massachusetts institute of technology — artificial intelligence laboratory

Learning with Deictic Representation

Sarah Finney, Natalia H. Gardiol,
Leslie Pack Kaelbling and Tim Oates

AI Memo 2002-006 April 2002

Abstract

Most reinforcement learning methods operate on propositional representations of the world state. Such representations are often intractably large and generalize poorly. Using a deictic representation is believed to be a viable alternative: they promise generalization while allowing the use of existing reinforcement-learning methods. Yet, there are few experiments on learning with deictic representations reported in the literature. In this paper we explore the effectiveness of two forms of deictic representation and a naïve propositional representation in a simple blocks-world domain. We find, empirically, that the deictic representations actually worsen performance. We conclude with a discussion of possible causes of these results and strategies for more effective learning in domains with objects. [1]

[1] This report describes work done within the Artificial Intelligence Lab at Massachusetts Institute of Technology. The research was sponsored by the Office of Naval Research contract N00014-00-1-0298, by the Nippon Telegraph & Telephone Corporation as part of the NTT/MIT Collaboration Agreement, and by a National Science Foundation Graduate Research Fellowship.

1 Introduction

Humans speak, and probably think, of the world as being made up of objects, such as chairs, pencils, and doors. Individual objects have features, and sets of objects stand in various relations to one another. If artificial agents are to successfully interact with our world, they will need to represent and reason about objects, their features, and relations that hold among them. We are interested in how agents embedded in inherently relational environments, such as physically embodied robots, can learn to act.

What learning algorithms are appropriate? What representations are appropriate? The answers to these questions interact. First order logic (FOL) immediately comes to mind as a possible representation due to its compactness and ability to generalize over objects via quantification. It is likely that we will have to develop learning algorithms that use truly relational representations, as has been done generally in inductive logic programming [19], and specifically by Dzeroski et al [10] for relational reinforcement learning. However, before moving to more complex mechanisms, it is important to establish whether, and if so, how and why, existing techniques break down in such domains. In this paper, we document our attempts to apply relatively standard reinforcement-learning techniques to an apparently relational domain.

One strategy from AI planning [13], for example, is to propositionalize relational domains. That is, to specify a (hopefully small) finite domain of objects, and to describe the domain using a large set of propositions representing every possible instantiation of the properties and relations in the domain. This method may be inappropriate for learning because it does not give any representational support for generalization over objects.

Starting with the work of Agre and Chapman [1], and gaining momentum with the debate over the fruitcake problem in the late 1980's (see the Winter, 1989 issue of AI Magazine for the culmination of this debate), arguments have been made for the viability of deictic representations in relational domains. Such representations point to objects in the world and name them according to their role in ongoing activity rather than with arbitrary identifiers, as is typically the case with first order representations [3]. Despite the fact that deictic representations lack quantification, they allow for some of the generalization that FOL affords without the attendant computational complexity. Perhaps most importantly, they can be used with some existing reinforcement learning algorithms.

Even though deictic representations appear to be attractive, there has been no systematic investigation of the circumstances under which they result in better performance compared to naive propositional representations when the underlying domain is relational. This paper presents the results of a set of experiments in which two forms of deictic representations are compared to propositional representations with respect to a learning problem in a blocks-world domain. Results for two reinforcement learning algorithms are described: neuro-dynamic programming [5], and Chapman and Kaelbling's G algorithm [7].

The remainder of the paper is organized as follows. Section 2 briefly reviews reinforcement learning and the impact that the choice of representation has on the nature of the learning problem. Section 3 reviews deictic representations and the impact they have as well. Section 6 describes a set of experiments aimed at understanding when deictic representations are advantageous, and section 7 reviews related work and concludes.

2 Reinforcement Learning and Representations

The term *reinforcement learning* (RL) is used to denote both a set of problems and a set of algorithms. Reinforcement learning problems are framed in terms of an agent, an environment, and rewards. The agent can observe the environment and can affect the environment by taking actions. Each time the agent takes an action it receives a scalar reward signal that indicates how good it was to have taken that particular action given the current state of the environment. The task facing the agent is to learn a mapping from observations (or histories of observations) to actions that maximizes the reward that it receives over time.

The mapping learned by the agent is called a policy and is denoted $\pi(s, a)$, where s is an observation and a is an action; the value of $\pi(s, a)$ is the probability of taking action a given observation s. Observations are often called states, thus the use of the symbol s, but they may or may not correspond to underlying states of the environment. If an agent's perceptions are in reliable one-to-one correspondence with states of the environment, the environment is said to be fully observable. Often, however, observations are in a one-to-many correspondence with states and multiple distinct states appear to be the same to the agent. Such environments are said to be partially observable. They yield much more difficult learning problems when the optimal action or value for two conflated states differs.

There are many different ways to represent states, each with its own advantages and disadvantages. To make the discussion concrete, consider a blocks world in which each block can be one of three different colors and can be on another block or a table.

One way of representing states is atomically, with one atom for each unique configuration of the blocks. It might then be possible to learn, for example, that taking action a_2 in state 2397 yields a reward of 0.5 and leads to state 78512. This representation is simple in the sense that a single number can represent an entire state and learning can be implemented using tables indexed by states. The price to be paid for this simplicity is high. First, atomic representations can be intractably large, even for small numbers of objects. Given n blocks, there are 3^n possible assignments of colors to blocks and $O(n!)$ ways to stack the blocks. Second, because there is no way to determine whether two states are similar, there is no way to generalize to learn, for example, that if action a_2 is the best one to take in state 2397 then it is probably also the best one to take in state 78512.

Factored representations meliorate some of the problems with atomic representations. For example, the state could be represented as the set of propositions that are currently true, such as *block2-is-red* or *block12-is-on-block8*. The distance between states can be measured in terms of the number of bits that are different in their encodings. This makes generalization possible through the use of learning algorithms based on, for example, neural networks and decision trees.

The term *propositional representation* will be used throughout this paper to denote a naive propositionalization of a relational domain. For example, given n blocks there would be $3n$ propositions to specify the colors of the blocks and n^2 propositions to specify how the blocks are stacked.

Though propositional representations afford some opportunities for generalization, they must refer to objects by name (e.g., *block12* and *block8*) and are therefore unable to perform an important type of generalization. In many domains, objects of a given type are fungible: the identity of an object is unimportant as long as it has some particular property. For example, if the goal of the blocks-world agent is to pick up green blocks, it might want to learn the following rule expressed in first order logic:

$$\textbf{if } \exists x \texttt{color}(x, \texttt{green}) \wedge \forall y \neg \texttt{on}(y, x) \textbf{ then } \texttt{pickup}(x).$$

That is, if there is a block that is green and there is nothing on top of it, that block should be picked up. This rule applies regardless of the number of blocks in the environment, yet the size of the propositional equivalent to this rule is a polynomial function of the number of blocks. The propositional version contains a disjunction over all blocks to cover the existential quantification, and each disjunct contains a conjunction over all blocks to cover the universal quantification.

There is almost no work in RL on learning in domains where states are represented as expressions in FOL. This is partly because dealing with uncertainty in FOL is problematic. Also, because existing work has largely assumed atomic or propositional state representations, work on combining RL and FOL has required the introduction of novel data structures and learning algorithms [9]. This is unfortunate given the wealth of theory and algorithms that apply to the more standard representations. Deictic representations have the potential to bridge this gap, allowing much of the generalization afforded by FOL representations yet being amenable to solution (even in the face of uncertainty) by existing algorithms. Deictic representations are therefore the subject of the next section.

3 Deictic Representations

The word deictic derives from the Greek *deiktikos*, which means "able to show". It is used in linguistics, and was introduced into the artificial intelligence vernacular by Agre and Chapman [1], who were building on Ullman's work on visual routines [22]. A deictic expression is one that "points" to something; its meaning is relative to the agent that uses it and the context in which it is used. "The book that I am holding" and "the door that is in front of me" are examples of deictic expressions in natural language.

Two important classes of deictic representations are derived directly from perception and action relations between the agent and objects in the world. An agent with directed perception can sensibly speak of (or think about) *the-object-I-am-fixated-on.* An agent that can pick things up can name *the-object-I-am-holding.* Given a few primitive deictic names, such as those just suggested, we can make compound deictic expressions using directly perceptible relations. So, for example, we might speak of:

- *the-object-on-top-of(the-object-I-am-fixated-on)*
- *the-color-of(the-object-to-the-left-of(the-object-I-am-fixated-on)).*

Benson [4] describes an interesting system for synthesizing complex deictic expressions on demand.

Central to our use of deictic representations is the notion of a *marker* — an attentional resource under the control of the agent. The agent can mark objects (i.e. attend to them) and move markers about (i.e., shift its foci of attention). Perceptual information, both features and relations, is available only for those objects that are marked. All other objects are effectively invisible to the agent. Markers also influence activity, because only marked objects can be manipulated by the agent.

There are many ways that the perceptions and actions of an agent can be structured within the general framework of deictic representations. At one extreme, an agent with one marker for each object in its environment effectively has a complete propositional representation. At another extreme, an agent with one or two markers has a very limited perceptual window on its world. Though the environment may be fully observable given enough markers, it is only partially observable from the standpoint of the agent.

Two points on this spectrum might be characterized as *focused deixis* (FD) and *wide deixis* (WD). In FD there is a special marker called the focus and some number of additional markers. Perceptual information is only available about the object marked by the focus. There are perceptual actions for moving the focus about, moving one of the other markers to the object marked by the focus, and moving the focus to an object marked by one of the other markers. Also, only objects marked by the focus can be manipulated by the agent. WD is just like FD, except perceptual information is available about all objects that are marked, regardless of whether they are marked by the focus or one of the other markers.

There are a number of practical differences between focused deixis and wide deixis. There is less information about the state of the environment available with FD than with WD. This means that the space required by FD representations can be much less than that of WD representations, but it can also mean that the agent has to learn long sequences of perceptual actions to gather sufficient information about the state of the environment to behave effectively. This may require access to short state histories so the learner can in effect remember what it has seen and done in the recent past.

Our starting hypothesis was that deictic representations would ameliorate the problems of full-propositional representations. Some of the advantages include:

- **focused partial observability:** Though the agent only gets to see objects that are marked, those objects typically include ones that play some role in the current activity and are thus most likely to be relevant to whatever reward is ultimately received. Also, if irrelevant aspects of the world are not easily observed then learning is made easier because there is no need to generalize over them.

- **passive generalization:** Despite their lack of quantification, deictic representations are able to generalize over objects. For example, if I know what to do with *the-cup-that-I-am-holding*, it doesn't matter whether that cup is *cup3* or *cup7*.

- **biological plausibility:** Objects are named by the role they play in the agent's current activity. It is conjectured that deixis plays a role in the way humans represent the world [3].

In most deictic representations, and especially those in which the agent has significant control over what it perceives, there is a substantial degree of partial observability: in exchange for focusing on a few things, we lose the ability to see the rest. As McCallum observed in his thesis [16], partial observability is a two-edged sword: it may help learning by obscuring irrelevant distinctions as well as hinder it by obscuring relevant ones.

The thing that is missing in the literature is a systematic evaluation of the impact of switching from propositional to deictic representations with respect to learning performance. The next sections reports on a set of experiments that begin such an exploration.

4 The Experimental Domain: Blocks World

Our learning agent exists in a simulated blocks world and must learn to use its hand to remove any red or blue blocks on a green block so that block may be lifted. The choice of this problem domain was not arbitrary. Whitehead and Ballard [23] introduced it in their pioneering work on the use of deixis in relational domains. They developed the Lion algorithm to deal with the *perceptual aliasing* (partial observability) that resulted from using a deictic representation by avoiding the aliased states. McCallum [17] used the same domain to demonstrate that the number of markers required to solve the problem can be reduced by keeping a short history of observations. Finally, Martin [15] used the domain to motivate and evaluate an algorithm for learning policies that generalize over initial configurations.

4.1 Whitehead and Ballard's blocks world

Whitehead and Ballard's blocks world [23] differs from ours in several ways. First, Whitehead's representation has two markers, one for perceptual information and one for taking action. In our representation, there is only one marker which the agent can move arbitrarily, and this marker is used both for gaining perceptual information and for performing actions. Whitehead's algorithm required two separate perception and action markers to allow the agent to avoid ambiguous states, a restriction that did not apply to our algorithms. Since we did not need separate markers for action and perception, we combined the two to give that agent the advantage of naturally focusing on areas of the world that are relevant to current activity (see Section 3).

Second, Whitehead's action set includes primitives for looking to the top and bottom of a stack. We gave our agent a more basic primitive action set for moving its focus of attention in the hopes that it would learn to put them together in meaningful search actions according to the needs of the task. Our hope was that with a more general action set, we could arrive at a learner that would be able to solve a more diverse set of tasks in the domain.

Lastly, rather than dealing with ambiguous states by trying to avoid them, as did Whitehead and Ballard, we added history to our perceptual space. This assumes that the agent is always able to differentiate ambiguous states by looking back in history, but for the tasks we were interested in, this property was true.

4.2 McCallum's blocks world

McCallum's blocks world [17], in which he tested his Utile Suffix Memory algorithm, was based on Whitehead and Ballard's, and differed only in the addition of history for differentiating ambiguous state, as does ours, and in having only one marker, again like ours. Our world contains additional "memory" markers to allow the agent to solve more complex tasks, but like McCallum's has only one that can be moved around arbitrarily.

McCallum's use of the world, however, includes Whitehead's Learning by

Watching technique, in which is agent acts according to a teacher policy with some probability. This is used only to direct the exploration, however, and does not influence the agent's decisions. We believed that our agent should be able to arrive at the optimal policy on its own so we did not advise it during learning. Finally, the action set used by McCallum was slightly less general than ours and more specific to Whitehead's "pick up the green block" task.

4.3 Our blocks world

The experimental setup described here differs from previous empirical work with deictic representations in two important ways. First, our goal is to understand the conditions under which one representation (deictic or propositional) is better than the other. That is, we want to systematically compare the utility of propositional and deictic representations rather than evaluate any given learning algorithm designed to operate with one or the other representation. Second, we have not tuned the perceptual features or training paradigm to the task. Despite its apparent simplicity, reinforcement learning does not seem to work in this domain without the use of perceptual features that are very finely tuned to the specific task faced by the learning agent [15, 23] or without frequent examples of correct actions provided by a teacher [17]. We tried to develop a set of perceptual features that seemed reasonable for an agent that might be given an arbitrary task in a blocks world, and we provide no information to the agent beyond its observations and rewards.

4.4 Two Deictic Representations

While a deictic name for an object can be conceived as a long string like *the-block-that-I'm-holding*, the idea can be implemented with a set of markers. For example, if the agent is focusing on a particular block, that block becomes *the-block-that-I'm-looking-at*; if the agent then fixes a marker onto that block and fixes its attention somewhere else, that block becomes *the-block-that-I-was-looking-at*.

For our experiments, we developed two flavors of deictic representation.

In the first case, *focused* deixis, there is a focus marker and one additional marker. The agent receives all perceptual information relating to the focused block: its color (`red`, `blue`, `green`, or `table`), and whether the block is in the agent's hand. In addition, the agent can identify the additional marker if it is bound to any block that is above, below, left of, or right of the focus. The focused deixis scenario imposes a narrow focus of attention and requires deliberate care when deciding what to attend to.

The second case, *wide* deixis, is like focused deixis with some additional information. Perceptual information (color and spatially proximal markers) is available for all marked blocks, not just the focused block. In addition, because information is available about spatially proximal blocks for all pairs of marked objects, only the identity of markers below and to the right of any given marker is returned. Information about blocks that are above and to the left of a marker would be redundant in this case.

The action set for both deictic agents is:

- *move-focus(direction)*: The focus cannot be moved up beyond the top of the stack or down below the table. If the focus is to be moved to the side and there is no block at that height, the focus falls to the top of the stack on that side.
- *focus-on(color)*: If there is more than one block of the specified color, the focus will land randomly on one of them.
- *pick-up()*: This action succeeds if the focused block is a non-table block at the top of a stack.
- *put-down()*: Put down the block at the top of the stack being focused.

- *marker-to-focus(marker)*: Move the specified marker to coincide with the focus.
- *focus-to-marker(marker)*: Move the focus to coincide with the specified marker.

4.5 Full-Propositional Representation

In the fully-observable propositional case, arbitrary names are assigned to each block, including the table blocks. The agent can perceive a block's color (one of `red`, `blue`, `green`, or `table`), the location of the block as indicated by the index of its horizontal position on the table, and the name of the block that is under the block in question. In addition, there is a single bit that indicates whether the hand is holding a block. The propositional agent's action set is:

- *pick-up(block#)*: This action succeeds only if the block is a non-table block at the top of a stack.
- *put-down()*: Put down the block at the top of the stack under the hand.
- *move-hand(left/right)*: This action fails if the agent attempts to move the hand beyond the edge of the table.

5 Choice of Algorithms

In these experiments, we took the approach of using model-free, value-based reinforcement learning algorithms, because it was our goal to understand their strengths and weaknesses in this domain. In the conclusions, we discuss alternative methods.

Because we no longer observe the whole state in the deictic representation, we have to include some history in order to make the problem more Markovian. The additional information requirement renders the observation space too large for an explicit representation of the value function, like a look-up table. Thus, we required learning algorithms that can approximate the value function. we now have to include some history in order to make the problem Markovian.

We chose Q-learning with a neural-network function approximator (known as neuro-dynamic programming [6], or NDP) as a baseline, since it is a common and successful method for reinforcement learning in large domains with feature-based representation. We hoped to improve performance by using function approximators that could use history selectively, such as the G algorithm [7] and McCallum's U-Tree algorithm [16]. After some initial experiments with U-Tree, we settled on using a modified version of the simpler G algorithm. In neuro-dynamic programming, neural networks are used to approximate the Q-value of each state-action pair. G and UTree both look at reward distributions to determine which observation bits are relevant to predicting reward and divide the state space up accordingly.

We believed that G and UTree would have the advantage over neuro-dynamic programming in two respects. We hypothesized first that G and UTree would be able to learn faster by virtue of their ability to discern which parts of the observation vector were irrelevant to the task and ignore them, thereby decreasing the amount of distracting information. After some initial experiments with U-Tree, we settled on using a modified version of the simpler G algorithm.

5.1 Description of Neuro-Dynamic Programming

Our baseline for comparison was the neuro-dynamic programming (NDP) algorithm described by Bertsekas and Tsitsiklis [5].

In our NDP setup, we used one two-layer neural network for each action in the action set. The number of input nodes to each network was equal to the length of the current percept vector plus the past few pairs of actions and percept vectors as specified by the history parameter h. The number of hidden nodes was the number of input nodes divided by four. There was a single output unit whose value indicates the approximate Q-value of the corresponding action.

The learning rate was 0.1 and the discount factor was 0.9. We used SARSA(λ) to update the Q-values, with $\lambda = 0.7$. As has been observed by others [21, 2, 20], we found that SARSA led to more stable results than Q-learning because of the partial observability of the domain.

5.2 Description of the G Algorithm

The G algorithm makes use of a tree structure to determine which elements of the state space are important for predicting reward. The tree is initialized with just a root node which makes no state distinctions whatsoever, but has a fringe of nodes beneath it for each distinction that could be made. Statistics are kept in the root node and the fringe nodes about immediate and discounted reward received during the agent's lifetime, and a statistical test (the Kolmogorov-Smirnov test) is performed to determine whether any of the distinctions in the fringe are worth adding permanently to the tree. If a distinction is found to be useful, the fringe is deleted, the distinction nodes are added as leaves, and a new fringe is created beneath all of these new leaf nodes. Q-values are stored and updated in the leaf nodes of the tree.

Because our domains were not fully observable, we had to modify the original G algorithm. To begin with, we had to allow the fringe to include historical distinctions. However, at the beginning of a trial, the agent has no history, so the first few experiences will not match to a leaf node, which substantially complicates the bookkeeping. To solve this, we started each episode with as many observations in the past as our history window allowed the agent to consider in making splits. We then added a bit to our observation vector that indicated whether or not the agent was alive at the time of each observation, and appended a buffer of the necessary number of non-alive experiences to the beginning of each trial. The rest of these observation vectors were filled arbitrarily. Also because of the partial observability of the domains, we used SARSA both for arriving at Q-values and for generating the sampled Q-values used to determining when a split was important.

Lastly, we did not use the D statistic used in the original G algorithm [7]. Rather, we kept vectors of immediate and one-step discounted reward for the state represented by each leaf and fringe node, both for the state as a whole, and divided up by outgoing action. These vectors are compared to the vectors of the parent node in examining a possible split rather than to the other children, since our percept vectors include features with more than two possible values.

In our experiments, the learning rate was 0.1 and the discount factor was 0.9.

5.3 Issues with the UTree Algorithm

There were two main reasons we decided not to use the UTree algorithm.

The first has to do with the use of the Kolmogorov-Smirnov (KS) test when executing a fringe test. The objective of the fringe test, as in G, is to determine if there is a set of nodes representing an additional distinction whose Q-values differ significantly from their parent leaf-node. Basically, the fringe test compares the utility of making an additional distinction vs. not making the distinction.

For the fringe test, the KS test looks at the Q-values of each instance originally stored in the parent and compares them to the Q-values of the instances tentatively stored in a particular fringe node. The Q-values for each instance are calculated as

$$q_{I_t} = r_{I_t} + \gamma U_{I_{t+1}},$$

where q_{I_t} is the the Q-value for the instance recorded at time t, r_{I_t} is the immediate reward received at time t, γ is the discount factor, and $U_{I_{t+1}}$ is the utility of the tree node corresponding to the instance recorded at the next time step. Sampling the instances' Q-values (instead of directly using the Q-value calculated for each tree node, which already is a point-summary of the instances' Q-values) and then using a statistical test is an attempt to find significant differences while taking into account uncertainty in the learning problem.

However, there are really two kinds of uncertainty when learning with incremental state-representation algorithms such as UTree. First, there is the uncertainty about the values of and transition probabilities between states of the true underlying MDP. But also, there is uncertainty about values and transition probabilities between states in the approximated state representation encoded by the tree. The use of the statistical test as described above tries to get ahold of the uncertainty present in the problem, but it comingles the two types of uncertainty. We feel that the KS test, used in this way, may not be answering the right question.

The second reason we did not use UTree has to do with how the Q-values are calculated during the fringe test. In UTree, the fringe is dynamically created every time a fringe test is to be carried out. As we understood it from the text of McCallum's thesis, the steps in the fringe test are:

1. Under a particular leaf node, expand the fringe to a fixed depth with some combination of actions and observations.
2. Deposit each of the leaf's instances into the corresponding fringe. Update the fringe node's Q-values accordingly.
3. Compare the Q-value distribution of the instances in each fringe node with that of the parent's. Keep the new distinction if the distributions differ by some threshold; otherwise, try a new combination.

The difficulty here is that the sampled Q-value of each instance in the fringe, depends on the utility of the next state, as represented by some node in the tree. It is possible, however, that the utility of this other node depends, in turn, on a node in the newly created fringe. The problem is that the statistics for other nodes in the tree are not updated during the fringe test. This may result in incorrect values in the Q-value distributions being compared. The problem is especially acute in the early stages of learning, when the tree is relatively small and the likelihood of an outside node depending on the fringe is significant.

The correct approach is to recompute the Q-values for the entire tree with each fringe test and then compare the original values of the parent to the new values of the fringe. Unfortunately, this is extremely computationally expensive. The resulting performance hit was severe enough to render UTree impractical for our purposes.

With the modifications made to the G algorithm described above, G becomes essentially the same as U-Tree for the purposes of selective perception. The major distinction between them remains that U-Tree requires much less experience with the world at the price of greater computational complexity: it remembers historical data and uses it to estimate a model of the environment's transition dynamics, and then uses the model to choose a state to split.

6 Experiments

Propositional representations yield large observation spaces and full observability. Deictic representations yield small observation spaces and partial observability. Which makes learning easier? That is the question that this section explores empirically with experiments in two different blocks-world starting configurations (Figure 1).

6.1 Comparing the Sizes of the State and Action Spaces for Each Representation

We first compute the size of the underlying state space for the full-propositional and deictic representations. The size of this state space is computed as:

$$\begin{aligned}\textit{State Space} &= (\#\textit{Configurations}) \times (\#\textit{Ways to name the blocks}) \\ &\quad \times (\textit{Am I Awake}) \times (\textit{Is My Hand Full})\end{aligned}$$

where

$$\textit{Configurations} = \frac{\textit{Ways to arrange blocks}}{\textit{Ways to arrange blocks of each color}},$$

$$\textit{Am I Awake} = \textit{true} \text{ or } \textit{false},$$

and

$$\textit{Is My Hand Full} = \textit{true} \text{ or } \textit{false}.$$

Furthermore, in the full propositional case,

$$\textit{Ways to name the blocks} = (\#\textit{blocks})!,$$

and in the deictic case,

$$\textit{Ways to name the blocks} = (\#\textit{blocks})^{(\#\textit{markers})}.$$

In *blocks1*, there are five blocks in three colors. The number of configurations is 12. In *blocks2*, with six blocks, there are 60 configurations.

Thus, the underlying state space in the full-propositional is 5760 ground states in *blocks1* and 172,800 in *blocks2*. The underlying state space in the deictic case, with two markers, is 1200 ground states in *blocks1* and 8640 in *blocks2*.

Next, we compute the size of the observation spaces for the full-propositional and deictic representations.

In the full-propositional case, the agent observes the two boolean-valued *Am I Awake* and *Is My Hand Full* features. Furthermore, for each block, it observes:

- The block's color: four possible values, `red`, `blue`, `green`, or `table`.
- The name of the block underneath it: n+1 values, (*#blocks*)+(`noblock`))
- The position of the stack its in: three values, 0, 1, or 2.

The size of this observation space outpaces the number of ground states dramatically: roughly 10 billion in *blocks1* and roughly 3 trillion in *blocks2*.[1]

In both deictic cases, the agents observe the two boolean-valued *Am I Awake* and *Is My Hand Full* features. Furthermore, in the wide case, the agent observes, for each marker:

- The marked object's color: four values, `red`, `blue`, `green`, or `table`.
- The identity of any marker underneath it: $m + 1$ values, (*#markers*) + `no marker`
- The identity of any marker to the right of it: $m + 1$ values, (*#markers*) + `no marker`

[1] Note that this observation space corresponds to the size needed for a look-up table, and it includes many combinations of percepts that are not actually possible.

- Whether the marked object is in the agent's hand: two values, `true` or `false`.

In the focused the case, the agents observes:

- The focused object's color: four values, `red`, `blue`, `green`, or `table`.
- The identity of any marker underneath it: m values, $($*#non-focus markers*$) +$ `no marker`
- The identity of any marker to the right of it: m values, $($*#non-focus markers*$) +$ `no marker`
- The identity of any marker to the left of it: m values, $($*#non-focus markers*$) +$ `no marker`
- The identity of any marker to the below it: m values, $($*#non-focus markers*$) +$ `no marker`
- Whether the focused object is in the agent's hand: two values, `true` or `false`.

The size of these observation spaces is constant in both domains: the size of the focused deictic observation space is 512, and that of the wide deictic observation space is 4096.

The action set for the deictic representations (see Section 4.4) does not change with additional blocks, so it is constant at 12 actions. The full-propositional action set (see Section 4.5) requires a *pickup()* action for each block, so it has five possible actions in *blocks1* and six in *blocks2*.

6.2 Experimental Setup

The learning task was to pick up a green block that was covered with a red block. The first domain, *blocks1*, consists of a three-block long table, a green block, and a red block. The second domain, *blocks2*, has an additional blue block as a distractor.

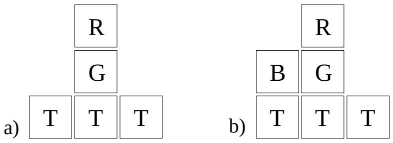


Figure 1: The two blocks-world configurations: a) *blocks1* and b) *blocks2.* The table on which the blocks are positioned is made up of unmovable `table`-colored blocks.

The agent receives a reward of 5.0 whenever it picks up a green block, a reward of -0.2 if it takes an action that fails (e.g., attempting to move its hand off the left or right edge of the world, or attempting to pick up the table), and a reward of -0.1 otherwise.

The performance of the learning algorithms was measured as follows. Given a configuration of blocks, the assignment of names to blocks was randomized in the fully observable case and the assignment of markers to blocks was randomized in the deictic cases. The agent was then allowed to take 200 actions while learning. If at any time the agent picked up the green block, the original configuration was restored and the names/markers were reassigned. At the end of each epoch of 200 actions the state of the learning algorithm was frozen and the agent took 100 additional actions during which the total accumulated reward was measured.

The results were plotted as follows. Because of the different lengths of the optimal action sequences for the deictic and propositional agents, we scale each result to reflect the optimal performance with respect to each agent. A data point for an agent is computed by taking its accumulated reward over the 100 testing steps, adding the maximum penalty that could be obtained by consistently executing the worst possible action (so that "0" is the worst possible score) and then dividing it by the maximum possible reward that could be obtained by the optimal policy in 100 steps. So, for each curve in the figures, a consistently optimal agent would score a "1.0" on every trial, and a consistently terrible agent would score "0.0" on every trial.

6.3 Initial Results

Here are the learning curves for each algorithm and each representation. The figures show that the NDP propositional (non-deictic) learner performed the best (Figure 2 and Figure 3).

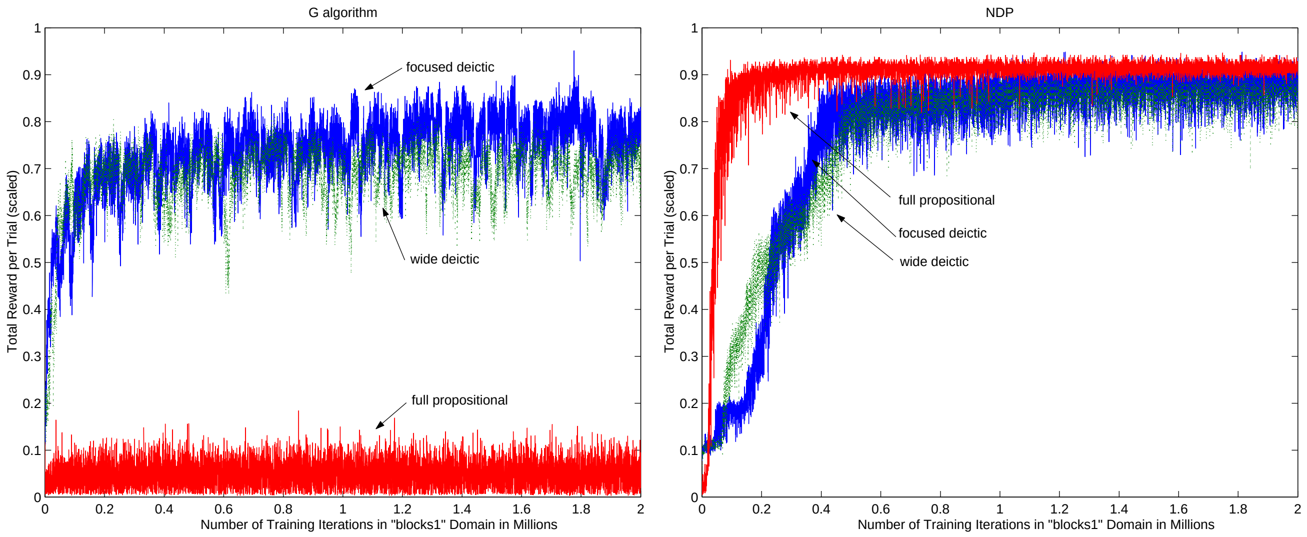


Figure 2: Learning curves for both algorithms in *blocks1* domain.

Our original hypothesis was that the additional block would distract the propositional learner, which had to represent the block explicity, more than it would the deictic learner, which did not. Even in the case of NDP, if the performance of the non-deictic learner was better than the deictics' in *blocks1*, according to our hypothesis any gap should certainly get smaller in *blocks2*. In fact, at some point, after the addition of sufficiently many blocks, the learning curves should "cross over" once the deictic learner's reduced observation space gave it the upper hand. The figures tell another story, however.

Given that its task is the same in each configuration and that the size of the observation space is the same even with the additional block, why would learning performance in NDP in the second configuration be so degraded with the deictic representation? The next section explores this question.

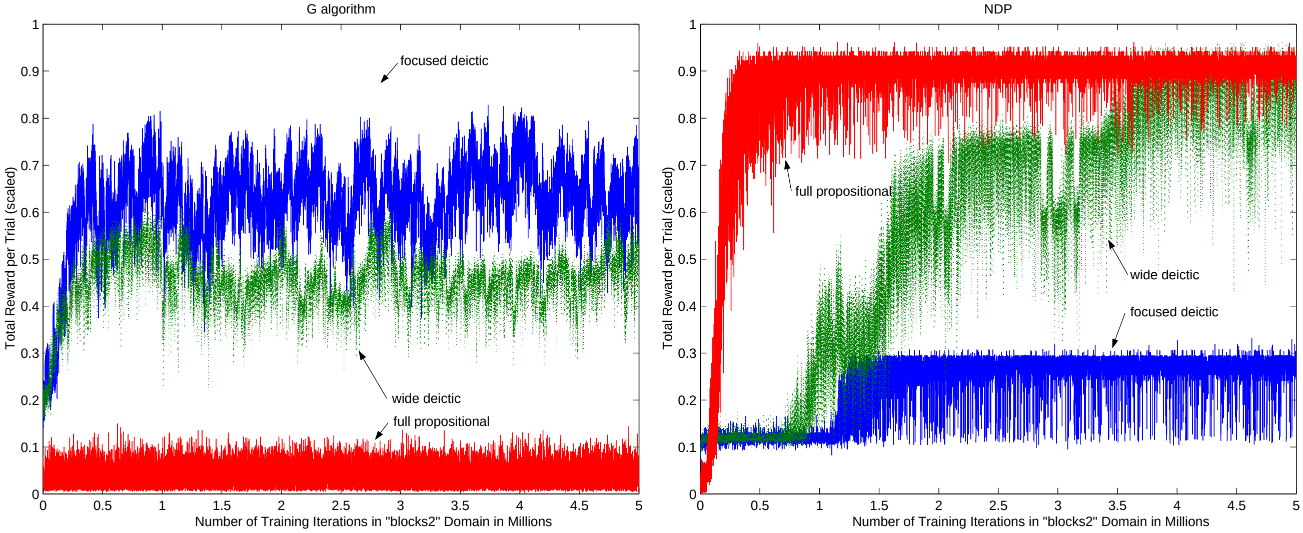


Figure 3: Learning curves for both algorithms in *blocks2* domain.

Furthermore, despite our hypothesis that G would perform better than NDP, we discovered that the opposite was true. Clearly, the agent using the G algorithm learns more slowly than the agent using NDP once the distractor block is added. More importantly, in both domains, the G agent is never able to reach a solution that is optimal in all cases, unless the tree is allowed to get very large (the trees plotted Figures 2 and 3 were capped at 10,000 nodes, including fringe nodes). Section 7.2 investigates why the trees need to be so large.

7 Discussion

The next two sections dissect our two counter-intuitive results and present explanations for them.

7.1 Comparing Deictic and Propositional Representations with NDP

Initially, we hypothesized that a deictic representation, because of its compact observation space and its ability to focus in on interesting parts of the state space, would have an advantage over a propositional representation. Furthermore, we conjectured that this advantage would grow more pronounced as the amount of distractive state information increased. Figures 2 and 3 show clearly, however, that our hypothesis was not true.

To really understand the trade-off between deictic and non-deictic representations, then, we need to understand how the problem gets harder with the addition of distractor blocks. Our starting hypothesis is that, as blocks are added, the exploration problem becomes much more difficult for the deictic learner. Even though its space of observables stays the same across configurations, the true space— which consists of the block configurations *and* the locations of all the markers—-grows quickly. Furthermore, the deictic actions are different from the non-deictic actions in that they are very conditional on which blocks the markers are bound to. Because the deictic learner does not have direct access to the focus location, an increase in complexity in this important, yet inaccessible, part of the state space might reduce its likelihood of arriving at a solution through exploration.

We set out to quantify, for each representation, how long it would take a random agent to stumble upon a solution. Our metric was a quantity called *mean time-to-goal*, which tries to quantify the difficulty of the exploration problem by counting the number of actions taken by the random agent to arrive at the goal. We measured the mean time-to-goal in the following way: for each representation (non-deictic, wide-deictic, and focused-deictic), we set up an agent to take actions randomly. We increased the number of distractor blocks each time by placing an additional blue block to either side of the original stack of green and red. For various block configurations, the time-to-goal was measured as the number of steps the random agents took to reach the goal. The results were averaged across 10,000 trials for each agent and each configuration.

As the left graph in Figure 4 shows, the number of steps taken to randomly

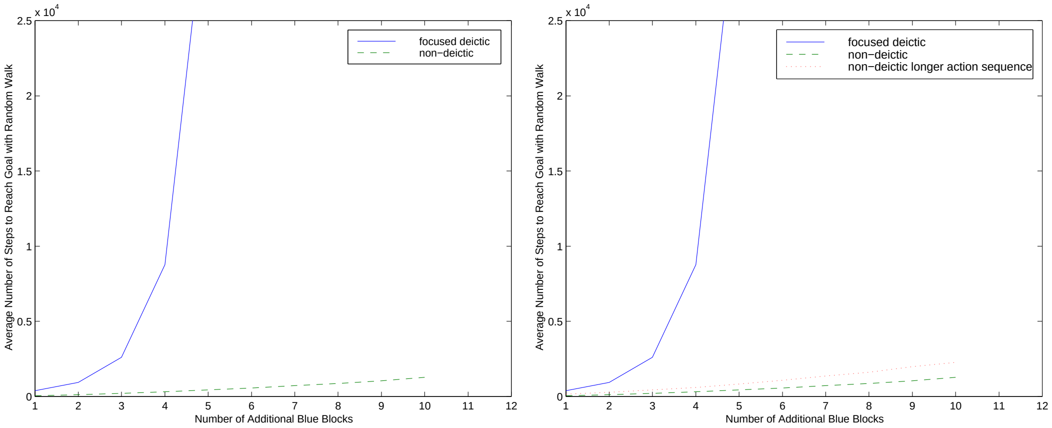


Figure 4: The mean time-to-goal for the various representations as the number of distractor blocks is increased. All agents start with the *blocks1* configuration, and blue blocks are successively added to either side of the original stack.

arrive at the solution in the deictic case outpaces the non-deictic case dramatically. If its exploration task gets so much harder with each extra block, it's little wonder then that its learning performance worsens, too.

Our next set of experiments involve trying to pin down how the choice of action sets affects the exploration difficulty.

The first experiment investigated how the length of the optimal action sequence affects the exploration difficulty. In the original action set, the deictic agent required six steps, in the best case, to reach the goal. The non-deictic agent, on the other hand, required only four steps to reach the goal. To even out the number of actions required to reach the goal, we tested the non-deictic agent with a task that required seven steps to reach the goal. The task was to pick up a green block initially covered by two red blocks. We expected to see, with this longer optimal action sequence, that the mean time-to-goal in the non-deictic case would now grow as in the deictic case. The right graph in Figure 4 shows that, while the time-to-goal grew somewhat faster than before, it was certainly not by the same order of magnitude as in the deictic case. We concluded it was not merely the length of the optimal action sequence that caused the difficulty of the problem to increase so much in the deictic case.

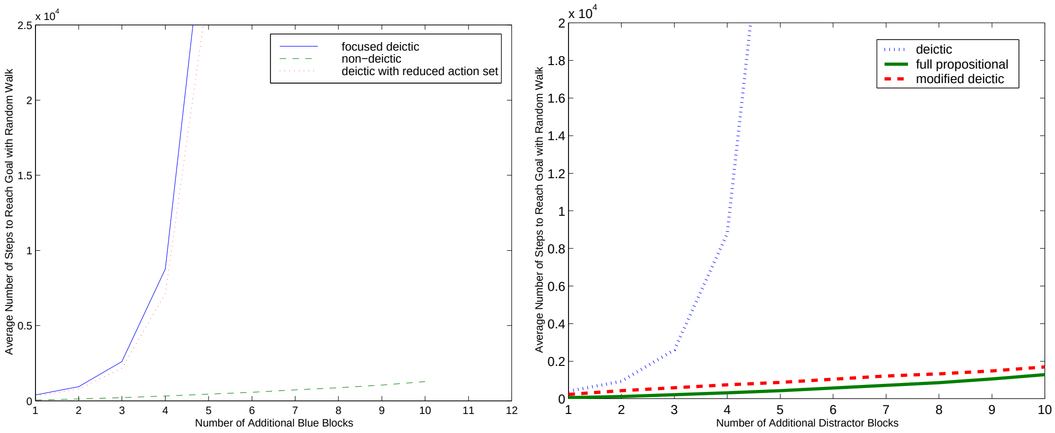


Figure 5: The mean time-to-goal for each representations as the number of distractor blocks is increased, now with different action sets.

The second experiment investigated the how the size of the action set (that is, the number of possible actions) affects exploration difficulty. To this end, we removed two actions from the deictic action set that we believed were distracting to the deictic agent: *focus-to-marker()* and *marker-to-focus()*. We compared a random deictic agent with this new action set (called the *reduced* action set, consisting of 10 actions rather than 12) to a random non-deictic agent with the original non-deictic action set (consisting of 8 actions). The left side of Figure 5 shows the result for this experiment. The rate of growth was very slightly slower, but it still seemed to go up by the same order of magnitude as before.

In the last experiment, we changed both the deictic and the non-deictic action sets in order to zero-in on the characteristic of the deictic action set that was making exploration so difficult. We created two new action sets (called the *modified* action sets). The idea was to modify the non-deictic action set so that it also suffered from what we thought was a difficult characteristic of the deictic action set: the division between focusing on a block and picking it up – the original non-deictic action set, on the other hand, had *pick-up(block#)*, which essentially did both tasks in one swoop. Here are the actions in the *modified* action set. For deictic [2]:

[2] Interestingly, this modified action set is similar to the set used by McCallum in his blocks-world experiments [17].

- *focus(direction)*: move the focus up, down, left or right. This version of the action is stricter than in the original set. If there is no block in the specified direction for the focus to land on, the focus will stay put.
- *look(color)*: focus on a green, red, blue, or table colored block.
- *pick-up-top()*: pick up the block at the *top* of the stack being marked by the focus, rather than the exact block being focused on.
- *put-down()*: put down the block being held on top of the stack being marked by the focus.

For non-deictic:

- *move-hand(block#)*: move the hand over the stack containing the specified block.
- *move-hand(direction)*: move the hand left or right.
- *pick-up-top()*: pick up the block at the top of the stack that is underneath the hand.
- *put-down()*: put down the block being held on top of the stack that is underneath the hand.

The right graph of Figure 5 shows the result. Surprisingly, instead of the non-deictic agent's time-to-goal growing like the deictic's, the opposite happened: both *modified* action sets' time-to-goal grew roughly at the same rate as the original non-deictic set. What changed?

The main difference between the two deictic action sets is as follows: The *modified* action set, via the *pick-up-top()* action, reduces a great deal of the dependence of the actions on the focus location. The focus location is crucial, yet it is not observeable by the agent – indeed, as the number of distractor blocks increases, this unobserveable part of the state space grows exponentially. Removing the strict dependence on the focus location increases the likelihood that the agent can use exploration to make progress. With the *modified* action set, the probability of choosing successful actions increases, and the probability of choosing harmful actions decreases. Please see the appendix for an informal analysis of the relative "distractability" of the various action sets.

Follow-up learning experiments (see Figure 6) with the *modified* action sets show the deictic agents learn comparably with the full-propositional agent in *blocks1* and even slightly faster than the full-propositional agent in *blocks2*.

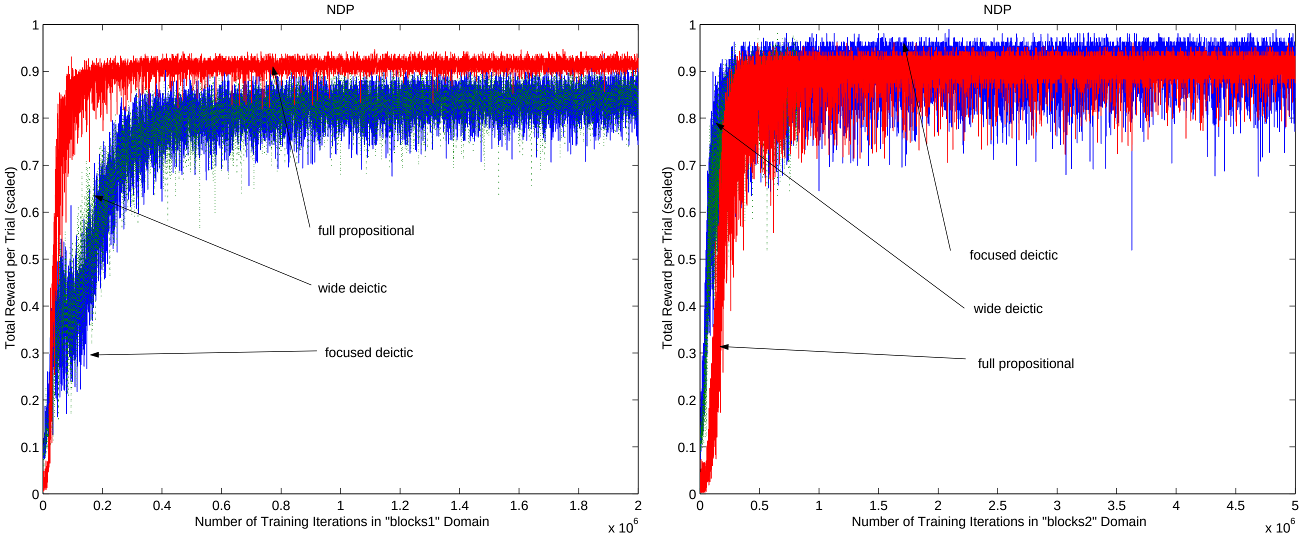


Figure 6: Learning curves for the three representations in *blocks2*, where the two deictic agents are using the *modified* action set. Compare to the right-hand sides of Figures 2 and 3.

Our over-arching conclusion is that if one wants to use a deictic representation, either the set of actions must be chosen carefully or a reasonable action-set bias must be provided. It was unsatisfying to note that in order to render the deictic action set tractable, we had to remove the dependence on what made deictic representations initially so appealing—the focus of attention. One option for providing the necessary structure might be to use hierarchical or temporally extended actions. Indeed, using a clever action hierarchy can do much to abstract away the partial observability that exists at the level of atomic actions [11]. Hierarchical or temporally-extended actions might also provide a more appropriate level at which to make decisions, by virtue of abstracting away exploratory "blips" (see next section) and providing a more nearly deterministic action space [14].

7.2 Comparing Neuro-dynamic Programming and G

In order for the G agent to learn a policy that is optimal in all cases, our initial results showed that it must grow a tree that seems unreasonably large (5,000-10,000 nodes). The trees seemed to keep growing without reaching a natural limiting state. For this reason, to avoid running out of memory, we had to add an arbitrary cap on the size of the trees. After observing these results, we developed the following theory as to why the trees need to be so large.

The G tree needs to have enough nodes to distinguish all the states that are different from one another with respect the reward received. That is, all the states which have a different value should be represented as a distinct leaf in the tree. However, since the policy that is being followed changes throughout the learning process, the tree must grow to be large enough to include leaves for all the states with distinct values in *all* of the policies followed during learning. A simple mazeworld example should make the problem clear. The next section considers the simple maze shown in Figure 7.

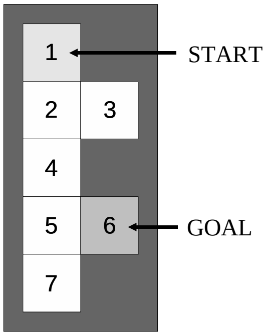


Figure 7: Mazeworld example. State 1 is the start state, and state 6 is the goal state.

7.2.1 Unnecessary Distinctions

In this simple maze, the agent can only observe whether there is a wall in each cardinal direction, so this environment is partially observable; states *2* and *5* look identical, but they require different actions to reach the goal. At each time step, the agent receives a new percept vector and may move in one of the four directions. The agent is rewarded for reaching the goal, penalized for each step that did not lead to the goal, and penalized slightly more for attempting to move into a wall.

In one trial, the G algorithm made distinctions in the following way. The first split distinguishes states based on whether south is clear or blocked. This separates states *1*,*2*,*4* and *5* from states *3* and *7* (state *6* is never actually observed by the agent, since the trial is restarted). The first of these subgroups is then further split according to whether east is blocked, separating states *2* and *5* from states *1* and *4*. Since a large reward is received for going east from state *5*, the leaf for states *2* and *5* contains a policy for going east. Figure 8 shows the tree at this point during the learning process. Clearly this policy is not optimal, since we have not yet learned to tell states *2* and *5* apart.

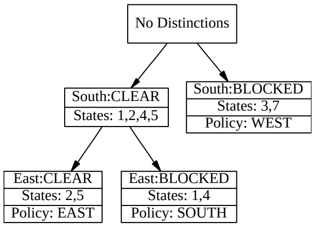


Figure 8: Example G tree after first two splits.

Intriguingly, the next distinction is made on the previous action under the node for states *3* and *7*, as shown in Figure 9. At first glance, this looks like a meaningless distinction to make: the only additional information it seems to yield is about the previous state. We already know that we are in state *3* or *7*, and knowing the last action merely gives us information about the preceding state: if we went east and we are now in state *3* or *7*, then we might have been in states *2*, *3*, or *7* previously, whereas a south action means we were in states *3*, *5*, or *7* previously, etc. There seems to be no utility to this knowledge, since subsequent policy decisions will depend only on the current state.

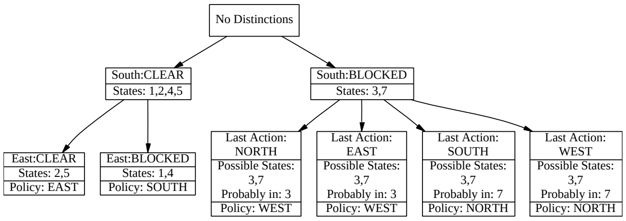


Figure 9: Example G tree after the first three splits.

However, by examining the policy in effect when the agent makes this split, the distinction begins to make sense. When the agent is in state *2*, the policy says to go east. Thus, unless it explores, it will go into state *3*. Once in state *3*, it will oscillate from west to east until exploration leads the agent south into state *4*. Thus, when the agent visits state *3*, it has generally just performed an east action. On the other hand, when the agent is in state *7*, it has most likely performed either a south or a west action. Thus, splitting on the previous action, with the current policy, actually disambiguates with high probability states *3* and *7* and yields a reasonable policy, one that goes north from state *7*, as shown in Figure 9.

Once we have made a distinction based on our history, our policy may then lead the algorithm to make more distinctions in an attempt to fully represent the value function under this new policy. For example, if we are executing the policy shown in Figure 10, we do in fact need two different values for state *2*, depending on whether we are visiting it for the first or second time. This is how the trees become much larger than they would need to be if they were only required to store the value function for the optimal policy.

The last experiment we did to confirm our explanation for the large trees was to fix the policy of the agent and allow the tree to grow. As expected, we obtain trees that contain few or no unnecessary splits for representing the value function for the fixed policy. While we cannot avoid making the occasional bad split due to the statistical nature of the problem, this is not the underlying cause of the very large trees. Note that this problem will almost certainly be exhibited by U-Tree, as well.

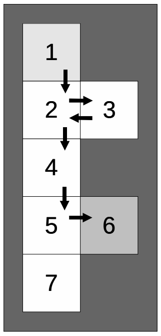


Figure 10: A sample policy that may complicate trees.

This problem of G growing unreasonably large trees in POMDPs seems very difficult to address. There is, fundamentally, a kind of "arms race" in which a complex tree is required to adequately explain the Q values of the current policy. But the new complex tree allows an even more complex policy to be represented, which requires an even more complex tree to represent its Q values.

We believe that the reason that this wasn't encountered as a serious problem in previous work using G is that those problems did not require the addition of history in order to differentiate between states, so there was not the enormous explosion of names for the same state that we see when we add history. Furthermore, McCallum, when he used U-Tree on the New York Driving task [16], required history; but, he did not seem to end up with trees that were terribly large. One reason for this is that he terminated the training rather arbitrarily – he states that the trees would have grown larger had training continued. In any case, McCallum indeed also remarks that his trees were larger than he anticipated, and in fact the trees were larger (although better performing) than hand-written trees he used for comparison.

One possible way to solve the tree-size problem would be adopting an action/critic approach, alternating between developing a policy and and learning the value function for that policy from the beginning rather than trying to tack all the value functions together in one tree. We have not, however, explored this.

7.2.2 Redundant Leaves

There is another notable reason for trees growing large in POMDPs. Given the ability to characterize the current state in terms of historic actions and observations, the learning algorithm frequently comes up with multiple perceptual characterizations that correspond to the same underlying world state. For instance, the set of states described by *the focus was on a green block and then I looked up* is the same as those described by *the focus was on a green block and then I looked down, then up, then up*, etc.

Of course, it is clear to us that multiple action and observation sequences can indicate a single underlying state, but it is not so to the algorithm. It cannot agglomerate the data it gets in those states, and it is doomed to build the same sub-tree underneath each one. This leads us to conclude, below, that actual identification of the underlying world dynamics, is probably a prerequisite to effective value-based learning in POMDPs.

8 Conclusion

In the end, none of the approaches for converting an inherently relational problem into a propositional one seems like it can be successful in the long run. The naïve propositionalization grows in size with (at least) the square of the number of objects in the environment; even worse, it is severely redundant due to the arbitrariness of assignment of names to objects. The deictic approach also has fatal flaws: the relatively generic action set leads to hopelessly long early trials. Intermediate rewards might ameliorate this, but assigning intermediate values to attentional states seems particularly difficult. Additionally, the inherent dramatic partial observability poses problems for model-free value-based reinforcement learning algorithms. We saw the best performance with NDP using a fixed window of history; but as the necessary amount of history increases, it seems unlikely that NDP will be able to select out the relevant aspects and will become swamped with a huge input space. And, as we saw in the last section, the tree-growing algorithms seem to be precariously susceptible to problems induced by interactions between memory, partial observability, and estimates of Q-values.

It seems that we will have to change our approach at the higher level. There are three strategies to consider, two of which work with the deictic propositional representation but forgo direct, value-based reinforcement learning.

One alternative to value-based learning is direct policy search [24, 12], which is less affected by problems of partial observability but inherits all the problems that come with local search. It has been applied to learning policies that are expressed as stochastic finite-state controllers [18], which might work well in the blocks-world domain. These methods are appropriate when the parametric form of the policy is reasonably well-known *a priori*, but probably do not scale to very large, open-ended environments.

Another strategy is to apply the POMDP framework more directly and learn a model of the world dynamics that includes the evolution of the hidden state. Solving this model analytically for the optimal policy is almost certainly intractable. Still, an online state-estimation module can endow the agent with a "mental state" encapsulating the important information from the action and observation histories. Then, we might use reinforcement-learning algorithms to more successfully learn to map this mental state to actions.

A more drastic approach is to give up on propositional representations (though we might well want to use deictic expressions for naming individual objects), and use real relational representations. Some important early work has been done in relational reinforcement learning [10], showing that relational representations can be used to get appropriate generalization in complex completely observable environments. Given states represented as conjunctions of relational facts, the TILDE-RT system is used to induce a logical regression tree that tests first order expressions at internal nodes and predicts Q values at the leaves. Internal nodes can contain variables and thus generalize over object identities. Though there is some recent work on applying inductive logic programming techniques (of which TILDE-RT is an example) to stochastic domains, it is unclear how TILDE-RT would work in such cases. Recent work by Driessens et al [8] has adapted the G algorithm to the TILDE-RT system, allowing their relational reinforcement learner to be incremental. This is promising work. As they show, however, problems like we encountered with tree-based systems (i.e. committing to a less-than-optimal split early on and being forced to duplicate more useful subtrees under these splits) to are still in effect, despite the relational representation. Furthermore, in relational domains, it seems the complexity of the learning problem now shifts to the query-generation mechanism—a problem not present in propositional domains where proposed splits are generally just the next attribute in line.

Ultimately, it seems likely that we will have to deal with generalization over objects using relational representations, and deal with partial observability by learning models of the world dynamics. We plan to continue this work by pursuing such a program of indirect reinforcement learning—learning a model and doing state estimation—using relational representations with deictic names for objects in the world.

Acknowledgments

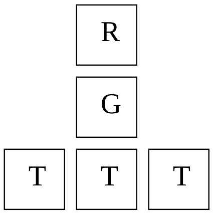
This work was funded by the Office of Naval Research contract N00014-00-1-0298, by the Nippon Telegraph & Telephone Corporation as part of the NTT/MIT Collaboration Agreement, and by a National Science Foundation Graduate Research Fellowship.

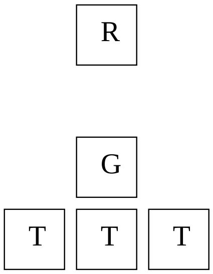
A Informal Analysis of the Different Action Sets

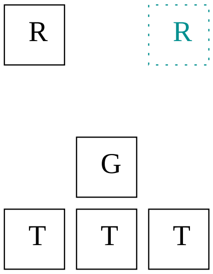
The next figures informally analyze the difficulty of using three action sets (original non-deictic, original deictic, and *modified* deictic) to move through the space of state configurations towards the goal. The first column shows the configurations of blocks seen by each agent on its way through the optimal policy to the goal. The next column lists the available actions, out of all the actions available in its action set (see Sections 4.4, 4.5, and 7.1 for a description of the action sets) that would move the agent strictly forward on its trajectory. Next to the list of forward-moving actions is the approximate probability of randomly selecting one of those actions (out of all possible actions in the action set). The last full column lists the available actions that would move the agent strictly backward (i.e. further from the goal), along with the probability of randomly selecting one of those actions.

Note that the probabilities listed in the figure are approximate; they are intended mainly to illustrate differences between the three action sets.

The take-home message here is that, in the original deictic action set, the likelihood of randomly choosing the right actions to advance with is very low. Compare this with the likelihoods of the full-propositional action set and the modified deictic action set. It is for this reason that using exploration becomes so difficult with the original deictic action set.

World Configuration	Likelihood of Advancing		Likelihood of Regressing	
	advancing actions	likelihood	regressing actions	likelihood
1. start: (probability 1.0)  	*pickup(red#)*	*0.13*	*(none)*	*0.0*

2.

 	
	hand_left()
	hand_right()
		0.25	*put_down()*	*0.13*

3.

 	
	put_down()	*0.13*	*hand_left()*
			hand_right()
				0.13

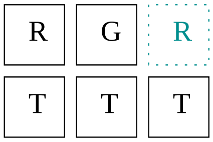
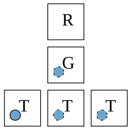
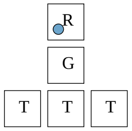
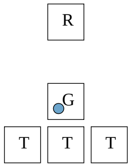
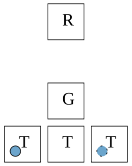
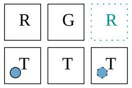
4.  	*pickup(green#)*	*0.13*	*pickup(red#)*	*0.13*

Figure 11: Trajectory for non-deictic agent and its original action set.

World Configuration	Likelihood of Advancing		Likelihood of Regressing	
	advancing actions	likelihood	regressing actions	likelihood
1a. start: (probability 0.8) 	*focus2marker()* $\frac{1}{5}$ *look(red)* *focus_up()* $\frac{1}{4}$	*0.088*	*(none)*	*0.0*

1b. start: (probability 0.2)

	pickup()	*0.83*	*look(green)*
			look(table)
			focus_down()
			focus_left()
			focus_right()
				0.48

2.

	focus_left()
	focus_right()
	look(table) $\frac{2}{3}$
		0.22	*put_down()*	*0.083*

3. 	*put_down()*	*0.083*	*look(green)* *look(table)* $\frac{1}{3}$ *look(red)* *focus2marker()* $\frac{3}{5}$ *focus_left()* $\frac{1}{2}$ *focus_right()* $\frac{1}{2}$	*0.32*
4a. 	*focus2marker()* $\frac{1}{5}$ *look(green)*	*0.1*	*(none)*	*0.0*

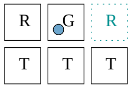
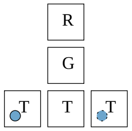
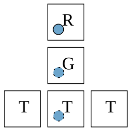
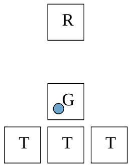
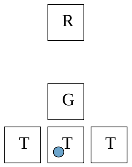
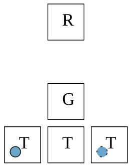
4b.

	pickup()	*0.083*	*focus_right()*
			focus_left()
			focus_down()
			look(table)
			look(red)
			focus2marker() $\frac{4}{5}$
				0.48

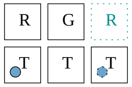
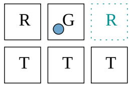
Figure 12: Trajectory for deictic agent with its original action set

Figure 13: Trajectory for deictic agent with *modified* action set

World Configuration	Likelihood of Advancing		Likelihood of Regressing	
	advancing actions	likelihood	regressing actions	likelihood
1a. start: (probability 0.4) 	*look(green)* *look(table)* $\frac{1}{3}$ *look(red)* *focus_left()* $\frac{1}{5}$ *focus_right()* $\frac{1}{5}$	*0.46*	*(none)*	*0.0*
1b. start: (probability 0.6) 	*pickup_top()*	*0.17*	*look(table)* $\frac{1}{3}$ *focus_left()* $\frac{1}{3}$ *focus_right()* $\frac{1}{3}$	*0.33*
2a. 	*focus_down()* *look(table)* $\frac{2}{3}$	*0.28*	*put_down()*	*0.17*

2b.

	focus_left() $\frac{1}{2}$
	focus_right() $\frac{1}{2}$
	look(table) $\frac{2}{3}$
		0.28	*put_down()*
			look(green)
			focus_up()
				0.5

3.

	put_down()	*0.17*	*look(green)*
			look(red)
			focus_left() $\frac{1}{2}$
			focus_right() $\frac{1}{2}$
				0.5

4a. 	*look(table)* $\frac{1}{3}$ *look(green)*	*0.22*	*pickup_top()*	*0.17*
4b. 	*pickup_top()*	*0.17*	*focus_left()* $\frac{1}{2}$ *focus_right()* $\frac{1}{2}$ *look(red)* *look(table)* $\frac{2}{3}$	*0.44*

References

[1] Philip E. Agre and David Chapman. Pengi: An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, 1987.

[2] Leemon C. Baird. Residual algorithms: Reinforcement learning with function approximation. In *12th International Conference on Machine Learning*, 1995.

[3] Dana H. Ballard, Mary M. Hayhoe, Polly K. Pook, and Rajesh P.N. Rao. Deictic codes for the embodiment of cognition. *Behavioral and Brain Sciences*, 20, 1997.

[4] Scott Benson. *Learning Action Models for Reactive Autonomous Agents.* PhD thesis, Stanford University, 1996.

[5] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont, Massachusetts, 1995. Volumes 1 and 2.

[6] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, Massachusetts, 1996.

[7] David Chapman and Leslie Pack Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Sydney, Australia, 1991.

[8] Kurt Driessens, Jan Ramon, and Hendrik Blockeel. Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. In *European Conference on Machine Learning*, 2001.

[9] S. Dzeroski, L. de Raedt, and H. Blockeel. Relational reinforcement learning. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 136–143. Morgan Kaufmann, 1998.

[10] Saso Dzeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine Learning*, 43, 2001.

[11] Natalia H. Gardiol and Sridhar Mahadevan. Hierarchical memory-based reinforcement learning. In *13th Advances in Neural Information Processing Systems*, 2000.

[12] Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6), November 1994.

[13] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *10th European Conference on Artificial Intelligence*, 1992.

[14] Terran Lane and Leslie Pack Kaelbling. Nearly deterministic abstractions of markov decision processes. In *18th National Conference on Artificial Intelligence*, 2002. (to appear).

[15] Mario Martin. *Reinforcement Learning for Embedded Agents facing Complex Tasks*. PhD thesis, Universitat Politecnica de Catalunya, Barcelona, Spain, 1998.

[16] Andrew K. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, Rochester, New York, 1995.

[17] R. Andrew McCallum. Instance-based utile distinctions for reinforcement learning with hidden state. In *Proceedings of the Twelfth International Conference Machine Learning*, pages 387–395, San Francisco, CA, 1995. Morgan Kaufmann.

[18] Nicolas Meuleau, Kee-Eung Kim, Leslie Pack Kaelbling, and Anthony R. Cassandra. Solving POMDPs by searching the space of finite policies. (manuscript, submitted to UAI99), 1999.

[19] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 1994.

[20] Richard S. Sutton. Open theoretical questions in reinforcement learning. In *4th European Conference on Computational Learning Theory*, 1999.

[21] John N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 1997.

[22] Shimon Ullman. Visual routines. *Cognition*, 18:97–159, 1984.

[23] Steven D. Whitehead and Dana H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7(1):45–83, 1991.

[24] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.