Accelarating Imitation Learning in Relational Domains via Transfer by Initialization

Sriraam Natarajan, Phillip Odom, Saket Joshi*, Tushar Khot$^+$, Kristian Kersting**, Prasad Tadepalli*
Wake Forest University, USA
* Oregon State University, USA
$^+$ University of Wisconsin-Madison, USA
** Fraunhofer IAIS, Germany

Abstract

The problem of learning to mimic a human expert/teacher from training trajectories is called Imitation learning. To make the process of teaching easier in this setting, we propose to employ transfer learning (where one learns on a source problem and transfers the knowledge to potentially more complex target problems). We consider multi-relational environments such as real-time strategy games and use functional-gradient boosting to capture and transfer the models learned in these environments. Our experiments demonstrate that the use of relational functional gradients learns a very good initial model from the simple scenario and effectively transfers the knowledge to the more complex scenario thus achieving a jump start, a steeper learning curve and a higher convergence in performance.

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 [1], learning from demonstrations [2], programming by demonstrations [3], programming by example [4], apprenticeship learning [5], behavioral cloning [6], learning to act [7], and some others. One distinguishing feature of imitation learning from ordinary supervised learning is that the examples are not iid, but follow a trajectory. Nevertheless, techniques used from supervised learning have been successful for imitation learning [8]. 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 [9, 8, 10, 11, 12]. 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. We seek to take this approach in this paper.

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 [13], was used *deductively* to learn deterministic macro-operators [1, 14]. This deductive approach was generalized to stochastic domains under the name of symbolic dynamic programming (SDP) [15, 16, 17]. 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 [18]. 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 [19]. However, all these approaches have the drawback that they assume that a sound and complete model and reward function are available, which makes them inapplicable to most real world domains.

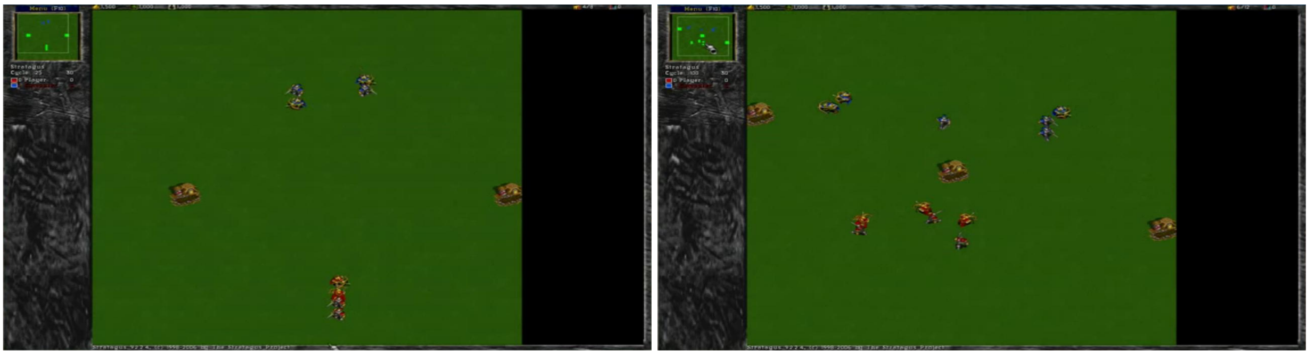

Figure 1: Wargus Scenarios *(left)* The two tower scenario where providing examples is easier. *right* The three tower scenario which is significantly more complicated and requires more training trajectories.

An alternative method was recently proposed [20]. This approach assumes an efficient hypothesis space for the policy function, and learns only policies in this space that are closest to the training trajectories [7, 21]. This approach is based on functional gradient boosting [22] in relational domains where a set of relational regression trees [23] are used to compactly represent a complex relational policy. This approach was demonstrated to be successful in many problems [20].

One of the key assumptions in the proposed approach is that the policies can be generalized across the objects in the domain. While one of the advantages of a logical representation is the generalization capability, it is also quite possible that in several large problems, the optimal policies can vary greatly as the number of the objects in the domains can increase. In such cases, the learner has to be provided with new example trajectories to learn the policies. And since the complexity of the domain has increased, the number of trajectories required for learning can also increase significantly. For instance, consider the two scenarios presented in Figure 1 where the goal is to defend the towers from being destroyed by the enemy units. In the left figure, there are two towers and two enemy and friendly footmen and archers. In the right figure, all the numbers increase by one. As we show empirically, the optimal policies for the two scenarios can be very different. More importantly, the number of trajectories required to converge to the optimal policy is higher in the case of the more complex scenario.

In order to train on such complex scenarios, we propose to employ transfer learning [24, 25, 26, 27, 28] for learning in a (simpler) source problem and then transferring the learned knowledge to a (more complex) target task. More precisely, we aim to employ transfer by initialization [29] where the models learned from the source task are used to initialize the models in the learning task. While Mehta et al. [29] initialized the value functions of a reinforcement learner, we initialize the policies of the target task. Following prior work [20], we perform search through the space of policies using functional gradient boosting but initialize the gradients with the models learned in the source task. Our hypothesis is that this initialization will allow the learner to explore more complex policy spaces that might not have been accessed easily if the search started out with uniform policies. We verify this claim empirically.

In summary, we consider the problem of imitation learning in relational domains where the optimal policies can be significantly different as the number of objects in the domain increases. Note that generalization of policies is still a desired property as the properties of the objects themselves can change across situations with the same number of objects. When the number of objects change, we propose to employ transfer learning by initialization to initialize the gradients in the target task. We evaluate the hypothesis in a real time strategy game and show that we are able to achieve a jump start, faster convergence to a more optimal policy.

The rest of the paper is organized as follows: we introduce the background and the prior work on relational imitation learning next. We then present our transfer algorithm for initialization and evaluate the algorithm on a complex RTS game and conclude the paper by outlining some challenges for 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 $p^a_{ss'}$ 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 imitation learning, 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.

3 Relational Imitation Learning

Following Ratliff et al. [30], 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 jth 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.

In our previous work [20], we employed Functional-Gradient Boosting for learning relational policies. The goal 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, archers or footmen in a real-time strategy game 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,

$$P(a_i|\mathbf{f_i};\psi) = e^{\psi(a_i;\mathbf{f_i})}/\sum_{a_i'} e^{\psi(a_i';\mathbf{f_i})}, \forall a_i \in \mathbf{A} \qquad (1)$$

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. Formally, functional gradient ascent starts with an initial potential ψ_0 and iteratively adds gradients Δ_i. Thus,

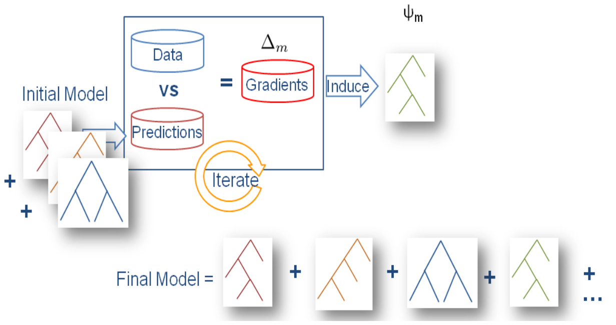

Figure 2: Relational Functional Gradient Boosting. This is similar to the standard FGB where trees are induced in stage-wise manner; the key difference being that the trees are relational regression trees. To compute the predictions, a query, x is applied to each tree in turn, and the numerical values at the leaf reached in each tree are summed to obtain $\psi(x)$.

$$\psi_m = \psi_0 + \Delta_1 + ... + \Delta_m \qquad (2)$$

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

$$\Delta_m = \eta_m \times E_{x,y}[\partial/\partial\psi_{m-1} log\, P(y|x;\psi_{m-1})] \qquad (3)$$

where η_m is the learning rate.

Note that in the functional-gradient presented in Equation 3, the expectation $E_{x,y}[..]$ 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). Hence, functional-gradient methods treat the data as a surrogate for the joint distribution. Instead of computing the functional-gradient over the potential function, the functional-gradients are computed for each training example (i.e., trajectories):

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

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. The main idea in the gradient-tree boosting is to fit a regression-tree on the training examples at each gradient step [31]. In our prior work, we replaced the propositional regression trees with relational regression trees. The idea of functional gradient boosting is presented in Figure 2.

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 $a_i^j = \hat{a}_j^i$ and 0 otherwise. 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 [32, 33, 34], we used *Relational Regression Trees* (RRTs)[23] to fit the gradient function at every feature in the training example [20]. The key idea in our prior 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. Hence, when computing $P(a(X)|f(x))$ 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. For example, X could be a particular unit in Wargus. or a certain block in the blocksworld.

4 Relational Transfer

In this work, we extend the previous work in imitation learning by employing the ideas for *inductive transfer* [24]. While the previous approach was able to achieve generalization in a imitation learning setting, the generalized policies might not be sufficient in some other variations of the problems. For instance, in the scenarios considered in Figure 1, the optimal policies for the three tower defense scenario can be significantly different from the easier task of two tower scenario. Also, since the three tower case is a harder task, as we show empirically, learning in this setting might require more example trajectories from the expert. In such cases, it is easier to transfer the knowledge gained from the two tower scenario to the three tower case for initialization and then improve upon the knowledge by learning in the three tower case. Our hypothesis is that this will enable the learner to: (a) learn a better policy than the one generalized from the two tower scenario and (b) converge to the optimal policy faster in the three tower scenario i.e., from a fewer trajectories when compared to learning with no knowledge.

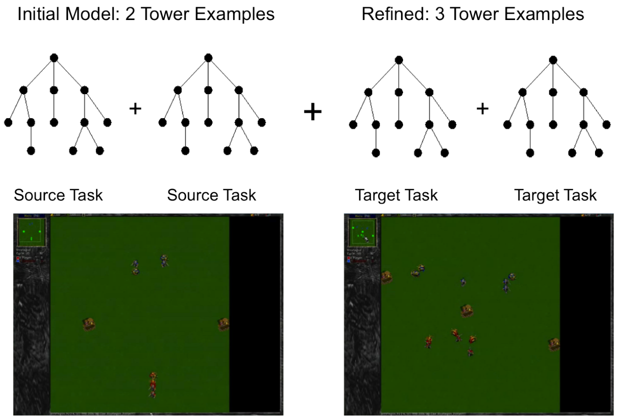

Figure 3: Proposed transfer approach. The key idea is to learn a small set of trees from the source task and use them to initialize the RFGB algorithm for the harder task.

The form of the functional gradients facilitate easy transfer. Since they perform gradient descent in function space, we can initialize the models (ψ_0 in Equation 2) for the three tower scenario with some of the trees learned in the two tower scenario. Conceptually, this is essentially the same as using the result of the first few gradient steps in the source problem while learning in the target problem. After initializing the gradient ascent with the initial set of trees, we propose to learn new set of trees in the target task that build upon the initial model. As mentioned earlier, this initial set of trees for the psi_0.

To perform learning in the target task, the trajectories must be weighted given the initial model. Similar to Equation 5, we compute the value of $I(a_i^j = \hat{a}_i^j|\mathbf{f_i^j}) - P(a_i^j|\mathbf{f_i^j};\psi_0)$ for each action of each trajectory, i.e., the weight of each observed action is the difference between the indicator function of that action and the marginal probability of that action given the initial potential function. Once these weights are computed for the given trajectory, they serve as the examples for learning new set of trees. This idea is presented in Figure 3. As can be seen, first we learn a few set of trees in the source task and then use them to initialize the models in the target task. We then learn a new set of trees in the target task.

Our proposed transfer learning algorithm is presented in Algorithm 1. The function *Transfer* is the main algorithm that takes as input trajectories from the source and the target tasks. The algorithm first calls the *TIL* function (which stands for $Tree-basedImitationLearning$) with an empty potential function (empty set of trees) and the source trajectories. The *TIL* function then learns a set of trees in the source task. The *TIL* function is the same one presented in [20] with the modification that it can use an initial set of trees. 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.

Algorithm 1 Transfer Learning Algorithm

```
function TRANSFER($T_{source}$,$T_{target}$)
    $\mathbf{\Lambda}_s$ = TIL($\{\}$,$T_{source}$)    ▷ Learn with source Trajectories
    $\mathbf{\Lambda}_t$ = TIL($\Lambda_s$,$T_{target}$)    ▷ Use learned models and learn on Target Trajectories return $\mathbf{\Lambda}_t$
end function
function TIL($\mathbf{\Lambda}$, Trajectories $T$)
    $\mathbf{\Lambda}_0 = \mathbf{\Lambda}$
    for $1 \leq k \leq |\ \mathbf{A}|$ do    ▷ Iterate through each action
        for $1 \leq m \leq M$ do    ▷ M gradient steps
            $S_k := GenExamples(k; T; \Lambda_{m-1}^k)$
            $\Delta_m(k) := FitRRT(S_k; L)$    ▷ Gradient
            $\Lambda_m^k := \Lambda_{m-1}^k + \Delta_m(k)$    ▷ Update models
        end for
        $P(A = k|\mathbf{f}) \propto \psi^k$
    end for
return $\mathbf{\Lambda}$
end function
function GENEXAMPLES($k, T, \Lambda$)
    $S := \emptyset$
    for $1 \leq j \leq |T|$ do    ▷ Trajectories
        for $1 \leq i \leq |S^j|$ do    ▷ States of trajectory
            Compute $P(\hat{a}_i^j = k|\mathbf{f_i^j})$    ▷ Probability of user action being the current action
            $\Delta_m(k; \mathbf{f_i^j}) = I(\hat{a}_i^j = k) - P(\hat{a}_i^j = k|\mathbf{f_i^j})$
            $S := S \cup [(\hat{a}_i^j, \mathbf{f_i^j}), \Delta(\hat{a}_i^j; \mathbf{f_i^j}))]$    ▷ Update relational regression examples
        end for
    end for
return $S$    ▷ Return regression examples
end function
```

Once the set of trees have been learned in the source task, a subset of those trees (typically we use 20 trees in our experiments), is then used as the initial model for the target task and the *TIL* function is called with this initial set and the trajectories. The function then returns a new set of trees which are then used for evaluating in the target task. It must be mentioned that when choosing to act in the target task, inference over the actions is performed using *all* the trees (the initial set of source trees plus the target trees). It is easy to see that we cannot ignore the transferred trees since they form the first step of the gradient ascent when learning the policy in the target domain.

5 Experiments

In this section, we present the empirical evaluation of our proposed algorithm on a real-time strategy game. We are particularly interested in the following questions:

Q1: How do the transferred models compare against the models that are generalized using the relational imitation learning algorithm?

Q2: How do the transferred models compare against the models that are learned directly on the target task with no prior models from source task?

5.1 Tower Defense in a RTS game

Experimental Setup: Stratagus is an open-source real-time strategy (RTS) game engine written in C based off the Warcraft series of games. Like all RTS games, it allows multiple agents to be controlled simultaneously in a fully observed setting, making an ideal test bed for imitation learning. A java client was written, revised at Oregon State University[1], to connect to the Stratagus game engine via a socket connection. The client collects all of the game information from the game engine and can issue detailed commands to all units of a player in the game. This client allows for the learned policies to be executed directly in the game environment as opposed to simulation creating more realistic performance metrics. The client is written so that player strategies are java classes that can be called by the client.

The setting in which transfer is being tested is the tower defense scenario shown in Figure 1. The map used for the experiments consisted of 6 x 6 grid world. Our scenarios consist of two opposing teams-one attacking, one defending- each with two kinds of units. Footman have more health but must be close to an enemy to attack them while ranged archers are easily killed but can attack from a distance. Towers exist on the map in set locations. The defending team must prevent the attacking team from destroying the towers while the attacking team must destroy as many towers as they can. The defending team must divide its units

[1] http://beaversource.oregonstate.edu/projects/stratagusai

among the various towers to prevent one tower from falling while another is being saved. This dynamic creates complex policies.

We used the following features to describe the state: the strength (high, medium, low) and location of all friendly units, the type (footman, archer) of all units in the game, which tower each enemy unit is currently attacking, and the enemy unit that all friendly units were attacking the previous state. Friendly units are unaware of the strength of enemy units or their exact location. The full set of information given at every state is included in Table 1. The actions available to the friendly units are to move to a location and attack a particular unit. The nature of the objects and the relationships between the objects in this game naturally allow for a relational representation. Each type of unit (footman or archers) shares traits such as their attacking range and their total health so certain policy rules will naturally apply to all units of that type.

Predicates	Description
friendlyobject	The type of defending unit
enemyobject	The type of attacking unit
dead	Enemy unit that is dead
locationId	Location of friendly unit
strength	Strength (hit points) of friendly unit
distance	Distance of a friendly unit to a tower
enemyattower	Tower that enemy unit is attacking
attacking	Enemy unit that a friendly unit was attacking in the previous state

Table 1: Features that describe the state in the two scenarios. We omit the arguments of the predicates for brevity.

As mentioned earlier, the goal of this experiment is to learn to protect two towers in a source scenario and transfer the learned knowledge to a target scenario. The attacking team's strategy is as follows: At the start of each game, each member of the attacking team randomly selects a tower to attack. However, if approached by an enemy archer or footmen, they will change their target to eliminate the opposing player's offensive units. After destroying one tower, they will randomly select another tower to attack until there are no towers left and the game ends. The goal of the game is to defend the towers. The game ends if either the enemy team manages to destroy all of the towers or the friendly team kills all of the enemy units. The number of friendly and attacking units vary between the source and target scenarios. In the source scenario, there are two footmen and two archers while in the target scenario, there are three footmen and three archers. In both the scenarios, the towers cannot defend themselves.

Results: We used two performance metrics that are based on the number of towers saved. The 3-tower win percentage is the percentage of games in which all three towers were saved. This is a difficult task because it requires the friendly team to defend all three towers simultaneously; else one tower may fall while they are all defending the others. The 1-tower win percentage is the easier metric, only requiring 1 of the 3 towers to be saved. We used randomly selected samples of 30, 50, 70, 100, and 150 3-tower games from a pool of approximately 1000 expert games for training in the target scenario. For the source scenario, we used 100 games for training. This experiment was repeated 10 times and the average percentage of winnings games were computed.

We used three models for evaluation – the transfer model, the non-transferred model which learns only on the target task and the generalization model which learns only on the source and not the target task. We learned 20 trees on the source task and learned a further 20 on the target task. Hence, the final transfer model had 40 trees while the other two models had 20 trees each.

The results are presented in Figures 4 corresponding to saving at least one tower and three towers respectively. As can be seen from the figures, the transfer model is superior to both the generalization and the pure imitation learner in the twin tasks. Also, the results follow the original transfer learning goals of jump start, steeper learning curve and better convergence. It is fairly clear that in defending at least one tower, the use of the initial models from the source scenario provides a bigger jump start than in defending all towers. On the other hand, when saving all towers, the jump start is not fairly high but the learning curve is steeper. It appears that the use of initial models allow for the learner in the target scenario to explore a space of policies that might not have been otherwise reached from an uniform policy. Similarly, the difference between transfer and non-transfer models is higher in the one tower case than saving all the towers though the difference in statistically significant in both the cases. Also, it must be mentioned that even when the non-transferred models were provided with a lot more trajectories, it is not able to match the performance of the transferred model. This suggests that using initial policies in some cases is more useful than obtaining more expert trajectories. It would be interesting to evaluate these results in other domains as well.

In summary, our experiments answer both the questions affirmatively in that the transfer models dominate both the original imitation learning models in the two tower and three tower cases.

6 Discussion and Conclusion

In this paper we address the issue of sample complexity in imitation learning settings. In scenarios where the expert's time is expensive/valuable and we have access to only a few training examples from the expert our approach is to divide the expert's time between simple (smaller domain size) and harder (larger domain size) problems. Although policies induced from the simpler problem training instances can be employed to solve the larger domain via relational generalization, in scenarios we provide (such as Wargus) this does not translate to better performance. We have presented transfer learning by using the simpler policy as our initial models and building an updatable relational model by learning from the harder examples, as a solution to this problem. We observe not only a superior performance to generalization but also a drastic reduction in the sample complexity as compared with the naive method of directly inducing a model on the complex examples.

Imitation learning encounters two major problem when dealing with large state spaces. First, assuming a tabular representation of the policy to be learned is likely to exceed memory due to the large state and action space. Second, it can only make use of a limited amount of expert traces compared to the excessive amount of possible traces. The implicit feedback gained by the expert's traces on the best action to take in a state might be so sparse that a well-generalizing policy will only be discovered slowly.

The first problem can be solved using relational imitation learning (RIL) for structural domains. However, the problem of sparse feedback has not been addressed by RIL yet. For relational reinforcement learning, there is a compelling and simple solution to this problem: inject traces of execution of a reasonable policy for the task at hand [35]. Unfortunately, this does not work for imitation learning. The input consists already of traces of execution of a reasonable policy, namely the policy of the expert. Thus, we do not gain anything despite enlarging the training set as can be seen from our results. The non-transfer model seemed to have converged to a inferior policy.

To overcome this problem, we intuitively propose to inject traces of a policy of a reasonably well related task. Specifically, we directly inject the complete "related" policy into a functional gradient boosting approach to RIL. This appears to be an interesting result in that sometimes prior policies have a better impact on the performance compared to more trajectories. Our immediate challenge is to validate this hypothesis on other more complex domains. Another interesting direction is the possibility of employing active learning methods for extracting the best complex examples given the initial model thereby further improving on the performance of transfer.

7 Acknowledgement

Sriraam Natarajan and Tushar Khot gratefully acknowledge the support of Defense Advanced Research Projects Agency (DARPA) Machine Reading Program under Air Force Research Laboratory (AFRL) prime contract no. FA8750-09-C-0181. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the DARPA, AFRL, or the US government. Phillip Odom was supported by the CS department at Wake Forest University. Prasad Tadepalli acknowledges the support of ONR grant N000141110106. Saket Joshi gratefully acknowledges the support of NSF CI Fellows award CIF-C-211 Kristian Kersting was supported by the Fraunhofer ATTRACT fellowship STREAM and by the European Commission under contract number FP7-248258-First-MM.

References

[1] A. Segre and G. DeJong. Explanation-based manipulator learning: Acquisition of planning ability through observation. In *Conf on Robotics and Automation*, 1985.

[2] B. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57:469–483, 2009.

[3] S. Calinon. *Robot Programming By Demonstration: A probabilistic approach.* EPFL Press, 2009.

[4] H. Lieberman. Programming by example (introduction). *Communications of the ACM*, 43:72–74, 2000.

[5] A. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *ICML*, 2000.

[6] C. Sammut, S. Hurst, D. Kedzier, and D. Michie. Learning to fly. In *ICML*, 1992.

[7] R. Khardon. Learning action strategies for planning domains. *Artificial Intelligence*, 113:125–148, 1999.

[8] N. Ratliff, A. Bagnell, and M. Zinkevich. Maximum margin planning. In *ICML*, 2006.

[9] P. Abbeel and A. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.

[10] G. Neu and C. Szepesvari. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Proceedings of UAI*, pages 295–302, 2007.

[11] U. Syed and R. Schapire. A game-theoretic approach to apprenticeship learning. In *NIPS*, 2007.

[12] M. Babes, V. Marivate, K. Subramanian, and M. Littman. Apprenticeship learning about multiple intentions. In *ICML*, pages 897–904, 2011.

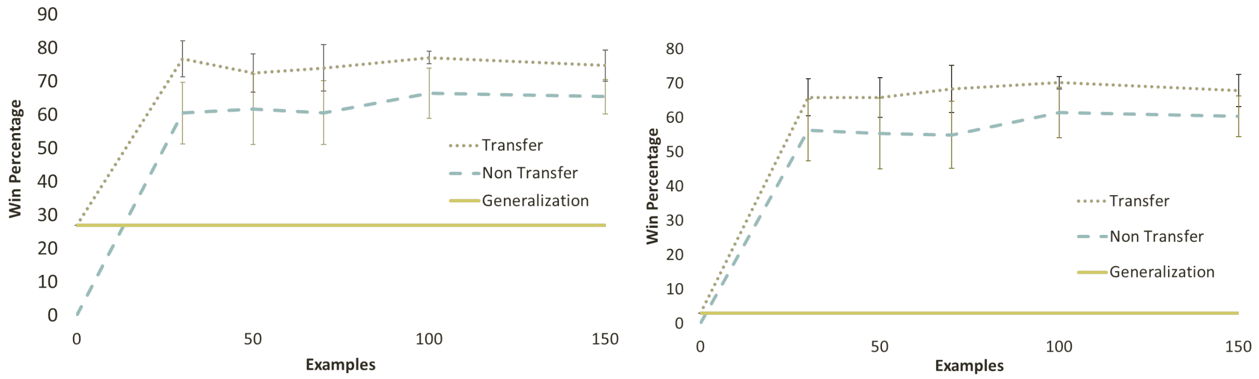

Figure 4: Results of saving at least one tower (left) and saving all three towers(right). The transferred models dominate the one learned from two towers (generalization) and the one learned on three towers without an initial policy (non-transfer)

[13] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[14] J. Shavlik and G. DeJong. BAGGER: An EBL system that extends and generalizes explanations. In *AAAI*, 1987.

[15] C. Boutilier. Symbolic dynamic programming for first-order mdps. In *IJCAI*, 2001.

[16] K. Kersting, M. Van Otterlo, and L. De Raedt. Bellman goes relational. In *In ICML*, 2004.

[17] S. Joshi and R. Khardon. Stochastic planning with first order decision diagrams. In *ICAPS*, 2008.

[18] S. Minton. *Learning Search Control Knowledge: An Explanation-based Approach*. Kluwer Publishers, 1988.

[19] C. Gretton and S. Thibaux. Exploiting first-order regression in inductive policy selection. In *UAI*, 2004.

[20] S. Natarajan, S. Joshi, P. Tadepalli, K. Kersting, and J. Shavlik. Imitation learning in relational domains: A functional-gradient boosting approach. In *IJCAI*, 2011.

[21] S. Yoon, A. Fern, and R. Givan. Inductive policy selection for first-order mdps. In *UAI*, 2002.

[22] J.H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29, 2001.

[23] H. Blockeel. Top-down induction of first order logical decision trees. *AI Commun.*, 12(1-2), 1999.

[24] S. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22:1345–1359, 2010.

[25] L. Torrey and J. Shavlik. Transfer learning, 2008.

[26] M. Taylor. *Transfer in Reinforcement Learning Domains*, volume 216 of *Studies in Computational Intelligence*. Springer, 2009.

[27] Stephan Al-Zubi and Gerald Sommer. Imitation learning and transferring of human movement and hand grasping to adapt to environment changes. In *Human Motion*, volume 36 of *Computational Imaging and Vision*, pages 435–452. 2008.

[28] B. Banerjee and P. Stone. General game learning using knowledge transfer. In *IJCAI*, pages 672–677, 2007.

[29] N. Mehta, S. Natarajan, P. Tadepalli, and A. Fern. Transfer in variable-reward hierarchical reinforcement learning. *Machine Learning*, 73(3):289–312, 2008.

[30] N. Ratliff, D. Silver, and A. Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Autonomous Robots*, pages 25–53, 2009.

[31] T.G. Dietterich, A. Ashenfelter, and Y. Bulatov. Training conditional random fields via gradient tree boosting. In *ICML*, 2004.

[32] B. Gutmann and K. Kersting. TildeCRF: Conditional random fields for logical sequences. In *ECML*, 2006.

[33] S. Natarajan, T. Khot, K. Kersting, B. Guttmann, and J. Shavlik. Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning*, 2012.

[34] K. Kersting and K. Driessens. Non–parametric policy gradients: A unified treatment of propositional and relational domains. In *ICML*, 2008.

[35] K. Driessens and S. Dzeroski. Integrating guidance into relational reinforcement learning. *Machine Learning*, 57(3):271–304, 2004.