

Imitation Learning in Relational Domains: A Functional-Gradient Boosting Approach

Sriraam Natarajan and Saket Joshi⁺ and Prasad Tadepalli⁺ and Kristian Kersting[#] and Jude Shavlik^{*}
Wake Forest University School of Medicine, USA ⁺ Oregon State University, USA
[#] Fraunhofer IAIS, Germany, ^{*} University of Wisconsin-Madison, USA

Abstract

Imitation learning refers to the problem of learning how to behave by observing a teacher in action. We consider imitation learning in relational domains, in which there is a varying number of objects and relations among them. In prior work, simple relational policies are learned by viewing imitation learning as supervised learning of a function from states to actions. For propositional worlds, functional gradient methods have been proved to be beneficial. They are simpler to implement than most existing methods, more efficient, more naturally satisfy common constraints on the cost function, and better represent our prior beliefs about the form of the function. Building on recent generalizations of functional gradient boosting to relational representations, we implement a functional gradient boosting approach to imitation learning in relational domains. In particular, given a set of traces from the human teacher, our system learns a policy in the form of a set of relational regression trees that additively approximate the functional gradients. The use of multiple additive trees combined with relational representation allows for learning more expressive policies than what has been done before. We demonstrate the usefulness of our approach in several different domains.

1 Introduction

It is common knowledge that both humans and animals learn new skills by observing others. This problem, which is called *imitation learning*, can be formulated as learning a representation of a policy – a mapping from states to actions – from examples of that policy. Imitation learning has a long history in machine learning and has been studied under a variety of names including learning by observation [Segre and DeJong, 1985], learning from demonstrations [Argall *et al.*, 2009], programming by demonstrations [Calinon, 2009], programming by example [Lieberman, 2000], apprenticeship learning [Ng and Russell, 2000], behavioral cloning [Sammut *et al.*, 1992], learning to act [Khordon, 1999], and some others. One distinguishing feature of imitation learning from ordinary supervised learning is that the examples are not iid, but

follow a meaningful trajectory. Nevertheless, techniques used from supervised learning have been successful for imitation learning [Ratliff *et al.*, 2006]. We follow this tradition and investigate the use of supervised learning methods to learn behavioral policies.

Inverse reinforcement learning is a popular approach to imitation learning studied under the name of apprenticeship learning [Abbeel and Ng, 2004; Ratliff *et al.*, 2006; Neu and Szepesvari, 2007; Syed and Schapire, 2007]. Here one would assume that the observations are generated by a near-optimal policy of a Markov Decision Process (MDP) with an unknown reward function. The approaches usually take the form of learning the reward function of the agent and solving the corresponding MDP. While this approach is justified when there is reason to believe that there is a simple reward function whose expected long-term value can be nearly optimized, in many cases the policy itself is simple enough to learn directly from observation, and the expected long-term value is difficult to optimize. For example, in chess, one could easily learn a few good opening moves by imitation, while optimizing them to improve the probability of winning appears impossible in the foreseeable future.

Our focus is on relational domains where states are naturally described by relations among an indefinite number of objects. Examples include real time strategy games such as Warcraft, regulation of traffic lights, logistics, and a variety of planning domains. Imitation learning in relational domains is not new. Indeed, one of the earliest approaches to imitation learning was explanation-based learning where a complete and correct model of the domain, usually in the form of STRIPS operators [Fikes and Nilsson, 1971], was used *deductively* to learn deterministic macro-operators [Segre and DeJong, 1985; Shavlik and DeJong, 1987]. This deductive approach was generalized to stochastic domains under the name of symbolic dynamic programming (SDP) [Boutilier, 2001; Kersting *et al.*, 2004; Joshi and Khordon, 2008]. However, SDP is computationally quite expensive and leads to highly complex representations of value functions or rules that are expensive to use even in deterministic domains [Minton, 1988]. This problem is somewhat mitigated by performing an approximate or partial SDP, and employing induction over multiple examples to learn an empirically valid approximate value function [Gretton and Thibaux, 2004]. However, all these approaches have the drawback that

they assume that a sound and complete domain model and reward function are available, which makes them inapplicable to most real world domains.

An alternative *inductive* approach is to assume an efficient hypothesis space for the policy function, and learn only policies in this space that are closest to the training trajectories. Positive empirical results have been obtained in this paradigm in several planning domains such as blocks world and logistics with appropriately designed policy languages in both deterministic [Kharon, 1999] and stochastic [Yoon *et al.*, 2002] cases. Theoretical results show that in deterministic domains when there is a target policy in the hypothesis space that is consistent with the training data, it is possible to learn it with a polynomial number of examples with a small error with high probability [Kharon, 1999].

Unfortunately, it is difficult to guarantee realizability of target policies in most real-world problems. Moreover, good policies are likely to be highly complex and not easily representable as relational decision lists [Kharon, 1999; Yoon *et al.*, 2002]. To overcome these limitations, we employ two ideas. First, instead of a deterministic policy, we learn a stochastic policy where the probability of an action given a state is represented by a sum of potential functions. Second, we leverage the recently developed functional-gradient boosting approach to learn a set of regression trees, each of which represents a potential function. The functional gradient approach has been found to give state of the art results in many relational problems [Gutmann and Kersting, 2006; Karwath *et al.*, 2008; Kersting and Driessens, 2008; Natarajan *et al.*, 2011]. Together, these two ideas allow us to overcome the limited representational capacity of earlier approaches, while also giving us an effective learning algorithm.

Indeed, the functional-gradient approach to boosting has already been found to yield excellent results in imitation learning in robotics in propositional setting [Ratliff *et al.*, 2009]. The contributions of this work are to generalize this approach to relational setting, and give empirical results in multiple domains, including the blocksworld, a simple traffic control domain that involves multiple agents, a resource gathering subdomain of a real-time strategy game that involves learning hierarchical relational policies, and the break-away subtask of the Robocup soccer that requires handling of continuous relational features such as distance between objects, angles etc. We show that our approach outperforms both learning a single relational regression tree and performing propositional functional gradient to represent the policy in all domains.

The rest of this paper is organized as follows. Section 2 formally introduces the problem of imitation learning. Section 3 describes our functional gradient approach to imitation learning. Section 4 describes the experimental results in several domains, followed by Section 5 which summarizes the conclusions and future work.

2 Background

An MDP is described by a set of discrete states \mathbf{S} , a set of actions \mathbf{A} , a reward function $r_s(a)$ that describes the expected

immediate reward of action a in state s , and a state transition function $p_{ss'}^a$ that describes the transition probability from state s to state s' under action a . A policy, π , is defined as a mapping from states to actions, and specifies what action to execute in each state. In the learning from imitations setting, we assume that the reward function is not directly obtained from the environment. Our input consists of S, A and supervised trajectories is generated by a Markov policy, and we try to match it using a parameterized policy.

Following Ratliff *et al.* [2009], we assume that the discount factor are absorbed into the transition probabilities and policies are described by $\mu \in \mathbf{G}$ where \mathbf{G} is the space of all state-action frequency counts. We assume a set of features \mathbf{F} that describe the state space of the MDP and the expert chooses the action a_i at any time step i based on the set of feature values $\langle f_i \rangle$ according to some function. For simplicity, we denote the set of features at any particular time step i of the j th trajectory as \mathbf{f}_i^j and we drop j whenever it is fairly clear from the context.

The goal of our algorithm is to learn a policy that suitably mimics the expert. More formally, we assume a set of training instances $\{\langle \mathbf{f}_i^j, a_i \rangle_{i=1}^{m^j}\}_{j=1}^n$ that is provided by the expert. Given these training instances, the goal is to learn a policy μ that is a mapping from \mathbf{f}_i^j to a_i^j for each set of features \mathbf{f}_i^j . The key aspect of our setting is that the individual features are relational i.e., objects and relationships over these objects. The features are denoted in standard logic notation where $p(X)$ denotes the predicate p whose argument is X . The problem of imitation learning given these relational features and expert trajectories can now be posed as a regression problem or a supervised learning problem over these trajectories. Before we present our algorithm in the next section, we introduce the functional-gradient boosting method.

2.1 Functional-Gradient Boosting:

Functional-gradient methods have been used previously to train conditional random fields (CRF) [Dietterich *et al.*, 2004] and their relational extensions (TILDE-CRF) [Gutmann and Kersting, 2006]. Assume that the training examples are of the form (\mathbf{x}_i, y_i) for $i = 1, \dots, N$ and $y_i \in \{1, \dots, K\}$. The goal is to fit a model $P(y|\mathbf{x}) \propto e^{\psi(y, \mathbf{x})}$. The standard method of supervised learning is based on gradient-descent where the learning algorithm starts with initial parameters θ_0 and computes the gradient of the likelihood function. Dietterich *et al.* used a more general approach to train the potential functions based on Friedman's [2001] gradient-tree boosting algorithm where the potential functions are represented by sums of regression trees that are grown stage-wise. Since the stage-wise growth of these regression trees are similar to the Adaboost algorithm [Freund and Schapire, 1996], Friedman called this as *gradient-tree boosting*.

More formally, functional-gradient ascent starts with an initial potential ψ_0 and iteratively adds gradients Δ_i . This is to say that after m iterations, the potential is given by

$$\psi_m = \psi_0 + \Delta_1 + \dots + \Delta_m \quad (1)$$

Here, Δ_m is the functional-gradient at episode m and is

$$\Delta_m = \eta_m \times E_{x,y}[\partial/\partial\psi_{m-1} \log P(y|x; \psi_{m-1})] \quad (2)$$

where η_m is the learning rate. Dietterich et al. [2004] suggested evaluating the gradient at every position in every training example and fitting a regression tree to these derived examples i.e., fit a regression tree h_m on the training examples $[(x_i, y_i), \Delta_m(y_i; x_i)]$. Dietterich et al. [2004] point out that although the fitted function h_m is not exactly the same as the desired Δ_m , it will point in the same direction (assuming that there are enough training examples). So ascent in the direction of h_m will approximate the true functional-gradient. The same idea has later been used to learn relational dependency networks [Natarajan et al., 2011] and policies [Sutton et al., 2000] and their relational extensions [Kersting and Driessens, 2008], relational CRFs [Gutmann and Kersting, 2006] and relational sequences [Karwath et al., 2008].

3 Tree-Boosting for Relational Imitation Learning (TBRIL)

Recall that the goal of this work is to find a policy μ that is captured using the trajectories (i.e., features \mathbf{f}_i^j and actions a_i^j) provided by the expert, i.e., the goal is to determine a policy $\mu = P(a_i | \mathbf{f}_i; \psi) \forall a, i$ where the features are relational. These features could define the objects in the domain (squares in a gridworld, players in robocup, blocks in blocksworld etc.), their relationships (type of objects, teammates in robocup etc.), or temporal relationships (between current state and previous state) or some information about the world (traffic density at a signal, distance to the goal etc.). We assume a functional parametrization over the policy and consider the conditional distribution over actions a_i given the features to be,

$$P(a_i | \mathbf{f}_i; \psi) = e^{\psi(a_i; \mathbf{f}_i)} / \sum_{a'_i} e^{\psi(a'_i; \mathbf{f}_i)}, \forall a_i \in \mathbf{A} \quad (3)$$

where $\psi(a_i; \mathbf{f}_i)$ is the potential function of a_i given the grounding \mathbf{f}_i of the feature predicates at state s_i and the normalization is over all the admissible actions in the current state.

Note that in the functional-gradient presented in Equation 2, the expectation $E_{x,y}[\dots]$ cannot be computed as the joint distribution $P(\mathbf{x}, \mathbf{y})$ is unknown (in our case, y 's are the actions while x 's are the features). Since the joint distribution is unknown, functional-gradient methods treat the data as a surrogate for the joint distribution. Hence, instead of computing the functional-gradient over the potential function, the functional-gradients are computed for each training example (i.e., trajectories):

$$\Delta_m(a_i^j; \mathbf{f}_i^j) = \nabla_{\psi} \sum_j \sum_i \log(P(a_i^j | \mathbf{f}_i^j; \psi)) |_{\psi_{m-1}} \quad (4)$$

These are point-wise gradients for examples $\langle \mathbf{f}_i^j, a_i^j \rangle$ on each state i in each trajectory j conditioned on the potential from the previous iteration (shown as $|_{\psi_{m-1}}$). Now this set of local gradients form a set of training examples for the gradient at stage m . Recall that the main idea in the gradient-tree boosting is to fit a regression-tree on the training examples at each gradient step. In this work, we replace the propositional regression trees with relational regression trees.

Proposition 3.1. *The functional-gradient w.r.t $\psi(a_i^j; \mathbf{f}_i^j)$ of the likelihood for each example $\langle \mathbf{f}_i^j, a_i^j \rangle$ is given by:*

$$\frac{\partial \log P(a_i^j | \mathbf{f}_i^j; \psi)}{\partial \psi(\hat{a}_i^j; \mathbf{f}_i^j)} = I(a_i^j = \hat{a}_i^j | \mathbf{f}_i^j) - P(a_i^j | \mathbf{f}_i^j; \psi) \quad (5)$$

where \hat{a}_i^j is the action observed from the trajectory and I is the indicator function that is 1 if $a_i^j = \hat{a}_i^j$ and 0 otherwise.

The expression is very similar to the one derived in [Dietterich et al., 2004]. The key feature of the above expression is that the functional-gradient at each state of the trajectory is dependent on the observed action \hat{a} . If the example is positive (i.e., it is an action executed by the expert), the gradient $(I - P)$ is positive indicating that the policy should increase the probability of choosing the action. On the contrary if the example is a negative example (i.e., for all other actions), the gradient is negative implying that it will push the probability of choosing the action towards 0.

Following prior work [Gutmann and Kersting, 2006; Natarajan et al., 2011; Kersting and Driessens, 2008], we use *Relational Regression Trees* (RRTs) [Blockeel, 1999] to fit the gradient function at every feature in the training example. These trees upgrade the attribute-value representation used within classical regression trees. In RRTs, the inner nodes (i.e., test nodes) are conjunctions of literals and a variable introduced in some node cannot appear in its right sub-tree (variables are bound along left-tree paths). Each RRT can be viewed as defining several new feature combinations, one corresponding to each path from the root to a leaf. The resulting potential functions from all these different RRTs still have the form of a linear combination of features but the features can be quite complex [Gutmann and Kersting, 2006].

At a fairly high level, the learning of RRT proceeds as follows: The learning algorithm starts with an empty tree and repeatedly searches for the best test for a node according to some splitting criterion such as weighted variance. Next, the examples in the node are split into *success* and *failure* according to the test. For each split, the procedure is recursively applied further obtaining subtrees for the splits. We use weighted variance on the examples as the test criterion. In our method, we use a small depth limit (of at most 3) to terminate the search. In the leaves, the average regression values are computed. We augment RRT learner with aggregation functions such as *count*, *max*, *average* that are used in the standard Statistical Relational Learning (SRL) literature [Getoor and Taskar, 2007] in the inner nodes thus making it possible to learn complex features for a given target. These aggregators are pre-specified and the thresholds of the aggregators are automatically learned from the data. We restrict our aggregators to just the three mentioned earlier. In the case of continuous features such as distance between objects or angles between objects, we discretize them into bins. The data is used to automatically define the bins.

The key idea in this work is to represent the distribution over each action as a set of RRTs on the features. These trees are learned such that at each iteration the new set of RRTs aim to maximize the likelihood of the distributions w.r.t ψ . Hence, when computing $P(a(X) | \mathbf{f}(X))$ for a particular value

of state variable X (say x), each branch in each tree is considered to determine the branches that are satisfied for that particular grounding (x) and their corresponding regression values are added to the potential ψ . For example, X could be a certain block in the blocksworld, a player in Robocup or a square in the gridworld.

Algorithm 1 TBRIL

```

1: function TBOOST(Trajectories  $T$ )
2:   for  $1 \leq k \leq |\mathbf{A}|$  do      ▷ Iterate through each action
3:     for  $1 \leq m \leq M$  do      ▷  $M$  gradient steps
4:        $S_k := \text{GenExamples}(k; T; \Lambda_{m-1}^k)$ 
5:        $\Delta_m(k) := \text{FitRRT}(S_k; L)$   ▷ Gradient
6:        $\Lambda_m^k := \Lambda_{m-1}^k + \Delta_m(k)$   ▷ Update models
7:     end for
8:      $P(A = k | \mathbf{f}) \propto \psi^k$ 
9:   end for
10: return
11: end function
12: function GENEXAMPLES( $k, T, \Lambda$ )
13:    $S := \emptyset$ 
14:   for  $1 \leq j \leq |T|$  do      ▷ Trajectories
15:     for  $1 \leq i \leq |S^j|$  do    ▷ States of trajectory
16:       Compute  $P(\hat{a}_i^j = k | \mathbf{f}_i^j)$  ▷ Probability of user
       action being the current action
17:        $\Delta_m(k; \mathbf{f}_i^j) = I(\hat{a}_i^j = k) - P(\hat{a}_i^j = k | \mathbf{f}_i^j)$ 
18:        $S := S \cup [(\hat{a}_i^j, \mathbf{f}_i^j), \Delta(\hat{a}_i^j; \mathbf{f}_i^j)]$   ▷ Update
       relational regression examples
19:     end for
20:   end for
21: return  $S$                     ▷ Return regression examples
22: end function

```

Our algorithm for imitation learning using functional-gradient boosting is called as *TBRIL* and is presented in Algorithm 1. Algorithm *TBoost* is the main algorithm that iterates over all actions. For each action (k), it generates the examples for our regression tree learner (called using function *FitRRT*) to get the new regression tree and updates its model (Λ_m^k). This is repeated up to a pre-set number of iterations M (typically, $M = 20$). We found empirically that increasing M has no effect on the performance as the example weights nearly become 0 and the regression values in the leaves are close to 0 as well. Note that the after m steps, the current model Λ_m^k will have m regression trees each of which approximates the corresponding gradient for the action k . These regression trees serve as the individual components ($\Delta_m(k)$) of the final potential function.

A key point about our regression trees is that they are not large trees. Generally, in our experiments, we limit the depth of the trees to be 3 and the number of leaves in each tree is restricted to be about 8 (the parameter L in *FitRRT*). The initial potential Λ_0^1 is usually set to capture the uniform distribution in all our experiments. The function *GenExamples* (line 4) is the function that generates the examples for the regression-tree learner. As can be seen, it takes as input the current predicate index (k), the data, and the current model (Λ). It iterates over all the examples and for each example,

computes the gradient based on the observed user action using the expression from Proposition 3.1 .

Our algorithm is attractive for imitation learning due to several reasons: (1) The trees allow for complex features for predicting the user policy as against the standard gradient descent. (2) The use of aggregators such as count,max,min allow for richer features to be used in the model. (3) The tree learner is able to handle continuous domains by automatically discretizing the space based on the data. (4) The regression tree learner allows to include background knowledge (such as most relevant predicates, less relevant predicates etc.) that can guide the search through the space of trees as with any logic-based learner such as *Aleph* [Srinivasan, 2004] or *TILDE* [Blockeel, 1999].

To summarize, our setting extends Ratliff *et al.*'s [2009] propositional “learning to search” setting to the relational case. In a sense, it establishes “learning to search at a lifted level.” On the other hand, it is much simpler than the relational reinforcement learning setting considered e.g. in [Kersting and Driessens, 2008], which is to find a policy that maximizes reward. Whereas Kersting and Driessens’ relational boosting approach receives feedback on the performance only in terms of a reward at the end of an action sequence and in turn faces the problem of estimating the value functions, we learn to directly imitate the expert and side-step the problems of both reward estimation and value optimization. As we will see, this approach can be quite powerful in domains where value optimization is significantly more difficult than policy imitation.

4 Experiments

For our experiments we chose four domains. These domains range from blocksworld which has a long tradition in probabilistic planning, to a grid world domain that has a rich relational hierarchical structure to a multi-agent system for traffic control to Robocup domain that has continuous features. We compare the performance of our *TBRIL* system to a classical RRT learner, *TILDE* [Blockeel, 1999] and to propositional functional-gradient boosting (*PFGB*) and empirically show the superiority of our method. *PFGB* is the method proposed by Dietterich *et al.* [2004] where regression trees are learned instead of relational regression trees. For each domain, we obtained traces of an expert’s policy for training and test sets.

Our performance metric is the area under the precision-recall (PR) curve in the test set. The key reason for employing this metric is due to the fact in several domains it is not possible to evaluate the exact performance of the imitation learner. In these cases, it would suffice to emulate the expert as much as possible. Hence, we used the area under the PR curve to measure the closeness of the learned policy to that of the expert policy. Recent work has indicated PR curves to be a more conservative measure of performance than ROC curves [Davis and Goadrich, 2006]. The goal of our experiments is to answer two specific questions:

1. Is the proposed boosting based approach (TBRIL) better than a non-boosted relational learning approach (i.e., does boosting really help)?

2. Is the proposed approach better than the propositional boosting approach (i.e., are relational models necessary)?

4.1 Blocksworld

This domain (Figure 1) and its variants have a long history in Planning research and has consistently featured as one of the domains at the biennial International Planning Competition (IPC). The world consists of blocks stacked into towers on the table in some configuration. The objective is to start from one configuration and act sequentially to achieve a target, typically specified in terms of certain conditions that have to be true (e.g. block a has to be on top of block b). The action set typically includes actions where the agent can pick up blocks (from other blocks or from the table) and put them down (on other blocks that are clear or on the table).

In our experiments the goal is to reach a target configuration where there is no tower of more than three blocks. Thus, we restrict the action space to only two actions, namely $putdown(x)$ (move block x from its current location to the table) and $noop$. The optimal policy is: $putdown(x)$ where x is a clear block with at least three blocks under it (choosing uniformly at random among all such x s) and $noop$ if there is no such x . Such a policy can be very compactly captured using a relational representation. A propositional representation, on the other hand, must represent all possible towers of four blocks. We encoded this optimal stochastic policy as the expert’s policy and generated training and test examples for randomly generated planning problems. We used the random problem generator from the IPC [Younes *et al.*, 2005].

Training examples were generated with 5 to 15 blocks and test examples were generated with 25 blocks (for which a propositional policy would have to represent 303,600 towers). Hence, there is a need to learn the model at the relational level. Figure 5(a) shows a comparison between the learning curves generated by *TBRIL* and by *TILDE*. We did not plot the performance of *PFGB* because it was consistently below 0.35. The plot shows the superiority of *TBRIL* over *TILDE* to be consistent over the number of training examples. The results were averaged over ten runs.

4.2 Gridworld

In this domain (Figure 2) the agent is in a grid of size 7×7 . Each grid cell has four doors and may contain resources (e.g. gold, food), locations (e.g. enemy castles, granaries, banks) or dragons. The agent’s objective is to *gather* resources and *kill* enemies. To gather a resource it has to be *collected* and *deposited* to a designated location (e.g. gold in a bank, food in a granary). There are two locations for each resource and its storage. There are two kinds of enemies, red and blue. The agent has to *kill* the enemy dragon and *destroy* an enemy castle of the same color as the dragon. The goals and subgoals are specified using a relational hierarchy. The episode ends when the user achieves the highest level goal. The actions that the user can perform are to move in 4 directions, open the 4 doors, pick up, put down and attack.

This domain is inherently relational as there are different objects and relations between them. The “type” of the objects

constrain the goal-subgoal combination. It is also hierarchical since there are goals and subgoals. In our experiments, the domain hierarchies and relations are provided as part of the background knowledge and not learned from trajectories. So the features of the state would include the current highest level goal, the subgoal completed, its type information etc. This domain is challenging as the learner must induce a “relational hierarchical policy” from the trajectories given the domain hierarchy. A propositional representation would require 25 times larger state space and hence at least as many examples. We collected expert trajectories from 3 real users who played the game. The data (100 trajectories) from two users were used for training and the resulting imitated policy was evaluated on the third user’s trajectories (40 trajectories).

As pointed out in [Li *et al.*, 2004], when learning from trajectories, not all the states (or actions) are equally important. For instance, if the current state has a particular door open (as a result of open action in the previous state), the obvious optimal policy is to walk through it. Hence, we measure the performance in only the non-move actions (opening different doors) as the policy for the move actions is very trivial (i.e., walk through an open door). Figure 5(b) shows a comparison between *TBRIL*, *TILDE* and *PFGB*. The area under the PR curve is averaged over the area for each action. As can be seen, *TBRIL* is able to mimic the expert quite reasonably in this domain and outperforms both *TILDE* and *PFGB*. It is a particularly challenging problem as there could be several different paths to navigate the grid and several goal-subgoal combinations.

4.3 Traffic Signal domain

In this domain (Figure 3) the world consists of four traffic lights. Each traffic light is an autonomous agent placed at an intersection, whose objective is to reduce traffic congestion as much as possible. There are four possible modes in which traffic could be flowing - between north and south, between east and west, south to east and north to west, and east to north and west to south. There are four actions available to the traffic light corresponding to the four modes of traffic flow. Each action allows traffic in one mode to proceed while disallowing others. We discretize the density of traffic in each mode to be an integer between 1 and 10 (1 being the lowest density and 10 being the highest).

We employed a simulator that generates traffic at each step in each mode according to a Poisson distribution. Our hand coded expert policy simply allowed the traffic in the densest mode to flow while blocking others. Ties were broken uniformly at random. We gathered data from four traffic lights following the same expert stochastic policy. This domain is an excellent example of “parameter sharing” i.e., the policies can be shared across different agents while learning and can be generalized across the different signals. For *PFGB*, we had to have a separate state space for each light and hence the parameters cannot be directly shared. We ran 5-fold cross validation over all trajectories in this domain. Figure 5(c) shows a comparison between the learning curves generated by *TBRIL* and by *TILDE*. The performance of *PFGB* was never greater than 0.27 and hence we do not show it explicitly in the Figure. As with the blocksworld domain, the results are obtained

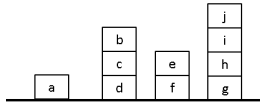


Figure 1: Blocksworld

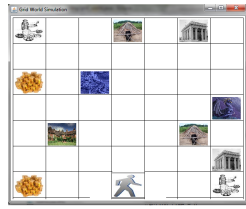


Figure 2: Gridworld

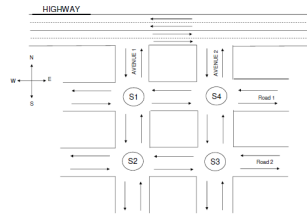


Figure 3: Traffic signal domain

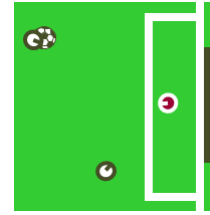


Figure 4: 2-on-1 Breakaway

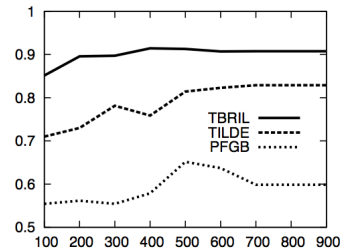
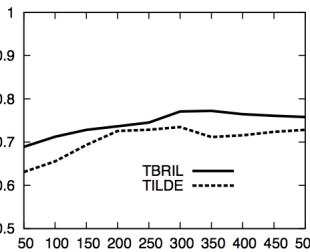
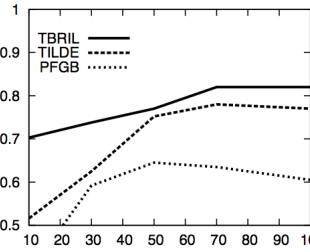
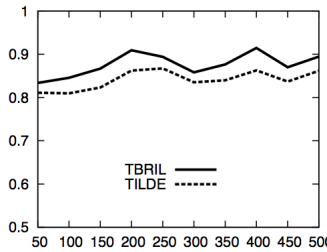


Figure 5: AUC-PR vs. number of trajectories (a) Blocksworld (b) Gridworld (c) Traffic Signal (d) 2-on-1 Breakaway

over 10 runs. Again, *TBRIL* outperforms *TILDE*.

4.4 Robocup

The Robocup domain (Figure 4) [Stone and Sutton, 2001] is a widely used domain for evaluating reinforcement learning algorithms. The objective here is to create a soccer team of robots. Programmed robots then compete with each other. The state space is continuous and there is inherent uncertainty in the effects of actions. An action could, therefore, take the agent to one of a large number of next states. This is again a good domain for imitation learning because it is hard to explain what a good policy should look like but it is relatively easy to obtain expert data. Simplified versions of this game are run through a simulator which we use in this work.

For our experiments we collected data generated through the simulation of an expert policy for the M-on-N BreakAway subdomain of Robocup [Torrey *et al.*, 2007] where M attackers try to score a goal against N defenders. In our experiments, $M = 2$ and $N = 1$. The attacker who has the ball may choose to move (ahead, away, left, or right with respect to the goal center), pass to a teammate, or shoot (at the left, right, or center part of the goal). RoboCup tasks are inherently multi-agent games, but a standard simplification is to have only one learning agent. This agent controls the attacker currently in possession of the ball, switching its focus between attackers as the ball is passed. Attackers without the ball follow simple hand-coded policies that position them to receive passes. The state information specifies the distances and angles between players and the goal post. We used the policies learned from an RL agent from the algorithm presented by Torrey *et al.* [2007] as trajectories after the learner’s policy has stabilized. This is particularly a challenging domain for RL since it involves mainly continuous features such as *distance* and *angle* between objects (players, ball and goal) and *time* which can lead to a large state space. As mentioned, our RRT learner can discretize these features into bins automatically based on

the distribution of the data.

Figure 5(d) shows a comparison between the learning curves generated by *TBRIL*, *TILDE* and *PFGB*. The area under the PR curve is averaged over the area for each of the seven actions for ten runs. Yet again, the plot shows the superiority of *TBRIL*. The results in this problem show that, as the problem complexity and hardness increase, so does the difference between the methods. *TILDE* and *PFGB* converge to a sub-optimal policy for the user while *TBRIL* has a very high precision and recall (close to 1) indicating that it can effectively mimic the user.

In summary, the two questions can be answered affirmatively. The fact that *TBRIL* dominates *TILDE* in all the experiments shows that the boosting approach is indeed superior to the other relational learning method. Similarly the significantly better performance over *PFGB* establishes that it is necessary to model the objects and relations explicitly.

5 Conclusion

To our knowledge this is the first adaptation of a Statistical Relational technique for the problem of learning relational policies from an expert. In that sense, our work can be understood as making a contribution both ways: applying gradient boosting for relational imitation learning and identifying one more potential application of SRL. We demonstrated that the expert can be imitated in a variety of domains with varying characteristics using a relational functional-gradient boosting method. One of the key areas of future work is to generalize across multiple experts. For instance, each expert could possess a different expertise in the domain and can act optimally in certain situations. A key challenge will then be to learn a single model from the different experts. Yet another interesting problem is to combine the expert trajectories with exploration. Finally, it will be useful to verify the generalization capabilities of the algorithm in a transfer learning setting where learning is performed from an expert on an easy problem and transferred to a harder problem.

6 Acknowledgements:

Sriram Natarajan acknowledges the support of Translational Science Institute, Wake Forest University School of Medicine. Saket Joshi is funded by the CI Fellowship (2010), under NSF subaward No. CIF-B-211. Prasad Tadepalli gratefully acknowledges the support of NSF under grant IIS-0964705. Kristian Kersting is supported by Fraunhofer ATTRACT fellowship STREAM and by the EC under contract number FP7-248258-First-MM. Jude Shavlik gratefully acknowledges the support of Defense Advanced Research Projects Agency under DARPA grant HR0011-07-C-0060. Views and conclusions contained in this document are those of the authors and do not necessarily represent the official opinion or policies, either expressed or implied of the US government or of DARPA.

References

- [Abbeel and Ng, 2004] P. Abbeel and A. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.
- [Argall *et al.*, 2009] B. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57:469–483, 2009.
- [Blockeel, 1999] H. Blockeel. Top-down induction of first order logical decision trees. *AI Commun.*, 12(1-2), 1999.
- [Boutilier, 2001] C. Boutilier. Symbolic dynamic programming for first-order mdps. In *IJCAI*, 2001.
- [Calinon, 2009] S. Calinon. *Robot Programming By Demonstration: A probabilistic approach*. EPFL Press, 2009.
- [Davis and Goadrich, 2006] J. Davis and M. Goadrich. The relationship between precision-recall and ROC curves. In *ICML*, 2006.
- [Dietterich *et al.*, 2004] T.G. Dietterich, A. Ashenfelter, and Y. Bulatov. Training conditional random fields via gradient tree boosting. In *ICML*, 2004.
- [Fikes and Nilsson, 1971] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
- [Freund and Schapire, 1996] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *ICML*, 1996.
- [Friedman, 2001] J.H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29, 2001.
- [Getoor and Taskar, 2007] L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning*. MIT Press, 2007.
- [Gretton and Thibaux, 2004] C. Gretton and S. Thibaux. Exploiting first-order regression in inductive policy selection. In *UAI*, 2004.
- [Gutmann and Kersting, 2006] B. Gutmann and K. Kersting. Tilde-CRF: Conditional random fields for logical sequences. In *ECML*, 2006.
- [Joshi and Khardon, 2008] S. Joshi and R. Khardon. Stochastic planning with first order decision diagrams. In *ICAPS*, 2008.
- [Karwath *et al.*, 2008] A. Karwath, K. Kersting, and N. Landwehr. Boosting relational sequence alignments. In *ICDM*, 2008.
- [Kersting and Driessens, 2008] K. Kersting and K. Driessens. Non-parametric policy gradients: A unified treatment of propositional and relational domains. In *ICML*, 2008.
- [Kersting *et al.*, 2004] K. Kersting, M. Van Otterlo, and L. De Raedt. Bellman goes relational. In *ICML*, 2004.
- [Khardon, 1999] R. Khardon. Learning action strategies for planning domains. *Artificial Intelligence*, 113:125–148, 1999.
- [Li *et al.*, 2004] L. Li, V. Bulitko, and R. Greiner. Batch reinforcement learning with state importance. In *ECML - Poster*, 2004.
- [Lieberman, 2000] H. Lieberman. Programming by example (introduction). *Communications of the ACM*, 43:72–74, 2000.
- [Minton, 1988] S. Minton. *Learning Search Control Knowledge: An Explanation-based Approach*. Kluwer Publishers, 1988.
- [Natarajan *et al.*, 2011] S. Natarajan, T. Khot, K. Kersting, B. Guttman, and J. Shavlik. Gradient-based boosting for Statistical Relational Learning: The Relational Dependency Network Case. *MLJ*, 2011.
- [Neu and Szepesvari, 2007] G. Neu and C. Szepesvari. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proceedings of UAI*, pages 295–302, 2007.
- [Ng and Russell, 2000] A. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *ICML*, 2000.
- [Ratliff *et al.*, 2006] N. Ratliff, A. Bagnell, and M. Zinkevich. Maximum margin planning. In *ICML*, 2006.
- [Ratliff *et al.*, 2009] N. Ratliff, D. Silver, and A. Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, pages 25–53, 2009.
- [Sammur *et al.*, 1992] C. Sammur, S. Hurst, D. Kedzier, and D. Michie. Learning to fly. In *ICML*, 1992.
- [Segre and DeJong, 1985] A. Segre and G. DeJong. Explanation-based manipulator learning: Acquisition of planning ability through observation. In *Conf on Robotics and Automation*, 1985.
- [Shavlik and DeJong, 1987] J. Shavlik and G. DeJong. BAGGER: An EBL system that extends and generalizes explanations. In *AAAI*, 1987.
- [Srinivasan, 2004] A. Srinivasan. *The Aleph Manual*, 2004.
- [Stone and Sutton, 2001] P. Stone and R. Sutton. Scaling reinforcement learning toward RoboCup soccer. In *ICML*, 2001.
- [Sutton *et al.*, 2000] R. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 2000.
- [Syed and Schapire, 2007] U. Syed and R. Schapire. A game-theoretic approach to apprenticeship learning. In *NIPS*, 2007.
- [Torrey *et al.*, 2007] L. Torrey, J. Shavlik, T. Walker, and R. Maclin. Relational macros for transfer in reinforcement learning. In *ILP*, 2007.
- [Yoon *et al.*, 2002] S. Yoon, A. Fern, and R. Givan. Inductive policy selection for first-order mdps. In *UAI*, 2002.
- [Younes *et al.*, 2005] H. Younes, M. Littman, D. Weissman, and J. Asmuth. The first probabilistic track of the international planning competition. *JAIR*, 24:851–887, 2005.