Deictic Option Schemas

Balaraman Ravindran
Dept. of Computer Science and Engineering
IIT Madras, India
ravi@cse.iitm.ac.in

Andrew G. Barto
Dept. of Computer Science
University of Massachusetts, Amherst
barto@cs.umass.edu

Vimal Mathew
Dept. of Computer Science and Engineering
IIT Madras, India
vml@cse.iitm.ac.in

Abstract

Deictic representation is a representational paradigm, based on selective attention and pointers, that allows an agent to learn and reason about rich complex environments. In this article we present a hierarchical reinforcement learning framework that employs aspects of deictic representation. We also present a Bayesian algorithm for learning the correct representation for a given sub-problem and empirically validate it on a complex game environment.

1 Introduction

Agre [1987; 1988] introduced deictic representations to the AI community as a way of reasoning and learning in large complex domains. It is a representational paradigm based on using a small set of pointers that lets the agent focus on parts of the state space relevant to solving the problem at hand.

Deictic pointers might be simple physical locators or they might encode complex semantics. Agre and Chapman [1987] employ pointers that let the agent precisely locate important components of the system, but their method needs substantial pre-processing and domain knowledge. While solving the arcade game Pengo, their agent Pengi employs complex pointers such as *bee-attacking-me*, *ice-cube-next-to-me*, etc. The actions of the agent are then defined with respect to these pointers, for example, *push ice-cube-next-to-me* toward *bee-attacking-me*. In general, deictic representations can be used in rich environments with incredible amounts of detail. They are also useful in systems where there are physical limitations on the sensory capabilities so that some form of attentional mechanism has to be employed [Minut and Mahadevan, 2001].

There has been considerable interest in employing deictic representations in a reinforcement learning (RL) framework [Whitehead and Ballard, 1991; McCallum, 1995; Finney *et al.*, 2002]. The attractions of such a synthesis are obvious and can lead to a trial-and-error approach that can work in large environments. In this article we develop a hierarchical deictic RL framework based on relativized options [Ravindran and Barto, 2003a]. Specifically we extend our earlier approach to factored MDPs and show that certain aspects of deixis can be modeled as finding structured homomorphic reductions of the problem. We present a Bayesian algorithm for learning the correct configuration of pointers and validate our approach on a simulated game domain inspired by Agre and Chapman [1987].

We employ Markov Decision Processes (MDPs) as our basic modeling paradigm. First we present some notation regarding factored MDPs (Section 2), a brief summary of relativized options (Section 3) and introduce deictic option schemas(Section 4). Then we present our Bayesian algorithm to choose the right pointer configuration to apply to a sub-task (Section 5). We relate our approach to an existing family of deictic algorithms in Section 6. In Section 7 we describe some related work, and we conclude in Section 8 by discussing some future directions for research.

2 Notation

A structured (finite) MDP is described by the tuple $\langle S, A, \Psi, P, R\rangle$, where S is a finite set of states, A is a finite set of actions, $\Psi \subseteq S \times A$ is the set of admissible state-action pairs, $P : \Psi \times S \rightarrow [0, 1]$ is the transition probability function with $P(s, a, s')$ being the probability of transition from state s to state s' under action a, and $R : \Psi \rightarrow \mathbb{R}$ is the expected reward function, with $R(s, a)$ being the expected reward for performing action a in state s. The state set S is given by M features or variables, $S \subseteq \prod_{i=1}^{M} S_i$, where S_i is the set of permissible values for feature i. Thus any $s \in S$ is of the form $s = \langle s_1, \ldots, s_M\rangle$, where $s_i \in S_i$ for all i.

The transition probability function P is usually described by a family of two-slice *temporal Bayesian networks* (2-TBNs), with one TBN for each action. A 2-TBN is a two layer directed acyclic graph whose nodes are $\{\mathbf{s_1}, \ldots, \mathbf{s_M}\}$ and $\{\mathbf{s'_1}, \ldots, \mathbf{s'_M}\}$. Here $\mathbf{s_i}$ denotes the random variable representing feature i at the present state and $\mathbf{s'_i}$ denotes the random variable representing feature i in the resulting state. Many classes of structured problems may be modeled by a 2-TBN in which each arc is restricted to go from a node in the first set to a node in the second. The state-transition probabilities can be factored as:

$$P(s, a, s') = \prod_{i=1}^{M} \text{Prob}(s'_i|\text{Parents}(\mathbf{s'_i}, a)),$$

where Parents$(\mathbf{s'_i}, a)$ denotes the parents of node $\mathbf{s'_i}$ in the 2-TBN corresponding to action a and each

$\text{Prob}(s'_i|\text{Parents}(\mathbf{s'_i}, a))$ is given by a conditional probability table (CPT) associated with node $\mathbf{s'_i}$. In computing the conditional probabilities it is implicitly assumed that the nodes in Parents($\mathbf{s'_i}$,a) are assigned values according to s.

An option (or a temporally extended action) [Sutton *et al.*, 1999] in an MDP $\mathcal{M} = \langle S, A, \Psi, P, R\rangle$ is defined by the tuple $O = \langle \mathcal{I}, \pi, \beta\rangle$, where the initiation set $\mathcal{I} \subseteq S$ is the set of states in which the option can be invoked, π is the option policy, and the termination function $\beta : S \rightarrow [0, 1]$ gives the probability of the option terminating in any given state. The option policy can in general be a mapping from arbitrary sequences of state-action pairs (or histories) to action probabilities. We restrict attention to Markov sub-goal options in which the policies are functions of the current state alone, and an option terminates on reaching a set of pre-defined (sub)goal states. The states over which the option policy is defined is known as the *domain* of the option. In such cases the option policy can be defined in a sub-MDP, known as the option MDP, consisting of the states in the domain of the option.

3 Relativized Options

A relativized option combines MDP homomorphisms [Ravindran and Barto, 2003b] with the options framework to compactly represent a family of related options. An MDP homomorphism is a transformation from an MDP $\mathcal{M}$ to a reduced model $\mathcal{M}'$ such that a solution to $\mathcal{M}'$ yields a solution to $\mathcal{M}$. Notationally, an MDP homomorphism, h, is defined as a surjection from Ψ to Ψ', given by a tuple of surjections $\langle f, \{g_s|s \in S\}\rangle$, with $h((s, a)) = (f(s), g_s(a))$, where $f : S \rightarrow S'$ and $g_s : A_s \rightarrow A'_{f(s)}$ for $s \in S$. $\mathcal{M}'$ is called the *homomorphic image* of $\mathcal{M}$ under h. An optimal policy in $\mathcal{M}'$ when lifted to $\mathcal{M}$ yields an optimal policy in $\mathcal{M}$.

In a relativized option, the policy for achieving the option's sub-goal is defined in the image of a partial homomorphism defined only over a subset of S. This image is called the option MDP. When the option is invoked, the current state is projected onto the option MDP, $\mathcal{M}_O = \langle S_O, A_O, \Psi_O, P_O, R_O\rangle$, and the policy action is lifted to the original MDP based on the states in which the option is invoked. A relativized option is defined as follows:

Definition: A *relativized option* of an MDP $\mathcal{M} = \langle S, A, \Psi, P, R\rangle$ is the tuple $O = \langle h, \mathcal{M}_O, \mathcal{I}, \beta\rangle$, where $\mathcal{I} \subseteq S$ is the initiation set, $\beta : S_O \rightarrow [0, 1]$ is the termination function and $h = \langle f, \{g_s|s \in S\}\rangle$ is a partial homomorphism from the MDP $\langle S, A, \Psi, P, R'\rangle$ to the option MDP $\mathcal{M}_O$ with R' chosen based on the sub-task.

In other words, the option MDP $\mathcal{M}_O$ is a partial homomorphic image of an MDP with the same states, actions and transition dynamics as $\mathcal{M}$ but with a reward function chosen based on the option's sub-task. The homomorphism conditions hold only for states in the domain of the option O. The option policy $\pi : \Psi_O \rightarrow [0, 1]$ is obtained by solving $\mathcal{M}_O$ by treating it as an episodic task. When lifted to $\mathcal{M}$, π is suitably transformed into policy fragments over Ψ.

A relativized option can be viewed as an *option schema* where a template of an option policy is specified using a parameterized representation of a family of sub-problems. When the option is invoked, a particular instantiation of the schema is chosen by binding to the appropriate resources. Given a set of possible bindings, [Ravindran and Barto, 2003a] presented a Bayesian approach to choosing the right binding to apply in a given context based on experience gathered in solving the task. It assumed that the set of bindings were given by a family of transformations, $\mathcal{H}$, applied to Ψ and maintained a heuristic weight vector, $w_n(h, \psi(s))$, which is a measure of the likelihood of transformation $h \in \mathcal{H}$ being the right transformation in the context represented by $\psi(s)$.[1] The weight vectors are updated using:

$$w_n(h, \psi(s)) = \frac{\overline{P_O}((f(s), g_s(a), f(s'))w_{n-1}(h, \psi(s))}{\mathcal{K}} \quad (1)$$

where $\overline{P_O}(s, a, s') = \max(\nu, P_O(s, a, s'))$, and $\mathcal{K} = \sum_{h' \in \mathcal{H}} \overline{P_O}((f'(s), g'_s(a), f'(s'))w_{n-1}(h', \psi(s))$ is a normalizing factor. Since the projected transition probability is lower bounded by ν, this approach works even when the homomorphic image is not exact. In this article we extend our earlier work to cases where the bindings are specified by deictic pointers.

4 Deictic Option Schemas

The set of candidate transformations, $\mathcal{H}$, can be specified by a set of deictic pointers together with their possible configurations. The agent learns to place the pointers in specific configurations to effect the correct bindings to the schema. We call such option schema together with the set of pointers a *deictic option schema*. Formally a deictic option schema is defined as follows:

Definition: A *deictic option schema* of a factored MDP $\mathcal{M} = \langle S, A, \Psi, P, R\rangle$ is the tuple $\langle \mathcal{D}, O\rangle$, where $O = \langle h, \mathcal{M}_O, \mathcal{I}, \beta\rangle$, is a relativized option in $\mathcal{M}$, and $\mathcal{D} = \{D_1, D_2, \cdots, D_K\}$ is the set of admissible configurations of the deictic pointers, with K being the number of deictic pointers available. For all i, $D_i \subseteq 2^{\{1,\cdots,M\}}$ is the collection of all possible subsets of indices of the features that pointer i can project onto the schema, where M is the number of features used to describe S.

The set D_i indicates the set of objects that pointer i can point to in the environment. For example, in a blocks world domain this is the set of all blocks. In our example in the next section, it is the set of all possible adversaries.

Once a deictic option schema is invoked the decision making proceeds in two alternating phases. In the first phase the agent picks a suitable pointer configuration, using a heuristic weight function similar to the one given in Section 3. In the second phase, the agent picks an action, based on the perceptual information obtained from the pointers, using a Q-function. This calling semantics is similar to that followed by Whitehead and Ballard [1991].

Each member of $\mathcal{H}$ has a state transformation of the form $\prod_{i=1}^{K} \rho_{J_i}$, where $J_i \in D_i$ for all i and ρ_J, is the projection of S onto the subset of features indexed by J. If h is

[1] Frequently, $\psi(s)$ is some simple function of the state space, like a projection onto a few features.

known a priori then the pointer configurations can be chosen appropriately while learning. In the absence of prior knowledge the Bayesian algorithm developed in [Ravindran and Barto, 2003a] can be used to determine the correct bindings to the schema from among the possible pointer configurations. But, the algorithm is not entirely suitable for deictic option schemas for the following reason.

The algorithm assumes that the candidate transformations are not structured and maintains a monolithic weight vector, $w_n(\cdot,\cdot)$. In the case of deictic option schemas the transformations are structured and it is advantageous to maintain a "factored" weight vector, $w_n(\cdot,\cdot) = \langle w_n^1(\cdot,\cdot), w_n^2(\cdot,\cdot), \cdots \rangle$. Ideally each component of the weight vector should be the likelihood of the corresponding pointer being in the right configuration. But usually there is a certain degree of dependence among the pointers and the correct configuration of some pointers might depend on the configuration of other pointers.

Therefore, three cases need to be considered. Assume that there are only two pointers, i and j, for the following discussion, but the concepts generalize to arbitrary number of pointers.

1. *Independent pointers:* For every $J \in D_i$, ρ_J satisfies the homomorphism condition on transition probabilities. Then, the right assignment for pointer i is independent of the other pointers and there is one component of the weight vector corresponding to pointer i and the updates for this components depends only on the features indexed by some $J \in D_i$.
2. *Mutually dependent pointers:* For each $J \in D_i$ and $J' \in D_j$, $\rho_J \times \rho_{J'}$ satisfies the homomorphism conditions. But ρ_J and $\rho_{J'}$ do not satisfy the homomorphism conditions for some $J \in D_i$ and $J' \in D_j$. Thus, they cannot be treated separately and the composite projections given by their cross-products has to be considered. There is one component of the weight vector that corresponds to this cross-product projection. The update for this component will depend on the features indexed by some $J \in D_i$ and $J' \in D_j$.
3. *Dependent pointer:* For each $J \in D_i$ and $J' \in D_j$, $\rho_J \times \rho_{J'}$ satisfies the homomorphism conditions, as does ρ_J. But $\rho_{J'}$ does not satisfy the homomorphism conditions for at least some value of $J' \in D_j$. This means pointer i is an independent pointer, while j is a dependent one. There is a separate component of the weight vector that corresponds to pointer j, but whose update depends on the features indexed by some $J \in D_i$ and $J' \in D_j$.

The weight vector is chosen such that there is one component for each independent pointer, one for each dependent pointer and one for each set of mutually dependent pointers. Let the resulting number of components be L. A modified version of the update rule in [Ravindran and Barto, 2003a] is used to update each component l of the weight independently of the updates for the other components:

$$w_n^l(h^i, \psi(s)) = \frac{\overline{P_O^l}((f^i(s), g_s^i(a), f^i(s')) \cdot w_{n-1}^l(h^i, \psi(s))}{\mathcal{K}} \quad (2)$$

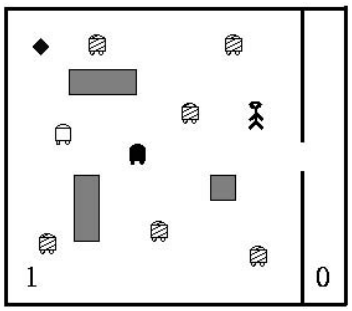

Figure 1: A game domain with interacting adversaries and stochastic actions. The task is to collect the black diamond. The adversaries are of three types—benign (shaded), retriever (white) and delayers (black). See text for more explanation.

where $\overline{P_O^l}(s,a,s') = \max\left(\nu, P_O^l(s,a,s')\right)$, $\psi(s)$ is again a function of s that captures the features of the states necessary to distinguish the particular sub-problem under consideration, and $\mathcal{K} = \sum_{h'^i \in \mathcal{H}} \overline{P_O^l}(f'^i(s), g_s'^i(a), f'^i(s'))$ $w_{n-1}(h'^i, \psi(s))$ is the normalizing factor. $P_O^l(s,a,s')$ is a "projection" of $P_O(s,a,s')$ computed as follows. Let J be the set of features that is required in the computation of $w_n^l(h^i, \psi(s))$. This is determined as described above for the various cases. Then $P_O^l(s,a,s') = \prod_{j \in J} \text{Prob}(s'_j | \text{Parents}(s'_j, a))$.

5 Experimental Illustration in a Game Environment

We now apply a deictic option schema to learning in a modified version of the game environment introduced in [Ravindran and Barto, 2003a]. The layout of the game is shown in Figure 1. The environment has the usual stochastic gridworld dynamics. There is just one room in the world and the goal of the agent is to collect the diamond in the room and exit it. The agent collects a diamond by occupying the same square as the diamond. A Boolean variable *have* indicates possession of the diamond.

The room also has 8 autonomous adversaries. The adversaries may be of three types—benign, delayer or retriever. If the agent happens to occupy the same square as the delayer it is *captured* and is prevented from moving for a random number of time steps determined by a geometric distribution with parameter *hold*. When not occupying the same square, the delayer pursues the agent with probability *chase*. The benign robots execute random walks in the room and act as mobile obstacles. The retriever behaves like the benign adversary till the agent picks up the diamond. Once the agent picks up the diamond, the retriever's behavior switches to that of the delayer. The main difference is that once the retriever occupies the same square as the agent, the diamond is returned to the original position and the retriever reverts to benign behavior. The retriever returns to benign behavior if the agent is also "captured" by the delayer. None of the adversaries leave the room, and thus the agent can "escape" from the room by exiting to the corridor. The agent is not aware of the types of the

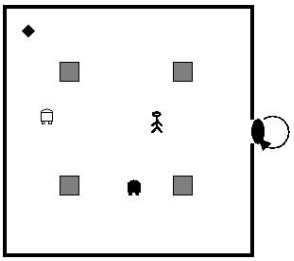

Figure 2: The option MDP corresponding to the sub-task *get-object-and-leave-room* for the domain in Figure 1. There is just one delayer and one retriever in this image MDP.

individual adversaries.

The option MDP (Figure 2) is a symmetrical room with just two adversaries—a delayer and a retriever with fixed *chase* and *hold* parameters. The features describing the state space of the option MDP consists of the x and y coordinates relative to the room of the agent and of the adversaries and a boolean variable indicating possession of the diamond. The room in the world does not match the option MDP exactly and no adversary in the world has the same chase and hold parameters as the adversaries here.

The deictic agent has access to 2 pointers: a *delayer* pointer that projects one of the adversaries onto the delayer in the image MDP and a *retriever* pointer that projects one of the adversaries onto the retriever in the image MDP. The *delayer* pointer is an independent pointer and the *retriever* pointer is dependent on the delayer pointer. The sets $D_{delayer}$ and $D_{retriever}$ are given by the 8 pairs of features describing the adversary coordinates.

In addition to the pointers the agent also has access to some background information, such as its own location (which can be formalized as a self pointer) and whether it has the diamond or not. Note that since the option MDP is an approximate homomorphic image, the homomorphism conditions are not strictly met by any of the projections. Therefore, in computing the weight updates, the influence of the features not used in the construction of the image MDP are ignored by marginalizing over them.

5.1 Experimental Results

The performance of the deictic agent is compared with a relativized agent (monolithic agent) that employs the same option MDP but chooses from a set $\mathcal{H}$ of 64 monolithic transformations, formed by the cross product of the 8 configurations of the deictic pointers. Both agents employ hierarchical SMDP Q-learning, with the learning rates for the option and the main task set to 0.1. The agents are both trained initially in the option MDP to acquire an approximate initial option policy that achieves the goal some percentage of the trials, but is not optimal. Both agents use ϵ greedy exploration. The results reported are averaged over 10 independent runs.

On learning trials both agents perform similarly (Figure 3), but the monolithic agent has a marginally better initial performance. To understand this we look at the rates at which the transformation weights converge (Figures 4, 5, and 6 are for

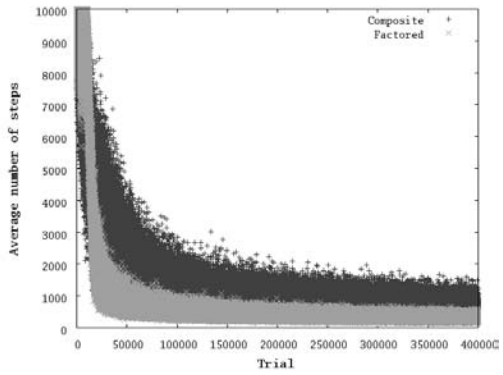

Figure 3: Average number of steps per episode taken by both agents for solving the task shown in Figure 1.

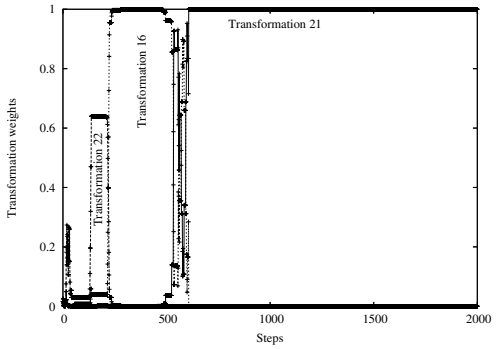

Figure 4: Typical evolution of a subset of weights of the monolithic agent on the task shown in Figure 1.

a single typical run). Figure 5 shows that typically the deictic agent identifies the *delayer* quite rapidly. In fact it takes only an average of 52 update steps to identify the delayer, over the 10 independent runs. The monolithic agent takes much longer to identify the right transformation (number 21 in our encoding), as seen from Figure 4. On an average it takes around 2007 update steps. As Figure 6 shows, identifying the *retriever* is harder and the deictic agent takes about 3050 update steps on an average.

This result is not surprising, since the correct position for the *retriever* depends on position of the *delayer* pointer. Therefore, while the *delayer* is being learned, the weights for the *retriever* receive inconsistent updates and it takes a while for the weights to get back on track. For much of the time we are interested in only identifying the *delayer* correctly and hence the deictic agent seems to have lower variability in its performance. Overall a single run with the composite agent takes around 14 hours of CPU time on a Pentium IV, 3.4 GHz machine, while a single run of the deictic agent takes around 4 hours. Further, the monolithic agent considers all possible combinations of pointer configurations simultaneously. Therefore, while it takes fewer update steps to converge to the right weights, the deictic agent makes far fewer number of weight updates: 130,000 vs. 85,000 on an average.

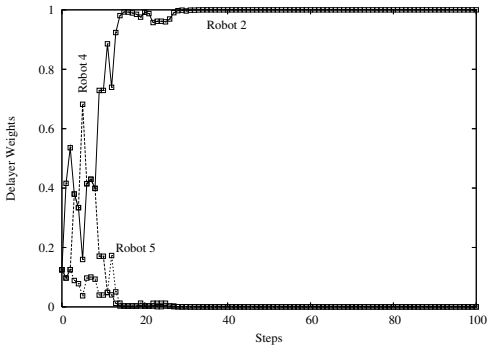

Figure 5: Typical evolution of a subset of the *delayer* weights of the deictic agent on the task shown in Figure 1.

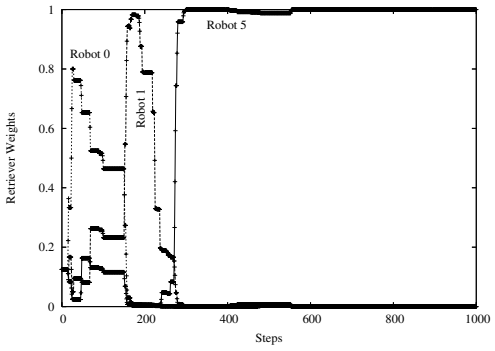

Figure 6: Typical evolution of a subset of the *retriever* weights on the task shown in Figure 1.

Comment

The algorithm used above updates the weights for all the transformations after each transition in the world. This is possible since the transformations are assumed to be mathematical operations and the agent could use different transformations to project the same transition onto to the option MDP. But deictic pointers are often implemented as physical sensors. In such cases, this is equivalent to sensing every adversary in the world before making a move and then sensing them after making the move, to gather the data required for the updates. Since the weights converge fairly rapidly, compared to the convergence of the policy, the time the agent spends "looking around" would be a fraction of the total learning time.

6 Perceptual Aliasing and Consistent Representations

The power of deixis arises from its ability to treat many perceptually distinct states in the same fashion, but it is also the chief difficulty in employing deictic representations. This phenomenon is known as *perceptual aliasing* [Whitehead and Ballard, 1991]. One approach to overcome perceptual aliasing is a class of methods known as *Consistent Representation Methods*. These methods split the decision making into two phases: in the perceptual phase the agent looks around the environment to find a consistent representation of the underlying state. A consistent representation [Whitehead and Ballard, 1991] of a state is one such that all states that map to the representation have the same optimal action-value function. In the overt phase the agent picks an action to apply to the environment based on the current sensory input. Learning takes place in both phases. The *Lion* algorithm [Whitehead and Ballard, 1991] is an example of a consistent representation algorithm. Here Q-learning is used in the overt phase and a simple learning rule based on one step error information is used to train the sensory phase. If the one step error in the Q update rule for a particular configuration is negative then that representation is considered perceptually aliased and is ignored in the future. This simple rule limits the applicability of this algorithm to deterministic settings alone.

If the representation used is a homomorphic image then it is a consistent representation, as mentioned in [Ravindran and Barto, 2003b]. By restricting the definition of deictic option schema to employ partial homomorphic images as option MDPs, it is guaranteed that a consistent representation is always employed. In the absence of knowledge of the option homomorphism, finding the right transformation to employ constitutes the search for a consistent representation and we employ Bayesian learning in this phase. As with the Lion algorithm, a form of Q-learning is used in the overt phase.

7 Related Work

Deixis originates from the Greek word *deiknynai* which means to show or to point out. It is employed by linguists to denote the pointing function of certain words, like *here* and *that*, whose meaning could change depending on the context. Deixis was introduced to the AI community by Agre. As mentioned earlier, [Agre and Chapman, 1987] used deictic representations to design an agent, Pengi, that plays the arcade game Pengo. Pengi was designed to play the game from the view point of a human player and hence used visuals from a computer screen as input.

[Whitehead and Ballard, 1991] were the first to use deictic representations in a RL system, with their *Lion* algorithm. Unfortunately, the method the Lion algorithm employs to determine consistency works only in deterministic environments. [McCallum, 1995] takes a more direct approach to overcoming perceptual aliasing. He employs deixis to solve a car driving task and models the problem as a partially observable MDP. He uses a tree structure, known as U-trees, for representing "states" and identifies the necessary distinctions so that the resulting representation is consistent. But his approach is not divided into explicit perceptual and overt phases. There has not been much work on using hierarchical RL and deixis. The only work we are aware of is by [Minut and Mahadevan, 2001]. They develop a selective attention system that searches for a particular object in a room. It operates by identifying the most salient object in the agent's visual field and shifting its visual field to center and focus on that object. They employ an option to identify the most salient object in the current visual field. Though they do not state it thus, this is a "deictic" option, whose effect depends on the

current visual field.

A systematic study on using deictic representations with RL was reported by [Finney *et al.*, 2002]. They employ a straightforward deictic representation with two pointers on a blocks world task. They use the G-algorithm to represent past information as a tree. They report that their approach does not work well for a variety of reasons. First the tree grows very large rapidly. The deictic commands are defined with respect to the two focus pointers. When long sequences of actions are required with a small number of pointers, it is easy to lose focus. While they try to address this by redesigning the pointers, they do not have much success. One way to alleviate this problem is by adopting a hierarchical approach as we do in this work. If the number of pointers required by each deictic level to maintain focus is not large, we can avoid some of the problems encountered by [Finney *et al.*, 2002].

8 Discussion and Future Work

While deixis is a powerful paradigm ideally suited for situations that are mainly reactive, it is difficult to employ a purely deictic system to solve complex tasks that require long-range planning. Our hierarchical deictic framework allows us to employ deictic representations in lower levels of the problem to leverage their power and generalization capacities, while at the higher levels we retain global context information in a non-deictic fashion. Mixing such representations allows us to exploit the best of both worlds and to solve tasks that require maintaining long term focus. It is our belief that there is no pure deictic system in nature. While it has been established that humans employ deixis in a variety of settings, we certainly maintain some higher level context information. While gathering ingredients for making tea, we might be using deictic pointers for accessing various containers [Land *et al.*, 1998], but we also are continuously aware of the fact that we are making tea.

The Bayesian approach we outlined requires that we have a complete model of the option MDP, which we assumed was available apriori. To learn this would require a lot more experience with the option MDP than we would need to learn a reasonable policy. Currently we are experimenting with learning a partial model of the option MDP, such that it is sufficient to identify the correct configuration of the deictic pointers.

The various approaches to learning with deictic pointers [Whitehead and Ballard, 1991; Finney *et al.*, 2002] usually employ simple physical locators.[Agre, 1988] uses complex pointers, but hand codes the policy for maintaining the focus of these pointers. For example, a set of rules are used to determine which is the *bee-attacking-me* and the pointer is moved suitably. Our approach falls somewhere in between. We start by defining a set of simple pointers. As learning progresses the agent learns to assign these pointers consistently such that some of them take on complex roles. We can then assign semantic labels to these pointers such as *robot-chasing-me*.

It should be noted that a homomorphic image implies a consistent representation but the reverse is not true. The notion of a consistent representation is a more general concept than a homomorphic image and corresponds to optimal-action value equivalence discussed in [Givan *et al.*, 2003]. By restricting ourselves to homomorphic images we are limiting the class of deictic pointers that we can model. Further work is needed to extend our framework to include richer classes of deictic pointers and to employ memory based methods. Nevertheless in this article we have taken the first steps in accommodating deictic representations in a hierarchical decision theoretic framework.

References

[Agre and Chapman, 1987] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of AAAI-87*, 1987.

[Agre, 1988] Philip E. Agre. The dynamic structure of everyday life. Technical Report AITR-1085, Massachusetts Institute of Technology, 1988.

[Finney *et al.*, 2002] S. Finney, N. H. Gardiol, L. K. Kaelbling, and T. Oates. That thing we tried didn't work very well: Deictic representation in reinforcement learning. In *Proceedings of the 18th UAI*, 2002.

[Givan *et al.*, 2003] R. Givan, T. Dean, and M. Greig. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147(1–2):163–223, 2003.

[Land *et al.*, 1998] Michael F. Land, N. Mennie, and J. Rusted. Eye movements and the roles of vision in activities of daily living: making a cup of tea. *Investigative Ophthalmology and Visual Science*, 39(S457), 1998.

[McCallum, 1995] A. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Computer Science Department, University of Rochester, 1995.

[Minut and Mahadevan, 2001] Silviu Minut and Sridhar Mahadevan. A reinforcement learning model of selective visual attention. In *Proceedings of the Fifth International Conference on Autonomous Agents*, Montreal, 2001.

[Ravindran and Barto, 2003a] Balaraman Ravindran and Andrew G. Barto. Relativized options: Choosing the right transformation. In *Proceedings of ICML 2003)*, pages 608–615, August 2003. AAAI Press.

[Ravindran and Barto, 2003b] Balaraman Ravindran and Andrew G. Barto. SMDP homomorphisms: An algebraic approach to abstraction in semi-Markov decision processes. In *Proceedings of the Eighteenth IJCAI*, pages 1011–1016, August 2003. AAAI Press.

[Sutton *et al.*, 1999] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[Whitehead and Ballard, 1991] Steven D. Whitehead and Dana Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7:45–83, 1991.