ways of ruling out superfluous predictions and their associated datasets as well as principled manners of applying our methodology to stochastic domains.

7 Acknowledgement

This work was supported in part by DARPA under "Hierarchical, Probabilistic Planning and Learning for Collaboration," grant number W911NF-15-1-0503.

References

[Abel et al. 2015] Abel, D.; Hershkowitz, D. E.; Barth-Maron, G.; Brawner, S.; O'Farrell, K.; MacGlashan, J.; and Tellex, S. 2015. Goal-based action priors. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling*.

[Bollini et al. 2013] Bollini, M.; Tellex, S.; Thompson, T.; Roy, N.; and Rus, D. 2013. Interpreting and executing recipes with a cooking robot. In *Experimental Robotics*, 481–495. Springer.

[Brafman and Tennenholtz 2003] Brafman, R. I., and Tennenholtz, M. 2003. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res.* 3:213–231.

[Dietterich 2000] Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Intell. Res.(JAIR)* 13:227–303.

[Diuk, Cohen, and Littman 2008] Diuk, C.; Cohen, A.; and Littman, M. L. 2008. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, 240–247. New York, NY, USA: ACM.

[Geramifard et al. 2013a] Geramifard, A.; Walsh, T. J.; Roy, N.; and How, J. P. 2013a. Batch-ifdd for representation expansion in large mdps. *CoRR* abs/1309.6831.

[Geramifard et al. 2013b] Geramifard, A.; Walsh, T. J.; Tellex, S.; Chowdhary, G.; Roy, N.; and How, J. P. 2013b. A tutorial on linear function approximators for dynamic programming and reinforcement learning. *Foundations and Trends in Machine Learning* 6(4):375–451.

[Hall et al. 2009] Hall, M.; Frank, E.; Holmes, G.; Pfahringer, B.; Reutemann, P.; and Witten, I. H. 2009. The weka data mining software: An update. *SIGKDD Explor. Newsl.* 11(1):10–18.

[Hinton and Salakhutdinov 2006] Hinton, G. E., and Salakhutdinov, R. R. 2006. Reducing the dimensionality of data with neural networks. *Science* 313(5786):504–507.

[Jetchev, Lang, and Toussaint] Jetchev, N.; Lang, T.; and Toussaint, M. Learning grounded relational symbols from continuous data for abstract reasoning.

[Knepper et al. 2013] Knepper, R.; Tellex, S.; Li, A.; Roy, N.; and Rus, D. 2013. Single assembly robot in search of human partner: Versatile grounded language generation. In *Human-Robot Interaction (HRI), 2013 8th ACM/IEEE International Conference on*, 167–168.

[Konidaris, Kaelbling, and Lozano-Perez 2014] Konidaris, G.; Kaelbling, L. P.; and Lozano-Perez, T. 2014. Constructing symbolic representations for high-level planning. *Proceedings of the Twenty-Eighth Conference on Artificial Intelligence* 1932–1940.

[Konidaris, Kaelbling, and Lozano-Perez 2015] Konidaris, G.; Kaelbling, L. P.; and Lozano-Perez, T. 2015. Symbol acquisition for probabilistic high-level planning. *Proceedings of the Twenty Fourth International Joint Conference on Artificial Intelligence*.

[Mnih et al. 2013] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[Quinlan 1993] Quinlan, R. 1993. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers.

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