Integrating Human Instructions and Reinforcement Learners : An SRL Approach

Pradyot Korupolu V N
Computer Science Dept.
IIT Madras
India

S S Manimaran
Computer Science Dept.
IIT Madras
India

Balaraman Ravindran
Computer Science Dept.
IIT Madras
India

Sriraam Natarajan
Translational Science Institute
Wake Forest University
USA

Abstract

As robots and other intelligent systems move out from tailored environments and enter the real world they would have to adapt quickly and learn from their surroundings. Like humans, they would have to learn from various sources - experience, peers, teachers, trial-and-error etc. In earlier work, we proposed a framework that facilitates a teacher-student like relationship between humans and robots, wherein the human-robot interactions were restricted to certain classes of simple instructions. In this work we extend this framework to handle multiple interpretations of instructions. Instead of the human teacher taking pains to carefully instructing the learning agent (robot) in a manner that it can understand, the teacher instructs in his own natural style and the agent appropriately interprets them. The agent generalizes these instructions by building relational models of tasks using Statistical Relational Learning techniques. In addition to learning from teachers, the agent learns on its own using Reinforcement Learning (RL). The agent chooses the most appropriate among these knowledge models using certain confidence measures. The generalization power of SRL makes learning and transfer of knowledge to related objects easy. The trial-and-error nature of RL enables the agent to improve upon what it has been taught and in some cases unlearn human teaching.

1 Introduction

One of the long term goals of AI is to build adaptive systems that can interact and learn from humans. Reinforcement Learning [25] (RL) provides a natural framework for adaptive learning by receiving feedback from the environment in the form of rewards based on which it learns an action preference for each state that it visits. Standard RL methods use an atomic, propositional or propositional function representation to capture the current state and possible actions of the learner. Although this type of representation suffices for many applications as demonstrated by successful RL applications [27, 6], in many real world problems such as robotics, real-time strategy games, logistics and a variety of planning domains etc., there is a need for relational representations. In such domains, achieving abstraction or generalization by standard function approximators can pose significant difficulties in terms of representation and requires a large number of training examples.

On the other hand, these domains are naturally described by relations among an indefinite number of objects. Recently there have been algorithms proposed that directly operate on these relational domains [19, 20, 23, 30]. Sanner and Boutilier [23] used situation calculus to capture the dynamics and proposed a linear programming formulation for solving the action selection problem. Wang et al. [30] on the other hand, employed First Order Decision Diagrams (FODDs) to capture the domain dynamics and the reward and value functions. Action selection was then posed as manipulation of these diagrams using several arithmetic operators on FODDs that they defined. While these methods are attractive, they still suffer from the need to explore extensively in the environment in order to collect examples for learning. It is natural in many domains that there is an availability of a human expert who can provide guidance or instructions to the learner. In these RRL systems, the expert can be utilized to design the reward functions and even provide the models of the environment. This is the approach taken by Thomaz et al. [28] where the human teachers provide rewards for actions chosen by the learner. This is a cumbersome process in large domains compared to the possibility of the ex-

pert providing examples and direct instructions to the learner. Recently, a policy-gradient approach to learning in relational domains has been proposed that can use a small number of expert trajectories to initialize the policy [11]. While this method can use the initial trajectories, there is no interaction with the human beyond the initial model.

AI has a long history of methods that use expert trajectories for learning to act (*Imitation Learning*). Imitation learning has been studied under a variety of names including learning by observation [24], learning from demonstrations [2], programming by demonstrations [7], programming by example [15], apprenticeship learning [18], behavioral cloning [22], learning to act [12], 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 [21]. Recently, imitation learning has been posed in highly stochastic yet relational domains using a statistical relational learning (SRL) [10] formulation and has been solved using functional-gradient boosting [16]. Lately there has been a surge in interest in developing apprenticeship learning methods (also called as inverse reinforcement learning) [1, 21, 17, 26]. 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. All these methods discussed assume that the expert is optimal (or at least near-optimal) and the aim of these methods is to generalize from the example trajectories and emulate the expert in several scenarios.

We take a different approach to interacting and learning from humans. In our framework, the human expert provides *instructions* – suggestions for actions selection or specification of relevant features in learning a policy. This reduces the effort on the expert by not providing examples for every possible scenario. Instead, the expert can interact with the system and provide suggestions of actions or features periodically. As mentioned earlier, we are interested in relational domains where states are described using objects and relations. More precisely, we consider a sorter robotics setting where there are balls and baskets in room. In such a setting the advice provided by the expert concerns the ground objects or actions. For example, the expert could gesture to the red ball in the hand of the robot to be dropped in a red basket. While the expert gestures at the red basket, there is an implicit generalization in that he/she refers to dropping the ball in a basket of the same color. Hence, there is a need to generalize the gestures of the expert to objects in the domain. Also, the domain is noisy due to sensory noise of robot, stochastic effects of actions etc. To generalize in this stochastic environment, we propose the use of SRL to capture and reason with the instructions of the user. A related approach has been taken by Branavan et al. [5] where natural language instructions are mapped into rewards for the learner. We on the other hand, do not treat them as rewards but as direct relevance statements on states and actions.

The chief contribution of our work is the ability to handle multiple interpretations of instructions. As an example to realize the necessity for such a framework, consider a person searching for a misplaced key to his cupboard and one of his friends points to a heavy paper weight on a table nearby. The person will either interpret the friend's instruction as *break the lock with the paper weight* or as *search for the key near the paper weight*. As can be seen, interpretation of an instruction greatly varies the result of the task at hand. In this paper, we propose a novel framework that enables a learner to handle such instructions and choose the best possible interpretation based on specific utility measures. An additional contribution of our work is treating the problem of generalization in sequential decision problems as a teacher-student setup where the student retains the capability to unlearn what the teacher has taught.

In the next section, we introduce the Sorter Domain, a ball sorting domain, that we test our approach on. In the later sections we introduce the background to the problem, explain the different types of instructions we use and explain the algorithm in detail.

2 Sorter Domain

The Sorter Domain (Fig 1) consists of 3 objects and 3 baskets. The task of a sorter robot is to drop the objects into the basket with the same color i.e., a red object should be dropped into a red basket. The colors of the objects and baskets are chosen randomly such that every object has at least one basket with the same color. An episode is completed when every object has been dropped into a basket. Once an object is dropped into a basket it cannot be picked up anymore. Dropping an object into the correct basket is rewarded +50, a wrong match is rewarded -50 and any other action is rewarded -1. Each action takes a finite time to complete execution. An object or basket occupies 1 of 6 fixed positions.

Figure 1: The Sorter Domain

3 Markov Decision Process

Sequential problems are generally modeled as *Markov Decision Processes* (MDPs) in Reinforcement Learning. The MDP framework forms the basis of our definition of instructions.

A MDP is a tuple $\langle S, A, \psi, P, R \rangle$, where S is a finite set of states, A is a finite set of actions, $\psi \subseteq SXA$ is the set of admissible state-action pairs, $P : \psi \rightarrow [0, 1]$ is the transition probability function with $P(s, a, s')$ being the probability of transition from state s to state s' by performing action a. $R : \psi \rightarrow \mathbb{R}$ is the expected reward function with $R(s, a)$ being the expected reward for performing action a in state s (this sum is known as return). $A_s = \{a|(s, a) \in \psi\} \subseteq A$ be the set of actions admissible in state s. We assume that A_s is non-empty for all $s \in S$. $\pi : \psi \rightarrow [0, 1]$ is a stochastic policy, such that $\forall s \in S \sum_{a \in A_s} \pi(s, a) = 1$. $\forall (s, a) \in \psi$, $\pi(s, a)$ gives the probability of executing action a in state s.

The value of a state-action pair conditioned on policy π, $Q_\pi(s, a)$, is the expected value of a sum of discounted future rewards of taking action a, stating from state s, and following policy π thereafter. The optimal value functions assign to each state-action pair, the highest expected return achievable by any policy. A policy whose value function is optimal is an optimal policy π^*. Conversely, for any stationary MDP, any policy greedy with respect to the optimal value functions must be an optimal policy : $\pi^*(s) = argmax_a Q^*(s, a) \forall s \in S$, where $Q^*(s, a)$ is the optimal value function. If the RL agent knew the MDP, it could be able to compute the optimal value function, and from it extract the optimal policy. However, in the regular setting, the agent is only aware of ψ, the state-action space and must learn Q^* by exploring. The Q-learning algorithm learns the optimal value function by updating its current estimate,$Q_k(s, a)$, of $Q^*(s, a)$ using this simple update [32],

$$Q_{k+1}(s, a) = Q_k(s, a) + \alpha \big[r + \gamma max_{a'} Q_k(s', a') - Q_k(s, a) \big] \quad (1)$$

α is the learning rate of the algorithm, $\gamma \in [0, 1]$ is the discounting factor and a' is the greedy action in s'.

An option (o) is a temporally extended action [3] or a policy fragment. Q-learning applies to options too and is referred to as SMDP (Semi Markov Decision Process) Q-learning [3].

$$Q_{k+1}(s, o) = (1 - \alpha) Q_k(s, o) + \alpha \big[R + \gamma^\tau max_{o'} Q_k(s', o') \big] \quad (2)$$

where R is the sum of the time discounted rewards accumulated while executing the option and τ is the time taken for execution.

A state s in the Sorter Domain is given by the features {*isCarrying, Carrying(O), Color(O1), inBasket(O1), Color(O2), inBasket(O2), Color(O3), inBasket(O3), Color(B1), Color(B2), Color(B3), Botat(X)*}. *isCarrying* - an indicator variable that is true if the robot is carrying an object, $Carrying(O)$ - object in robot's arm, $Color()$ - color of object/basket, $inBasket(O)$ - indicates if object O has been dropped into a basket and $Botat(X)$ - robot's current location such that $X = \{O1, O2, O3, B1, B2, B3\}$. The set of options are $O =$ {*Pickup, Drop, Goto(O1), Goto(O2), Goto(O3), Goto(B1), Goto(B2), Goto(B3)*}.

In real world robots, options such as *Goto(.)* are generally implemented using planners making them deterministic in terms of the next state reached on executing the option. But the time taken to execute the plan (option) can vary depending on the time taken to navigate to the target making them stochastic in τ. The option *Goto(O2)* will take different times to execute depending on the robot's current position. The sensor's noise is a chief contributor to this stochasticity as it greatly affects the robot's ability to observe the world and localize itself. SMDP Q-learning helps by providing a learning framework that can accommodate the stochasticity mentioned above. By abstracting out options as plans we also free ourselves from the hassles of a domain being continuous. In essence, we work with a discrete state space now.

4 Instructions

We define *Instructions* as inviolate external inputs to the RL agent which affect its decision making or direct its exploration or alter its belief on a policy. For example, an agent learning to wash cups can be instructed to *use the scrubber*.

Instructions have also been used to specify regions of the state space or objects of interest. In [8], a human directs Sonja, a game player, using instructions. For example, he instructs Sonja to *Get the top most amulet from that room*. This instruction binds the region of search and the object of interest yet withholding direct

information about solving the task.

4.1 Mathematical Formulation

This section introduces a mathematical formulation for instructions. Representing the policy $\pi(s,a)$ as shown below makes it easy to understand the two types of instructions that we use in this paper :

$$\pi(s,a) = G(\Phi(s),a) \tag{3}$$

where $\Phi(s)$ models operations on the state space. $G(.)$ is a mapping from $(\Phi(s),a)$ to the real interval $[0,1]$. $\Phi(s)$ can either model mappings to a subspace of the current state space or model projections of the state s onto a subset of features.

4.1.1 π-Instructions

Instructions of this type are in the form of action or option to be performed at the current state : $I_\pi(s) = a$, where $a \in A_s$. For example, in the Sorter Domain, the instruction *"Goto(O3)"* is a π-Instruction. If policy models are built over such instructions, their effect on the policy would be $\pi(s,a) \simeq 1$.

4.1.2 Φ-Instructions

Instructions of this type are given as structured functions [33] denoted by I_Φ. In this paper, we restrict ourselves to using only projections, a class of structured functions. Such an instruction, I_Φ would be captured by $\Phi(s)$ as $\rho_{D'}(s), D' \subseteq D$. D is the set of features representing the state set S and $\rho_{D'}$ is the projection operation. $D' \subseteq D$ captures the possibility that some features in a state representation are inconsequential in learning the optimal policy. For example, in the Sorter Domain, the instruction *"Object 3"* is a Φ-Instruction. The effect of this instruction is to set D' to {*isCarrying, Carrying(O), Color(O3), inBasket(O3), Botat(X)*}. The other objects and baskets are ignored.

5 Proposed Approach

In earlier work, we have shown in independent experiments that using each of the instruction types mentioned above is very effective in speeding up RL. Choosing the best instruction type to be used proved a challenge though. As explained in the "paper weight" example in the introduction, interpretation of instructions is crucial. In this paper, we attempt to overcome this dilemma by proposing an algorithm that can handle multiple interpretations. In this section, we explain our approach and the heuristics used. We assume a human instructs the agent in a manner that enables the agent to interpret the instruction both as a $\pi - instruction$ and a $\Phi - instruction$. Using pointing gestures is one such instructing mechanism, where pointing to an Object can either be interpreted as *"Goto(Object)"* or $D' = \{Object features\}$.

Although this difference in interpretation does not affect the instructor, it greatly influences the learner. For instance, a $\Phi - instruction$ results in projecting the state space S onto a reduced feature space. The projected state space S' is smaller and usually the applicable set of actions $A_{S'}$ is also smaller. Learning the optimal policy on S' is thus quicker. Whereas $\pi - instructions$ give the optimal action at a state that can be generalized over similar states. As explained, although both types aid in learning, their effects are very different. Hence it is very important for the learner to choose carefully.

In the following approach, we build both models in parallel (Algo 1). The algorithm is split into two phases. During the training phase, the learner accumulates instructions, if available, to be used later to train an instruction model. This model is used to select actions in the post-training phase. The learner maintains an individual set of instructions for both π and Φ Instructions, D_π and D_Φ. Every instruction, $I(s)$, is interpreted as an action $(\pi - Ins)$ and as a binding on the state space $(\Phi - Ins)$. $I(s) =$ *Pointing gesture towards an object* is interpreted as an action $I_\pi(s) =$ *Goto(Object)* as well as a projection operation $I_\Phi(s) = \rho_{D'}(s)$, where $D' = \{ObjectFeatures\}$. Although both D_π and D_Φ are updated, the agent executes $I_\pi(s)$. This is to make it easy to use the framework on real world agents. Ideally, each instruction model should suggest an action and both need to be performed subject to being in the exact initial states, same random seeds etc. This is not easy to setup in a real world application and hence we choose to perform $I_\pi(s)$ at a state. This arrangement is only during the training phase. In the case that instructions are unavailable at any step, the learner chooses an action suggested by a simple SMDP Q-Learner [4] that is independent of the instruction models.

Once the training period is over, the models $\hat{\Pi}_\pi$ and $\hat{\Pi}_\Phi$ are learned using the training sets. Learning these models requires us to represent the probabilistic dependencies among attributes of the related objects. We use Markov Logic Networks (MLN) [9] to perform the generalization as they can succinctly represent these dependencies resulting in sample-efficient learning and inferring. Combining MLNs and RL is not new and has been done successfully in the past. Torrey et al. [29] have successfully used MLNS and RL to transfer knowledge from a simple 2-on-1 Breakaway task to a 3-on-2 Breakaway task in the Robocup

simulated-soccer domain. Wang et al. [31] approximate an RL agent's policy using MLNs where they update the weights of the MLN using Q-values. On similar lines, we use MLNs to model a human instructor's preference where the inputs to the MLN are either actions or attention pointers.

Algorithm 1 *LearnWithInstructions*

while *Training Period* **do**
  s is the current state
  if $I(s)$ *available* **then**
    $a \leftarrow I_\pi(s)$
    $\mathcal{D}_\pi \leftarrow \mathcal{D}_\pi \cup \{s, I_\pi(s)\}$
    $\mathcal{D}_\Phi \leftarrow \mathcal{D}_\Phi \cup \{s, I_\Phi(s)\}$
  else
    $a \leftarrow \hat{\Pi}_Q(s)$
  end if
  Update $Q(s, a)$
end while
$Train(\hat{\Pi}_{\pi Ins}, \mathcal{D}_\pi)$
$Train(\hat{\Pi}_{\Phi Ins}, \mathcal{D}_\Phi)$
while *Episode not terminated* **do**
  $\hat{\Pi}(s) \leftarrow \max \texttt{conf}(\hat{\Pi}_{\pi Ins}(s), \hat{\Pi}_{\Phi Ins}(s), \hat{\Pi}_Q(s))$
  $a \leftarrow \hat{\Pi}(s)$
  Perform option a
  Update $Q(s, a)$
end while

In the second phase, instructions are absent and the trained models are used to choose actions. The available action models are $\hat{\Pi}_{\pi Ins}(s)$, $\hat{\Pi}_{\Phi Ins}(s)$ and $\hat{\Pi}_Q(s)$ ($\pi - Ins, \Phi - Ins$ and $Q - Learner$). The action suggested by the most confident model is used. The confidence a model is measured by

$$\texttt{conf} = \max_a \hat{\Pi}(s, a) - \max_{b \neq a^*} \hat{\Pi}(s, b) \tag{4}$$

where $a^* = \arg\max_a \hat{\Pi}(s, a)$. The selected action is performed and the $Q - Learner$ is accordingly updated.

5.1 Using the $\hat{\Pi}_{\Phi Ins}$ model

The $\Phi - Instructions$ result in a projection $\rho_{D'}$ of the state space onto a reduced space. By learning the optimal policy in the reduced space, the optimal policy in the original space can be derived by lifting actions suitably. Since in this paper we work with simple projections, the lifting of actions is trivial. An action in the reduced space is lifted to be the same in the original state space. A detailed explanation of learning using ΦIns has been given in our earlier work [14].

5.2 Learning the models

We transform the state features into a set of predicates and ground them using the feature values. The instructions at a state are also transformed into predicates. For example, the $\pi - Instruction$ *Go to object* is transformed into a predicate *Go-to-object* that is set to *true* along with the set of ground predicates describing the state.

We use the techniques proposed in [9] to learn the structure of the MLNs and for inference. Every time a new state is encountered, an action ($\pi(s)$) and a correct binding ($\Phi(s)$) are inferred by treating the ground predicates representing the state as the evidence. In our experiments, we use the Alchemy package [13] for the tasks mentioned above.

The state features are represented as predicates {*isCarrying, Carrying(O), Color(O1,c), inBasket(O1), Color(O2,c), inBasket(O2), Color(O3,c), inBasket(O3), Color(B1,c), Color(B2,c), Color(B3,c), Botat(X)*}. *isCarrying* - true if robot is carrying an object, $Carrying(O)$ - true if object O is in robot's arm, $Color(., c)$ - true if color of corresponding object/basket is $c \in \{c1, c2, c3, c4, \ldots\}$, $inBasket(O)$ - true if object O has been dropped into a basket and $Botat(x)$ - true if robot's current location is x such that $x \in \{O1, O2, O3, B1, B2, B3\}$. The set of actions A = {*Pickup, Drop, Goto(O1), Goto(O2), Goto(O3), Goto(B1), Goto(B2), Goto(B3)*}. The *colorequal(a,b)* predicate is True if a and b have the same color. We assume background knowledge such as $\forall a, isBasket(a) \Rightarrow \neg isObject(a)$ and $isCarrying \Rightarrow \neg Pickup$.

The $\pi - Instruction$ model ($\hat{\Pi}_{\pi Ins}(s)$) is a set of MLNs, where each MLN represents one among the options *Pickup, Drop, Goto(basket), Goto(object)*. In addition to the $\pi - Ins$ model, we also learn a $\Phi - Ins$ model and pit the two against an SMDP Q-Learner. In other words, we choose between three candidate policies using the earlier confidence measure. The learner acts according to the chosen policy at the current state. We normalize the output probabilities of the instruction models to get their action policies and use an $\epsilon - Greedy$ action selection for the Q-Learner. We similarly choose a policy for the agent at each state that it visits. Some of the weights learned by the $\hat{\Pi}_{\pi Ins}(s)$ for the action *Goto(basket)* are shown in Table 1.

The table shows that the MLN has learnt that one of the important features of the task is that the agent has to "go to the basket that is of the same color as the object it is carrying". In addition, the MLN has also learnt the importance of the system dynamics such as "a basket is not an object" and "cannot be near an object that is already in a basket" etc.

Weight	Formula
17.015	¬ isBasket(a1) ∨ ¬isObject(a1)
16.4008	botat(a1)
-15.5892	¬Carrying(a1) ∨ ¬colorequal(a1,a2) ∨ ¬Goto(a2)
7.9959	isObject(a1)
8.28407	isBasket(a1)
-8.25381	Goto(a1)
-9.01076	colorequal(a1,a2)
10.1116	¬inBasket(a1) ∨ ¬botat(a1)

Table 1: Some formulas learned and their weights.

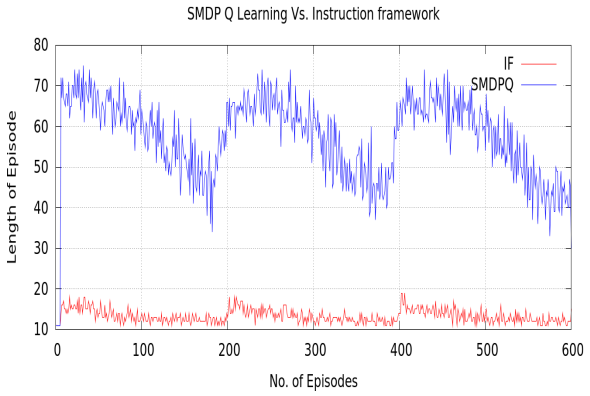

Figure 2: Comparison of IF with SMDP Q Learning

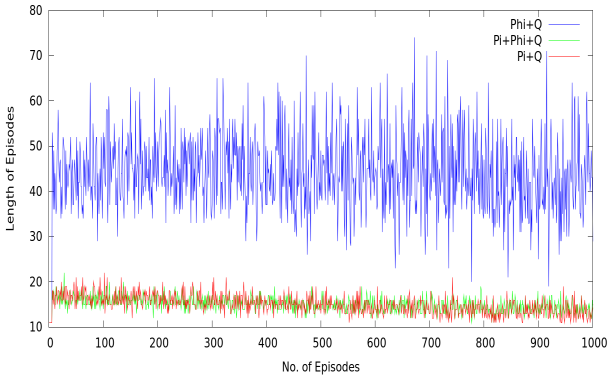

Figure 3: Comparison of $\hat{\Pi}_{\pi Ins}$, $\hat{\Pi}_{\Phi Ins}$ and IF.

6 Results

In this section, we discuss the performance of our approach on the Sorter Domain.

All experiments involved a training period of six episodes during which all actions performed were as instructed by a human. Our framework accumulates the training data and simultaneously updates its Q-Learners (one for the standard SMDP Q-Learner and one for the $\hat{\Pi}_{\Phi Ins}$ model's reduced space). It is to be noted here that we do not employ any kind of function approximators for representing value functions. The instruction models serve as policy approximators though. The standard SMDP Q-Learner does not use any relational features. We refer to our approach as the Instruction Framework (IF) henceforth.

6.1 Experiment 1

In this experiment, we compare the performance of IF with standard SMDP Q-Learning. The purpose of this experiment is to gauge the generalization ability of IF. The experiment is split into blocks of 200 episodes. The learners are initialized with the same starting state for each of the 200 episodes. At the end of every block, one of the colors is replaced with a new one resulting in a different set of states for the learners.

The SMDP Q-Learner's performance drops drastically (Fig 2) whereas IF generalizes very well. This is due to the introduction of new states for which the SMDP Q-Learner has to learn a policy from scratch. IF generalizes well due to the availability of the instruction models. Although there are no human inputs after the training phase, the performance of IF improves as the experiment proceeds. This is due to the increasing confidence of IF's Q-learner. As more states are visited, the Q-learner's estimates of the actions' returns improves and thus control slowly shifts from the instruction models to the Q-learner. Since the training examples were insufficient, the instruction models learnt were not complete resulting in them having low confidence at certain states. By combining the Q-learner with these models we overcome this insufficiency in training data.

6.2 Experiment 2

This experiment demonstrates the importance of interpreting instructions properly and how IF helps by choosing between possible interpretations.

In this experiment we use 3 different learners, one which interprets instructions as $\pi - Instructions$ alone (A), one that interprets them as $\Phi - Instructions$ alone (B) and one that interprets them as both (IF). They are plotted as "Pi+Q", "Phi+Q" and "Pi+Phi+Q" respectively in Fig 3. In this experiment each of these learners are provided with random starting states for each episode i.e., the colors of the baskets and objects are chosen randomly but ensuring that a solution exists. All three learners are given the same set of instructions during the training phase.

B performs the worst suggesting that interpreting the instructions as $\Phi - Ins$ was not entirely beneficial in this domain. A performs much better compared to B implying that interpreting the instructions as $\Pi -$

Ins was overall more beneficial in this domain. The performance of IF is very similar to A implying that our approach selects the better interpretation. At this point, it is to be noted that IF chooses interpretations at each state independent of its choice at other states.

6.3 Importance of Confidence Measures

On carefully analyzing the working of IF, we noticed that although the combined framework seems to be overcoming the weaker interpretation, it does not completely eliminate it. There are certain states when IF chooses the weaker interpretation.

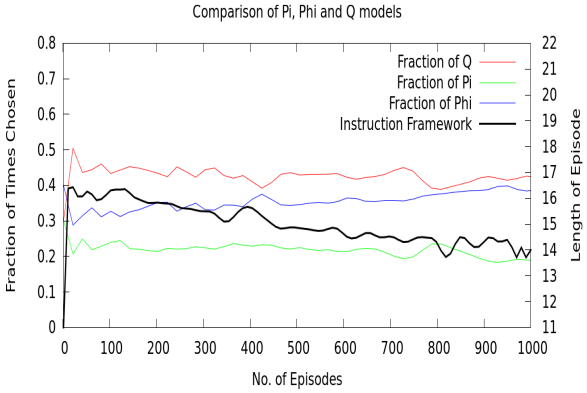

Figure 4: Comparison of percentage of times the action recommended $\hat{\Pi}_{\pi Ins}$, $\hat{\Pi}_{\Phi Ins}$ and SMDP Q-Learner models is taken. The proposed framework's performance (no. of steps taken to solve the task) has also been plotted. The plots have been Bezier smoothed for visibility purposes. The trends are evident in the unsmoothed data.

In Fig 4, we have plotted the performance of IF (in terms of length of episodes i.e, no. of steps taken to solve the task) and the fraction of states in which each of the models $\hat{\Pi}_{\Pi Ins}$, $\hat{\Pi}_{\Phi Ins}$ and $\hat{\Pi}_{Q}$ are preferred. As learning progresses, although the performance of IF improves, the fraction of times $\hat{\Pi}_{\Phi Ins}$ is preferred is slightly increasing. Although the Sorter Domain tasks can be solved in 11 steps, IF solves them in 13 (at the end of 1000 episodes). We doubt this loss in performance is due to preferring $\hat{\Pi}_{\Phi Ins}$ at a few states. One possible reason for this could be the confidence measure that we are using. The current confidence measure is naive and does not capture the confidence of the instruction models effectively. We are currently testing our approach with better confidence measures and have some initial results that show that performance improves with better confidence measures.

We have also been running a few simple experiments with noisy instruction models. We notice that the with better confidence measures our approach seems to be able to overcome the incorrect models quite well, i.e., our approach retains the ability to unlearn incorrect instruction models. The observations are only from initial experimenting though and are not presented here.

7 Conclusion

We have shown that handling multiple interpretations is advantageous and have proposed a novel framework that handles this effectively. We have tested our proposed framework on a reasonably challenging domain highlighting the generalization capabilities achieved by using MLNs and importance of properly interpreting instructions.

As mentioned earlier, a future direction of research would be to use better measures of confidence and integrate the earned returns into the confidence calculation.

References

[1] P. Abbeel and A. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of ICML 04*, 2004.

[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] Steven J. Bradtke and Michael O. Duff. Reinforcement learning methods for continuous-time markov decision problems. In *NIPS*, pages 393–400, 1994.

[4] Steven J. Bradtke and Michael O. Duff. Reinforcement learning methods for continuous-time markov decision problems. In *Proceedings of NIPS*, 1994.

[5] S. R. K. Branavan, Harr Chen, Luke S. Zettlemoyer, and Regina Barzilay. Reinforcement learning for mapping instructions to actions. In *Proceedings of ACL*, ACL '09, pages 82–90, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[6] Mark Brodie and Gerald DeJong. Learning to ride a bicycle using iterated phantom induction. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 57–66, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

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

[8] David Chapman. *Vision, instruction, and action.* MIT Press, 1991.

[9] Pedro Domingos, Stanley Kok, Hoifung Poon, Matthew Richardson, and Parag Singla. Unifying logical and statistical ai. In *Proceedings of the 21st national conference on Artificial intelligence - Volume 1*, AAAI'06.

[10] L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning.* MIT Press, 2007.

[11] Kristian Kersting and Kurt Driessens. Nonparametric policy gradients: a unified treatment of propositional and relational domains. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 456–463, New York, NY, USA, 2008. ACM.

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

[13] S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, D. Lowd, and P. Domingos. The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington.

[14] Pradyot Korupolu V N, Manimaran Sivamurugan S, and Balaraman Ravindran. Instructing a reinforcement learner. In *the Proceedings of the 25th FLAIRS Conference*, FLAIRS '12.

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

[16] Sriraam Natarajan, Saket Joshi, Prasad Tadepalli, Kristian Kersting, and Jude W. Shavlik. Imitation learning in relational domains: A functional-gradient boosting approach. In *In Proceedings of IJCAI '11.*

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

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

[19] Martijn Van Otterlo. A survey of reinforcement learning in relational domains. Technical report, CTIT Technical Report Series, 2005.

[20] Robert Givan Prasad Tadepalli and Kurt Driessens. Relational reinforcement learning: An overview. In *In Proceedings of the ICML04 Workshop on Relational Reinforcement Learning*, 2004.

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

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

[23] Scott Sanner and Craig Boutilier. Approximate linear programming for first-order mdps. In *In Proc. UAI05, 509 517*, 2005.

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

[25] Richard Sutton and Andrew Barto. *Reinforcement Learning : An Introduction.* MIT Press, 1998.

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

[27] Gerald Tesauro. Temporal difference learning of backgammon strategy. In *ML*, pages 451–457, 1992.

[28] A.L. Thomaz, G. Hoffman, and C. Breazeal. Reinforcement learning with human teachers: Understanding how people want to teach robots. In *The 15th IEEE International Symposium on Robot and Human Interactive Communication, 2006. ROMAN 2006.*, pages 352 –357, sept. 2006.

[29] Lisa Torrey, Jude Shavlik, Sriraam Natarajan, Pavan Kuppili, and Trevor Walker. Transfer in reinforcement learning via markov logic networks. In *Proceedigns of AAAI workshop on Transfer Learning for Complex Tasks 2008.*

[30] Chenggang Wang, Saket Joshi, and Roni Khardon. First order decision diagrams for relational mdps. *JAIR*, 31(1):431–472, March 2008.

[31] Weiwei Wang, Yang Gao, Xingguo Chen, and Shen Ge. Reinforcement learning with markov logic networks. In *Proceedigns of European Workshop on Reinforcement Learning 2008*, 2008.

[32] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 1992.

[33] Bernard P. Zeigler. Toward a formal theory of modeling and simulation: Structure preserving morphisms. *J. ACM*, 19, 1972.