ECAI 2016
G.A. Kaminka et al. (Eds.)

doi:10.3233/978-1-61499-672-9-1283

Learning the Structure of Dynamic Hybrid Relational Models

Davide Nitti[1 2] and **Irma Ravkic**[1 2] and **Jesse Davis**[2] and **Luc De Raedt**[2]

Abstract. Typical approaches to relational MDPs consider only discrete variables or else discretize the continuous variables prior to inference or learning. In contrast, we consider hybrid relational MDPs, which are represented as probabilistic programs and specify the probability density function of the continuous variables. Our key contribution is that we introduce a technique for learning their structure (and parameters) from data. The learned models contain rich relational descriptions as well as mathematical equations. We demonstrate the utility of our approach by learning a model that accurately predicts the effects of robot-arm actions. The learned model is then used for planning tasks.

1 Introduction

Markov Decision Processes (MDPs) are the standard representation used in probabilistic planning and reinforcement learning [19, 24, 22]. Relational MDPs integrate these processes with principles of statistical relational artificial intelligence [6], resulting in models that make abstraction of sets of states, transitions and can compactly represent and generalize across states having a variable number of objects through the use of relations. While relational MDPs are popular in planning and learning and there has been steady progress, their application to domains such as robotics is still severely limited by the emphasis on purely symbolic representations and the lack of general methods for dealing with subsymbolic (e.g., numeric) information. Approaches that have combined relational MDPs and robotics include learning probabilistic relational planning rules [17, 16], relational reinforcement learning [20, 13, 3], imitation learning [12], inverse reinforcement learning [11], and learning relational affordance models in multi-object manipulation tasks [10]. However, in all these cases the low-level numeric information had to be converted into a symbolic representation, often prior to learning and inference, leading to a loss of information, potentially affecting the quality of the obtained solutions. In another case, the approach required experts to provide partial code for the relational MDP [9].

We propose a different "hybrid" approach to relational MDPs in which numeric features are first-class citizens and the models are fully learned from data. Rather than discretizing numeric features and relations, we explicitly represent their probability density in an expressive probabilistic programming language, the dynamic distributional clauses (DDCs) [15], which has already been used in a robotics context and for which a planner, called HYPE [14], exists. Specifically, we address the problem of learning a state transition model from a set of trajectories collected from a robot-arm performing actions. Our central contribution is an algorithm to learn the structure and the parameters of a DDC model representing a hybrid relational MDP. In order to realize this, we leverage statistical relational learning (SRL) techniques for learning hybrid probabilistic relational models (e.g., [23, 5, 18]). One novelty from an SRL perspective is that the learned DDCs include expressive features defined as mathematical equations involving the continuous variables, which is useful for finding, for instance, the object closest to the moved one. We demonstrate the utility of our DDC-TL algorithm (DDC Tree Learner) by applying the learned model to perform planning with HYPE [14] in a simple robotics scenario.

By making continuous features first class citizens in relational MDPs, we hope to contribute towards bridging the gap between symbolic and numeric approaches and to facilitate the application of relational MDPs to robotics.

2 Background

Next we introduce the necessary background needed to explain our learner of hybrid dynamic relational models and how the learned hybrid models are used for planning.

2.1 MDP

The problem of planning under uncertainty can be modeled as a Markov decision process (MDP). In an MDP, an agent interacts with its environment, described using a set of *states* S, a set of *actions* A that the agent can perform, a *transition function* $p : S \times A \times S \to [0, 1]$, and a *reward function* $R : S \times A \to \mathbb{R}$. That is, when in state s_t and performing action a_t, the probability of reaching s_{t+1} is given by $p(s_{t+1}|s_t, a_t)$, for which the agent receives the reward $R(s_t, a_t)$. It is assumed that the agent operates over a finite number of time steps $t = 0, 1, \ldots, T$, with the goal of maximizing the expected reward: $\mathbb{E}[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)]$, where s_0 is the start state, a_0 the first action, and $\gamma \in [0, 1]$ is a discount factor. In this paper we consider goal-oriented MDPs where there is a goal $g \subset S$ to reach, and the reward is high if we reach the goal, and low otherwise.

For planning, we shall use HYPE [14], a recently proposed planner for hybrid relational MDPs. It is a sample-based planner that exploits the model to simulate action effects and to determine the action in each state that maximizes the expected total reward. As input, HYPE requires both the starting state s_0 and the MDP specification of the domain described using DDCs. That is, it must be given the state transition model and the reward function.

2.2 Distributional Clauses

We assume some familiarity with standard terminology of statistical relational learning and logic programming [4]. Briefly, in logic

[1] These authors contributed equally to this work.
[2] Department of Computer Science, KU Leuven, Belgium

programming symbols can be terms and predicates (often called relations). A term is a constant, a logical variable (logvar) or an n-ary functor f applied to a tuple of terms $t_1, t_2, ..., t_n$, that is, $f(t_1, t_2, ..., t_n)$. A constant term refers to a single object in the domain of interest. A logical variable (logvar) $\mathtt{X}$ is a variable ranging over terms $\mathtt{x} \in \mathcal{C}$, where $\mathcal{C}$ is the set of possible ground terms (Herbrand universe). An atom is of the form $\mathtt{P}(\mathtt{t}_1, ..., \mathtt{t}_n)$ where $\mathtt{P}/n$ is a predicate with arity n and each $\mathtt{t}_i$ is a term. For example $\mathtt{near}(1, 2)$ is an atom that describes that objects 1 and 2 are close to each other. A ground atom (or expression) does not contain logvars. A literal is an atom or its negation.

In distributional clauses (DCs) [5, 15], an interpretation I assigns a value $I(\mathtt{a})$ to each random variable $\mathtt{a}$. While in logic programming that value of ground atoms will be true or false, in DCs the values can also be discrete or numeric. A substitution $\theta = \{\mathtt{v}_1/\mathtt{t}_1, ..., \mathtt{v}_n/\mathtt{t}_n\}$ assigns terms $\mathtt{t}_i$ to variables. A substitution θ can be applied to an expression $\mathtt{E}$ yielding the expression $\mathtt{E}\theta$ where all variables $\mathtt{v}_i$ in $\mathtt{E}$ are simultaneously replaced by the corresponding terms $\mathtt{t}_i$ in θ. A grounding substitution for an expression maps each logvar occurring in that expression to a term without logvars. The set of all grounding substitutions for an expression $\mathtt{E}$ is denoted $grsub(\mathtt{E})$.

Formally, a *distributional clause* (DC) is a formula of the form $\mathtt{h} \sim \mathcal{D} \leftarrow \mathtt{b}_1, \ldots, \mathtt{b}_n$, where the $\mathtt{b}_i$ are literals and $\sim$ is a binary predicate written in infix notation. The name of the random variable h and the distribution $\mathcal{D}$ are formally terms. The intended meaning of a distributional clause is that each ground instance of the clause $(\mathtt{h} \sim \mathcal{D} \leftarrow \mathtt{b}_1, \ldots, \mathtt{b}_n)\theta$ defines the random variable $\mathtt{h}\theta$ with distribution $\mathcal{D}\theta$ whenever all the $\mathtt{b}_i\theta$ are true, where θ is a substitution. A distributional clause is a template to define conditional probabilities: $p(\mathtt{h}\theta|(\mathtt{b}_1, \ldots, \mathtt{b}_n)\theta) = \mathcal{D}\theta$. The term $\mathcal{D}$ can be nonground, i.e., values, probabilities, or distribution parameters can be related to conditions in the body. Furthermore, given a random variable $\mathtt{r}$, the term $\simeq(\mathtt{r})$ constructed from the reserved functor $\simeq/1$ represents the value of $\mathtt{r}$.

Dynamic distributional clauses (DDCs) associate a time index to each random variable to capture temporal information. It is easy to specify an MDP using DDC as described in [14].

Example 1. Let us consider a scenario of pushing an object, where there is an object on the table and the robot has to move it in a given region. This scenario is modeled with the following MDP:

$$\mathtt{pos}(\mathtt{ID})_{\mathtt{t}+1} \sim \mathtt{gaussian}(\simeq(\mathtt{pos}(\mathtt{ID})_{\mathtt{t}}) + (\mathtt{DX}, \mathtt{DY}), \Sigma) \leftarrow \mathtt{push}(\mathtt{ID}, (\mathtt{DX}, \mathtt{DY})). \quad (1)$$

$$\mathtt{stop}_{\mathtt{t}} \leftarrow \mathtt{dist}(\simeq(\mathtt{pos}(\mathtt{ID})_{\mathtt{t}}), (0.6, 1.0)) < 0.1. \quad (2)$$

$$\mathtt{reward}(100)_{\mathtt{t}} \leftarrow \mathtt{stop}_{\mathtt{t}}. \quad (3)$$

$$\mathtt{reward}(-1)_{\mathtt{t}} \leftarrow \mathtt{not}(\mathtt{stop}_{\mathtt{t}}). \quad (4)$$

The DDC clause (1) defines the state transition model, i.e. the next position $\mathtt{pos}(\mathtt{ID})_{\mathtt{t}+1}$ of an object $\mathtt{ID}$ after a push action with displacement $(\mathtt{DX}, \mathtt{DY})$. The deterministic clause (2) defines when the goal is reached (e.g., an object is close to point $(0.6, 1.0)$) and the remaining clauses define the reward function.

Note that $\mathtt{pos}(1)_{\mathtt{t}+1}$ represents a random variable, i.e., the position of object 1 at time $t + 1$, with predicate-like notation. But unlike standard relational representations, a random variable can have a continuous (or categorical) range. Thus, in DCs and DDCs an interpretation assigns each random variable a value in its range.

Static inference in DCs is performed using importance sampling, while filtering in dynamic models is performed using particle filtering methods [15].

This paper explores how to learn a DDC program that describes the state-transition model by using DDC-TL, a relational tree learner inspired by learner of local models (LLM) [18] for hybrid relational dependency networks.

3 Learning State Transition Models

We propose an algorithm (DDC-TL) for learning a state-transition model, expressed as a DDC, from data described by continuous variables, discrete variables, and relations (e.g., $\mathtt{nextTo}$). Concretely, given a set of discrete-time trajectories $(\mathbf{s}_0, a_0, \mathbf{s}_1, a_1, ..., \mathbf{s}_{T-1}, a_{T-1}, \mathbf{s}_T)$, where $\mathbf{s}_t$ is the state (i.e., an interpretation that could describe object properties and relations such as position, orientation, type, and color) and a_t is an action at time t, the goal is to learn DDCs that define a state transition model of the following form:

$$\mathtt{Q}_{t+1} \sim \mathcal{D}(f_c(\mathbf{s}_t)) \leftarrow \mathtt{body}_{\mathtt{t}} \quad (5)$$

Each such clause defines the distribution $\mathcal{D}(f_c(\mathbf{s}_t))$ of a relational random variable $\mathtt{Q}_{t+1} \in \mathbf{s}_{t+1}$ in terms of a set of (continuous) relational features $f_c(\mathbf{s}_t)$ whenever $\mathtt{body}_{\mathtt{t}}$ holds, where $\mathtt{body}_{\mathtt{t}}$ is a conjunction of literals (discrete conditions) that refer to the current state $\mathbf{s}_t$ and action a_t. The relational features are essentially the random variables defined in the DDC. Note that DDCs defined in this manner result in a stratification where predicates at time $t+1$ only depend on predicates from the previous time step t. Stratification is required for DDCs to be well-defined [5].

In this paper, we consider a robot arm that moves objects on a table. The actions considered are grasping followed by a vertical or horizontal movement, pushing or tapping, while the features will be positions of objects and all the derived relationships. However, unlike related work, concepts like $\mathtt{rightOf}$, $\mathtt{closeTo}$ or $\mathtt{aboveOf}$ are not manually defined, but are indirectly learned with equational features, which we will describe later. We assume inverse kinematics and a motion planner are available to execute the described actions.

3.1 The Learned Model

The key insight to develop an algorithm for learning DDCs is that we can leverage ideas from the learner of local models (LLM) approach for learning hybrid relational dependency networks (HRDNs) [18]. HRDNs are able to model dependencies in relational domains that have both continuous and discrete variables. Like DDCs, HRDNs use a variant of first-order logic as a template language for defining conditional probability distributions. An HRDN uses a set of local distributions to approximate a joint distribution over a set of random variables defined by grounding atoms constructed over a set of predicates $\mathcal{P}$ and terms in $\mathcal{C}$. In this paper, we are interested in learning the dependencies that represent Formula (5). Hence, $\mathcal{Q}_{t+1}$ denote the set of predicates for which we learn dependencies. We use $\mathcal{P}_t$ to denote the set of predicates describing the previous time step, which are used to construct the features that appear in $Parents(\mathtt{Q}_{\mathtt{t}+1})$. Each local distribution is quantified by a dependency $\mathtt{Q}_{\mathtt{t}+1} \mid Parents(\mathtt{Q}_{\mathtt{t}+1})$, where $\mathtt{Q}_{\mathtt{t}+1} \in \mathcal{Q}_{t+1}$ is a predicate and $Parents(\mathtt{Q}_{\mathtt{t}+1})$ is a set of relational features that describe how $\mathtt{Q}_{\mathtt{t}+1}$ depends on the other predicates in the domain. Such local distributions can be learned using relational regression trees [2], but unlike standard (relational) regression trees, each leaf does not contain a constant but instead has a linear or logistic regression model. The key observation is that the distribution modeled by a relational regression tree can be represented with DDCs. Therefore, techniques for learning relational regression

trees can be adapted for learning (certain classes of) distributional clauses.

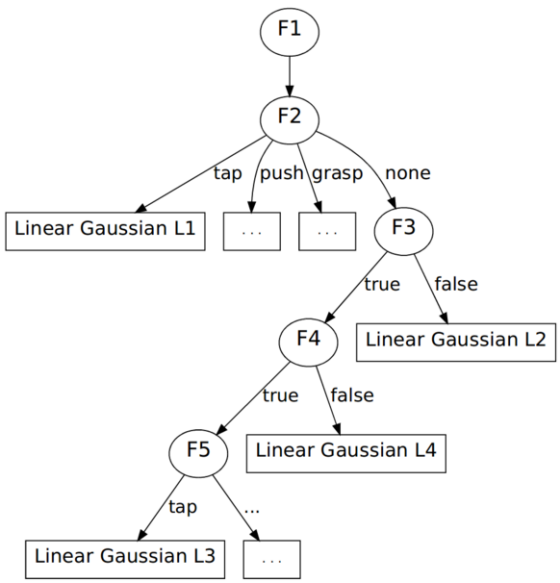

Figure 1. A simplified snapshot of a relational regression tree for predicting $\texttt{pos}(\texttt{ID})_{t+1}$. Ellipses represent internal nodes corresponding to learned relational features, and rectangles represent density functions for $\texttt{pos}(\texttt{ID})_{t+1}$ as defined in Example 2.

Example 2. Figure 1 shows a simplified example of a relational regression tree learned by DDC-TL. Each root-to-leaf path can be mapped onto one DDC, where each internal node defines a condition b_i that appears in the body of the rule or a numerical feature $f \in f_c$, and the leaf contains the probability distribution (density) $\mathcal{D}(f_c)$ that defines the random variable $\texttt{Q}_{t+1}$ in terms of the features f along the path.

The example tree in Figure 1 corresponds to the following set of DDCs (some clauses are omitted):

$$\begin{aligned}
\text{L1} \quad & : \texttt{pos}_\texttt{x}(\texttt{ID})_{t+1} \sim \texttt{gaussian}(\texttt{P} + \texttt{DX}, .02) \leftarrow \\
\text{F1} \quad & : \texttt{P is} \simeq(\texttt{pos}_\texttt{x}(\texttt{ID})_t), \\
\text{F2-Tap} \quad & : \texttt{action}(\texttt{ID}, \texttt{tap}, \texttt{DX}, \texttt{DY})_t.
\end{aligned} \tag{6}$$

$$\begin{aligned}
\text{L2} \quad & : \texttt{pos}_\texttt{x}(\texttt{ID})_{t+1} \sim \texttt{gaussian}(\texttt{P}, .004) \leftarrow \\
\text{F1} \quad & : \texttt{P is} \simeq(\texttt{pos}_\texttt{x}(\texttt{ID})_t), \\
\text{F2-None} : \quad & \texttt{not}(\texttt{action}(\texttt{ID}, \texttt{Action}, \texttt{DX}, \texttt{DY})_t), \\
\text{F3-False} : \quad & \texttt{not}(\texttt{min}\{(\simeq(\texttt{pos}_\texttt{y}(\texttt{ID})_t) - \simeq(\texttt{pos}_\texttt{y}(\texttt{ID}_2)_t)), \\
& \qquad \texttt{ID} \neq \texttt{ID}_2\} < 0.07).
\end{aligned} \tag{7}$$

$$\begin{aligned}
\text{L3} \quad & : \texttt{pos}_\texttt{x}(\texttt{ID})_{t+1} \sim \texttt{gaussian}(\texttt{P} + \texttt{DX}_2, .03) \leftarrow \\
\text{F1} \quad & : \texttt{P is} \simeq(\texttt{pos}_\texttt{x}(\texttt{ID})_t), \\
\text{F2-None} : \quad & \texttt{not}(\texttt{action}(\texttt{ID}, \texttt{Action}, \texttt{DX}, \texttt{DY})_t), \\
\text{F3-True} : \quad & \texttt{min}\{(\simeq(\texttt{pos}_\texttt{y}(\texttt{ID})_t) - \simeq(\texttt{pos}_\texttt{y}(\texttt{ID}_2)_t)), \\
& \qquad \texttt{ID} \neq \texttt{ID}_2\} < 0.07, \\
\text{F4-True} : \quad & \texttt{min}\{(\simeq(\texttt{pos}_\texttt{y}(\texttt{ID})_t) - \simeq(\texttt{pos}_\texttt{y}(\texttt{ID}_2)_t)), \\
& \qquad \texttt{ID} \neq \texttt{ID}_2\} > -0.07, \\
\text{F5-Tap} \quad & : \texttt{action}(\texttt{ID}_2, \texttt{tap}, \texttt{DX}_2, \texttt{DY}_2)_t.
\end{aligned} \tag{8}$$

$$\begin{aligned}
\text{L4} \quad & : \texttt{pos}_\texttt{x}(\texttt{ID})_{t+1} \sim \texttt{gaussian}(\texttt{P}, .004) \leftarrow \\
\text{F1} \quad & : \texttt{P is} \simeq(\texttt{pos}_\texttt{x}(\texttt{ID})_t), \\
\text{F2-None} : \quad & \texttt{not}(\texttt{action}(\texttt{ID}, \texttt{Action}, \texttt{DX}, \texttt{DY})_t), \\
\text{F3-True} : \quad & \texttt{min}\{(\simeq(\texttt{pos}_\texttt{y}(\texttt{ID})_t) - \simeq(\texttt{pos}_\texttt{y}(\texttt{ID}_2)_t)), \\
& \qquad \texttt{ID} \neq \texttt{ID}_2\} < 0.07, \\
\text{F4-False} : \quad & \texttt{not}(\texttt{min}\{(\simeq(\texttt{pos}_\texttt{y}(\texttt{ID})_t) - \simeq(\texttt{pos}_\texttt{y}(\texttt{ID}_2)_t)), \\
& \qquad \texttt{ID} \neq \texttt{ID}_2\} > -0.07).
\end{aligned} \tag{9}$$

The aggregation function $\texttt{min}\{\texttt{U}, \texttt{C}\}$, where $\texttt{U}$ is a mathematical equation, returns the minimum of the $\texttt{U}$'s values obtained by applying all possible substitutions θ for the unbound logvars in $\texttt{U}$ (i.e., logvars not in the target) such that the condition $\texttt{C}\theta$ holds.

In a nutshell, this program models that the next x position is the current position plus the action displacement of a tap action if one occurred (clause (6)). If there is no tap action on the object and if its y position is 7 cm higher (or lower) than that of any other object, then the object's x position will basically not change (clauses (7) and (9)). If the object is close to another object that is tapped, then its position will be affected as well (clause (8)).

One advantage of using relational regression trees to learn DDCs is that they satisfy the mutual exclusiveness property required by DCs, which states that if there are two distributional clauses defining the same continuous random variable, their bodies must be mutually exclusive. This is guaranteed by relational regression trees.

We shall now first introduce the type of features and distributions used by our learning algorithm, and then provide a description of the relational regression tree learning algorithm.

3.2 Relational Features

Each learned DDC will have the form shown in Formula (5) and will correspond to a root-to-leaf path in a relational regression tree. There are two types of internal nodes in such a tree:

- logical conditions which are tests that evaluate to true or false as in standard decision trees. These nodes contribute a condition to $\texttt{body}_t$; and
- continuous features which specify a parameter that appears in the conditional distribution $\mathcal{D}(f_c(\mathbf{s}_t))$ contained in all leaf nodes below this node.

In contrast to standard decision and regression trees, nodes containing a continuous feature do not specify a test. Hence, they do not split the data and only have one child in the tree.

We now define the distributions and features used in our DDC-TL algorithm.

Distributions. Leaf nodes, and hence $\mathcal{D}(f_c(\mathbf{s}_t))$ in Formula (5), contains one of the following distributions:

- A **linear Gaussian distribution** defining $\mathcal{D}(f_c(\mathbf{s}_t)) = \mathcal{N}(\mu, \sigma)$ where $\mu = \alpha + \sum_{f \in f_c(\mathbf{s}_t)} f \cdot \beta_f$ if $\mathtt{Q}_{t+1}$'s range is continuous
- A **softmax** or normalized exponential model defining $\mathcal{D}(f_c(\mathbf{s}_t))$ as a $\mathtt{softmax}(\alpha^{(j)} + \sum_{f \in f_c(\mathbf{s}_t)} f \cdot \beta_f^{(j)})$ if $\mathtt{Q}_{t+1}$ ranges over discrete values j, where the $\beta_f^{(j)}$ are the weights for the features $f \in f_c(\mathbf{s}_t)$ and value j in $\mathtt{Q}_{t+1}$'s range.

The set $f_c(\mathbf{s}_t)$ contains the numeric (continuous) features encountered on the root-to-leaf path. Notice that these distributions degenerate into a $\mathcal{N}(\alpha, \sigma)$ and a probability mass function when there are no numeric features, that is, when $f_c(\mathbf{s}_t) = \emptyset$.

Features. Given the (target) random variable $\mathtt{Q}_{t+1}(\mathtt{A}_0, ..., \mathtt{A}_n)$ with logical variables $A = \{\mathtt{A}_0, ..., \mathtt{A}_n\}$, any term U representing a random variable with logical variables $\mathtt{B}_i$ such that $\forall \mathtt{B}_i \in \mathtt{A}$, can be used as a feature in a distributional clause. For example, given the target random variable $\mathtt{pos}_x(\mathtt{ID})_{t+1}$, any random variable with logical variable ID is an admissible feature in f_c (if continuous) or in $\mathtt{body}_t$ (if discrete), e.g., $\mathtt{pos}_x(\mathtt{ID})_t, \mathtt{pos}_y(\mathtt{ID})_t, \mathtt{color}(\mathtt{ID})$. Continuous features can be discretized and added as conditions in $\mathtt{body}_t$, e.g., $\mathtt{pos}_x(\mathtt{ID})_t > 2$.

If the random variable U has some logical variable $\mathtt{B}_i$ that does not appear in A, then we allow aggregations of the form:

$$\mathtt{agg}(\mathtt{U}, \mathtt{C})$$

where agg is an aggregation function, U is a term representing a random variable, and C is a conjunction of atoms that evaluates to true or false (i.e., a condition). The feature computes the specified aggregate agg over values returned as the result of aggregation over all possible groundings of $\mathtt{U}\theta$ that satisfy $\mathtt{C}\theta$. Sometimes the aggregation might not be defined. For example, if there are no objects satisfying a specific condition C, then aggregation is over an empty set. In case the node represents discretization of an aggregation feature applied to an empty set, the discretized feature is assumed to be *false* as, for example, F3-False in Example 2. If at some node a continuous feature evaluates to $undefined$ during learning, that feature is discarded from the set of candidate features for that node and all its children.

Constructing high-level concepts from low-level numeric sensor data often requires performing mathematical operations (e.g., addition, multiplication, etc.) on the raw data. As our goal is to enable automated discovery of these types of concepts, we allow features involving two atoms $\mathtt{U}_1$ and $\mathtt{U}_2$ that both have numeric ranges. These features have the following form:

$$\mathtt{agg}\{(\mathtt{U}_1 \ \mathtt{op} \ \mathtt{U}_2), \mathtt{C}\}$$

where op is a mathematical operator $(+, -, *, /)$, and agg and C are defined as before.

Note that an admissible aggregation over U needs at least one logical variable $\mathtt{B}_i$ bound with a logical variable in the target $\mathtt{Q}_{t+1}(\mathtt{A}_0, ..., \mathtt{A}_n)$, i.e. $\exists i, j : \mathtt{B}_i = \mathtt{A}_j$.

Example 3. An example of feature that includes an equation is:

$$\mathtt{min}\{(\mathtt{pos}_y(\mathtt{ID})_t - \mathtt{pos}_y(\mathtt{ID}_2)_t), \mathtt{ID} \neq \mathtt{ID}_2\}$$

For a specific grounding of ID bound with the target, this feature calculates the minimum distance in the y-plane between that object and other objects around it. By placing a threshold on this feature, we could distinguish when two objects are close. Thus, these features can represent important concepts that are not explicitly encoded in the raw data.

The condition C can also be a conjunction, for example:

$$\mathtt{min}\{(\mathtt{pos}_y(\mathtt{ID})_t - \mathtt{pos}_y(\mathtt{ID}_2)_t), (\mathtt{ID} \neq \mathtt{ID}_2, \mathtt{action}(\mathtt{ID}_2, \mathtt{push}, \mathtt{DX}, \mathtt{DY})_t)\},$$

which calculates the minimum distance between two distinctive objects if the pushing action is performed on the object $\mathtt{ID}_2$. If the condition is not satisfied, the feature's value is $false$.

Logical Conditions. Features can easily be turned into logical conditions. If a feature f is discrete, the condition $f = v$ (with a value v in the range of f) can serve as an atom in the body of a distributional clause. When a discrete feature is included in a node, there will be one subtree for each possible value v of that node in the decision tree, as usual. If f is continuous, we can discretize it by picking some threshold v in the range of f and building a feature such as $f > v$ or $f \leq v$.

3.3 Learning Relational Regression Trees

Algorithm 1 DDC-TL

```
1: function FEATURESPACE(Q_{t+1}, data, P, op, aggrs)
2:     F_{Q_{t+1}} ← CONSTRUCT(Q_{t+1}, data, P, op, aggrs)
3: end function
4: function GROWTREE(tree, Q_{t+1}, data, score, F_{Q_{t+1}})
5:     f, scoref ← FINDBESTFEATURE(Q_{t+1}, data, F_{Q_{t+1}})
6:     if stopcond(scoref) then
7:         ADDLEAF(tree)
8:     end if
9:     if is_continuous(f) then                          ▷ Continuous range
10:        tree ← ADDNODE(f, tree)
11:        GROWTREE(tree, Q_{t+1}, data, score, F_{Q_{t+1}} \ f)
12:    else
13:        tree ← ADDNODE(f, tree)                       ▷ Discrete range
14:        for each a in domain(f) do
15:            filtered ←filter(data, f, a)  ▷ data such that f = a
16:            GROWTREE(tree[a], Q_{t+1}, filtered, score, F_{Q_{t+1}} \ f)
17:        end for
18:    end if
19: end function
```

Algorithm. For each predicate Q_{t+1} occurring in a state, we learn a relational regression tree that defines the distribution according to a set of clauses of the form defined in Formula (5). Algorithm 1 outlines a top-down procedure for learning the tree. First, the function FEATURESPACE constructs a set of candidate relational features $\mathcal{F}_{Q_{t+1}}$ that can appear in the internal nodes when learning the tree for a target predicate Q_{t+1} (legal features will be described in detail later). Starting from the empty tree, it adds internal nodes as follows. It calls FINDBESTFEATURE which iterates through the set of candidate features and tries using each feature as the current internal node. It calculates the difference in score between the new tree and the old tree using five fold internal cross validation on the training data (the score function is discussed in detail later). If no feature improves the score (that is, stopcond*(scoref)* is true), then a leaf node is added. Otherwise, the algorithm greedily selects the highest scoring feature

f to include as the internal node. The selected feature is removed from the set of candidate features. If the selected feature has a continuous range and never evaluates to undefined in the current branch, the procedure recurses and passes all data to the next node. If the selected feature has a discrete range, one branch is constructed for each value of the discrete feature (yielding a logical condition). The data is divided over the branches according to the feature's value, and the procedure recurses along each branch. When no feature improves the score, the recursion stops, a leaf node is added, and the parameters of the leaf's probability distribution or density function are estimated.

Defining Legal Features. We now describe in more detail the CONSTRUCT function in Algorithm 1, which receives as input a target predicate $\mathtt{Q}_{t+1}$, a set of aggregation functions (*aggrs*), mathematical operators *op*, and $data$ from which the ranges of predicates are extracted. The features of the form $\mathtt{agg}(\mathtt{U}, \mathtt{C})$ are constructed by exhaustively enumerating all combinations of $\mathtt{agg} \in aggrs$, $\mathtt{U}$, and $\mathtt{C}$ that meet the following constraints. The atom $\mathtt{U}$ does not appear in $\mathtt{C}$, and $\mathtt{C}$ is a conjunction with fewer than $k \leq N$ conjuncts. Each conjunct in $\mathtt{C}$ is either a randvar-value test or an (in)equality constraint between two logvars. Each condition in $\mathtt{C}$ is "linked" to $\mathtt{U}$ via a path of shared logvars, that may pass also through other conditions. As aggregation functions, we consider $\mathtt{min}, \mathtt{max}, \mathtt{avg}$, and $\simeq$. Recall, that $\simeq(\mathtt{U})$ simply returns the value of $\mathtt{U}$ in the data and that it is only applicable if there is exactly one grounding substitution of $\mathtt{U}$ for each grounding substitution θ of $\mathtt{C}$ for which $\mathtt{C}\theta$ is true in the data.

We construct candidate features of the form $\mathtt{agg}\{(\mathtt{U}_1\ \mathtt{op}\ \mathtt{U}_2), \mathtt{C}\}$ in an analogous manner. Both $\mathtt{U}_1$ and $\mathtt{U}_2$ must have continuous ranges. We consider $\mathtt{op} \in \{+, -, *, /\}$ and the aforementioned aggregation functions.

For all features that return a real value, we construct two variants: 1) the feature itself to be used as a parameter of the distribution, and 2) discretized version that can be used as a logical condition in the body of a rule. The second alternative is implemented by discretizing the feature into a number of different bins where bin widths are set based on the training data. In the experiments we used five bins.

Example 4. To illustrate how to threshold a continuous value to create a discrete feature, consider again the following feature

$$\mathtt{min}\{(\mathtt{pos_y(ID)_t} - \mathtt{pos_y(ID_2)_t}), \mathtt{ID} \neq \mathtt{ID_2}\}$$

which calculates the minimum distance in the y-plane between two distinct objects. We also consider features with the following template:

$$\mathtt{min}\{(\mathtt{pos_y(ID)_t} - \mathtt{pos_y(ID_2)_t}), \mathtt{ID} \neq \mathtt{ID_2}\} \bowtie \mathtt{Thresh}$$

where $\bowtie \in \{<, >\}$, and $\mathtt{Thresh}$ is a threshold determined from the training data. If $\bowtie$ is $<$ and $\mathtt{Thresh} = 0.07$, we get Feature F3 from Example 2.

Score Function. In Algorithm 1 we use the loglikelihood as the scoring function. The data consists of a set of traces of the form $(\mathbf{s}_0, a_0, \mathbf{s}_1, a_1, ..., \mathbf{s}_{T-1}, a_{T-1}, \mathbf{s}_T)$, and since we are interested in learning the state transition model $p(\mathbf{s}_{t+1}|\mathbf{s}_t, a_t)$ with the Markov assumption, we split the traces into triples of the form $(\mathbf{s}_t, a_t, \mathbf{s}_{t+1})$.

Each triple can be viewed as an interpretation e that assigns values to each of the logical atoms and random variables that appear in the triplet at times $t+1$ and t. The loglikelihood of a triple $e = (\mathbf{s}_t, a_t, \mathbf{s}_{t+1})$ for a DDC program $\mathbb{P}$ is

$$score(\mathbb{P}, e) = logp_{\mathbb{P}}(\mathbf{s}_{t+1}|\mathbf{s}_t, a_t) = \sum_{(h_{t+1} \sim \mathcal{D} \leftarrow body_t) \in \mathbb{P}} \sum_{\theta : body_t\theta \cup h_{t+1}\theta \subseteq e} logp_{\mathcal{D}}(e(h_{t+1}\theta)|e(body_t\theta)),$$

where $e(a)$ denotes the value assignment of random variable a in interpretation e. In this definition we assume that the transition probability factorizes as follows: $p(\mathbf{s}_{t+1}|\mathbf{s}_t, a_t) = \prod_{q_i \in \mathbf{s}_{t+1}} p(q_i|\mathbf{s}_t, a_t)$, that is, every variable in the next state $q_i \in \mathbf{s}_{t+1}$ depends only on the previous state and action.

Scoring a DDC program first requires estimating the parameters for the CPDs $\mathcal{D}(f_c(\mathbf{s}_t))$. For linear Gaussian distributions, parameter learning requires estimating the weight vector for the linear regression model. This can be done via standard maximum likelihood techniques (e.g., ridge regression [1]). Similarly, softmax parameter estimation requires learning the weight vectors for the logistic regression model. We follow standard gradient ascent approach to maximize the loglikelihood [1].

3.4 Related Work

In the literature there are several relational approaches that try to learn a model and use it for planning. However, most approaches only support relational (binary) random variables. For example, Pasula et al. [17, 16] learn noisy indeterministic deictic (NID) rules that define the state in terms of facts (true or false statements). Such representation has been used with the planner PRADA [8]. Similarly, Moldovan et al. [10] learn relational affordance models in multi-object manipulation tasks. In such works, the training data is discretized and the model is relational, without the possibility to include continuous variables. This makes the model abstract but useful low level information is lost. Moreover, in robotics applications the discretization is generally handcrafted. In contrast, our approach tries to learn the best features for the prediction task.

Other approaches, based on RL, try to directly learn the Q-function or the policy without learning the state transition model. Among the works with a relational representation there is an approach for performing policy gradient boosting [7], and approach for imitation learning [12]. Both methods require a regression tree learner, thus our approach can be easily used in such settings.

Few works consider learning hybrid relational domains. One simplified attempt is the one of Moldovan at al. [9], that learns the structure of a Bayesian network to model two object effects and convert it into DDC clauses. However, this fixed conversion might not generalize well on more than two objects. In contrast, our approach directly learns a hybrid relational model, which allows to combine data collected with a different number of objects, and thus provides a better generalization. Moreover, their work considers only simple conditional linear Gaussian models without aggregation or equational features and they evaluate only plans of horizon one.

This work is based on the learner of local models (LLM), an approach for learning the structure of HRDNs [18]. Our approach differs from the LLM algorithm used for HRDNs in several crucial ways. Most importantly, we provide support for automatically learning relationships that involve equational features. This is a key capability in terms of being able to model spatial relationships based on low-level continuous data. Additional differences include that we take into account time (and dynamics) and use a tree-based representation for the distributions/densities. Furthermore, the trees are represented using a set of DDCs, enabling their direct use for planning with HYPE.

Beyond incorporating mathematical equations, our tree representation also differs from existing relational regression trees such as TILDE [2] and its extension towards learning aggregate functions by [21]. Another difference is that we use distributions in the leaves. Finally, there may be features on a path that are not used to "split" the data but will be used as features in the distributions.

4 Experiments

This section empirically evaluates our approach for learning dynamic hybrid transition models. Specifically we want to answer the following questions:

Q1) How accurate are our learned state transition models for prediction tasks?
Q2) How does DDC-TL compare to the propositional hybrid learners?
Q3) What is the performance of the planner that executes our learned models?

To evaluate these questions, we consider a robotics scenario that consists of a table with cubes and a Kinova MICO robot arm that can manipulate the cubes as shown in Figure 2. The learner will only have access to the x, y position of each object. Hence, the learned model must automatically discover important intermediate concepts like `closeTo` by constructing features that build upon the low-level positional data. The learned model is then used for planning.

The learning was performed on Intel(R) Xeon(R) CPU 2.40GHz machines with 128 Gb memory. The planning was performed on a laptop Intel(R) i7 CPU 2.40GHz with 4 Gb memory.

4.1 Experimental Setup and Data Generation

To generate data, a sequence of actions is randomly generated and executed on the objects and their effects (positions) are stored, that is, $(\mathbf{s}_t, a_t, \mathbf{s}_{t+1})$ triples. The state is the x, y position of each object. The number of objects used in the experiments are two, three and four. The actions considered are grasping followed by moving horizontally, grasping followed by moving vertically, pushing and tapping. Each action refers to an object and is parameterized by the displacement, a continuous value. The action might fail, e.g., when the object is out of the range of the robot, or the object is too close to other objects. In such cases, the objects will not move or move differently than expected from a successful action execution. The concept of failure is not explicitly encoded and the learner must learn how to distinguish the different effects according to the state and the action.

4.2 Methodology

We perform the experiments with the following learners:

- **Basic Propositionalization.** We propositionalize the state by constructing one feature for each fact. We denote these with B.
- **Advanced Propositionalization.** We propositionalize the state by using exactly the candidate features considered by DDC-TL. We denote these with A.
- **DDC-TL.** Our approach for learning hybrid dynamic state transition models as introduced in Section 3.
- **DDC-TL-50.** A version of DDC-TL using feature selection when learning the tree. This means that when deciding on the split the learner uses only the top 50 selected features.

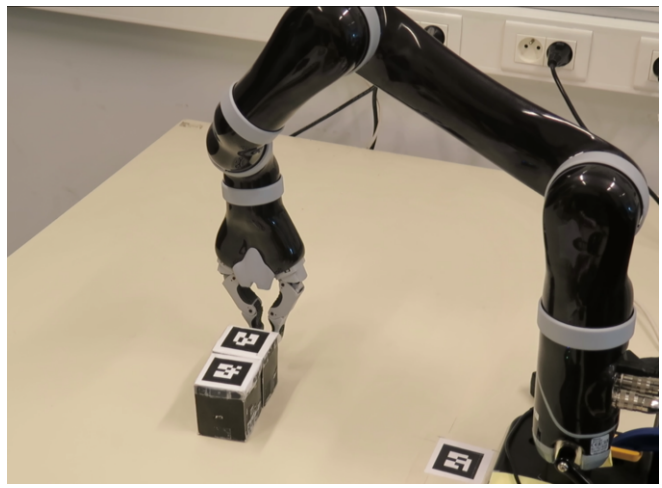

Figure 2. MICO arm performing a left tap action of the object on the right. This action moves the other object as well.

In Basic Propositionalization, each interpretation with n objects is converted in n training examples, where the target is the next position of an object $\texttt{pos(x)}_{t+1}$, and the features are the positions of all objects at time t. In this scheme, we need to ensure that for each training example with target $\texttt{pos(x)}_{t+1}$, the features corresponding to object `x` are in the same position.

Example 5. Given the triplet interpretation $\{\texttt{pos(1)}_{t+1}, \texttt{pos(2)}_{t+1}, \texttt{pos(3)}_{t+1}, \texttt{pos(1)}_{t}, \texttt{pos(2)}_{t}, \texttt{pos(3)}_{t}\}$ (the values and the actions are omitted for compactness), the Basic Propositionalization extracts three training examples:

$$\texttt{pos(1)}_{t+1} \mid \texttt{pos(1)}_{t}, \texttt{pos(2)}_{t}, \texttt{pos(3)}_{t}$$
$$\texttt{pos(2)}_{t+1} \mid \texttt{pos(2)}_{t}, \texttt{pos(1)}_{t}, \texttt{pos(3)}_{t}$$
$$\texttt{pos(3)}_{t+1} \mid \texttt{pos(3)}_{t}, \texttt{pos(2)}_{t}, \texttt{pos(1)}_{t}$$

Note that the feature $\texttt{pos(x)}_{t}$ of the target object `x` is always the first feature selected. The non-target object features do not have a fixed order.

In DDC-TL we use all the available features for learning. In contrast, DDC-TL-50 uses feature selection in each node to overcome some of the known issues of greedy search and control against overfitting. To select a subset of the features, we use Lasso regression, ARD linear regression, and Random Forests for regression. We pick the top 50 features according to the absolute value of the weights in the linear models, and the feature importance metric of Random Forests.

We use the following propositional learners for our experiments: Lasso, regression trees, and gradient boosted regression trees. We consider three different setups: the data only contains two objects, the data only contains three objects, and the data containing two or three objects. We also evaluate the models learned on two and three objects by applying them to test data that contain four objects. In each case, the goal of the model is to predict the x and y positions of each object in the next time step.

4.3 Evaluation Settings

We consider two different evaluation setups. First, we perform 10-fold-cross-validation, and report the average root-mean-squared-error (RMSE) over all held-out folds and learning time. For the

propositional learners, hyper-parameter optimization is performed using a grid search with internal cross-validated on the training set.

Second, we use the DDC-TL models with the best performance on 10-fold-cross-validation to perform planning using HYPE. We use a reward of 100 if the goal is achieved and -1 otherwise, and consider the following goals:

Region This requires moving any object in a specific region (distance around 20 cm from the closest object).

Swap This entails moving an object to the right (or left) side of another object.

The presence of additional objects can make actions fail. Moreover, if an object is out of reach it can be moved only indirectly by acting first on the other objects that can influence its position. For this reason, it is important that the action effects and the interactions between objects are captured in the learned model. For each goal, a set of 15 experiments is performed from different starting positions and the average number of steps is provided.

We assume that every action is applicable (i.e., executable), even when the action does not provide an effect. Obviously, the planner will need to select the actions that are useful to reach the goal. Actions that do not produce movement (i.e., fail) will probably not be selected.

Videos of actions executed by the robot are available at `https://dtai.cs.kuleuven.be/ml/systems/DC/`

4.4 Results and Discussion

Table 1 presents the RMSE of the x and y positions for all approaches, and Table 2 summarizes the learning time. Note that the basic propositionalization approach is not applicable when combining the two and three objects data as this approach only works for a fixed number of input variables and hence a fixed number of objects. DDC-TL learns a model that has an error smaller than most of propositional models tested (Q1). In the two objects case, DDC-TL-50 beats all the propositional models (Q2). On the three objects setting, DDC-TL has the best result. On the dataset combining the data of both two and three objects, DDC-TL and Gradient Boosting have the same performance. Note that Gradient Boosting has access to the same features as DDC-TL and is an ensemble approach which provides it with an advantage.

Moreover, for the propositional models learned with the advanced feature set and DDC-TL, we used the models learned on data about two and three objects data and evaluated them on test data containing a number of objects (four) that was never seen in the training data. The result for this setting is shown in the last column of Table 1. The superior performance in this setting shows an important advantage of our approach, which is that it is able to generalize to new scenarios involving a different number of objects.

Table 2 shows the learning time. We report the time needed to calculate the values of all the features in the feature space and the time to actually learn the model. The calculated features are also used when learning propositional approaches. The proposed DDC-TL is more complex, thus generally slower than propositional learning methods. However, our approach has not been optimized and there are number of ways to improve performance (e.g., use feature selection, use a beam search, etc.).

The models have been also qualitatively tested on some actions. For example, Figure 3 shows the predicted positions of two objects (based on sampling) after a tap action. The visual inspection confirms

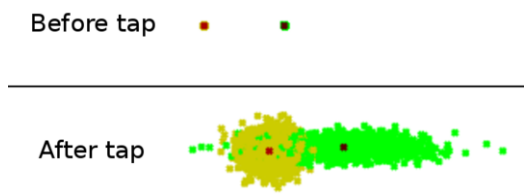

Figure 3. Prediction of tap action with two objects. The object on the left is tapped on the right, this moves the other object. 1000 samples of object positions are shown with the mean in dark color.

that the model captures the interactions between objects. In particular, the model contains relevant features such as the closeness of an object to the object being moved (as shown in Figure 1) and the concept 'not reachable', that is, the object is far away from the arm.

Table 3 presents results about using the learned model with the HYPE planner. For the experiments we use the model learned by DDC-TL with the full feature set. For the **Region** task it has an average success rate of 61% and needs 3.4 steps on average (when it succeeds); whereas for the **Swap** task, it has an average success rate of 62% and needs 3.8 steps on average (Q3). These results confirm that DDC-TL is able to learn a model that is sufficiently accurate for performing simple planning tasks. However, the plans do not always succeed for two reasons. First, there are some scenarios where the objects interactions are not properly modeled. This is expected given that we have a limited amount of training data (196 interpretations for the setup with two objects, and 376 interpretations for the setup with three objects). Second, to keep the planning time reasonable, we limited the number of samples used by the planner to 250. Using more samples could improve the planner's performance.

	RMSE			
	Num. of objects			
Learners	2	3	2 + 3	2 + 3 → 4
DDC-TL	0.029	**0.022**	**0.024**	0.023
DDC-TL-50	**0.027**	0.026	0.026	**0.019**
Lasso-B	0.030	0.026	NA	NA
Regression Tree-B	0.037	0.030	NA	NA
Gradient Boosting-B	0.029	0.025	NA	NA
Lasso-A	0.031	0.026	0.027	0.023
Regression Tree-A	0.037	0.029	0.032	0.030
Gradient Boosting-A	0.029	0.024	**0.024**	0.023

Table 1. The RMSEs of predicting the next x and y positions, based on the learned models, averaged over 10 folds.

5 Conclusions

To the best of our knowledge, this paper is the first approach that can learn a dynamic statistical relational state transition model in a hybrid domain and then apply an MDP planner to the learned model. The central contributions of the paper are: adapting HRDN learning

	Runtime (seconds)		
	Num. of objects		
Learners	2	3	2 + 3
Feature computation	1072.5	2099.5	3186
DDC-TL	3501.5	58322.0	100159.0
DDC-TL-50	1436.0	14846.5	25646.0
Lasso-B	1.8	2.7	NA
Regression Tree-B	0.2	0.7	NA
Gradient Boosting-B	19.9	75.2	NA
Lasso-A	33.7	98.1	160.6
Regression Tree-A	7.2	19.6	26.6
Gradient Boosting-A	636.5	1824.5	2490.5

Table 2. The learning time measured in seconds for learning the models for predicting the next x and y positions averaged over 10 folds. Feature computation is the time DDC-TL needs to calculate the values of all the features in the feature space. These calculated feature values are also used for learning propositional models.

	Avg. # steps (max 5)	Avg. Reward	Success Rate
Region	3.4	56.8	61%
Swap	3.8	57.3	62%

Table 3. Planning results using planner HYPE with the model learned by DDC-TL. 'Avg. # steps' refers to the average number of steps when the plan succeeds.

for learning dynamic distributional clauses, extending the rich relational feature space to include mathematical equations involving the continuous variables, planning with the learned hybrid models, and identifying a potential robotics application for SRL. Empirically, we demonstrated the merits of our approach using scenario involving a real robotic arm. One of the future directions is to perform learning with partial observability or an unknown number of objects. Another future direction is to explore whether the features that we select in one scenario and that are deemed as interesting can be re-used in future scenarios. This might speed up the learning and increase the expressivity and compression of the models as the number of scenarios grows.

Acknowledgements

This work has been supported by the Research Foundation Flanders, by the Chist-Era ReGround project, and by the BOF funds of the KULeuven.

References

[1] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.
[2] Hendrik Blockeel and Luc De Raedt, 'Top-down induction of first-order logical decision trees', *Artificial intelligence*, **101**(1), 285–297, (1998).
[3] Tom Croonenborghs, Jan Ramon, Hendrik Blockeel, and Maurice Bruynooghe, 'Online learning and exploiting relational models in reinforcement learning.', in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 726–731, (2007).
[4] *Probabilistic Inductive Logic Programming — Theory and Applications*, eds., L. De Raedt, P. Frasconi, K. Kersting, and S. Muggleton, volume 4911 of *Lecture Notes in Artificial Intelligence*, Springer, 2008.
[5] B. Gutmann, I. Thon, A. Kimmig, M. Bruynooghe, and L. De Raedt, 'The magic of logical inference in probabilistic programming', *Theory and Practice of Logic Programming*, **11**(4-5), 663–680, (2011).
[6] Saket Joshi, Roni Khardon, Prasad Tadepalli, Alan Fern, and Aswin Raghavan, 'Relational Markov decision processes: Promise and prospects.', in *AAAI Workshop: Statistical Relational Artificial Intelligence*, (2013).
[7] Kristian Kersting and Kurt Driessens, 'Non-parametric policy gradients: A unified treatment of propositional and relational domains', in *Proceedings of the 25th International Conference on Machine Learning*, pp. 456–463, (2008).
[8] Tobias Lang and Marc Toussaint, 'Planning with Noisy Probabilistic Relational Rules', *Journal of Artificial Intelligence Research*, **39**, 1–49, (2010).
[9] Bogdan Moldovan and Luc De Raedt, 'Learning relational affordance models for two-arm robots', in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 2916–2922. IEEE, (2014).
[10] Bogdan Moldovan, Plinio Moreno, Martijn van Otterlo, José Santos-Victor, and Luc De Raedt, 'Learning relational affordance models for robots in multi-object manipulation tasks', in *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4373–4378. IEEE, (2012).
[11] Thibaut Munzer, Bilal Piot, Matthieu Geist, Olivier Pietquin, and Manuel Lopes, 'Inverse reinforcement learning in relational domains', in *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, (2015).
[12] Sriraam Natarajan, Saket Joshi, Prasad Tadepalli, Kristian Kersting, and Jude Shavlik, 'Imitation learning in relational domains: A functional-gradient boosting approach', in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*, volume 22, p. 1414, (2011).
[13] Vien Ngo and Marc Toussaint, 'Model-based relational reinforcement learning when object existence is partially observable', in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 559–567, (2014).
[14] Davide Nitti, Vaishak Belle, and Luc De Raedt, in *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD) 2015, Part II*, volume 9285.
[15] Davide Nitti, Tinne De Laet, and Luc De Raedt, 'A particle filter for hybrid relational domains', in *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2764–2771, (2013).
[16] Hanna Pasula, Luke S Zettlemoyer, and Leslie Pack Kaelbling, 'Learning probabilistic relational planning rules.', in *International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 73–82, (2004).
[17] Hanna M Pasula, Luke S Zettlemoyer, and Leslie Pack Kaelbling, 'Learning symbolic models of stochastic domains', *Journal of Artificial Intelligence Research*, 309–352, (2007).
[18] Irma Ravkic, Jan Ramon, and Jesse Davis, 'Learning relational dependency networks in hybrid domains', *Machine Learning*, **100**(2), 217–254, (2015).
[19] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.
[20] Prasad Tadepalli, Robert Givan, and Kurt Driessens, 'Relational reinforcement learning: An overview', in *Proceedings of the ICML-2004 Workshop on Relational Reinforcement Learning*, pp. 1–9, (2004).
[21] Anneleen Van Assche, Celine Vens, Hendrik Blockeel, and Sašo Džeroski, 'First order random forests: Learning relational classifiers with complex aggregates', *Machine Learning*, **64**(1-3), 149–182, (2006).
[22] Luis Gustavo Rocha Vianna, Leliane N. de Barros, and Scott Sanner, 'Real-time symbolic dynamic programming', in *Proceedings of the 29th Conference on Artificial Intelligence (AAAI)*, pp. 3402–3408, (2015).
[23] Jue Wang and Pedro Domingos, 'Hybrid Markov Logic Networks.', in *Proceedings of the 23rd Conference on Artificial intelligence (AAAI)*, volume 2, pp. 1106–1111, (2008).
[24] M. Wiering and M. van Otterlo, *Reinforcement Learning: State-of-the-Art*, Adaptation, Learning, and Optimization, Springer, 2012.