Beyond Rewards : Learning from Richer Supervision

A THESIS

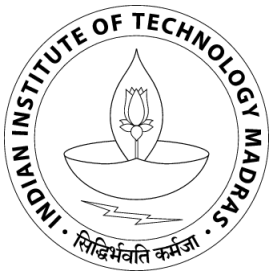
submitted by

K.V.N. PRADYOT

for the award of the degree

of

MASTER OF SCIENCE
(by Research)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.

August 2012

THESIS CERTIFICATE

This is to certify that the thesis entitled **Beyond Rewards : Learning from Richer Supervision**, submitted by **K.V.N. Pradyot**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Science (by Research)**, is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. B. Ravindran
Research Guide
Associate Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Place: Chennai

Date: June 24, 2013

ACKNOWLEDGEMENTS

I express my sincere thanks to my advisor Dr. B. Ravindran, without whose support this work would not have been brought forth. His untiring support and guidance was present all throughout this work. He was always approachable, ready to clear my doubts and taught me a lot regarding research and life. I am grateful for his patience during difficult times and guidance ensuring I was on the right path always. I cherish the conversations I have had with him, including those outside my research work. I also thank CAIR, DRDO, Bengaluru, for funding and supporting my research work.

I thank Dr. Sriraam Natarajan of Wake Forest University for providing useful pointers on Statistical Relational Learning and for clarifying my doubts regarding the same. I thank my GTC members - Dr. D. Janakiram, Dr. Sridharan K and Dr. Anurag Mittal for their valuable suggestions regarding my work. I thank Dr. C. Chandra Sekhar for introducing me to Dr. Ravindran. I thank Manimaran S S for helping me in almost all of my work and for all the discussions.

I thank Balaji for introducing me to research and guiding me over the last few years. I also thank all my friends and RISE Lab members, especially Abhishek, Ananda, Anish, Anshul, Arun, Ashok, Deepak, Kalyan, LN, Prahasaran, Priya, Ranga, Saarang, Sadagopan, Shivashankar, Swapna, Tripti and Vasanth for their help and support. I thank the RISE Robotics Group for helping me implement my framework on the Sorter Robot. I thank everyone who has made my stay at IIT Madras a memorable one. I especially thank my parents for all that they have done for me.

Finally, I thank the administration of IIT Madras and the CSE department for providing me the facilities and environment for conducting research.

ABSTRACT

KEYWORDS: Learning from humans, Markov Logic Networks, Reinforcement Learning, Robot Learning

Robots have captivated human imagination for a long time. Numerous science fictions have promised that the era of robots is beckoning and the day androids walk shoulder to shoulder with us is not too far away. If this were to happen, the intelligent systems that would populate our environment will have to adapt and learn fast, possibly from humans too. For such systems, human teachers present a narrow but critical learning environment. Even between humans, the interactive teacher-student relationship is known to be effective. Endowing robots with human-like learning capabilities would facilitate a similar human-machine interaction, resulting in effective utilization of human knowledge by the robot. We model a subset of these interactions as instructions and propose a framework that enables humans to instruct robots and robots to exploit these supervisory inputs. In all of our experiments, the systems are based on Reinforcement Learning (RL).

In RL, rewards have been considered the most important feedback in understanding environments. However, recently there have been interesting forays into other modes such as sporadic human instructions. We utilize these instructions to identify structural regularities in the environment that can aid in taking behavioural decisions in unfamiliar situations. An important aspect of working with instructions is their proper interpretation. Our approach accommodates multiple interpretations of instructions and provides a handle to choose the best. In this regard, we have tested our approach on several domains and implemented it on a real robotic system.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

CLL	Conditional Likelihood
DOS	Deictic Option Schemas
FODD	First Order Decision Diagram
FOL	First Order Logic
IF	Instruction Framework
i.i.d	Independent and identically distributed
kNN	k-Nearest Neighbour
MDP	Markov Decision Process
MLN	Markov Logic Network
PCL	Point Cloud Library
RL	Reinforcement Learning
ROS	Robot Operating System
RRL	Relational Reinforcement Learning
RRT	Rapidly-Exploring Random Trees
SRL	Statistical Relational Learning
SMDP	Semi Markov Decision Process
VFH	Viewpoint Feature Histogram

NOTATION

$\pi(s, a)$	Policy of RL agent.
π**-Ins or** $I_\pi(s)$	Instruction in the form of action/option.
Φ**-Ins or** $I_\Phi(s)$	Instruction in the form of an operation on state space.
$\hat{\Pi}_{\pi Ins}$	Model learned over given π-Ins.
$\hat{\Pi}_{\Phi Ins}$	Model learned over given Φ-Ins.
$Q(s, a)$	Action value estimate for a in state s.
$\rho_{D'}(s)$	Projection of state s onto a subset of features given by D'.
$\mathcal{D}_I$	Dataset of received instructions.

CHAPTER 1

Introduction

One of the long term goals of AI is to develop systems that can interact and learn from humans. With the recent surge in robots moving out of tailored environments into the real world, the need for such systems has increased. Reinforcement Learning (RL) (Sutton and Barto, 1998), a trial-and-error based learning approach, is a favourite choice among researchers working on such systems since it resembles the human method of learning in many aspects. An RL agent perceives the world as states and interacts with it by performing actions. On performing an action, the agent receives feedback in the form of a numerical reward and a state transition. The agent looks to maximize this reward accumulated over time. Building such an action preference would require inordinate amounts of exploration in the state-action space thus increasing the learning time. There have been earlier approaches that speed up learning by transferring human knowledge to robots such as advice taking (Utgoff and Clouse, 1991; Clouse and Utgoff, 1992), transfer learning (Torrey *et al.*, 2009, 2007), supervised actor-critic (Rosenstein and Barto, 2004) and imitation learning. Imitation learning 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*a*), behavioral cloning (Sammut *et al.*, 1992), learning to act (Khardon, 1999), and 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 (Ratliff *et al.*, 2006). Recently, imitation learning has been posed in highly stochastic yet relational domains using a statistical relational learning (SRL) (Getoor and Taskar, 2007) formulation and has been solved using functional-gradient boosting (Natarajan *et al.*). Lately there has been a surge in interest in developing

apprenticeship learning methods (also called as inverse reinforcement learning) (Abbeel and Ng, 2004; Ratliff *et al.*, 2006; Neu and Szepesvari, 2007; Syed and Schapire, 2007).

With a similar objective, we model human inputs as instructions and propose an approach to incorporate them in the agent's learning process.

1.1 Motivation

Almost all of the above mentioned approaches use trajectories given by a human expert. In an RL setting, a trajectory is a temporal sequence of state-action pairs. These approaches look to optimize an agent's behaviour based on observations of rewards and transition dynamics associated with these trajectories. All these methods assume that the expert is optimal (or at least near-optimal) hence limiting the learner's performance to the efficiency of the expert. Also, generating such human trajectories is expensive and requires the expert's continuous attention over the entire training period.

David Chapman developed a game player, Sonja (Chapman, 1991), that used simple human instructions. Sonja used ideas from Deixis (Agre, 1988) to bind these instructions to relevant objects or regions in the game. In (Utgoff and Clouse, 1991), the authors propose an approach where the learning agent requests advice from an expert to which the expert responds with state preferences. The agent learns this state preference model from a few such interactions and uses this to speed up learning. Motivated by this success in incorporating instructions into the learning process and effectively exploiting them, we model human-robot interactions as two classes of instructions - suggestions for action selection or specification of relevant features in learning. We use these instructions to identify structural regularities in the world. The expert inputs that we work with are more general than all the previous apprenticeship and imitation learners. This generality of instructions reduces the cognitive load on the expert. Also, our approach works with intermittent inputs not requiring the expert's continuous attention.

The key contribution of our work is that we observe beyond the usual rewards and

transition dynamics that almost all the earlier mentioned methods exploit. As an example consider a robot learning to cook. If the robot touches a hot bowl, a simple RL agent would perceive it as an action with a negative reward and accordingly update its action preference. Most of the methods mentioned in the introduction would observe an expert cooking and as a consequence probably manage to stay away from the bowl since the expert stayed away. A robot using our approach would be learning through exploration until it approaches a hot bowl, when the expert would instruct it to *keep away*. The robot would follow this instruction and also learn that $heat \implies keepaway$. In other words, the agent learns properties of the world other than rewards by generalizing these limited expert instructions.

Although working with such instructions is beneficial, the generality can result in ambiguous interpretations resulting in a loss in performance. Our approach is equipped to handle such multiple interpretations and prunes out the suboptimal ones.

1.2 Organization of the Thesis

The next few sections of this chapter discuss earlier work that is related to the work presented in this thesis and highlight our contributions. Chapter 2 introduces the necessary background. Chapter 3 [1] introduces instructions and the formulation that we propose. It also explains our framework and discusses the experiments that we performed on individual classes of instructions. Chapter 4 discusses an extension to the framework that handles multiple interpretations of instructions. Chapter 5 summarizes our work and discusses possible directions of future research.

1.3 Brief Description of Related Work

In this section, we discuss those approaches that we feel are closely related to our work.

[1] Please note that the chapter organization is slightly different from that given in the synopsis, with chapters 3-5 there merged to a single chapter 3.

1.3.1 Learning from demonstration

The obvious approach to learning from humans would seem to be demonstration. Christoper Atkeson and Stefen Schaal (Atkeson and Schaal, 1997) outline an approach in which the robot learns a reward function by observing a demonstration and learns the task model by repeatedly attempting to perform the task. The task in their case was balancing an inverted pendulum (the segway is also an inverted pendulum). Due to modelling limitations, a purely imitation based approach did not provide good results, thus creating the need for a learning procedure. They proposed an approach that could be divided into 2 phases:

- An initial human demonstration that provided a starting point for learning and defined the goal task.
- The second phase, model learning, was further divided into swinging the pole up and balancing, which requires planning and direct-task level learning.

Their observations put forth interesting points such as the inability of non-parametric methods to eliminate the need for task-level direct learning, the role of a priori knowledge in learning and its implications on RL, especially on the debate of whether model based approaches are better or non-model based ones.

1.3.2 Advice

Any external input to the control algorithm that could be used by the agent to take decisions about and modify its behaviour can be called advice.

The general approach to learning from advice, as suggested in (Hayes-Roth *et al.*, 1981), consists of five steps:

1. Requesting or receiving the advice.
2. Converting the advice into an internal representation.
3. Converting advice into a usable form.
4. Integrating the reformulated advice into the agents knowledge base.

5. Judging the value of advice.

Each step poses challenges that need to be dealt with in a manner appropriate to the problem at hand. For example, step 1 raises the issue of whether advice should be given on request or as and when the advisor feels necessary. This further raises doubts over whether the advisor has good advice when requested resulting in a requirement to judge the value of advice (step 5). Similarly step 4, integrating advice into the agent's knowledge base is addressed by Maclin and Shavlik (Maclin and Shavlik, 1998). They use KBANN (Knowledge Based Neural Networks), which involves the use of hidden units to record state information. The authors accept certain rules for action selection and incorporate them into a neural-network Q-function model 2.1. Their observations increase the confidence in the idea of advice taking RL being effective and exploitable.

Another method of incorporating advice could be the use of deixis 2.2.

1.3.2.1 Who initiates Advice?

Paul Utgoff and Jeffery Clouse have conducted experiments that suggest that an external expert is helpful in speeding up learning. They address the problem with two different approaches.

In the first one, the learner queries the expert for advice (Utgoff and Clouse, 1991). The authors talk about two kinds of training information for learning an evaluation function. The corresponding methods to exploit each are the State Preference (SP) method and the Temporal Difference (TD) method. SP learns a value function that represents the pairwise state preference of an expert whereas the TD method learns a future value of the state in terms of cost to goal. They also present a method to integrate both. In the second method, the expert offers advice whenever he feels necessary(ACE/ASE algorithm - teacher modifies action preference) (Clouse and Utgoff, 1992). Both the experiments show considerable reduction in time taken for learning.

We model our approach by using ideas that combine the benefits of both these ap-

proaches. Similar to their first approach, we model the expert but restrict ourselves to approximating the expert's action preference, also called policy. Also, their approach only handles state preferences and control decision preferences whereas our approach facilitates the integration of a special type of information that we call $\Phi - instructions$ (explained in a later section). Like in their second approach, an expert instructs at his convenience.

I would advice you to read the footnote on page 10 for some extra information.

Those of you who now know that my surname is Korupolu, chose to take my advice, whereas the others chose to ignore it. Did following the advice mean the trust on the advisor was more? What motivated your decision to heed or ignore the advice? The next section discusses this dilemma of when to heed and when to ignore advice.

1.3.2.2 How absolute?

An important question that arises next is the absoluteness of advice, i.e., can the agent override the expert's advice? The approach suggested by Utgoff, in which the expert modifies an action's preference, seems reasonable. In this method, the agent performs the action when advised and learns from guidance by reinforcing its belief on the expert through feedback.

Mike Rosenstein (Rosenstein and Barto, 2004) developed a method of integrating Error Signals (Supervised Learning) and Evaluation Function (Reinforcement Learning), the two forms of feedback available to agents in most real world tasks. He proposed a Supervised Actor-Critic RL Framework that computes a composite action that is a weighted average of the action suggested by the supervisor (a^S) and the exploratory action suggested by the evaluation function (a^E). He introduces an interpolation factor (k) that decides the extent of control (autonomity) that the supervisor (learner) exercises. The amount of control is reduced as the evaluation function gains experience.

$$a \leftarrow ka^E + (1 - k)a^S \tag{1.1}$$

The idea was implemented on various task domains and was found to aid considerably in the learning process. When this advice is absolute or inviolate, we call it an instruction. All expert inputs in our work are modelled as instructions.

1.3.2.3 Nature of Supervision

The nature of supervision defines the form in which advice would be given to the learner. Advice could be:

1. The next action to be chosen at a given state.
2. An initial action selection model that the agent could improve upon.
3. The region of state space to visit next.

1.3.3 Shaping

Dorigo and Colombetti (Dorigo and Colombetti, 1994) explore the use of RL to *shape* a robot to perform a predefined target task. Their work broadly lies in the line of research that deals with developing autonomous agents that are strongly coupled with the physical world, and are called situated or embedded agents (agents employed by Chapman, mentioned in 2.2).

A first, fundamental requirement is that agents must be grounded, in that they must be able to carry on their activity in the real world and in real time. Another important factor is that adaptive behaviour cannot be considered as a product of an agent considered in isolation from the world, but can only emerge from a strong coupling of the agent and its environment.

The second approach relies on automatic learning to dynamically develop a situated agent through interaction with the world. The idea is that the interactions between an agent and its environment soon become very complex, and their analysis is likely to be a hard task.

The approach they advocate is intermediate. They first design the agent exploiting knowledge about interactions with environment and then translate suggestions from an external trainer into an effective control strategy that will help them reach the goal. They call this approach shaping, as opposed to the more classical unsupervised RL approach, in which an organism increasingly adapts to its environment by directly experiencing the effects of its activity. The problem was to come up with a balance between design and learning & training, that is how much knowledge do they craft into the agent and how much should they leave for the robot to find out. They approach this problem by extensive experimentation with various design choices and learning strategies.

Unlike this approach, we do not shape an agent but rather let the agent track patterns in the world based on the expert inputs it receives. Shaping requires considerable effort on the expert's part unlike intermittent instructions that reduce the expert's cognitive load.

1.3.4 Inverse Reinforcement Learning

Informally, the Inverse Reinforcement Learning (IRL) Problem (Ng and Russell, 2000*b*) can be considered as the task of determining the reward function being optimized given measurements of the agent's behaviour over time in various situations, measurements of its sensory inputs if necessary, and a model of the world, if available. Motivation for this approach lies in the presupposition that the reward function is the most precise and transferable definition of the task and that the reward functions of most real world problems cannot be modelled completely resulting in the need to observe experts. Unlike in IRL, the approach we propose does not require the agent to observe an expert's behaviour over several episodes. Instead the expert gives occasional inputs in the form of instructions only. We do not look to optimize an agent's behaviour based on an estimated reward function as in IRL. Instead, we generalize the information contained in an instruction over the entire search space. We carefully make use of this generalization to speed up learning.

1.3.5 Implicit Imitation

Accelerating Reinforcement Learning through Implicit Imitation (Price and Boutilier, 2003) proposes the Implicit Imitation model. This model allows the agent to observe an expert's behaviour and use the observed state transitions to update its estimates of state values and actions. The agent does not explicitly imitate the trajectory of the expert. In addition to learning the domain dynamics, the agent can also take cues from the expert in terms of regions of state space worth exploring etc.

To summarize, extensive work has been done in the field of learning from experts and it has been shown that expert input can be effectively utilized in speeding up learning. Taking inspiration from the work on Advice taking Learners and Chapman's work with Sonja, we model human inputs as instructions (absolute advice). We also address the limitations of IRL and Implicit Imitation by looking beyond the traditional reward function and transition dynamics. Also, our formulation of expert inputs reduces the effort required from the experts, unlike approaches similar to Shaping.

1.4 Contributions of Thesis

- Explored a hitherto less explored aspect of learning beyond rewards and transitions in RL. Showed that it is beneficial to observe such system patterns to speed up learning.
- Classification of human-robot interactions into two types of instructions. Provided a mathematical formulation of the same for ease of integration with RL. This formulation covers the two most important aspects of RL - actions and states.
- Explored the benefits of using each type of instruction in challenging learning tasks - Transporter Domain and Game Domain. The observations made from these experiments highlight the ease with which naive supervised learning techniques can be combined with RL using our formulation and also achieve good speed up in learning.
- Discussed the ambiguity arising due to use of general instructions and provided a framework for efficiently handling the same. Tested the framework on a real world Sorting task with a real robot.
- Provided an SRL (Statistical Relational Learning) approach to integrating instructions and reinforcement learners. Most real world tasks are relational and our ap-

proach provides a very effective handle to combine the representational advantages of SRL with RL. The generalizing power of SRL techniques makes learning quick making our approach well suited for real world tasks.

2

[2]The K in Pradyot KVN stands for Korupolu.

CHAPTER 2

Background

2.1 Reinforcement Learning

RL is a machine learning technique, that falls neither under supervised nor unsupervised learning methods. In supervised learning (SL), there is a predefined input-output pair representation, i.e., at a given state (input), a supervised learner would be told what action to take next (output). Reinforcement learners are not bound by any such input-output representation. Also, unlike unsupervised learning, there is a notion of feedback in terms of rewards and next state. Initially an RL agent chooses randomly from the action space, receiving rewards from the environment for every action selected. Based on the experience gained over repeated runs, the agent computes the desirability of choosing a particular action. The agent looks to maximize the reward accumulated over time, i.e., it looks to choose actions that promise higher returns in the long run. Returns are defined as a time-discounted sum of rewards. The value of a state is defined as the expected return starting from that state. Thus, the expected return of selecting an action is the weighted sum of the immediate reward and the value of the expected next state. Sequential problems are generally modeled as *Markov Decision Processes* (MDPs) in RL. 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, $\psi \subseteq S\,XA$ 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 (Watkins and Dayan, 1992),

$$Q_{k+1}(s, a) = Q_k(s, a) + \alpha[r + \gamma max_{a'} Q_k(s', a') - Q_k(s, a)] \tag{2.1}$$

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 (Bradtke and Duff, 1994) or a policy fragment. Q-learning applies to options too and is referred to as SMDP (Semi Markov Decision Process) Q-learning (Bradtke and Duff, 1994).

$$Q_{k+1}(s, o) = (1 - \alpha) Q_k(s, o) + \alpha[R + \gamma^\tau max_{o'} Q_k(s', o')] \tag{2.2}$$

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

In real world robots, options such as `Exit Room` 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 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. Also, the option `Exit Room` will take different times to execute depending on the robot's current position in the room. SMDP Q-learning helps by providing a learning framework that can accommodate the stochasticity mentioned above.

Options can be also understood as macro-actions or subroutines which enable the use of a hierarchical learning framework where the RL agent can use these subroutines to take high level decisions. An agent can decide to `Exit Room` assuming the lower control of executing every move to reach the exit is taken care by a low level learner.

2.2 Deixis

David Chapman and Philip Agre (Agre and Chapman, 1989) have given a sound reasoning of the role of plans and contrast two views of plans, *plan-as-program* and *plan-as-communication*. The plan-as-program approach considers a plan to be a series of steps to be executed blindly, whereas the program-as-communication treats plans as just any other resource required for decision making (advice). The approach is explained using Pengi, a system that uses novel kinds of perceptions and representations in playing a video game, Pengo. An important contribution here is a participatory theory of representation the authors call indexical-functional or deictic pointers. The theory describes a causal relationship between the agent and indexically and functionally individuated objects. Unlike in classical representation, where objects are referred to as object1, object2 and so on; in representation using Deixis, objects are represented as the-object-AGENT-is-looking-at. This participatory nature ensures that the agent deals with its environment through a constant interaction rather than through construction and manipulation of models. This method of representation will be of immense use in dynamic environments such as the real world.

Deictic representation, because of its compact observation space and its ability to focus

on interesting parts of the state space, could have an advantage over other forms of representation. In his thesis, Michael Cleary (Cleary, 1997) details the benefits of deixis using a deictically controlled wheel chair. Deictic commands based navigation is very similar to the way humans move around in the world, bringing the problem closer to natural learning. For example, when we give a person directions to reach a destination, we regularly use landmarks such as buildings and towers. In some cases, the listener may not even have seen the landmark (or pointer) earlier, and need not know its functionality or any other information as long it can recognize that it is the required landmark.

In their paper on Diectic Option Schemas (Ravindran *et al.*, 2007), Ravindran.B, Andrew Barto and Vimal Matthew present a hierarchical RL framework that makes use of Deictic representation that results in the time taken for looking around being only a fraction of the total learning time. On the other hand, Sarah Finney, Natalia H. Gardiol, Leslie Pack Kaelbling and Tim Oates (Sarah Finney and Oates, 2002) show with an example of a Blocks world that learning with deictic representation need not always yield better results. This opens up the question on how effective would deixis be in aiding advice incorporation and in case it is, what issues would we face in developing such a system.

We classify certain instructions as focus pointers similar to the Deictic pointers introduced in this section. Such instructions are incorporated as operations on the state space and aid in identifying influential features of the state. Working on this reduced feature set results in good speed up in learning.

2.3 Structured States

The set of states S is *structured* by representing it as a cross-product of an indexed family $\{S_\alpha | \alpha \in D\}$, where D is the set of state features (Zeigler, 1972). In general $\alpha \in D$ is referred to as the coordinate and S_α is its state set. The structure assignment is a one-one mapping from S to $\prod_{\alpha \in D} S_\alpha$. Thus, a state $s \in S$ is represented as $(s_{\alpha_1}, s_{\alpha_2}, \ldots, s_{\alpha_i}, \ldots)$ where s_{α_i} is a value of the feature set S_{α_i}.

Let f be a set of indexed functions such that $\{f_i : S \rightarrow B_i | i \in E\}$, where E is a different set of coordinates and B_i is the corresponding state set. The cross product function $\prod_{i \in E} f_i : S \rightarrow \prod_{i \in E} B_i$ is defined by $\prod_{i \in E} f_i(s) = (f_1(s), f_2(s), \dots)$. Coordinate projections, are one such special class of indexical functions operating on S, that we use to model certain instructions. $\{\rho_\alpha | \alpha \in D\}$ where $\rho_{\alpha_i} : S \rightarrow S_{\alpha_i}$ such that $\rho_{\alpha_i}(s_{\alpha_1}, s_{\alpha_2}, \dots, s_{\alpha_i}, \dots) = s_{\alpha_i}$. Extending the above cross product function to projections of S : For $D' \subseteq D, \rho_{D'} : S \rightarrow \prod_{\alpha \in D'} S_\alpha$ is given by $\rho_{D'} = \prod_{\alpha \in D'} \rho_\alpha$. For example, $\rho_{\{color,shape\}}(green, heavy, cube, wet) = (green, cube)$.

Every subset $D' \subseteq D$ induces a partition $K_{D'}$ on S such that two states $s, s' \in S$ belong to the same block B_i only if: $\rho_{D'}(s) = \rho_{D'}(s')$ and is denoted by

$$[s]_{K_{D'}} = [s']_{K_{D'}} \tag{2.3}$$

A partition can also be represented as $[s]_f$. The states $s, s' \in S$ belong to the same block in the partition B caused by f only if $f(s) = f(s')$.

The focus pointers from the previous section are incorporated into learning as projections ($\rho_{D'}$). In general, our approach can work with instructions that can be represented as structured functions.

2.4 Markov Logic Networks

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 (Tesauro, 1992; Brodie and DeJong, 1999), 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 (Otterlo, 2005; Prasad Tadepalli and Driessens, 2004; Sanner and Boutilier, 2005; Wang *et al.*, 2008*a*). Sanner and Boutilier (Sanner and Boutilier, 2005) used situation calculus to capture the dynamics and proposed a linear programming formulation for solving the action selection problem. Wang et al. (Wang *et al.*, 2008*a*) used First Order Decision Diagrams (FODDs) that generalize Algebraic Decision Diagrams (Bahar *et al.*, 1993) to capture the domain's dynamics and represent the reward and value functions. Action selection was posed as a manipulation of these diagrams. They defined several arithmetic operators on FODDs for the same. Although these methods are attractive, they still suffer from the large exploration required to collect enough training samples for learning. In many real world domains, it is natural to have access to a human expert who can provide guidance or instructions to the learner. In these Relational RL (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. (Thomaz *et al.*, 2006) 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 expert 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 (Kersting and Driessens, 2008). While this method can use the initial trajectories, there is no interaction with the human beyond the initial model.

In some of our work we use Markov Logic Networks (MLN) (Domingos *et al.*, 2006) to represent the world of interest and use human instructions to learn the probabilistic relations among objects populating the world. These relations represent the dynamics of the world that we make use of to speed up learning.

In general, a set of possible worlds can be represented using a first order Knowledge Base. These formulas act as a set of hard constraints. Violating even a single formula would deem the world impossible. The idea in using Markov Logic is to relax these constraints by associating the formulas with weights. Violating constraints would lessen the

probability of that particular world but not make it impossible. A higher weight implies a stronger constraint. This set of formulas and real valued weights (F_i, w_i) are represented using MLNs. Together with a set of constants $C = \{c_1, c_2, \ldots, c_{|c|}\}$, a MLN can be instantiated as a Markov Network with a node for every ground predicate and a feature for every ground formula. The same weight is assigned to every grounding of the same formula, resulting in the following joint probability distribution :

$$P(X = x) = \frac{1}{Z} \exp(\sum_i w_i n_i(x)) \tag{2.4}$$

where $n_i(x)$ is the number of times the formula i is satisfied by the world x and Z is a normalizing constant (like Markov Networks).

We transform the state features into a set of predicates and instantiate them using the feature values. The instructions at a state are transformed into predicates too. 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.

If the constraints governing the environment are known, these can be accordingly transformed into formulas and corresponding weights be learned. Weight learning is done generatively by maximizing the likelihood of a relational database (Eqn. 2.4). This method is known as Generative Weight Learning (Domingos *et al.*, 2006). We use the thus learned weights to infer the probability of an action being optimal at a newly encountered state (instantiation of predicates).

A basic inferencing task is to find the most probable truth assignments for predicates given a partial assignment. The partial assignment is called evidence. This is known as MAP inference. In MLNs, inferring reduces to finding the truth assignment that maximizes the sum of weights of satisfied clauses or formulas. In our experiments, we learn the model using the (state,instruction) pairs collected during the training period. 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 a state as the evidence. Since we know beforehand, the predicates that would be queried, we could also use Discriminative Weight

Learning (Domingos *et al.*, 2006). This method exploits knowledge of query predicates by partitioning the ground atoms into evidence (X) and query (Y). A *conditional likelihood* (CLL) of Y given X is learned which is better compared to pseudo-likelihood.

Weight learning requires us to know the underlying formulas governing the world. In most learning tasks these are unknown and hence simple weight learning is insufficient. In other words, the structure of the MLN needs to be learned. In principle, the structure of an MLN can be learned using any inductive logic programming (ILP) technique. In (Kok and Domingos, 2005), the authors present a structure learning technique that starts with a set of unit clauses or an expert-supplied MLN and repeatedly adds clauses to the MLN using beam search. Similarly hand-coded clauses are modified by removing predicates. In our experiments, we use this technique to learn the structure of the MLN representing our task, using the Alchemy package (Kok *et al.*) .

Since all the instructions that a human provides in our approach are positive, we make the open-world assumption while learning the weights of our MLNs. This means we assume that we do not know anything about those grounded predicates that do not appear in the knowledge database. If we were to make a closed-world assumption, the weight learner would have assumed that all predicates absent in the database are false. This assumption has an effect on the quality of generalization achieved by the MLNs.

CHAPTER 3

Instructions

3.1 Introduction

We model a subset of human-robot interactions based on interactions between peers playing a multi-player video game. Consider the process of a kid learning to play the game. He develops most of his skills by learning through experimentation and inputs from his peers. Typical commands exchanged while playing a game are “Crouch”, “Jump”, “Walk slow”, “Shoot” etc. These exchanges are intermittent. Since the teacher instructs only occasionally, the window during which he must focus on the student is small. This makes it possible for the teacher to instruct the student while simultaneously performing a different task. This reduction in cognitive load on a human expert makes it beneficial to teach a real world robot through instructions. These interactions are also suited to collaborative tasks such as cooking where a human uses instructions to direct a robot aiding him and the robot learns to solve the task by following these instructions.

3.2 Definition and Examples

We define instructions as any inviolate external input to the RL agent, that it uses to make behavioural decisions. For example, an agent that is learning to cut vegetables can be instructed to *use the sharp edge of the knife*. Consider a human learning to throw a ball into a basket. Evaluative feedback will depend on how far the ball misses the target by. Whereas, instructive feedback will be a coach instructing him to throw harder or slower.

Instructions in an RL setting could be of various forms:

- The next action or an option to be chosen at a given state. For example, "Walk slowly" in the video game.
- A binding in the form of a state or region of state space to visit next. For example, "Keep to the left".
- An object that binds a policy to a goal. For example, for an agent that knows how to throw a ball, the instruction, " that red ball," would ground its policy and it would pick the red ball and throw it.

3.3 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 work :

$$\pi(s, a) = G(\Phi(s), a) \tag{3.1}$$

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.

3.3.1 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 $a \in A_s$. As an example, consider an RL agent learning to manoeuvre through a farm and is in a state with a puddle in front (s_{puddle}). The instruction *jump* is incorporated as $\pi\left(s_{puddle}, jump\right) = 1$. If policy models are built over such instructions, their effect on the policy would be $\pi(s, a) \simeq 1$.

3.3.2 Φ-Instructions

Instructions of this type are given as structured functions (Zeigler, 1972) denoted by I_Φ. In this work, 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, consider an RL agent learning to throw balls. The instruction, *"Ignore the ball's color"* will be incorporated as $D' = D - \{ballcolor\}$.

3.4 Proposed Framework

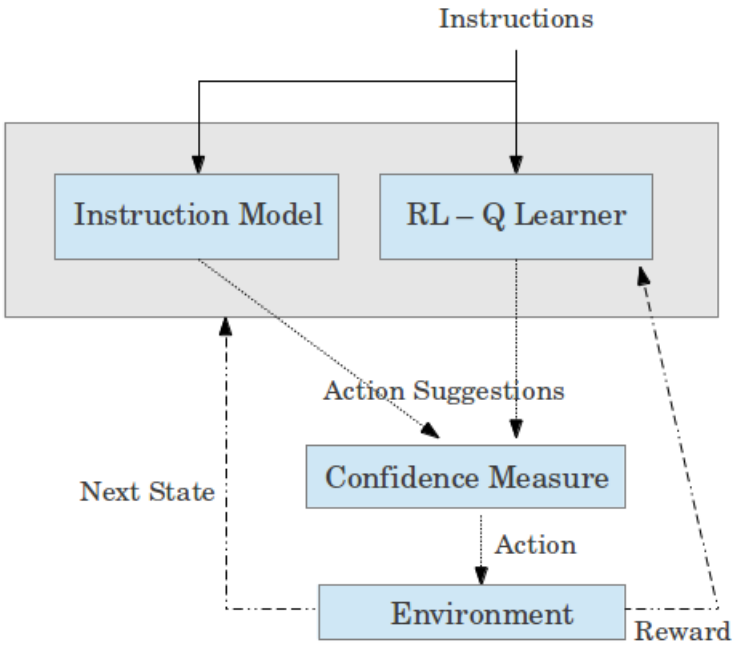


Figure 3.1: The Proposed Framework

Fig 3.1 shows the proposed approach as a block diagram. We propose a framework wherein an agent receives instructions from an expert, performs as commanded and simultaneously builds a model of these instructions (shown as solid lines). The instruction model is independent of the RL framework. When instructions are not available at a state, the agent chooses between the actions suggested by the instruction model and RL using certain confidence measures and accordingly performs an action (shown as dotted lines). The reward

received on performing the action is used by the agent to update its estimate of the action value. The agent transitions into a new state on performing the action and the cycle continues.

3.5 π - Instructions

This section explains how the proposed framework can be used to generalize π-instructions. We implement the framework on the Transporter Domain and use a k-NN classifier to generalize the instructions. In this domain, states that are spatially close to each other are behaviourally similar. In other words, an instruction given at a state is generally valid for states in an immediate neighbourhood. The presence of such a pattern in exploited by our approach resulting in quick generalization and early convergence times for learning. It is to be noted that our framework is domain independent and for the sake of clarity in the exposition, we have focused our discussion on one domain.

3.5.1 Transporter Domain

The layout of the Transporter Domain is shown in Fig 3.2. The task of the RL agent is to transport an object from the starting position to the goal position. The object can be a sphere, a cube or a cylinder weighing between 0 and 15 pounds. The path to the goal is 15 feet long. The first and last 5 feet are on level floor and the remaining are on a slope. The agent can transport an object by invoking 1 of 3 options. The options are *carry using 1 arm*, *carry using 2 arms* and *push*. All options move the agent towards the goal for a maximum of 5 feet.

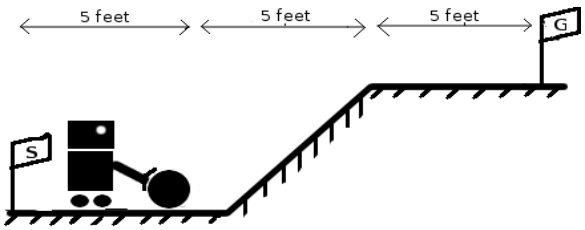


Figure 3.2: Transporter Domain

The dynamics of the options depend on the shape and weight of the object as well as the slope of the floor. Heavier objects take longer time to be transported. *Carry using 1 arm* is faster than *carry using 2 arms*, which in turn is faster than *push along the floor*. All options execute slower on the slope. *Pushing* a sphere or cylinder is faster than *pushing* a cube. Also, the time taken to pick up an object on the floor is proportional to its weight. An option may not execute to completion always. The agent might drop the object halfway, depending on the object properties and the slope of the floor. For example, a heavy object is dropped more easily than a light object. Similarly, carrying a cube using 2 arms is safer than carrying a cube with 1 arm. Also, the probability of dropping an object is greater on the slope. The exact dynamics of the domain along with the implementation is currently available at http://rise.cse.iitm.ac.in/wiki/index.php/TransporterDomain.

A learner would optimally *carry light objects using 1 arm*, *push heavy objects*, *carry cubes using 2 arms*, *push a sphere or a cylinder* and *carry objects using 2 arms on the slope*. The state features observed by the agent are $\langle obj-shape, obj-weight, current-position, obj-in-arm\rangle$, where $obj-in-arm$ indicates whether the object is being carried or is on the floor.

3.5.2 Instruction Framework

The standard RL approach uses SMDP-Q learning with an ϵ greedy exploration policy. The reward function used is the minimally informative and is described simply as $reward = -1$ till termination. Our algorithm is presented in Algo 1.

Occasionally(with some probability ζ), we provide the agent with π - instructions i.e.,

tell the agent the best option to perform in a given state. The agent generalizes over these instructions by using a standard classifier. This classifier outputs an option based on the given state. The set of all π - instructions seen so far forms the training data for this classifier. In this particular implementation we have used a k-NN classifier. Every time an instruction is given, $\{s, I_\pi(s)\}$ is added to the dataset $\mathcal{D}_I$. The flow of instructions is cut-off after a fixed number of episodes. Regular Q function updates continue to take place in parallel as in a standard Q-learner.

At every decision point, the agent chooses between the option recommended by the k-NN model and the Q-learner by comparing their confidences. The confidence of the k-NN model is computed as the inverse squared distance between the given state and its nearest neighbour (of the same class as predicted by the k-NN). The variance in the Q function is used to represent the confidence measure of the Q-learner. If the variance in the value of a particular state-option pair is very low, it implies that the value has converged to the final value defined by the policy. In other words, the confidence of the Q-learner in an option is inversely proportional to the variance of the Q function at that state-option pair.

Whenever the confidence of the k-NN model is high (as decided by a threshold) and the confidence of the Q-learner is low, it performs the action suggested by the k-NN model. Otherwise, it follows the policy represented by $Q(s, a)$. We assume that eventually the Q-learner might become more optimal than the k-NN model. This might be due to errors in generalization or a faulty instructor. Hence the Q-learner is given the benefit of the doubt. The threshold parameters *Qthresh* and *Cthresh* have to then be tuned.

3.5.3 Results and Analysis

Learning the optimal policy on this domain is hard due to the complex dynamics and variety in the objects. The performance of our approach is shown in Fig 3.3[1]. Our approach converges at around 2000 episodes, whereas standard SMDP Q learning takes more than 10000 episodes to show similar performance. This is because the standard Q learner's

[1]List of parameters used : $\epsilon = 0.07$, $\alpha = 0.1$, $k = 3$, $CThresh = 50$, $QThresh = 0.5$

Algorithm 1 LearnWithπInstructions($\mathcal{D}_I$)

while *episode not terminated* **do**
  s is the current state
  if $I_\pi(s)$ *available* **then**
    $a \leftarrow I_\pi(s)$
    $\mathcal{D}_I \leftarrow \mathcal{D}_I \cup \{s, I_\pi(s)\}$
  else
    if $\texttt{conf}(Q(s, \arg\max_b Q(s,b)) < Qthresh \& \texttt{conf}(kNN(s)) > Cthresh$ **then**
      $a \leftarrow$ *kNN-Classify*$(s; \mathcal{D}_I)$
    else
      $a \leftarrow \arg\max_b Q(s,b)$
    end if
    Perform option a
    Update $Q(s,a)$
  end if
end while

knowledge about solving the task is built only by exploration. Also, due to the stochastic nature of the world, the Q-learner's learning rate (α) was kept low resulting in the learner to perform an action several times over to get a strong estimate of the return associated with the action. A high learning rate in a stochastic world would result in large fluctuations in the learner's estimates of the action's returns. This would mean that the learner would take longer to converge to an optimal policy, thus requiring a low learning rate. For example, in the Transporter Domain, although a standard Q-learner would have explored all possible *options* for a *sphere* weighing 10 pounds, it would have to explore these *options* again for a *sphere* weighing 9 pounds although the best option for both could be the same one. Whereas in our approach, the k-NN model of the world built based on instructions captures world specific information such as behavioural similarity of neighbouring states due to which redundant exploration as explained earlier is avoided.

In order to make the comparison fair, the SMDP Q-learner follows the same set of instructions whenever available. The parameter ζ is the probability of receiving an instruction at any decision point. Thus a larger ζ implies more number of instructions overall and also faster convergence (Fig 3.3). Also note that our approach earns better rewards sooner than the standard SMDP Q-learner. In RL experiments, since the aim is to maximize the earned reward, a higher average reward normally implies better learning.

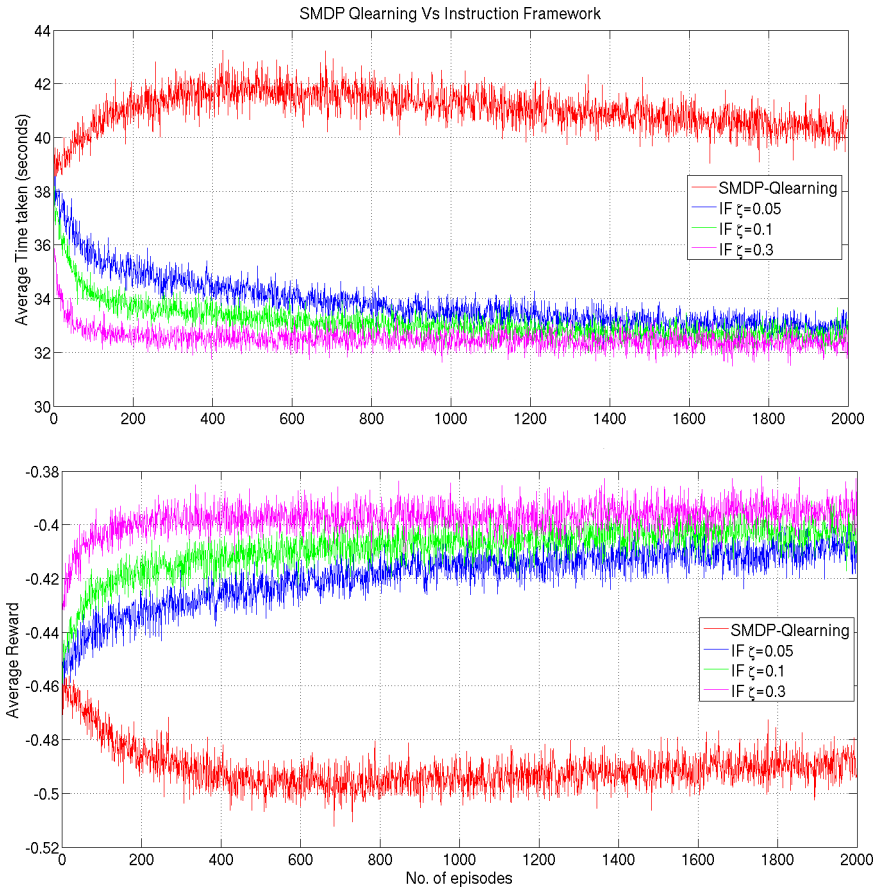


Figure 3.3: Comparison of SMDP Q-learning with instruction framework for various ζ

Approaches that employ function approximators (FAs) for the value function, proved to be difficult to setup for this domain. Using FAs such as a neural network and tile coding on this domain resulted in much longer learning periods than standard Q learning. Choosing options purely based on the k-NN classifier results in poorer performance than the above approach. This points out that our method is better than both generalizations of the policy and generalizations of the value function.

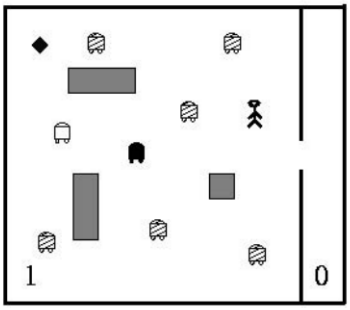
3.6 Φ - Instructions

This section highlights the advantages of generalizing Φ-instructions. Like in the previous section, our approach is not limited to this particular domain. We chose the Game Domain

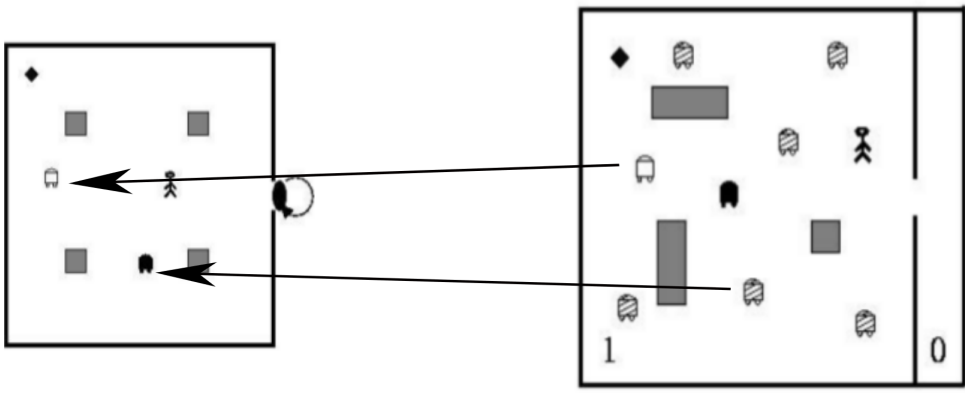
as it is well suited to showcase the advantages of using Φ-instructions. Solving this task requires the agent to carefully choose relevant features of the state space, which is one of the motivations for designing Φ-instructions. The agent uses these instructions to learn a pattern, if it exists, in this selection of features and exploits this knowledge to solve the task more efficiently.

3.6.1 Game Domain

The layout of the game is shown in Fig 3.4a. The environment has the usual stochastic gridworld dynamics and the *SLIP* parameter accounts for noise. The RL agent's goal is to collect the only diamond in the one room of the world and exit it. The agent collects a diamond by occupying the same square as the diamond. Possession of the diamond is indicated by a boolean variable, *have*.



(a)



(b)

Figure 3.4: (a) The Game domain. (b) Projections from the game world to the option world.

The room is also populated by 8 autonomous adversaries. They are of three types - benign, delayer or retriever. Of the 8, only one is a delayer and another one is a retriever, the other 6 are benign. If the RL agent occupies the same square as the delayer it is considered captured and is prevented from making a move for a random number of time steps determined by a geometric distribution with parameter *HOLD*. When in a different square from the agent, the delayer pursues the agent with probability *CHASE*. The benign adversaries execute random walks and behave as mobile obstacles. The retriever behaves like the benign adversary as long as the diamond is not picked by the agent. Once the agent picks up the diamond, the retriever behaves like a delayer. The important difference is that once the retriever and the agent occupy the same square, the diamond is returned to

its original position and the retriever reverts to being benign. The retriever also returns to being benign if the delayer has "captured" the agent. None of the adversaries can leave the room, and hence it is possible for the agent to "escape" from the room by exiting to the corridor. The agent is not aware of the types of the individual adversaries, nor is it aware of their *CHASE* and *HOLD* parameters. In every episode, a new pair of adversaries are chosen by the environment as the delayer and retriever. The RL agent can observe its own coordinates, the 8 adversaries' coordinates and the *have* variable.

3.6.2 Instruction Framework

Among the eight adversaries in the gameworld, only one is the delayer and one is the retriever. It is enough for the agent to observe these two adversaries to retrieve the diamond effectively. In other words, there are state features that can be ignored. Hence, we make projections of the states onto a subset of features resulting in a reduced world. A state in the game world is given by $s = \langle have, (x,y)_{agent}, (x,y)_{adv1}, \ldots, (x,y)_{adv8} \rangle$. The required state is given by $s_o = \langle have, (x,y)_{agent}, (x,y)_{del}, (x,y)_{ret} \rangle$. The projections used here are given by $\prod_{i \in D'} f_i(s)$, where $f_i : S \rightarrow S_i$ and D' is the reduced feature set. The agent uses Q-learning to learn the optimal policy on the reduced world.

The delayer and retriever change every episode and hence the corresponding projections also change. The agent does not know the true delayer and retriever. Φ type instructions are applicable here. We occasionally use these instructions to inform the agent about the indices of the adversaries that are the true delayer and true retriever. Suppose adv_k is the true delayer and adv_l is the true retriever for the current episode. The instruction $\Phi(s)$ gives the agent the $D' = \{have, (x,y)_{agent}, (x,y)_{adv_k}, (x,y)_{adv_l}\}$. When instructions are absent, the agent learns the correct (k,l) using a Bayesian weight update given by (Ravindran *et al.*, 2007).

3.6.2.1 Bayesian Weight Update

Consider the set of cross product functions f given by the subset of feature variables $D' = \{have, (x,y)_{agent}, (x,y)_{adv_i}, (x,y)_{adv_j}\}$. There are 8 possibilities for both adv_i and adv_j resulting in a set of 64 cross product functions f^m. The likelihood of any f^m being the required cross product function is maintained using a factored weight vector $\langle w_n^1(\cdot,\cdot), w_n^2(\cdot,\cdot)\rangle$, with one component each for the delayer and retriever. The retriever component captures the dependence of the retriever on the delayer.

$$w_n^l(f^m, \psi(s)) = \frac{\overline{P^l}((\rho_{J_m}(s), a, \rho_{J_m}(s')).w_{n-1}^l(f^m, \psi(s))}{\mathcal{K}} \tag{3.2}$$

where $\psi(s)$ is a function of s that captures the features of the states necessary to distinguish the particular sub-problem under consideration, s' is the next state in the gameworld, J_i is the corresponding subset of features to be used for projecting onto the reduced MDP. $\overline{P^l}(s,a,s') = max(v, P^l(s,a,s'))$. $\mathcal{K}$ is the normalizing factor. $P^l(s,a,s')$ is the projection of $P(s,a,s')$ onto the subset of features J_m required in the computation of $w_n^l(f^m, \psi(s))$. For details about the projection, refer to (Ravindran *et al.*, 2007).

3.6.3 Exploiting Instructions

In this section, we report additional experiments in which the agent exploits instructions in estimating *CHASE* and *HOLD* of the adversaries. The agent identifies the delayer and retriever assignment pattern in the environment based on these parameters. It uses this to reduce the number of updates required to identify the true delayer and retriever in the absence of instructions.

When the correct delayer-retriever pair is given as an instruction, the agent estimates the *CHASE* of the adversary that is the true delayer for the current episode. Similarly it can estimate the delayer's *HOLD* too. After a few such instructions, it would have good estimates of every adversaries' parameters. In order to show that this additional knowledge can be exploited effectively, we modify the game such that, for a given episode,

only an adversary with *CHASE* ≤ 0.7 is chosen by the environment to be the delayer. A retriever is chosen from those adversaries with *CHASE* more than 0.7. A classifier is used to learn this model as more and more instructions are received. Possible retrievers and delayers are predicted using this model. This prediction is used to reduce the number of updates the agent needs to perform. For example, let the *CHASE* of the adversaries be $\{0.5, 0.9, 0.6, 0.7, 0.4, 0.35, 0.8, 0.73\}$. An episode's delayer will be selected only from adversaries 1, 3, 4, 5 and 6 and the retriever from the rest. Once this classification is learnt, we can avoid updating the weights for $\{1, 3, 4, 5, 6\}$ while learning the correct retriever and $\{2, 7, 8\}$ while learning the correct delayer. Overall, the agent converges to the true likelihoods in lesser updates as shown for the delayer in Fig 3.5d.

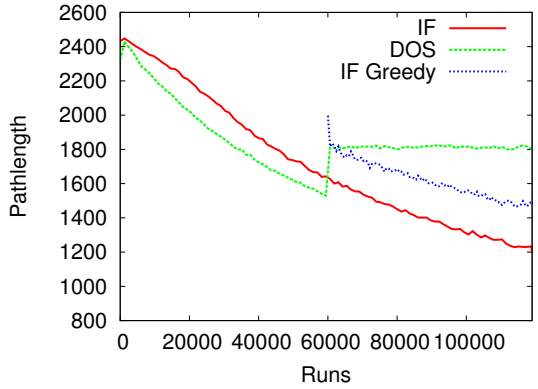
3.6.4 Deictic Option Schemas

Ravindran et al., (Ravindran *et al.*, 2007) proposed Deictic Option Schemas as an approach to solve tasks such as the Game Domain. In the DOS approach, the RL agent prelearns an optimal policy in a reduced MDP (training MDP) with $D' = \{have, (x, y)_{agent}, (x, y)_{adv_{del}}, (x, y)_{adv_{ret}}\}$. This is shown in Fig 3.4b. It lifts this optimal policy onto the gameworld MDP by choosing f^m according to the Bayesian weight updates. In this approach, f_i is $\rho_{D'}(s)$, where the projections are onto the option MDP. Here, learning takes place only in the training phase.

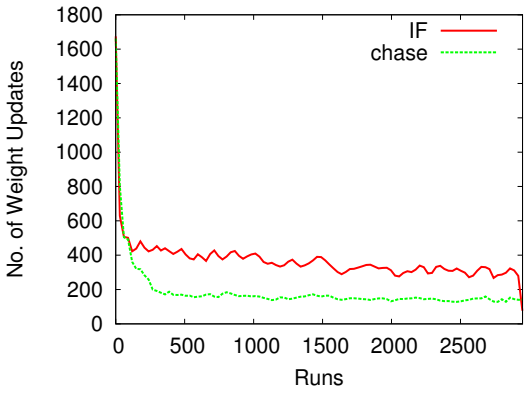
3.6.5 Results and Analysis

We compare the performance of our approach (labelled IF) with the DOS approach. In DOS, the agent works with the best estimate of the true delayer and retriever. Since the weights fluctuate heavily initially and we do not have a bound on when they converge, the agent would not know when to start estimating the *CHASE* of the true delayer. In some episodes, due to the randomness in the behaviour of the adversaries, the delayer likelihood estimates could be very wrong initially and takes a long time to converge to the correct ones. Hence it is not straight forward to estimate the adversaries' parameters in DOS.

DOS (Ravindran *et al.*, 2007) trains in the option MDP over 60000 episodes. IF does not have an exclusive learning phase, instead, depending on the availability of instructions, it alternates between learning using instructions and learning using the weight update. In this experiment, instructions were made available in randomly selected 60000 episodes of the total 120000. Hence all the weight updates shown for DOS occur after the learning phase. The graphs comparing the performance of Instruction Framework (IF) and DOS are shown in Fig 3.5.



(a) Pathlengths



(b) Exploiting Instructions

Figure 3.5: Graphs comparing IF and DOS. The graphs have been Bezier smoothed only for visibility purposes. The trends in data are evident even in the non-smoothed plots. All results have been averaged over 50 independent runs.

Fig 3.5a shows the number of time steps taken by each algorithm to solve the task successfully (collect the diamond and exit the room). It can be noticed that even though DOS performs well during the training phase, its performance drops in the game world. This is because of the differences in the training world and the game world in terms of obstacles, *CHASE* and *HOLD* parameters of adversaries. A major factor is the time taken

to identify the true delayer and true retriever, until which the agent has an incomplete understanding of the game world. The figure suggests that IF does not face this problem. Even though IF does not suffer from losses due to differences in worlds, it is affected by time taken to identify true delayer and retriever. This is masked in the plot as the episodes have been averaged over independent runs during which the same episode would have received instructions in some runs and would not have in other runs. In order to show that IF outperforms DOS, *IFgreedy* has been plotted.

At regular intervals, the IF algorithm was made to imitate DOS in the sense that IF behaved greedily based on the knowledge of the game world it had at that moment (including the knowledge gained due to the earlier instructions). In addition to this, there were no instructions available to IF during these episodes resulting in time being spent identifying the true delayer and the true retriever. It can be seen in the figure that even *IFgreedy* performs better than DOS proving that IF outperforms DOS.

The no. of weight updates required by IF to identify the true delayer and retriever are comparable to DOS. In Fig 3.5b, the no. of weight updates required to identify the true delayer using the classifier based on *CHASE* (plotted as chase) and without (plotted as IF) are shown. *chase* does not consider those adversaries classified as retrievers in identifying the delayer. Hence instead of updating the likelihood of 8 adversaries, it only updates 5. This implies lesser no. of weights to update and hence an earlier convergence to the true delayer. It can be seen that the no. of updates required is nearly half that required when we do not make use of *CHASE* values based classification.

Similar to the Transporter Domain experiments, here too an increase in fraction of instructed episodes would result in steeper learning curves and earlier convergence to the optimal policy. On the other hand, the behaviour change on decreasing this fraction is slightly complicated. As explained above, our approach also exploits instructions by learning patterns such as the *CHASE* based adversary selection. Scarce instructions can result in such models being faulty therefore having a detrimental effect on the learning process, therefore hindering the learner's progress rather than aiding it. This can be overcome by associating a confidence measure with the model similar to the approach employed in the

Transporter Domain.

CHAPTER 4

Multiple Interpretations

A 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 the following sections, we detail our observations of using a framework that enables a learner to handle such instructions and choose the best possible interpretation based on specific utility measures.

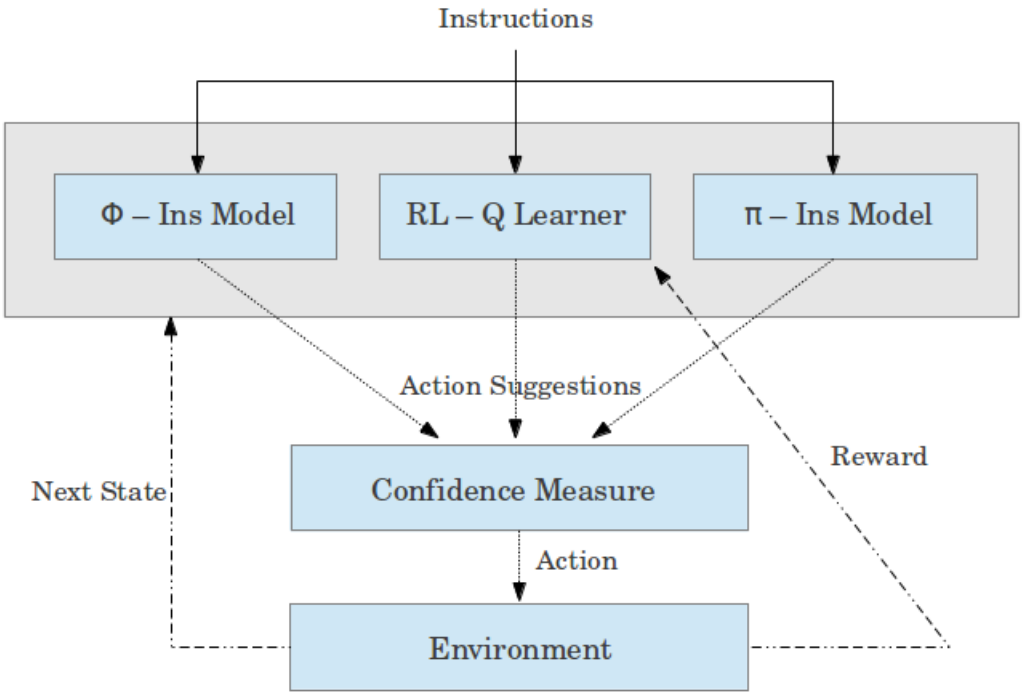


Figure 4.1: The Proposed Framework

The approach that we propose is shown in Fig 4.1 as a block diagram. Similar to the earlier proposed approach, an agent builds instruction models parallel to the RL framework. Unlike those approaches that dealt with only one type of instruction, here the agent interprets every instruction simultaneously as π-Ins and Φ-Ins. It builds independent models for each type and chooses the best among these two and the RL framework based on a confidence measure. We test this approach on the Sorter Domain.

4.1 Sorter Domain

The Sorter Domain (Fig 4.2) 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 any more. 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 4.2: The Sorter Domain

4.2 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 work, 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 $D' = \{Objectfeatures\}$.

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 2). 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 (Bradtke and Duff, 1994) that is independent of the instruction models.

Once the training period is over, the models $\hat{\Pi}_{\pi Ins}$ and $\hat{\Pi}_{\Phi Ins}$ 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) (Domingos *et al.*, 2006) 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. (Torrey *et al.*) 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. (Wang *et al.*, 2008*b*) 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.

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

$$\texttt{conf} = \max_a \hat{\Pi}(s,a) - \max_{b \neq a^*} \hat{\Pi}(s,b) \tag{4.1}$$

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

Algorithm 2 *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

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

The Φ – *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 work we use 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.

4.2.2 Learning the models

We transform the state features into a set of predicates as shown in Table 4.1 and ground them using the feature values. The instructions at a state are also transformed into predicates. For example, the π – *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 (Domingos *et al.*, 2006) to learn the structure of the

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 $c \in \{c1, c2, c3, c4, \dots\}$
inBasket(O)	TRUE if object O has been dropped into a basket
Botat(x)	TRUE if robot's current location is x such that $x \in \{O1, O2, O3, B1, B2, B3\}$

Table 4.1: State features as predicates

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 (Kok *et al.*) for the tasks mentioned above.

A state is represented thus : {*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)*}. 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 4.2.

The Table 4.2 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"

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 4.2: Some formulas learned and their weights.

etc.

4.3 Results and Analysis

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.

4.3.1 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

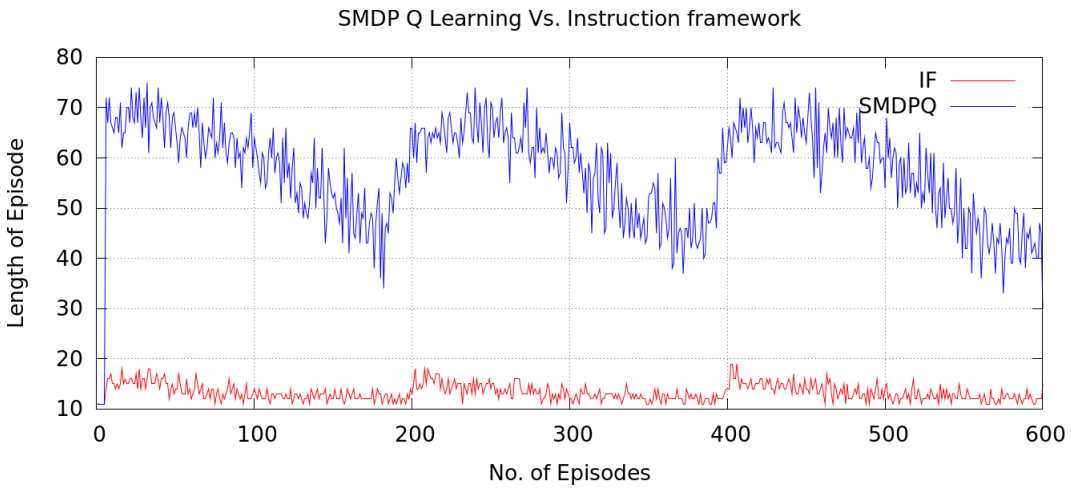


Figure 4.3: Comparison of IF with SMDP Q Learning

a new one resulting in a different set of states for the learners.

The SMDP Q-Learner's performance drops drastically (Fig 4.3) 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.

4.3.2 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

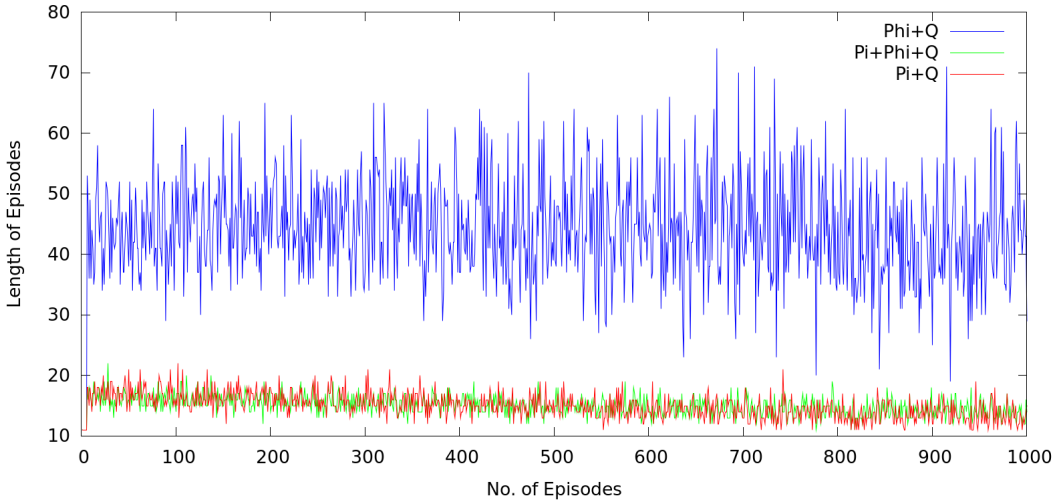


Figure 4.4: Comparison of $\hat{\Pi}_{\pi Ins}$, $\hat{\Pi}_{\Phi Ins}$ and IF.

that interprets them as both (IF). They are plotted as "Pi+Q", "Phi+Q" and "Pi+Phi+Q" respectively in Fig 4.4. 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.

It appears that B is stuck in a local maximum (40 steps taken to complete the task). This is a feature observed with several deixis (Agre, 1988; Sarah Finney and Oates, 2002) based approaches. The $\Phi - Instructions$ that we work with are similar to deictic operations as they bind the learner's focus onto specific objects.

4.3.3 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.

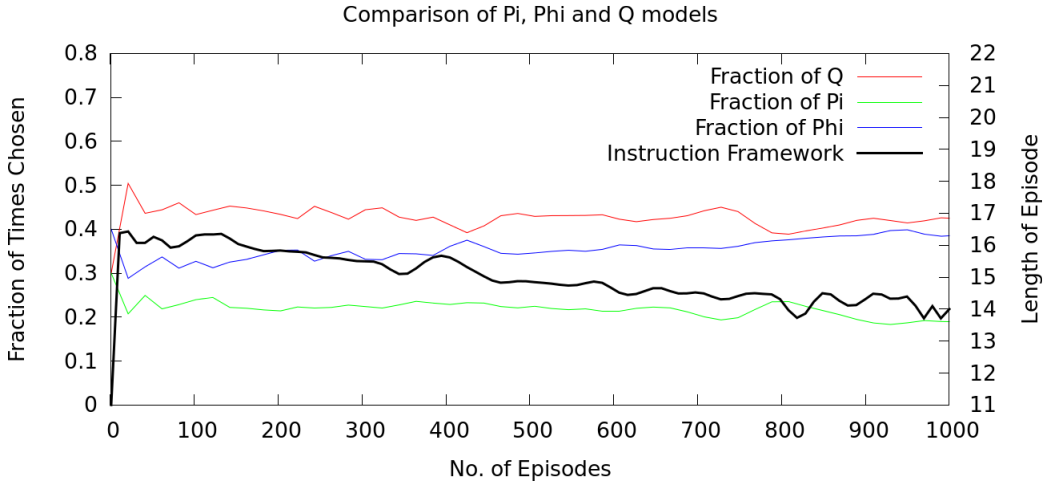


Figure 4.5: Comparison of percentage of times the action recommended $\hat{\Pi}_{\pi Ins}$, $\hat{\Pi}_{\Phi Ins}$ and SMDP Q-Learner models is taken. The proposed framework's performance (no. of steps taken to solve the task) has also been plotted. The plots have been Bezier smoothed for visibility purposes. The trends are evident in the unsmoothed data.

In Fig 4.5, 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. Another interesting observation from the above plots is that in Fig 4.4, "Pi+Q" and "Pi+Phi+Q" show very similar learning curves, although $\hat{\Pi}_{\Phi Ins}$ model is preferred nearly 20% of the time by "Pi+Phi+Q", it does not seem to affect the performance of IF

greatly. This is possibly because of the small state space and action space resulting in the $\hat{\Pi}_{\Pi Ins}$ model being able to easily compensate for the loss in performance due to the noisy action predictions of the $\hat{\Pi}_{\Phi Ins}$ model.

We ran 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 experimentation though.

4.4 The Sorter Robot

[1] The proposed framework was tested on our robot - Pioneer P3DX mobile robot base, Microsoft Kinect and a 5 DOF Robot arm (Fig 4.6). All robot controls were implemented using Robot Operating System (ROS) (Quigley *et al.*, 2009) packages. The various actions were executed using sampling based planners such as the Rapidly-Exploring Random Trees (RRT) (LaValle and Kuffner, 1999).

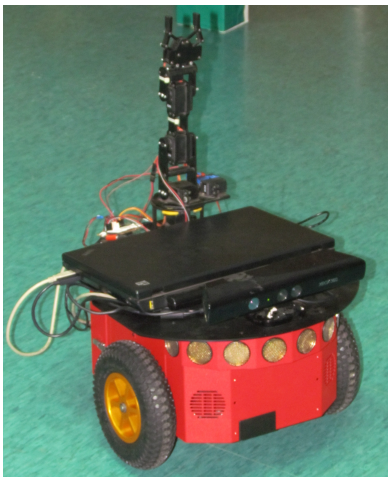
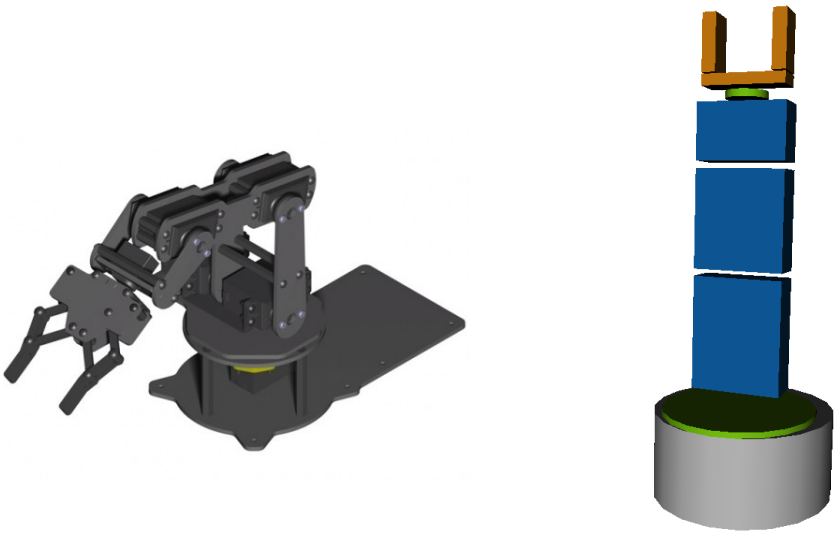


Figure 4.6: The robot - P3DX Mobile base, Microsoft Kinect sensor, 5-DOF Robot Arm

[1]The framework was implemented on this robot along with the RISE Robotics Group - S S Manimaran, Prahasaran Raja, Abhishek Mehta and Anshul Bhansal. A video of the robot in action can be found at http://bit.ly/riserobot.

4.4.1 5 DOF Robot Arm

We used a 5DOF Robot Arm to pickup and drop objects. The arm was controlled using the ROS openrave_planning stack and plans were generated using an RRT based planner. The arm's CAD model and visualization in OpenRave (Diankov and Kuffner, 2008) are shown in Fig 4.7.



(a) Arm Model (b) OpenRave Model

Figure 4.7: 5 DOF Robot Arm.

As mentioned earlier, we use the Kinect both as a sensor and an interaction device. Sensing is done by an on-board Kinect whereas a different Kinect placed outside the arena is used for instructing the robot.

4.4.2 Kinect - Sensing

The Kinect is used as a depth sensor to aid in navigation as well as identifying objects.

Navigation : The ROS Navigation stack (ROS) is used to control the motion of the P3DX. The stack makes use of the P3DX odometry information, depth information from the Kinect and the onboard Sonar range finder. The map of the world required for Naviga-

tion is generated using the ROS Gmapping stack (ROS) and is shown in Fig 4.8.

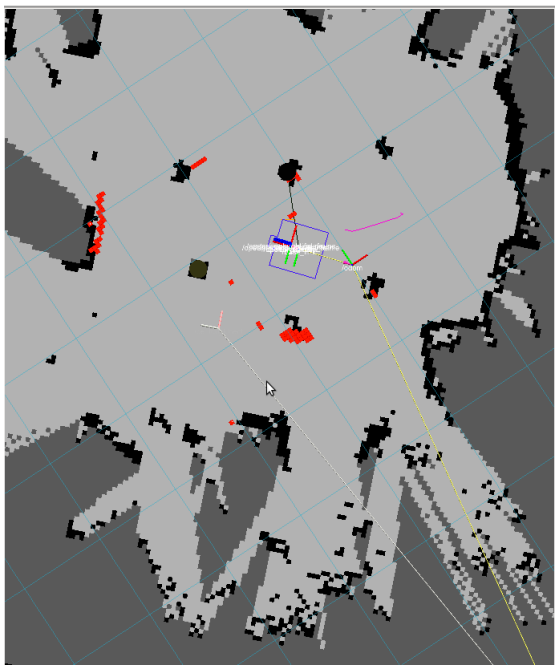
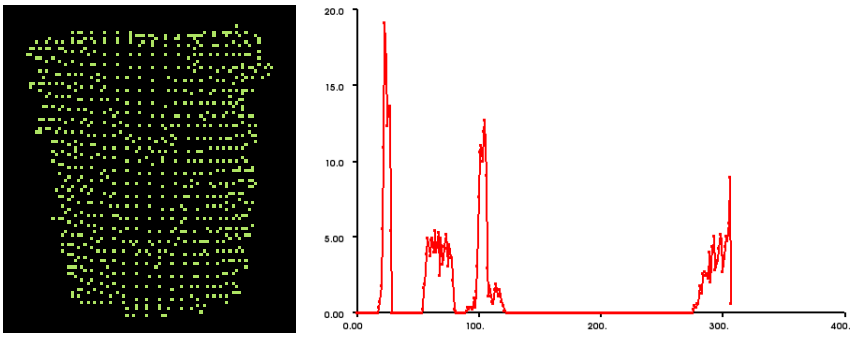


Figure 4.8: The map of the domain world generated using the Gmapping stack.

Object Detection : The Kinect sensor information is used to detect the objects and baskets. The Point Cloud Library (Rusu and Cousins, 2011) provides standard implementations for 3-D Object recognition and cluster extraction. The object recognition is done using the Viewpoint Feature Histogram (VFH) descriptors proposed by (Rusu *et al.*, 2010). We use cluster extraction to separate the objects in the scene. Further we compute the VFH descriptors for each object at various distances and use the object recognition implementation to recognize them. We combine the previously mentioned steps and port them to ROS as a robust online method for object identification. The depth image and VFH of a basket is shown in Fig 4.9.



(a) Depth Image (b) VFH Image

Figure 4.9: Basket.

4.4.3 Kinect - Interacting

We use the Kinect to recognize human gestures that serve as instructions to the robot. We use the ROS openni tracker package (OpenNI, 2010) for gesture recognition. This package recognizes a human, approximates him with a skeleton (Fig 4.10) and constantly tracks the skeleton's joints. We use the coordinates of the wrist and shoulder joints of the right hand to compute the direction being pointed at. By extending the line joining the two joints, we identify the target object. This constitutes an instruction $I(s)$.

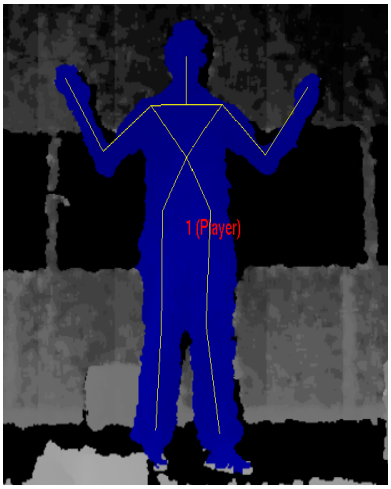


Figure 4.10: Tracking a human using OpenNI tracker.

The general flow of our experiment is given by the following steps :

- Robot scans the arena to identify objects and baskets and sample their colors.
- Instructor gives instructions.
- The Instruction models are built.
- Models are compared and actions are accordingly chosen over various episodes until optimality is achieved.

Although the Sorter domain requires an object to be put into a basket of the same color, while running the experiments on the robot we use identical objects placed on colored boxes. Detecting such tiny objects using the Kinect was unreliable. The task was slightly modified into putting an object on a particular box into a basket of the same color as the box. We spent some time on positioning the objects and baskets in the domain. This was to simplify the task of grasping the objects since we did not want to side track the main idea of the work by concentrating on other challenges.

CHAPTER 5

Conclusion

In this work, we introduced a classification of instructions and provided a mathematical formulation to incorporate the same into RL. In addition, we have presented a framework that enables the efficient exploitation of these instructions in identifying structural regularities in the world of interest. These patterns were used in generalizing the instructions thus speeding up learning. We work with inputs that are more general than earlier approaches resulting in possibly ambiguous interpretations. We extended our approach to handle such multiple interpretations and prune out the suboptimal ones. This framework was implemented using MLNs that proved to be effective in real world tasks. In this regard we implemented the framework on a real Sorter Robot. The chief contributions of the thesis can be summarized as :

- Explored a hitherto less explored aspect of learning beyond rewards and transitions in RL. Showed that it is beneficial to observe such system patterns to speed up learning.
- Classification of human-robot interactions into two types of instructions. Provided a mathematical formulation of the same for ease of integration with RL.
- Explored the benefits of using each type of instruction in challenging learning tasks - Transporter Domain and Game Domain. Highlighted the ease with which naive SL techniques can be combined with RL using our formulation for quicker learning.
- Discussed the ambiguity arising due to use of general instructions and provided a framework for efficiently handling the same.
- Provided an SRL approach to integrating instructions and reinforcement learners. The generalizing power of SRL techniques makes learning sample-efficient making our approach well suited for real world tasks.

Several earlier approaches to learning from humans employ trajectory based teaching. The learner assumes these trajectories to be optimal and optimizes its behaviour based on

observed rewards and transitions associated with the trajectories. In such approaches, the learner is restricted to the efficiency of the teacher. Also, the teacher is required to focus on demonstrating entire trajectories. In the real world, generating such training samples would be very expensive. In our approach, the teacher interacts with the learner through intermittent instructions thus relieving the teacher of extended periods of attention on the learner. This is beneficial especially in collaborative tasks, wherein the teacher and learner solve a task simultaneously and knowledge transfer happens through instructions. Since the interactions in our approach are intermittent, collecting these instructions in the real world from humans is comparatively less expensive than trajectories.

There have been earlier approaches that employ non-trajectory based interactions. As explained in the discussion on related work, our idea of instructions is motivated by these approaches (Chapman, 1991; Clouse and Utgoff, 1992). These approaches restrict the interactions to be pair-wise state preferences or use instructions to bind attention pointers or ground policies. In our work, we overcome the limitations of these approaches by providing a formulation of instructions that we believe is complete. Also we don't merely follow instructions but learn patterns that govern these instructions. These patterns aid in speeding up learning.

As mentioned in an earlier paragraph, most of these approaches are limited by the teacher's knowledge. Our approach combines building instruction models with exploration based learning (RL). Hence our learners retain their ability to learn on their own resulting in them improving upon the teacher's knowledge (Fig 4.3).

5.1 Limitations and Future Work

- Mapping human communication to instructions interpretable by a robot is a challenging task all by itself. In most of our experiments, we assume the existence of some mechanism that ensures human instructions are understood by the learner.
- We handle π-Ins and Φ-Ins independent of each other in all our experiments. In some real world tasks, useful instructions are a combination of both. For example, "Search in the black cupboard" - "Search" - π - Ins and "black cupboard" - Φ - Ins. Our approach currently cannot handle such complex instructions. An interesting direction

of future work would be to extend the current multiple interpretations approach to work with such instructions.

- The current confidence measures being used are qualitative at best. Working with better measures that can quantitatively compare the instruction models with the RL framework would aid in improving the approach's performance. Also, the current measures compute the confidence of the recommending model but do not take into account the rewards earned by following the chosen model's instructions. By including this information in the measures used, the current effectiveness of the instruction is also accounted for. This will be useful in domains that change over time.

- In the multiple interpretations framework, we disambiguate interpretation choice by choosing the interpretation model that is more confident about a particular option than others. Although this would work in most cases, it will result in a problem when there are more than one highly rewarding option. This raises the question about whether disambiguation based on such confidence measures is the right approach. This is a dilemma faced by most ensemble approaches wherein voting mechanisms have proved to be a better choice in some cases. Although in our work, the aim was to highlight the importance of proper interpretation of instructions as opposed to just finding the best options/actions, it would be interesting to explore the use of voting mechanisms wherein options that seem more profitable when confidence measures of both models are combined are chosen. In our setup, we do not expect this to make any difference since the Φ and Π models model the world independently and completely. Such approaches are applicable in scenarios where the models capture different aspects of the world and combining them supplements each other's knowledge.

- In the multiple interpretations framework, we use MLNs to learn the instruction models. Recently, there has been a surge in the number of SRL algorithms that learn from a small number of example trajectories. One could extend our current SRL based approach to use FODDs for generalization. As far as we are aware, there are no learning algorithms that exist for learning FODDs from example trajectories. It would be interesting to generalize instructions using FODDs.

- Learning from Oblivious instructors - Generalizing instructions from an expert who is oblivious to certain aspects of the world. For example, learning from an expert who is oblivious to the actual task but is confident about a few of the subtasks.

5.2 Publications Based on the Thesis

- Korupolu V N, P., and Ravindran, B. (2011) "Beyond Rewards: Learning with Richer Supervision". In the Proceedings of the 9th European Workshop on Reinforcement Learning (EWRL 2011).

- Korupolu V N, P., Sivamurugan S, M. and Ravindran, B. (2012) "Instructing a Reinforcement Learner". To appear in the Proceedings of the 25th Florida AI Research Society Conference (FLAIRS 2012).

- Korupolu V N, P., Sivamurugan S, M., Ravindran, B. and Natarajan, S. (2012) "Integrating Human Instructions and Reinforcement Learners : An SRL Approach" To appear in the Proceedings of the 2nd International Workshop on Statistical Relational AI (StarAI 2012).

REFERENCES

(). Ros stack list. URL http://www.ros.org/browse/list.php.

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

Agre, P. E. (1988). The dynamic strucutre of everyday life. Technical report.

Agre, P. E. and **D. Chapman**, What are plans for? *In Robotics and Autonomous Systems*, 17–34. MIT Press, 1989.

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

Atkeson, C. G. and **S. Schaal**, Robot learning from demonstration. *In Proceedings of the 14th ICML*, ICML '97. 1997.

Bahar, R. I., **E. A. Frohm**, **C. M. Gaona**, **G. D. Hachtel**, **E. Macii**, **A. Pardo**, and **F. Somenzi**, Algebraic decision diagrams and their applications. *In Proceedings of the 1993 ICCAD*, 188–191. IEEE Computer Society Press, Los Alamitos, CA, USA, 1993. ISBN 0-8186-4490-7. URL http://dl.acm.org/citation.cfm?id=259794.259826.

Bradtke, S. J. and **M. O. Duff**, Reinforcement learning methods for continuous-time markov decision problems. *In NIPS*, 393–400. 1994.

Brodie, M. and **G. DeJong**, Learning to ride a bicycle using iterated phantom induction. *In Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, 57–66. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999. ISBN 1-55860-612-2. URL http://dl.acm.org/citation.cfm?id=645528.657621.

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

Chapman, D., *Vision, instruction, and action*. MIT Press, 1991.

Cleary, M. E. (1997). *Systematic use of deictic commands for mobile robot navigation*. Ph.D. thesis, College of Computer Science, Northeastern University.

Clouse, J. A. and **P. E. Utgoff**, A teaching method for reinforcement learning. *In Proceedings of the 9th ICML*, ML92. 1992.

Diankov, R. and **J. Kuffner** (2008). Openrave: A planning architecture for autonomous robotics. Technical Report CMU-RI-TR-08-34, Robotics Institute, Pittsburgh, PA.

Domingos, P., **S. Kok**, **H. Poon**, **M. Richardson**, and **P. Singla**, Unifying logical and statistical ai. *In Proceedings of the 21st national conference on Artificial intelligence - Volume 1*, AAAI'06. 2006.

Dorigo, M. and **M. Colombetti** (1994). Robot shaping: developing autonomous agents through learning. *Artif. Intell.*, **71**, 321–370. ISSN 0004-3702. URL `http://portal.acm.org/citation.cfm?id=200949.200956`.

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

Hayes-Roth, F., **P. Klahr**, and **D. J. Mostow**, Advice-taking and knowledge refinement: An iterative view of skill acquisition. *In Cognitive Skills and Their Acquisition*. Lawrence Erlbaum Associates, 1981.

Kersting, K. and **K. Driessens**, Non-parametric policy gradients: a unified treatment of propositional and relational domains. *In Proceedings of the 25th international conference on Machine learning*, ICML '08, 456–463. ACM, New York, NY, USA, 2008. ISBN 978-1-60558-205-4. URL `http://doi.acm.org/10.1145/1390156.1390214`.

Khardon, R. (1999). Learning action strategies for planning domains. *Artificial Intelligence*, **113**, 125–148.

Kok, S. and **P. Domingos**, Learning the structure of markov logic networks. *In Proceedings of the 22nd international conference on Machine learning*, ICML '05, 441–448. ACM, New York, NY, USA, 2005.

Kok, S., **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.

LaValle, S. M. and **J. J. Kuffner**, Randomized kinodynamic planning. *In Proceedings IEEE International Conference on Robotics and Automation*, ICRA '99, 473–479. IEEE, 1999.

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

Maclin, R. and **J. W. Shavlik**, Creating advice-taking reinforcement learners. *In* **S.Thrun** and **L.Pratt** (eds.), *Learning to Learn*, 22, 251–281. Kluwer Academic Publishers, 1998.

Natarajan, S., **S. Joshi**, **P. Tadepalli**, **K. Kersting**, and **J. W. Shavlik**, Imitation learning in relational domains: A functional-gradient boosting approach. *In In Proceedings of IJCAI '11*.

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

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

Ng, A. Y. and **S. Russell**, Algorithms for inverse reinforcement learning. *In Proceedings of Seventeenth International Conference on Machine Learning*. 2000*b*.

OpenNI (2010). *OpenNI User Guide*. OpenNI organization. URL `http://www.openni.org/documentation`. Last viewed 19-01-2011 11:32.

Otterlo, M. V. (2005). A survey of reinforcement learning in relational domains. Technical report, CTIT Technical Report Series.

Prasad Tadepalli, R. G. and **K. Driessens**, Relational reinforcement learning: An overview. *In In Proceedings of the ICML04 Workshop on Relational Reinforcement Learning*. 2004.

Price, B. and **C. Boutilier** (2003). Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research*.

Quigley, M., **B. Gerkey**, **K. Conley**, **J. Faust**, **T. Foote**, **J. Leibs**, **E. Berger**, **R. Wheeler**, and **A. Y. Ng**, Ros: an open-source robot operating system. *In Proceedings of the Open-Source Software workshop at the International Conference on Robotics and Automation (ICRA)*. Los Angeles, USA, 2009.

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

Ravindran, B., **A. G. Barto**, and **V. Mathew**, Deictic option schemas. *In Proceedings of the Twentieth International Joint Conference on Artifical Intelligence*, 1023–1028. AAAI Press, 2007. URL `http://portal.acm.org/citation.cfm?id=1625275.1625441`.

Rosenstein, M. T. and **A. G. Barto**, Supervised actor-critic reinforcement learning. *In* **W. P. J. Si, A. Barto** and **D. Wunsch** (eds.), *Learning and Approximate Dynamic Programming: Scaling Up to the Real World*, 359–380. John Wiley and Sons, 2004. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.65.6362`.

Rusu, R. B., **G. Bradski**, **R. Thibaux**, and **J. Hsu**, Fast 3D Recognition and Pose Using the Viewpoint Feature Histogram. *In Proceedings of the 23rd IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Taipei, Taiwan, 2010.

Rusu, R. B. and **S. Cousins**, 3d is here: Point cloud library (pcl). *In IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China, 2011.

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

Sanner, S. and **C. Boutilier**, Approximate linear programming for first-order mdps. *In In Proc. UAI05, 509 517*. 2005.

Sarah Finney, L. P. K., Natalia H. Gardiol and **T. Oates**, That thing we tried didnt work very well: Deictic representation in reinforcement learning. *In Proceedings of the Eighteenth Uncertanity in Artificial Intelligence conference*. 2002.

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

Sutton, R. and **A. Barto**, *Reinforcement Learning : An Introduction*. MIT Press, 1998.

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

Tesauro, G., Temporal difference learning of backgammon strategy. *In ML*, 451–457. 1992.

Thomaz, A., **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.*, 352 –357. 2006.

Torrey, L., **J. Shavlik**, **S. Natarajan**, **P. Kuppili**, and **T. Walker**, Transfer in reinforcement learning via markov logic networks. *In Proceedigns of AAAI workshop on Transfer Learning for Complex Tasks 2008*.

Torrey, L., **J. Shavlik**, **T. Walker**, and **R. Maclin**, Relational macros for transfer in reinforcement learning. *In ILP*. 2007.

Torrey, L., **T. Walker**, **R. Maclin**, and **J. Shavlik** (2009). Advice taking and transfer learning: Naturally inspired extensions to reinforcement learning.

Utgoff, P. E. and **J. A. Clouse**, Two kinds of training information for evaluation function learning. *In Proceedings of the Ninth National Conference on Artificial intelligence - Volume 2*, AAAI'91. 1991.

Wang, C., **S. Joshi**, and **R. Khardon** (2008*a*). First order decision diagrams for relational mdps. *JAIR*, **31**(1), 431–472. ISSN 1076-9757. URL http://dl.acm.org/citation.cfm?id=1622655.1622668.

Wang, W., **Y. Gao**, **X. Chen**, and **S. Ge**, Reinforcement learning with markov logic networks. *In Proceedigns of European Workshop on Reinforcement Learning 2008*. 2008*b*.

Watkins, C. J. C. H. and **P. Dayan** (1992). Q-learning. *Machine Learning*. URL http://jmvidal.cse.sc.edu/library/watkins92a.pdf.

Zeigler, B. P. (1972). Toward a formal theory of modeling and simulation: Structure preserving morphisms. *J. ACM*, **19**.