Exploring Deictic Representations in Reinforcement Learning

by

Sarah Finney

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2003

Author .....
Department of Electrical Engineering and Computer Science
December 20, 2002

Certified by .....
Leslie P. Kaelbling
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by ..
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Exploring Deictic Representations in Reinforcement Learning

by

Sarah Finney

Submitted to the Department of Electrical Engineering and Computer Science
on December 20, 2002, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

Real-world domains involve objects, and if artificial agents are to interact successfully with our world, they will need to represent and reason about these objects, their attributes, and the relations that hold among them. Thus, the question of how to represent the world to the agent becomes important. This thesis compares two representations that are intended to allow for the use of reinforcement learning techniques in a domain that is essentially relational, a fully propositionalized representation, and a "deictic" one. The full propositional representation is impractical for large domains, and the hope was that a deictic representation would ameliorate many of the scaling problems. However, this thesis argues that a deictic representation raises a new set of problems that may make such a representation infeasible, even in fairly small domains.

Thesis Supervisor: Leslie P. Kaelbling
Title: Professor of Computer Science and Engineering

Contents

List of Figures

List of Tables

Chapter 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 attributes, 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 the relations that hold among them. In order to use established reinforcement learning techniques in an essentially relational domain, we must find some representation for the world that captures the notion of objects and their relations [7]. This thesis explores two such representations, full propositional and deictic.

A fully propositionalized representation of the world specifies a finite domain of objects, and describes the domain using a large set of propositions representing every possible instantiation of the properties and relations in the domain. This representation is known to be reasonably effective in small domains, but since the size of the observation space grows as the world expands to contain more objects, it becomes unwieldy fairly quickly. Also, since each object is given a unique name and reasoned about individually, there is no representational support for generalization over objects.

Starting with the work of Agre and Chapman [1], who were building on Ullman's visual routines [15], and gaining momentum with the debate over the fruitcake problem in the late 1980s (see the Winter, 1989 issue of AI Magazine for the culmination of the 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.

This thesis experimentally compares the performance of learning algorithms in a blocks-world domain using the two representations and analyzes the results. The remainder of this chapter describes more fully the representations used in the experiments, expands on the motivations for each, and outlines the conclusions eventually reached. Chapter 2 details the implementations of the domain, the representations, and the learning algorithms used in the experiments. Chapter 3 presents and discusses the results for the NDP experiments, and Chapter 4 similarly analyzes the results for the G algorithm. Finally, Chapter 5 draws conclusions about the effectiveness of the two representations and suggests some ideas for future work in this area. This thesis builds on work done by Kaelbling, Oates, Hernandez, and Finney [7].

1.1 Propositional Representations

The term *propositional representation* will be used throughout this paper to denote a naive propositionalization of a relational domain. That is, a set of propositions that are either true or false, such as *block2-is-red* or *block2-is-not-on-block3*. The truth values of these propositions are then encoded in a binary vector. For example, given n blocks, there would be kn propositions to specify the colors of the blocks (for k possible colors) and n^2 propositions to specify how the blocks are stacked. This gives us a total of $2^{kn}2^{n^2}$ distinct world states.

Since the objects in the domain are given unique names, this representation provides no mechanism for generalizing over objects. In many domains, objects of a given type are interchangeable: 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 block that is green, and there is nothing on top of it, that block should be picked up." [7]. This rule applies regardless of the particular identity of the green block, but the propositional version requires a separate version of the rule for each name the green block could have.

However, the full propositional representation gives the agent access to every piece of information available about the world. In fully observable domains, traditional reinforcement learning techniques are known to be convergent and to converge to the optimal solution. So, while the representation leads to a perhaps infeasibly large observation space, it is known that at least in small domains reinforcement learning will be successful, if perhaps slow.

1.2 Deictic Representations

The word deictic was introduced to the artificial intelligence vernacular by Agre and Chapman [1] who used a deictic representation in a video game domain. In their representation, the agent (a penguin) avoids the computational load of keeping track of every object in the world by having markers for each of the objects that are deemed useful by the agent's designer. For example, the penguin (who is trying to avoid being stung by a bee) has a marker for *the-bee-that-is-chasing-me* which always points to the correct object, if there is one that is appropriate. The agent then uses this marker, along with similar markers for other relevant objects in the game, to act effectively in the game.

Using a deictic approach in learning is attractive for a number of reasons. Such task-specific attentional control is similar to our own ability to filter out distracting information when trying to attend to and complete a particular task. Beyond this intuitive argument, there are several aspects of deictic representations that appear more promising than a full propositional representation.

First, the agent benefits from some passive generalization through its use of the attentional markers. That is, once the agent has learned to perform a task with the *the-cup-that-I-am-holding*, it will not matter *which* cup the agent is holding for the

task to be successful. In a full-propositional representation, each object must have a unique name, whether or not they share characteristics relevant to a particular task, so the agent must learn to perform the task on each object independently.

Second, since the size of the observation space in a deictic representation only grows with the number of attentional markers, an agent using such a representation could be expected to learn a policy in one world and then use that policy in a world with additional objects. A full propositional representation makes this more difficult due to the unique naming of each object in the world. This aspect of the representation is particularly attractive as a means for incorporating a teacher into the system. As with many human learners, the agent can be led through a series of increasingly complex worlds, performing a particular task, with a "real world" domain as the final goal. It is more difficult to do this with a full-propositional representation that grows in size as the world becomes more complex.

Lastly, since the agent's attentional resources are generally used both for perception and taking actions, the agent's attention is restricted to the part of the world where it is capable of taking action. This focused partial observability means that the focused objects are 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.

1.3 Related Work

While Agre and Chapman showed success in acting with such a representation, they did not attempt to do any learning. Similarly the fruitcake discussion deals only with whether or not it is possible to represent a reasonable policy using deictic markers, not how to learn one. Whitehead and Ballard did do learning in their work using deixis in a blocks-world domain [16], but their approach dealt with the inherent perceptual aliasing of a deictic representation by making the agent avoid those observations which seemed to be ambiguous with respect to underlying state. This limitation is fairly severe, as many tasks may *require* that the agent move through an ambiguous

observation in order to achieve the goal. In our experiments we try to arrive at a learning mechanism that allows the agent to learn a policy in the face of ambiguity that will ultimately allow the task to be completed.

In [9], Martín uses his CANDID algorithm to solve problems in the blocks-world domain. However, he solves the more complex problems by leading the agent through a series of easier tasks that can be used as macro actions in doing the harder task. While his approach is both interesting and successful, we are interested in finding out whether a deictic representation can be useful to an agent in learning to do complex tasks on its own.

1.4 Using A Deictic Representation

There are many ways that the perceptions and actions of an agent can be structured within the general framework of deictic representations. In Agre and Chapman's work, the agent is given markers that are designed to be useful for the task that the agent needs to perform. In our representation, we chose to give the agent a more general set of markers, since we hoped that the agent would be able to learn a variety of tasks in a particular domain, and Agre and Chapman's choice of markers was greatly influenced by the task. To this end we chose to give the agent a focus which it can move from one object to another in the world. We also gave the agent additional markers which it cannot move directly, but only relative to the focus. That is, the agent can either place a marker on the object currently in focus, or move the focus to a previously marked object.

In our representation, the deictic names as described above (i.e. *the-cup-I-am-holding* or *the-bee-that-is-chasing-me*) are determined by the way in which the markers are used rather than being predetermined by the agent's designer. In the blocks-world domain, the block on which the focus is currently placed can be thought of as *the-block-I-am-looking-at* from the agent's perspective. Similarly, if a marker M_1 is placed on the focused object, and the focus is subsequently moved, the object marked by M_1 becomes *the-block-I-was-looking-at.* The focused block can also be *the-block-I*

would-pick-up-on-a-pickup-action or *a-block-in-the-stack-that-would-grow-taller-on-a-putdown-action*, depending on the agent's intention. The example in Figure 1-1 shows the way in which an agent can use its attentional markers to perform a blocks-world task. In the example, the agent needs to uncover the blue block, so it must first remove the top block on the stack.

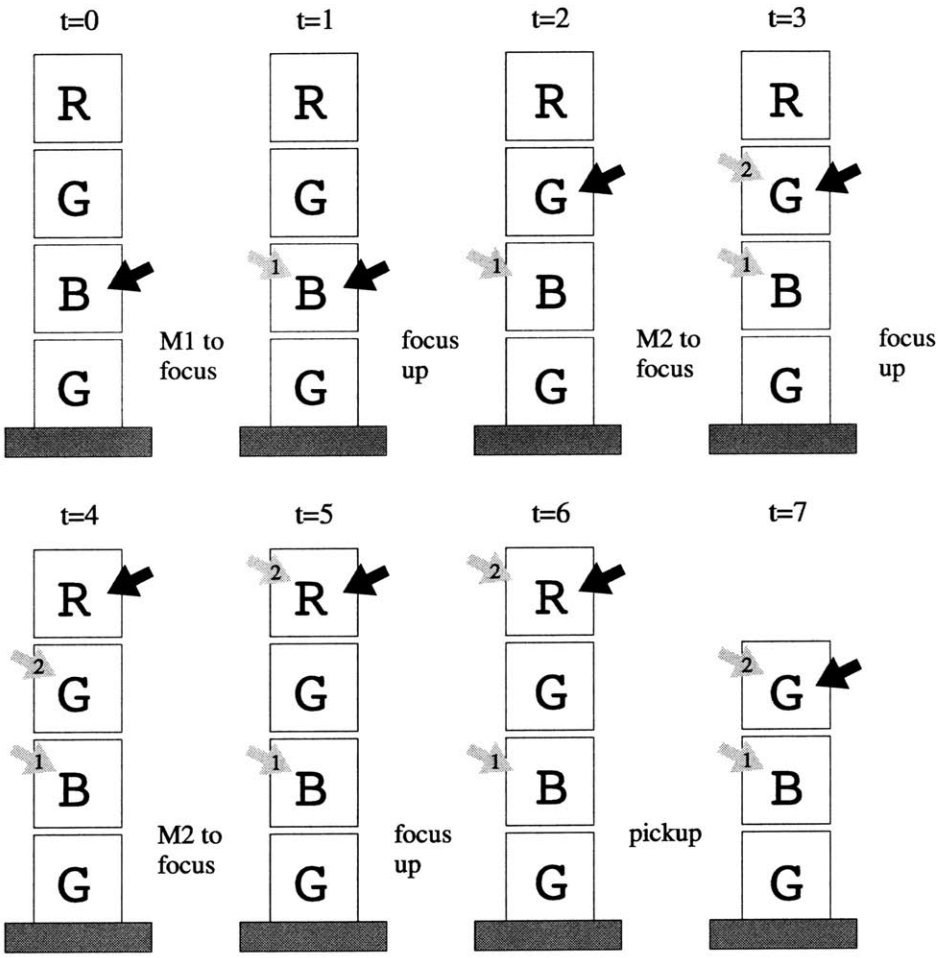


Figure 1-1: Use of markers in blocks-world task.

In time step 0 in Figure 1-1, the focus is used to the mark *the-block-I'm-trying-to-uncover*. In time step 1, marker M_1 is used to remember that location for later. The focus is then moved up in the next time step, becoming *the-block-that-might-be-the-top-block*. To see whether or not the focused block is the top block, M_2 is moved

to the same location (t=3), and focus is moved up again. Since M_2 and the focus do not mark the same block in time step 4, the focus is now *the-block-that-might-be-the-top-block* again. After the procedure is performed again (t=6), M_2 and the focus *do* mark the same block, so the focus now becomes *the-top-block* and can be picked up.

Clearly this procedure takes many more time steps than would be necessary with the full-propositional representation described above. In the propositional representation, each block would have a unique name and its location and color would be known to the agent, so it would be able to determine the top block on the stack containing the blue block and pick it up in one time step. However, to learn this policy the agent must learn it for all permutations of the names and locations of the blocks. What's more, once it has successfully learned to remove the red block from the top of the stack, the process of learning to remove the green block beneath it is made no easier for the experience. The deictic policy however, once learned, generalizes to any situation in which the top block of a stack needs to be removed.

1.5 Using History

Obviously, limiting the information that the agent has about its environment means that the world is no longer fully observable, so the agent may be unable to distinguish underlying states that require different actions to ultimately achieve the goal. However, by looking at its own recent history, the agent may be able to differentiate these important states. Consider the two deictic cases depicted in Figure 1-2, in which the agent's goal is to pick up the green block. The information available to the agent is identical in both cases (the focused block is red), but in the first case, it needs to pick up the red block, and in the second it needs to go find the green block and pick it up to complete the task.

However, by looking back a step in history, it is possible to tell the difference between the two states. Figure 1-3 shows two histories which make it clear which action to take. If the focus was on the table and then a put-down action was performed before the agent found itself looking at the red block, then the red block is now on the

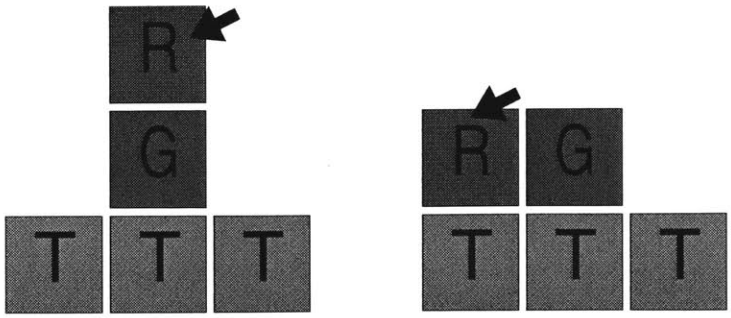


Figure 1-2: Two ambiguous deictic representations.

table, and it is time to go look for the green block. If the focus was on the green block and then was moved up before the agent saw the red block, then the red block must be removed before the green block can be picked up. Thus by adding a sufficiently long history window to the observations that the agent uses to determine its next action, we should allow the agent to arrive at a policy that achieves the goal.

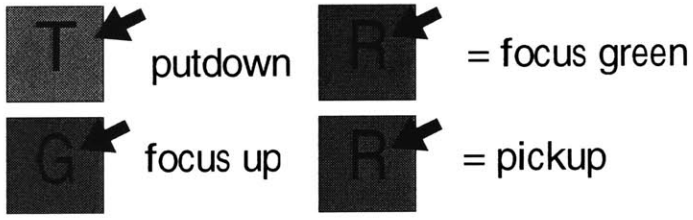


Figure 1-3: Using history to disambiguate two identical observations.

1.6 Selection of Learning Algorithms

In our 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. We chose to use a neural-network function approximator (known as neuro-dynamic programming [3], or NDP) as a baseline, since it is a common and successful method for reinforcement learning in large domains, such as backgammon [14] and elevator scheduling [5]. We hoped to improve performance further by using a function approximator that could use perceptual information selectively based on reward, such as the G algorithm [4] or McCallum's U-Tree algorithm [10]. After some initial experiments with U-Tree, we settled on using a modified ver-

sion of the simpler G algorithm. In neuro-dynamic programming, neural networks are used to approximate the Q-values of each observation-action pair. G and U-Tree both look at reward distributions to determine which observation bits are relevant to predicting reward and divide the state space up accordingly. The next chapter gives the details of both algorithms.

We initially believed that our modified G algorithm would have the advantage over NDP due to its ability to discern which parts of the observation vector were irrelevant to the task. Both algorithms are used to approximate the value functions for each observation-action pair, but the G algorithm uses the reward received to direct the search for relevant observation bits, whereas NDP is forced to learn a reasonable policy for each possible value of all features in the observation vector, regardless of whether of not the feature actually effects which action should be taken next. Given that the full observation space is too large to represent exactly, even in the deictic representation, it would seem that an approximator that tries to explicitly ignore the irrelevant bits in the input would be more effective than one that does not.

1.7 Conclusion Overview

As described above, this work was begun with the hopes that a deictic representation would give our reinforcement learning agent an advantage over one with a full propositional representation. Further, we believed that using an algorithm that tries to ignore the task-irrelevant parts of the observation space would boost our agent's performance still more. Unfortunately, our experiments uncovered a number of issues that make the problem more difficult than we had expected, and ultimately lead our approach to fail.

First, the question of optimality becomes more complex, as discussed in section 3.4. Second, the domain simply is not Markov, no matter how much history the agent is allowed to observe on each time step. Further, as the history window is increased, the learning appears to become more stable, but these policies are decreasingly robust to nondeterminism, again raising the question of what measure should

be used for evaluating a policy. Thus, both with NDP and with G, it seems that with straight-forward reinforcement learning strategies, we cannot learn a policy that is both stable and robust. This issue is addressed throughout the thesis, but particularly in Section 4.3. Lastly, the issue of determining which parts of the observation space are important to the task at hand becomes much more problematic when the agent must consider historical observations as well as current ones. The problem here is that the agent's history is only useful to it once it has a somewhat reasonable policy, but it is difficult or impossible to develop a reasonable policy without first making appropriate historical distinctions. These problems are described in detail in Chapter 4.

However, despite these difficulties, I remain convinced that some method for pruning the observation size of the agent in a task-specific way is necessary for successful learning in complex domains. The real world is simply too large for an approach that seeks to represent every possible distinct world state, but approximating the state may obscure necessary distinctions between individual states, causing many learning methods to fail. Thus a successful learning agent in the real world will need to develop policies that explicitly take actions to disambiguate observations enough to facilitate appropriate future actions. The work in this thesis is intended to help pinpoint the problems that need to be more specifically addressed by a learning algorithm designed to learn reasonable policies in such a domain, and several possible directions for continuing research are proposed in Chapter 5.

Chapter 2

Implementation

To test the usefulness of our deictic representation, we chose to work in a blocks-world domain, first introduced by Whitehead and Ballard [16]. This chapter describes in detail the implementations of the experimental domain, the representations being compared, and the different algorithms used to test the representations.

2.1 Blocks world

Our blocks-world implementation is a two-dimensional world that contains only block objects, each of which has a color and a location. Blocks can be red, green, blue, or table-colored and the table is made up of immovable, table-colored blocks (no other blocks are table-colored). A block that is covered by other blocks cannot be picked up until each of the covering blocks is removed. The world is completely deterministic.

In our experiments, each initial configuration has only one green block and the agent's task is to pick up that block. The agent receives a reward of +2 for successfully completing the task. Attempting to perform a failing action (such as trying to pick up a covered block or a table block) results in a reward of -0.2. Any other action results in a reward of -0.1. Regardless of the representation used for learning, the agent employs an ϵ-greedy exploration policy with an exploration rate of 10%. The domain has a discount factor of 0.9. Each learning algorithm is trained for 200 steps and then tested for 100. Experiments were done with exploration off during testing.

Field	Possible Values	Number Possible Values
ALIVE	true, false	2
HAND_FULL	true, false	2
COLOR_BLOCK0	red, green, blue, table	4
COLOR_BLOCK1	red, green, blue, table	4
⋮ (n entries)	red, green, blue, table	4
ON_BLOCK0	BLOCK1, BLOCK2, ...	$n-1$
ON_BLOCK1	BLOCK0, BLOCK2, ...	$n-1$
⋮ (n entries)	⋮	$n-1$
LOCATION_BLOCK0	1, 2, ... l	l
LOCATION_BLOCK1	1, 2, ... l	l
⋮ (n entries)	⋮	l

Table 2.1: Full Propositional Percept Vector

2.2 Full Propositional Representation

In the full propositional representation, the agent is given a vector of percepts representing all of the information known about each of the blocks. Each block is given a unique name that is determined randomly for each trial. Table 2.1 shows the information contained in each percept vector. The alive bit indicates whether or not the agent was active yet when this percept was observed, and its purpose is explained later. l is the length of the surface on which the blocks are placed (that is, the number of table-colored blocks in the scene), and n is the total number of blocks, including table blocks. The size of this percept vector clearly scales linearly with the number of blocks in the configuration presented to the agent. However, the total number of distinct observations is $2 * 2 * 4^n * (n-1)^n * l^n$. Thus the size of the observation space (that is, the size of table needed to store the value function explicitly) grows exponentially in n.

The full propositional agent also has an action set that refers to each object by its unique name. Table 2.2 shows the action set of the full propositional agent with a brief description of the effects of each action. Again, because of the unique naming, the size of the action set grows with the number of blocks, but only linearly.

Action	Description
HAND_LEFT	Moves hand one stack to the left. Fails if hand is already at far left.
HAND_RIGHT	Moves hand one stack to the right. Fails if hand is already at far right.
PICKUP_BLOCK0	Picks up blocked named "BLOCK0". Fails if block is part of table, hand is full, or block is covered by other block(s).
⋮ (n entries)	Pickup actions for each block.
PUTDOWN	Puts block down at current hand position. Fails if hand is empty.

Table 2.2: Full Propositional Action Set

2.3 Focused and Wide Deictic Representations

There are many different ways to implement a deictic representation, even given our previously stated decision to use generic markers that the agent is allowed to move from object to object. At one extreme, an agent could be given a marker for every object in its environment, effectively giving it a complete propositional representation. At another extreme an agent with one or two markers has a very limited perceptual window on its world. Given enough markers and the ability to place them effectively, the environment may be fully observable, but a more reasonable number of markers and a naive initial policy for moving them around make the world partially observable from the standpoint of the agent.

Two points on this spectrum might be characterized as *focused deixis* and *wide deixis*. In each of these, there is a special marker called the focus and some number of additional markers. Also in both, the agent has actions for moving the markers about, and for moving the object marked by the focus. Table 2.3 shows the action set for both of these representations. The difference between the two is the size of the percept vector. In focused deixis, the agent is given information only about the focus. In wide deixis, the agent is given information about all of the markers. Tables 2.4 and 2.5 show the information contained in each vector. Clearly the size of both observation spaces grows with the number of markers (k), but not with the number of blocks (n).

Action	Possible Arguments	Description
MOVE_FOCUS	UP, DOWN, LEFT, RIGHT	Move the focus one block in the designated direction.
FOCUS_TO_MARKER	M1, M2, ...	Moves the focus to the object marked by the marker given in the argument.
MARKER_TO_FOCUS	M1, M2, ...	Moves the designated marker to the object marked by the focus.
PICKUP	none	Picks up the block marked by the focus. Fails if the hand is full, the block marked is covered by other block(s), or the block marked is a table block.
PUTDOWN	none	Puts the held block down on top of the focused block's stack. Fails if the hand is empty.

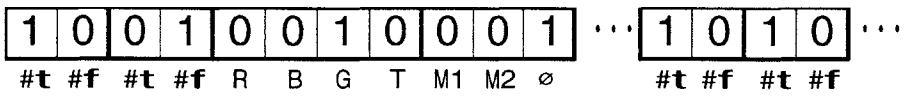
Table 2.3: Deictic Action Set

Field	Possible Values	Number Possible Values
ALIVE	true, false	2
HAND_FULL	true, false	2
FOCUS_COLOR	red, green, blue, table	4
FOCUS_ON	M1, M2, ...	k-1
FOCUS_UNDER	M1, M2, ...	k-1
FOCUS_ON_RIGHT	M1, M2, ...	k-1
FOCUS_ON_LEFT	M1, M2, ...	k-1

Table 2.4: Focused Deictic Percept Vector

Field	Possible Values	Number Possible Values
ALIVE	true, false	2
HAND_FULL	true, false	2
FOCUS_COLOR	red, green, blue, table	4
FOCUS_ON	M1, M2, ...	k-1
M1_ON	FOCUS, M2, ...	k-1
⋮ (k entries)	⋮	k-1
FOCUS_ON_RIGHT	M1, M2, ...	k-1
M1_ON_RIGHT	FOCUS, M2, ...	k-1
⋮ (k entries)	⋮	k-1

Table 2.5: Wide Deictic Percept Vector



h=0	ALIVE =	true
h=0	HAND_FULL =	false
h=0	FOCUS_COLOR =	green
h=0	FOCUS_ON =	no marker
⋮	⋮	⋮
h=1	ALIVE =	true
h=1	HAND_FULL =	true
⋮	⋮	⋮

Figure 2-1: NDP encoding of observation input.

2.4 Neuro-dynamic Programming

Our implementation of neuro-dynamic programming (NDP) uses a three-layer neural network for each of the agent's actions. The input for each network is a vector of bits encoding the current state and the observation-action pair for each time step in the history window. For each observation feature with k possible values, k bits of the input vector, only one of which is high in a given observation, represent that feature, as illustrated by Figure 2-1. This encoding was selected for its compatibility with neural network architecture, despite its obvious inefficiency.

The output of each network is a Q-value for the observation-action pair. SARSA(λ) is used to update these Q-values. The update rule for the parameters (θ) in the networks are as follows [13]:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha[r_{t+1} + \gamma Q(o_{t+1}, a_{t+1}) - Q(o_t, a_t)]\vec{e}_t$$

$$\vec{e}_t = \lambda\gamma\vec{e}_{t-1} + \nabla_{\vec{\theta}_t} Q(o_t, a_t)$$

The partial derivative is calculated using the back-propagation algorithm. For our experiments, we used $\lambda = 0.9$, $\alpha = 0.1$ and $\gamma = 0.9$.

2.5 Modified G Algorithm

The original G algorithm [4] makes use of a tree structure to determine which elements of the observation space are important for predicting reward. The tree is initialized

with just a root node which makes no distinctions, but has a fringe of nodes beneath it, one for each possible distinction that could be made. Statistics are kept in the root node and the fringe nodes about immediate and discounted future reward received during the agent's lifetime, and a statistical test (the Kolmogorov-Smirnov test, or K-S test) is performed on the stored reward data to determine whether any of the distinctions in the fringe is 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 each of the new leaf nodes. Q-values are stored and updated in the leaf nodes of the tree.

Because our domain was not fully observable (in the deictic case), we had to modify the original G algorithm. To begin with, we had to allow the fringe to include historical distinctions. To deal with the fact that the agent has no history at the beginning of a trial, we added a bit to each observation indicating whether or not the agent was alive at that time step, and filled the rest of the observation for those pre-historical time steps randomly. Thus every observation the agent receives matches to some leaf in the tree. As in NDP, we chose to use SARSA(λ) for our update rule, to help ameliorate the partial observability of the problem. The same parameters for λ and the learning rate were used as in NDP.

Lastly, we did not use the D statistic used in the original G algorithm. Rather, we kept vectors of immediate and one-step discounted reward for the set of observations represented by each leaf and fringe node, both for the leaf as a whole and divided up by outgoing action. These vectors in the fringe nodes are compared (using the K-S test) to the corresponding vectors in the parent leaf in examining a split rather than to the other children, since our percept vectors include features with more than two possible values. The K-S test gives us the probability that the reward data stored in the parent and child nodes are drawn from different distributions. If this probability is above a certain threshhold, the distinction is found to be significant and the fringe nodes are added as leaves.

An example may serve to more clearly illustrate the algorithm. Figure 2-2 shows an initial configuration of blocks that the agent is confronted with at the beginning

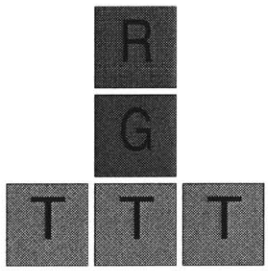


Figure 2-2: Initial configuration of blocks.

of each trial. The location of each of the markers is determined randomly for each trial.

At the start, the tree makes no distinctions, so the agent acts randomly. Actions that lead to failure more often than other actions will come to have lower Q-values, and so the root node will contain Q-values for a policy that takes the safest action(s), such as focus_color(green), which will never fail. After collecting reward statistics for some time, the agent notices that the reward for a pickup action is much lower if the agent's hand is already full, since this action always fails in this case. Thus, the tree grows, as shown in figure 2-3. The figure shows that the policy stored in the leaves reflects the fact that a putdown action fails if the hand is empty and a pickup action fails if the hand is full. The actual policy depends on the particular experience of the agent so far.

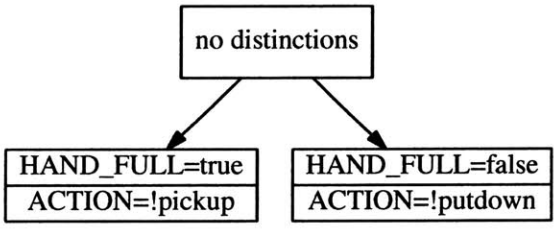


Figure 2-3: Example tree first split

After some more experience, the agent now notices that if the hand is empty, the color of the focus has some effect on the reward received for performing a pickup action. That is, occasionally when the hand is empty and the focus is on the green block a pickup action results in a reward of +2, which is seen under no other circumstances. Thus, another split is made and tree now looks like Figure 2-4.

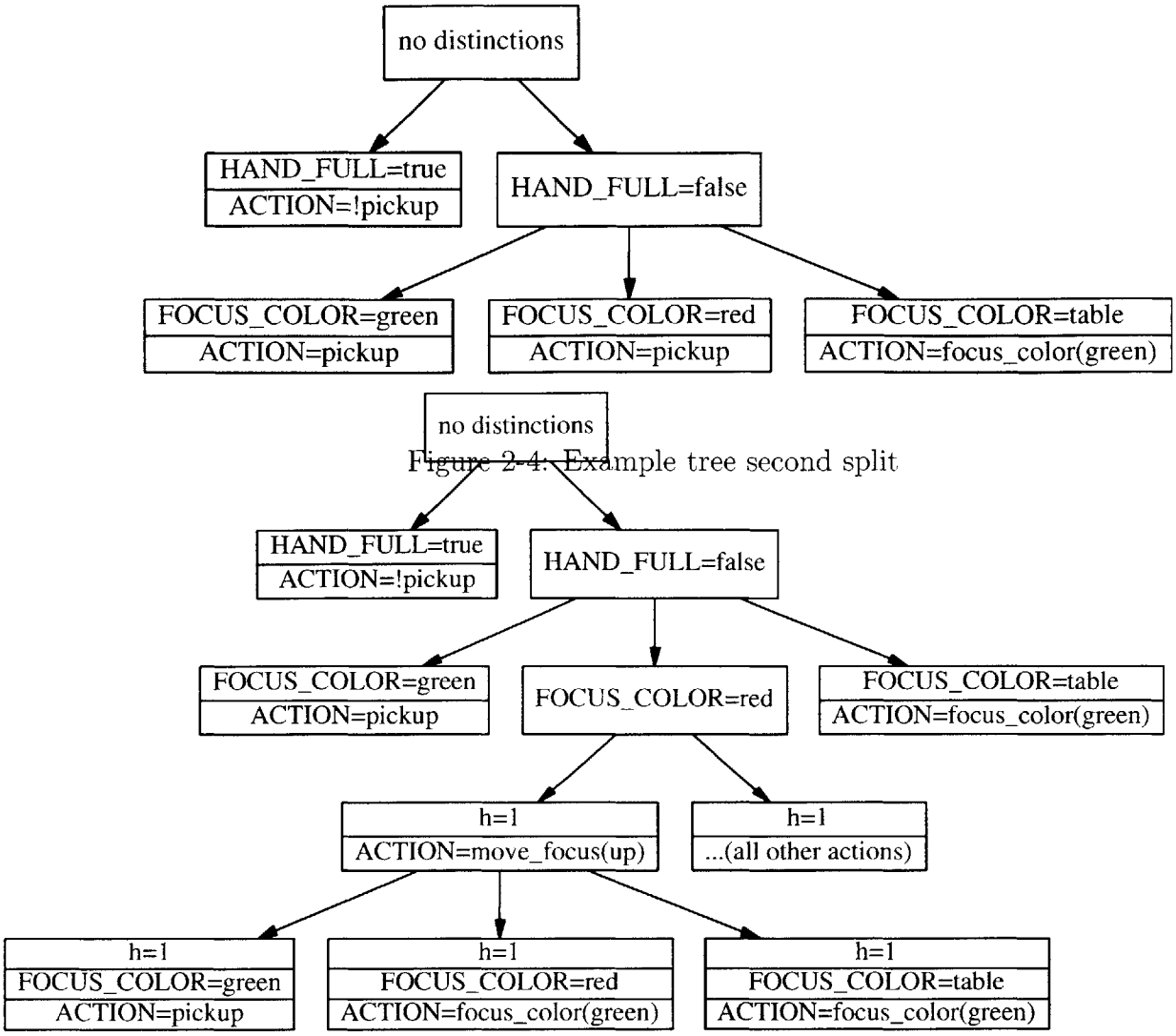


Figure 2-4: Example tree second split

Figure 2-5: Example tree with historical splits

To make the distinction described in Section 1.5, the tree will eventually need to include all of the splits shown in Figure 2-5. In the figure the nodes marked "h=1" represent either the action or percept immediately preceding the current one. Obviously this tree is still not sufficient to represent the optimal policy. More distinctions need to be made for the agent to know when to put down the red block, and when to move the focus so that the block isn't replaced on top of the green block. The idea behind the algorithm is that this process will continue, making distinctions based on statistical differences. Eventually, we hope that the tree will contain the necessary distinctions to represent the optimal policy.

For additional details about the implementations of the world, the representations, and the learning algorithm, see out technical report [7]. The remainder of this thesis presents the results of our experiments and some analysis.

Chapter 3

NDP Results

Our initial experiments were performed with the configuration shown in Figure 2-2. The results, shown in Figure 3-1 were surprising and disappointing. Each plot is the average of 10 different runs of the experiment plotted with a sliding window of 10 data points to smooth out the curves. In the deictic cases, the agent is given a history window of one time step, and in the full propositional the agent is given no history. An average reward per step of 0 represents taking a failing action at every time step, whereas an average reward per step of 1 is the reward associated with the optimal policy. Exploration was turned off during testing so that the policy could be better evaluated.

3.1 Initial Results

In our initial experiment, the most successful agent was the one using the full propositional representation. Clearly the deictic agents are learning much more slowly. However, it appears that they were still making progress when the experiments were truncated, so a longer set of experiments were also performed (Figure 3-2).

The results in the longer trials are even more disturbing. While it is true that the deictic agents continue to improve for some time, and it is also true that every trial eventually finds a good policy, the policy is unstable and the average performance of the deictic agents is bad. The propositional agents, however, seem to converge on the

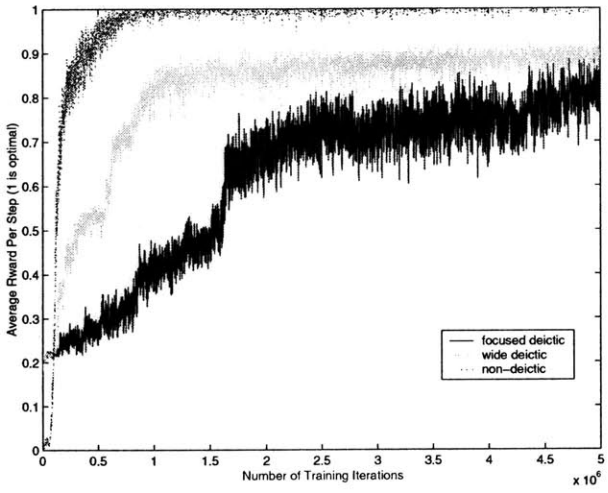


Figure 3-1: NDP results on initial configuration.

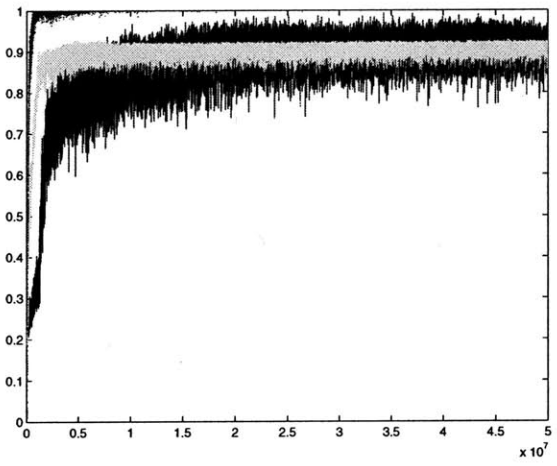


Figure 3-2: Longer trials.

optimal policy and stay there.

3.2 Scaling Up

The above results were discouraging, but our hypothesis about the potential success of the deictic representation argued that as the size of the world increased, the deictic agents would start to out-perform the full propositional ones. Thus, the same set of experiments were performed with the configuration shown in Figure 3-3, which is the configuration in the initial experiments with one additional distractor block. The graph in Figure 3-4 shows the results of these experiments. Clearly the deictic agents are not performing at all successfully.

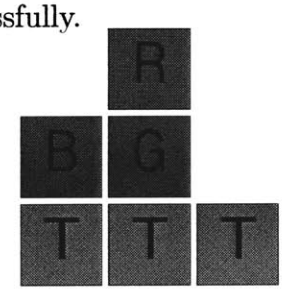


Figure 3-3: Second configuration of blocks.

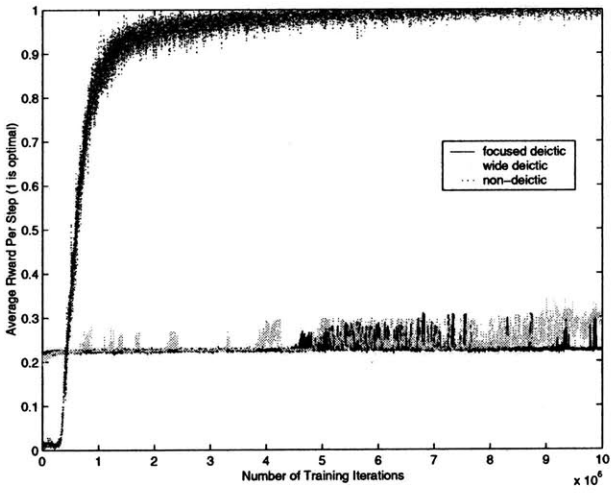


Figure 3-4: NDP results on second configuration.

However, giving the deictic agents longer history windows allows them to make some progress as seen in Figure 3-5. Still they perform much less well than the

propositional agent, but it is interesting that adding history improves the performance, despite increasing the number of distracting bits in the input.

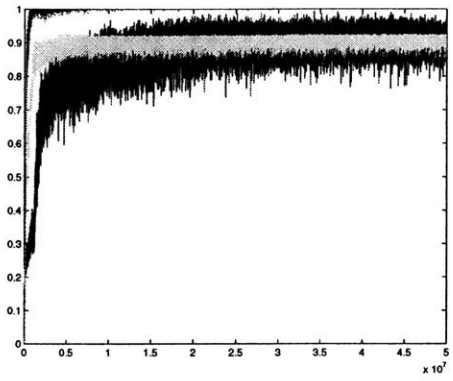


Figure 3-5: NDP results with longer history window.

3.3 Longer History Window

Our initial hypothesis about why the deictic representations failed with NDP was that neural networks are typically bad at learning functions with a large number of input bits, only a few of which matter. However, increasing the number of history steps in the window would seem to make this problem worse, causing the agent to learn more slowly, and this is not the case at all. When the number of history steps in the window is increased to 3 for the deictic case (Figure 3-5), those agents start to show some progress, although they are still not able to learn to solve the task in every trial, and some of the agents still haven't learned the task in the allotted number of training iterations. Increasing the history window still further showed a corresponding increase in performance, although the time required for trials with such large inputs prohibited running enough trials to warrant presenting the data here.

Although the agent only needs one historical time step in order to represent the optimal policy, additional history steps help in the learning process. If the agent needs to perform two actions sequentially during the trial in order to reach the goal, then with only one history step in its input, the agent can't learn anything useful

Shortest Action Sequence
look_color(red)
pickup
focus_left (or focus_right)
putdown
focus_color(green)
pickup

Figure 3-6: The action sequence with the fewest number of steps.

until it happens to perform the two actions one after the other. However, with 5 history steps, for example, the agent could perform the second action several steps after the first, and the fact that the first action was performed is still present in the input vector, so its relevance will not be completely lost. Once a policy is selected that has both actions in the right order, subsequent exploration will eliminate the unnecessary intervening steps, leading to a better policy. Thus, adding history to the input increases the performance of the learning algorithm. However, as the next section explores, the addition of more history also changes the nature of the learned policies.

3.4 Policies and Optimality

Examining the policies learned by the deictic agents in these experiments raises questions about the meaning of optimality in this domain. In terms of the number of steps, the optimal action sequence is shown in Figure 3-6. When the number of historical observations is increased, the agents that find successful policies at all find policies leading to action sequences like this one.

Clearly it is possible for an agent to execute such a policy knowing only the current observation and the last action, as shown in Figure 3-7. However, since the policy depends mostly on the previous action, if the agent ever takes an exploratory action, or if the world has some kind of goblin which puts the red block back on top of the green block after the agent has removed it, the agent will be totally flummoxed. Given

Last Observation	Last Action	Current Observation	Action
ALIVE=false	x	x	look_color(red)
x	look_color(red)	focus=red	pickup
x	pickup	focus=green	focus_left (or focus_right)
x	focus_left (or focus_right)	focus=table	focus_color(green)
x	focus_color(green)	focus=green	pickup

Figure 3-7: The policy leading to the action sequence in Figure 3-6.

Last Observation	Last Action	Current Observation	Action
x	!focus_up	focus=green	focus_up
focus=green	focus_up	focus=red	pickup
x	x	focus=green, hand=full	focus_left (or focus_right)
focus=green	focus_up	focus=green	pickup
All other observations			look_color(green)

Figure 3-8: A more robust, but longer, policy.

that it is looking at the green block, if it has not just performed a focus_color(green) action (or if the focus_color(green) action has been performed at an inappropriate time), it cannot know whether or not it should pick up the green block. Perhaps a better definition of optimal in this domain, then, is one which can start with the observation corresponding to an arbitrary underlying state and reach the goal reliably in the minimum number of steps if no further exploration or non-determinism occurs. That is, if the current state is ambiguous with respect to the action that needs to be performed, the policy will take appropriate actions to figure out enough about the underlying state for the agent to know what to do next to move closer to the goal.

The policy in Figure 3-8, which leads to the action sequence in Figure 3-9, has this property. This policy disambiguates the two different underlying states corresponding to looking at the green block by moving the focus up, thereby revealing whether or not that block is clear. If the next observation is red, the block is not clear, whereas if the next observations is green, it is. So, despite the fact that this policy requires two extra steps, it seems that it is the better policy in the presence of exploration or

More Robust Action Sequence
look_color(green)
focus_up
pickup
focus_left (or focus_right)
look_color(green)
focus_up
pickup

Figure 3-9: The action sequence generated by the policy in Figure 3-8.

non-determinism in the world. As Martín concludes in his thesis [9], since it is not possible to learn behaviors that are optimal in all cases, we need to learn policies that are acceptable in all cases.

Thus, the primary issue raised by the experiments using neuro-dynamic programming is one of measuring the success of a policy. If the only measure of success is the total number of time steps required to acheive the goal, the agent is motivated to develop a policy that takes risky actions when it observes ambiguous information about the world rather than safe, state-disambiguating actions. As is further explored in the next chapter, this contributes to the instability of learning in this domain.

Chapter 4

G Results

The results using the G algorithm, shown in Figure 4, were similarly disappointing. The full propositional agent is not able to learn a successful policy at all. The deictic agents succeed in learning an optimal policy in only a few out of the ten trials, and even then the policy is unstable and the agent spends much of its time acting according to sub-optimal, and even sometimes maximally bad, strategies; thus the average performance is very low. Adding a distractor block again fails to give the edge to our deictic agent over the NDP full propositional agents, as the additional block seems to cause more problems for the deictic agents than for the NDP propositional agent. The remainder of this chapter explores the problems that cause the G algorithm to fail.

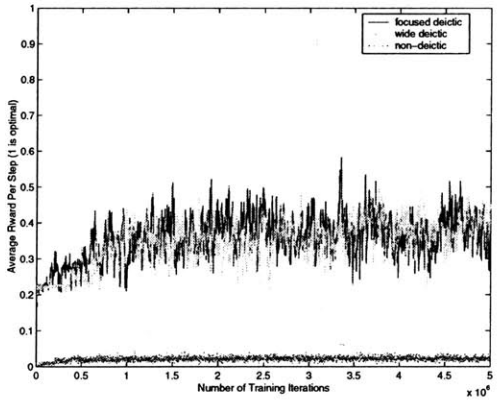


Figure 4-1: Results of the G algorithm.

4.1 Tree Size

To avoid running out of memory, we had to cap the total tree size, and this seems to be fatal to the full propositional case. This is because the tree needs to be very large indeed to account for all the potential renamings of the blocks. For example, given that the agent's hand is empty, the full propositional agent needs to know the names of the green and red blocks and whether or not the red block is on top of the green one in order to know which block to pick up. Figure 4-2 shows the beginning of a tree that is able to make this necessary distinction. In fact, this portion of the tree will have 113 leaf nodes, which gives us a total of 6667 fringe nodes, and this portion of the tree is only sufficient to determine which block should be picked up given that the hand is empty. Another similarly sized portion of the tree will be necessary to determine what to do if the hand is already full. The maximum practical tree size was 3000 nodes, so clearly this algorithm is not a viable solution in the full propositional case. Thus, the rest of the analysis of the G algorithm results focuses solely on the deictic representations.

4.2 Splitting

Initially, we thought that the problem with the deictic trees was similar to the problem with the full propositional trees: the tree simply wasn't allowed to get big enough to represent the optimal policy. In particular, since the domain is now partially observable, the value function depends on the policy being followed. Thus, the tree needs to be big enough to represent the value function for every policy that is followed during the course of learning. In fact, this problem can snowball, with the tree growing larger to represent the value function of the current policy, thereby creating a tree with room to represent a policy with a more complex value function, leading to a potentially infinitely large tree. To show this phenomenon more clearly, we present an example in a simpler maze-world domain [7]. The representation used in the maze-world domain was not particularly deictic, but the domain is partially observable, and

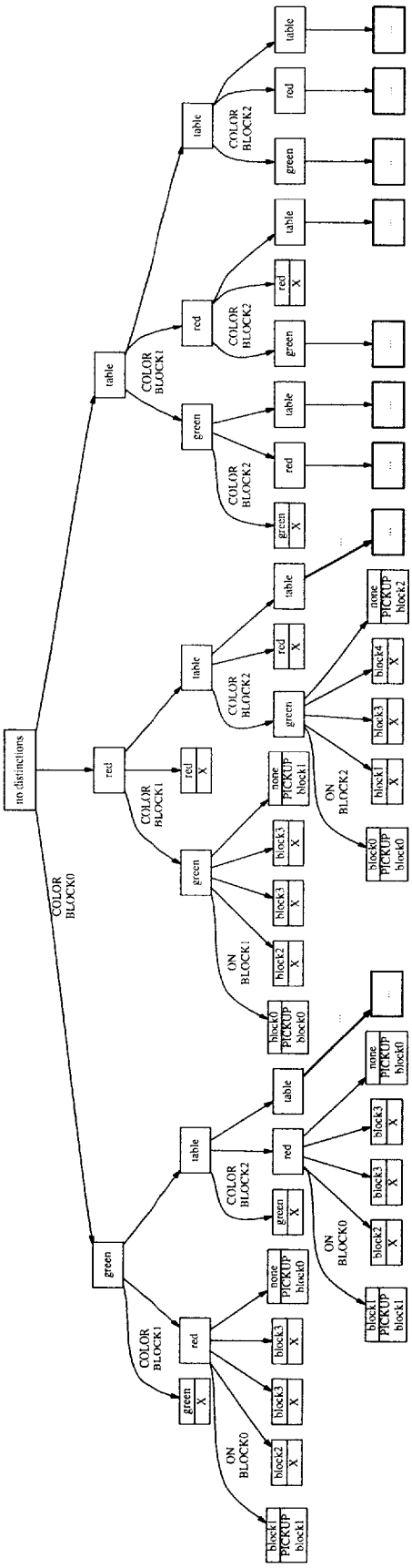


Figure 4-2: A Portion of the Full Propositional Tree

so demonstrates the problem that G has in dealing with partially observable domains.

4.2.1 Maze-world Example – Unnecessary Splits

In our simple maze-world domain, the agent is given a percept vector that tells it whether each of the cardinal directions relative to the agent is clear or blocked, and the action set consists of movements in each of the cardinal directions. The world is completely deterministic, and the percepts are always accurate. The experiments performed to observe the unnecessary splits phenomenon were done in the maze shown in Figure 4-3. This particular maze was chosen because it was the simplest maze we could concoct that was still partially observable, as the states *2* and *5* in Figure 4-3 have exactly the same percept vector, but require different actions to reach the goal state. The agent is positively rewarded for reaching the goal, penalized for each step that does not lead to the goal, and penalized more for attempting to move into a wall.

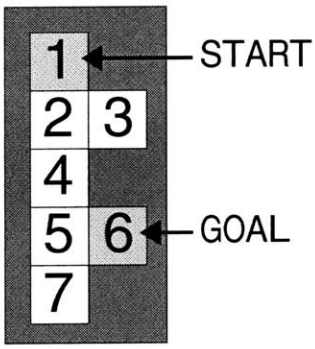


Figure 4-3: Maze-world domain.

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 4-4 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.

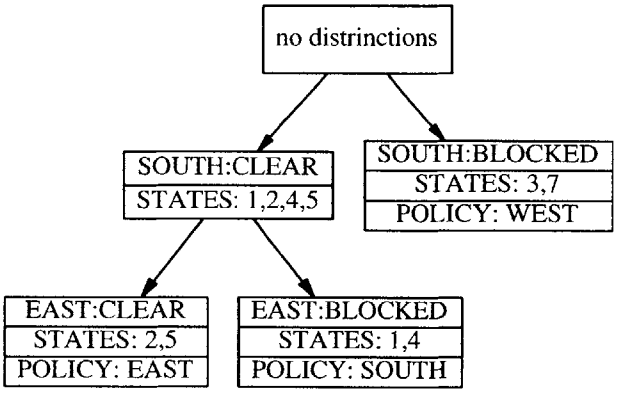


Figure 4-4: Example G tree after first two splits.

Intriguingly, the next distinction is made on the previous action under the node for states *3* and *7*, as shown in Figure 4-5. 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.

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, which takes the agent to 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

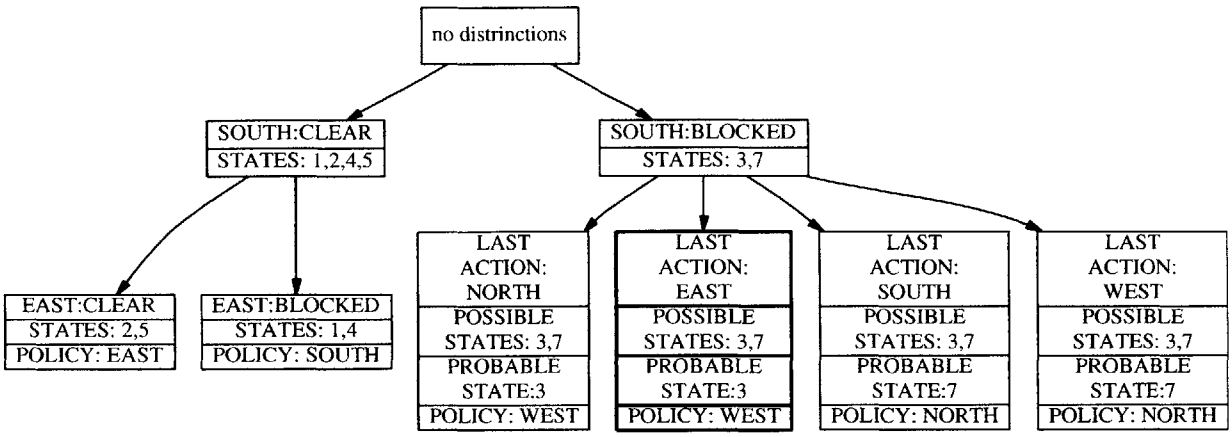


Figure 4-5: Example G tree after the first three splits.

high probability states *3* and *7* and yields a reasonable policy, one that goes north from state *7*, as shown in Figure 4-5.

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 4-6, we do in fact need two different values for state *2*, depending on whether we are visiting it for the first or second time. Later on in learning, the agent will in all likelihood get rid of the extra trip to state *3*, but since the tree does not have a mechanism for getting rid of splits once they are made, the tree contains a permanent split that is not useful. In fact, each unnecessary split made early in the learning process requires that subsequent useful splits be made redundantly under each subtree created by the useless split, which requires a still larger tree to get at the important distinctions. Thus the tree may need to be very large indeed in order to eventually allow for representation of the optimal policy. In fact, since there are an infinite number of possible policies, the tree could grow arbitrarily large.

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

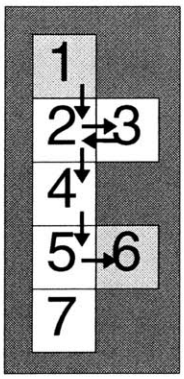


Figure 4-6: A sample policy that may complicate trees.

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.

This problem of G growing arbitrarily 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.

4.2.2 Meaningless Historical Splits

An obvious solution to this problem is to build in a mechanism for undoing splits if they are later found to be unnecessary. However, the nature of the tree structure makes this difficult since there may be additional splits under the children created by the useless split, and it is unclear how to merge these subtrees together.

Instead, in an attempt to solve this problem, we tried using two trees, allowing one to grow for a while and then using the policy arrived at in that tree to grow a new tree, thus eliminating any previously made splits that were found to be unnecessary for the learned policy. We tried this method in our blocksworld domain. While the experiments resulted in smaller trees, they showed no improvement in learning performance at all (see Figure 4.2.2). This result led us to believe that there is a still greater issue which causes the approach to fail.

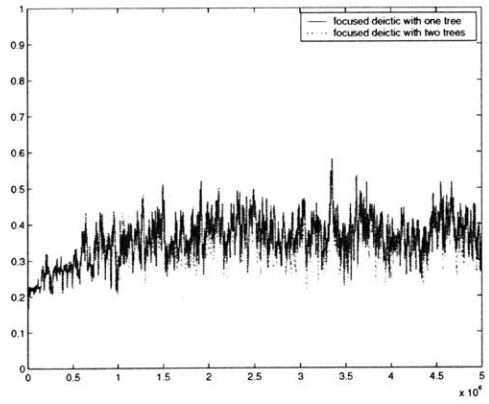


Figure 4-7: Comparison of original algorithm and algorithm with two trees.

Basically, the problem is that, as long as the agent has an ineffective policy, it is not fruitful to look at history to disambiguate important states. If the agent in our maze-world example is following a policy that selects a "west" action in states *2* and *5*, looking at history will not allow for the two states to be discriminated, as the history window will contain long strings of the same percept-action pairs. Only if the agent has a clever policy, such as one that moves south into both states, is it useful to look at the previous percept to tell which of the two underlying states is being perceived.

One way to fix this problem is to require that splits be made contingent upon the policy followed after the percept in question. That is, the mazeworld agent only cares about what it saw one time step ago in the case where it went south between then and now. Or, with the blocks-world agent, given that the focused block is red, the agent only cares about the color of the previously focused block if the intervening action was a focus-up action. However, this approach leads to a combinatorial explosion in the number of nodes in the fringe. Nonetheless, with only one time step of history needed, as in the blocks-world domain, the fringe is not unreasonably large, so we tried altering this algorithm in this way (Figure 4.2.2.

This approach is clearly more successful, but the agents still learn unstable policies. As before, while the agent is able to learn the optimal policy at some point in every trial, it is always unlearned later. To understand why this is the case, we decided to

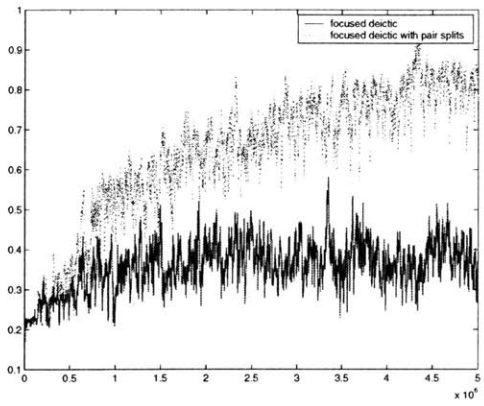


Figure 4-8: Comparison of original algorithm and algorithm with pair splits.

do away altogether with the splitting aspect of the problem and see if we could get the agent to learn stably in a hand-constructed tree. That is, we predetermined the structure of the tree and then tried to figure out why the learning was unstable and what could be done about it. The following section addresses the resulting issues.

4.3 Learning

To investigate the learning problem independent from the splitting problem, I hand-built the tree shown in Figure 4.3. The pair splits described in the last section were used in building this tree as it was a more compact way to describe the necessary leaves. The actions displayed in the leaves represent the optimal policy, according to the criterion described in Section 3.4. The most problematic portion of the tree is shown in Figure 4.3.

In this section of the tree, both of the leaves correspond to states in which the focus is on the green block and the hand is empty. The leaf on the left corresponds to the state where the focus was on the green block one time step ago and the agent chose to move the focus up. In this case, the green block is obviously clear, and the agent is able to learn a policy for that observation that picks up the block. The difficult part is the sibling leaf. This leaf corresponds to any state where either the focus was not on the green block one time step ago or the chosen action was not to

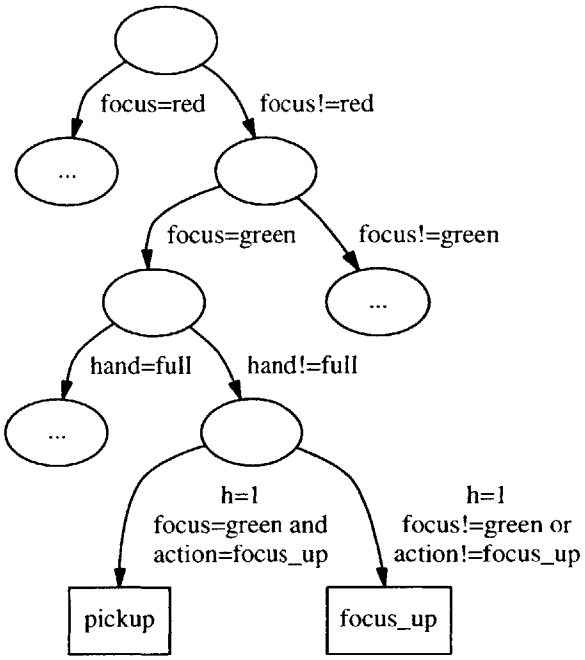


Figure 4-9: Problematic portion of hand-built tree.

move the focus up. There are many cases in which this might also correspond to the green block being clear, but this is not always the case. It is this uncertainty that seems to cause the problem in attempting to learn the optimal policy using this tree.

The results of using TD(λ) learning are shown in Figure 4-11. The average performance appears poor because once again, although each run results in the optimal policy, it is not at all stable. We also tried using Monte Carlo learning (results in Figure 4-12), which is significantly better in terms of stability, but much slower to achieve the optimal policy (note the different x-axis scales in the the two figures). In the Monte Carlo experiments, the agent was allowed to complete a trial before any learning took place. Only the last 100 steps of the trial were used in training, and the Q-values in the leaves were estimated as the average of all of the returns received during the agent's lifetime. However, again, examination of individual runs reveals that while the policies are much more stable than in TD learning, the agent's nonetheless occassionally revert to unsuccessful policies.

Looking only at the Q-values stored in the leaves shown in Figure 4.3, we can see

at least one source of the instability. The agent learns fairly quickly that pickup is the appropriate action in the left-side leaf in the figure. And soon thereafter if learns that moving the focus up is the best action in the other leaf. Once the agent has learned to look up, however, the value of the right leaf starts to go up. A pickup action in the right-side leaf either receives an immediate positive reward (if the green block happens to be free) or receives a penalty and results in a state which matches to the same leaf. Since the value of that leaf is now fairly high, both of these options look pretty good, so the agent eventually settles on a pickup action for the right-side leaf as well. Once the agent decides on this policy, however, the agent quickly learns that this is bad because it gets stuck continually trying to pick up the covered block, and the value for that action plummets. This continues forever, leading to an unstable policy.

In fact, we can analyze a slightly simpler system with one ambiguous observation, and see the source of the instability. The problem lies with the state occupation probabilities, which in turn are determined by the policy being followed. Consider the system shown in Figure 4-13. Each circle in the figure represents an observation that the agent sees. The large circle in the middle represents one observation that corresponds to more than one underlying state (s_1 and s_2). All the other circles represent observations that correspond to only one underlying state. If the agent is in the top state on the left, any action will put it into state s_1, in which action a_2 will cause the agent to stay in that state. However, if the agent comes from the lower of the two states on the left, any action will put it into state s_2 from which action a_2 leads directly to the goal state.

In the blocks-world environment used in our experiments, the ambiguous observation shown in Figure 4-13 corresponds to looking at the green block. If the red block has already been removed (state s_2), then a pickup action (action a_2) will lead directly to the goal. If the red block has not been removed yet (state s_1), then the pickup action (a_2) will not be successful, but will leave the agent in the same state. However, a focus_up action (a_1) will disambiguate the two states. In state s_1 a focus_up action will move the focus to the red block, which can then be picked up and moved off the

$$Q^{\pi}(s_1, a_1) = -0.1 - \gamma 0.1 + \gamma^2 Q^{\pi}(s_2, \pi(o))$$
$$Q^{\pi}(s_1, a_2) = -0.2 + \gamma Q^{\pi}(s_1, \pi(o))$$
$$Q^{\pi}(s_2, a_1) = -0.1 + \gamma 2.0$$
$$Q^{\pi}(s_2, a_2) = 2.0$$

Table 4.1: Q values for ambiguous states in simple system.

green block. In state s_2, a focus_up action will leave the focus on the green block meaning that the green block is free and can therefore be picked up. Now, analysis of this system shows why the learning is unstable.

The true Q-values for each state-action pair are shown in the Table 4.1. As in the blocks-world experiments, a reward of +2.0 and penalties for failing and non-failing actions of -0.2 and -0.1 respectively, is assumed. $\pi(o)$ is the action selected by policy π for observation o.

The Q-values stored in the relevant leaves of the tree will be a weighted average of the true Q-values for the underlying state, weighted according to the probability of occupying each of the underlying states. Thus, if p_1^{π} is the probability of occupying s_1 given that the agent is following policy π,

$$Q^{\pi}(o, a) = p_1^{\pi} Q^{\pi}(s_1, a) + (1 - p_1^{\pi}) Q^{\pi}(s_2, a)$$

.

We now have two questions to answer. First, if the agent is following the policy that always chooses a_2 (the bad action), will the Q-value for a_1 necessarily eventually be better than the Q-value for a_2. Second, once the agent is following the policy that always chooses a_1, will the Q-value for a_2 ever be higher, causing the agent to oscillate between the two policies. The answer to the first question is encouraging. Following policy π_2 (the policy that always chooses a_2), the Q-values are as follows.

$$Q^{\pi_2}(s_1, a_1) = -0.1 - \gamma 0.1 + \gamma^2 Q(s_2, a_2) = -0.1 - \gamma 0.1 + \gamma^2 2.0$$
$$Q^{\pi_2}(s_1, a_2) = -0.2 + \gamma Q(s_1, a_2) \rightarrow -2.0$$
$$Q^{\pi_2}(s_2, a_1) = -0.1 + \gamma 2.0$$
$$Q^{\pi_2}(s_2, a_2) = 2.0$$

For a probability of occupying s_1 of anything greater than 1%, a_1 will have the higher Q-value. Also, when policy π_2 is being followed, the agent will spend a lot of time in s_1, since it will get stuck in that state always performing action a_2. Thus, the Q-value for action a_1 will eventually be higher, and the agent will choose policy π_1 instead. What is more, this is caused by a property of this system that will generally be true of the type of partial observability introduced by a deictic representation. The ambiguity of this state is caused by the fact that the agent has lost its ability to tell whether the current true underlying state is one in which taking a risky action will lead to reward, or back to either the current state or one even further from the goal. Thus, by following the risky policy long enough, since it *never* reaches the goal in this way, it will always eventually decide to follow the more conservative but disambiguating, policy.

Unfortunately, the answer to the second question explains the instability observed during our experiments. Following policy π_1 (the good policy), the Q-values for s_1 are as follows:

$$Q^{\pi_1}(s_1, a_1) = -0.1 - \gamma 0.1 + \gamma^2 Q(s_2, a_1) = -0.1 - 0.1\gamma - 0.1\gamma^2 + 2.0\gamma^3$$
$$Q^{\pi_1}(s_1, a_2) = -0.2 + \gamma Q^{\pi_1}(s_1, a_1) = -0.2 - 0.1\gamma - 0.1\gamma^2 - 0.1\gamma^3 + 2.0\gamma^4$$
$$Q^{\pi_1}(s_2, a_1) = -0.1 + \gamma 2.0$$
$$Q^{\pi_1}(s_2, a_2) = 2.0$$

Clearly, the question of whether or not a_2 has a higher Q-value then a_1 depends on the actual values of γ and p. Let us fix γ at 0.9. Solving for p, we see that a_2 will have a higher Q-value than a_1 if $p < 48\%$. If the agent is performing perfectly, it will see each underlying state exactly once per trial, and things should be stable. However, due to the statistical nature of the problem, it is fairly certain that the probability will vary slightly over time and eventually the occupation probability for s_1 will drop below 48% and the agent will decide that action a_2 is better. As γ is lowered, it becomes more and more likely that the system will think that the bad action is better while following the good policy, until at a γ of just under 0.89, it is guaranteed to happen.

4.4 Conclusions

Our experiments with the G algorithm lead to two conclusions. First, selecting task-relevant pieces of the observation space is made much more complex by the addition of history to the observation space. Fundamentally this is because the history is dependent on the policy, but the policy is now dependent on which pieces of history are included in the observation. Second, the domain is simply not k-Markov, and therefore no amount of historical data will completely disambiguate the true underlying states. As has been noted before (by Baird [2] and McCallum [10], for example), partial observability can lead to instability in traditional reinforcement learning. Our experiments with the G algorithm show that the partial observability introduced by a deictic representation also leads model-free, value-based learning methods to be unstable, even when the agent is given a partitioning of the observation space that is sufficient for representing a good policy.

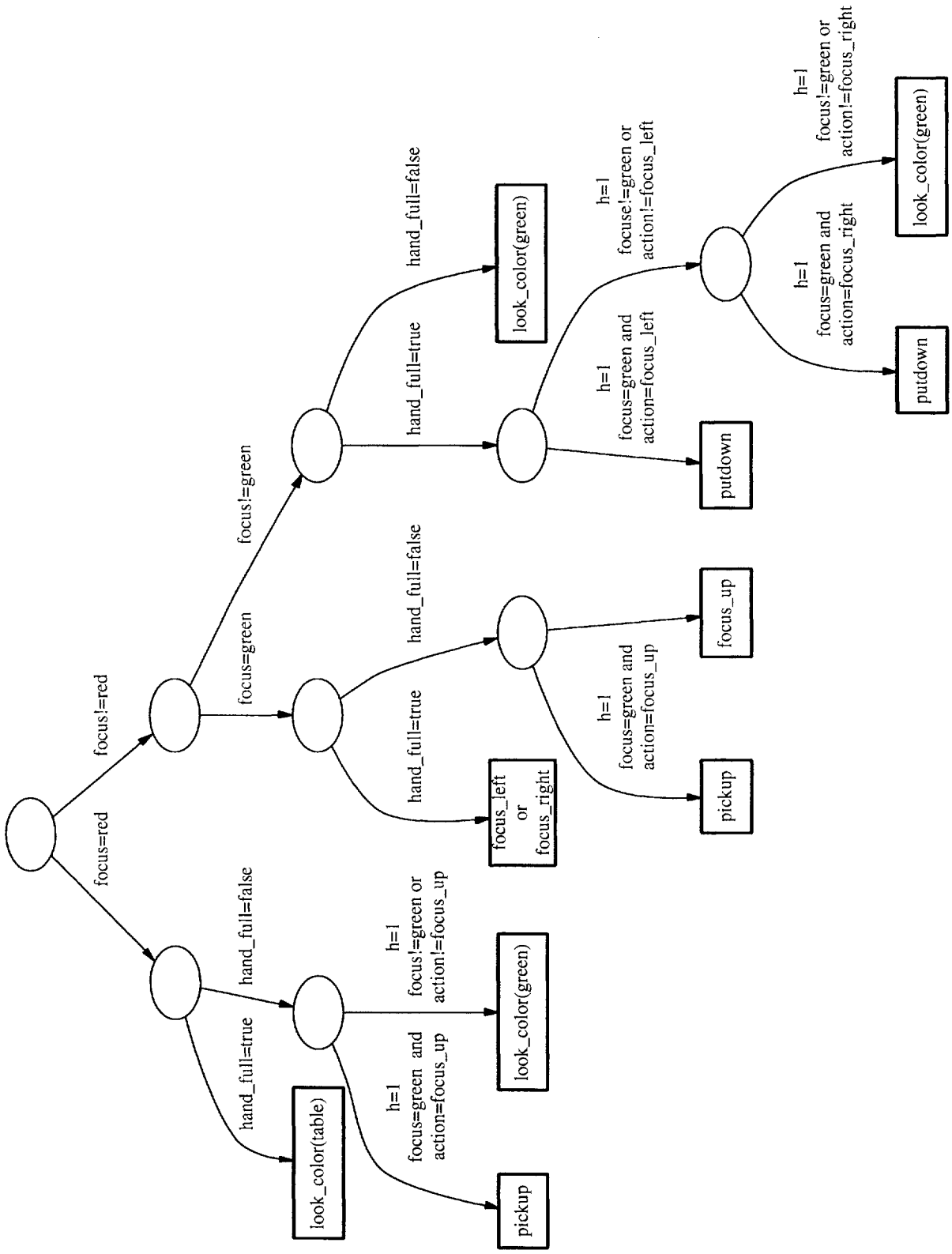


Figure 4-10: Hand built tree used for learning-only experiments.

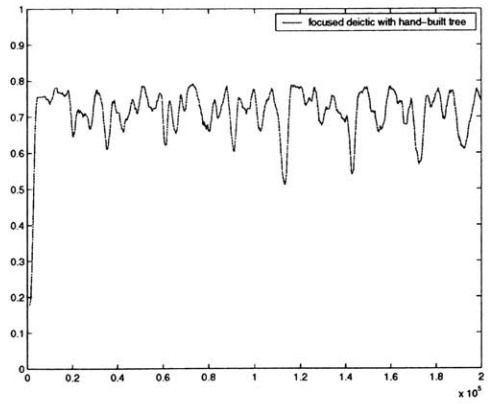


Figure 4-11: Hand-built tree TD(λ) learning results.

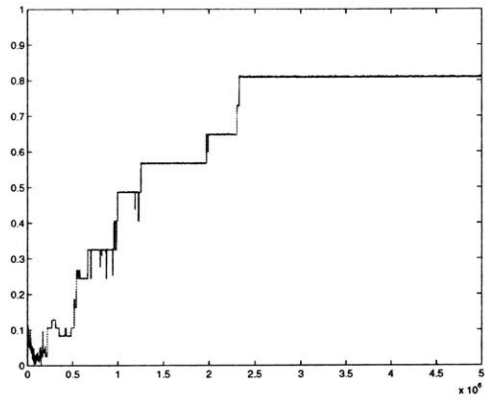


Figure 4-12: Hand-built tree Monte Carlo learning results.

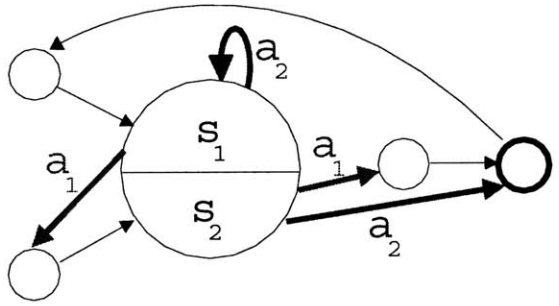


Figure 4-13: A simple system with one ambiguous observation

Chapter 5

Conclusions

5.1 Deixis Breaks Value-Based RL Techniques

The main conclusion of this thesis is that without some modification for explicitly dealing with partial observability, standard reinforcement techniques fail when used in conjunction with a deictic representation. The fundamental reason for this failure is that the deictic representation forces the domain to be partially observable. What's more, the domain is not k-Markov, so no amount of historical information will allow the agent to completely disambiguate similar looking states unless the agent already has a reasonable policy. Lastly, a deictic representation typically requires a certain amount of perceptual exploration to disambiguate states that would otherwise be confusing, and so completely avoiding observations that seem to correspond to more than one underlying state, as Whitehead and Ballard do [16], is not an option. In the case of the blocks-world example, the agent cannot simply decide to avoid looking at the green block at all since it can't tell whether or not the red block has been removed. But, once it sees the green block it must choose actions so that its history window *will* contain the necessary information for determining the appropriate actions.

However, despite this failure, it is clear that some representational change of this sort is necessary if reinforcement learning is to be successful in real-world domains. It is simply not feasible for an agent to learn while acting in the real world if it is not capable of determining which of its sensory inputs is relevant to the task at hand

and ignoring the rest. Thus, I believe that work along these lines is still potentially fruitful.

5.2 Possible Future Work

One aspect of this type of representation that we did not explore, although we initially saw it as a potential advantage, was the "teachability" of an agent with a deictic representation. Since the agent's observation space does not grow with the number of objects in the world, it is possible to construct a series of increasingly complex worlds in which the agent is given a constant task. It seems that such an approach would allow for an external teacher to guide the agent through the learning process in a fairly natural way, hopefully increasing learning performance.

Beyond this unexplored avenue for success with this type of representations, there are also several ways in which the algorithms we tried might be altered to lead to greater success. First, it might prove productive to consider using a continuous exploration policy. From previous work [12], we know that there exists a fixed point for Q-learning in partially observable domains when a continuous exploration policy is used, although it is not yet known whether the corresponding policy will be a good one. Initial experimentation with this change did not show significant improvement, but further exploration is warranted.

Second, it seems potentially fruitful to try to more explicitly look for ambiguous observations and choose actions that work to get rid of the ambiguity rather than always trying to optimize reward. This again raises the question of optimality with a deictic representation, and whether there is perhaps a better measure of what is a "good" action. Perhaps one measure of a good action would be one that leads the agent to an observation in which it knows with high certainty what it needs to do next. For example, if the agent is looking at the green block and doesn't know whether or not the red block is still on top of it, a focus_up action will consistently lead to an unambiguous observation, whereas assuming either of the possible underlying states consistently will cause the agent to get stuck.

Third, since it seems that the agent has little trouble finding the optimal policy using value-based reinforcement learning, but merely has difficulty sticking to it, another idea would be to use such a method during the learning process, but evaluate policies as a whole, as in policy search, to decide which policies to keep. Using such a measure, the agent would not shift from a good policy to an inferior one on the basis of incorrect Q-values, since the whole policy must be better to justify a change.

Another approach would be to apply Martín's CANDID algorithm to our blocks-world problem. Since our task requires an observation vector that is too large to explicitly store in a table as Martín does, we would still need to use some kind of function approximation. Thus is it not certain the algorithm would still perform favorably over other techniques in the face of the still greater partial observability introduced by the approximator, whether it is NDP, G, or some other approximation technique.

Each of the proposed solutions makes the assumption that the observation space has already been partitioned in some way such that a good policy can be represented with the paritions being used to determine the next action. With NDP, this is done by virtue of the function approximation done by the neural networks, so there is little more that can be done to improve this. With the G algorithm, however, we have not yet found a splitting criterion that leads to a reasonably sized tree for representing a good policy in the blocks-world domain. However, it is our belief that it will be more lucrative to look for a good splitting criterion once a successful learning technique has been developed for the kind of tree structure one could hope to have. In particular, detecting ambiguous observations may be useful both in developing good policies, as described above, and in deciding when a group of observations needs to be further split.

5.3 Different Approaches

One alternative to value-based learning is direct policy search [17, 8], 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 [11], 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 hidden state. Solving this model analytically for the optimal policy is almost certainly intractable. Still, an on-line 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 as been done in relational reinforcement learning [6], showing that relational representations can be used to get appropriate generalization in complex completely observable environments.

Bibliography

[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] L. Baird. Reinforcement learning through gradient descent, 1999.

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

[4] 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.

[5] Robert H. Crites and Andrew G. Barto. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 9.* The MIT Press, 1997.

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

[7] Sarah Finney, Natalia H. Gardiol, Leslie Pack Kaelbling, and Tim Oates. Learning with deictic representations. Technical Report (forthcoming), A.I. Lab, MIT, Cambridge, MA, 2002.

[8] Tommi Jaakkola, Satinder Singh, and Michael Jordan. Reinforcement learning algorithm for partially observable Markov decision problems. In *Advances in Neural Information Processing Systems 7*, 1994.

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

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

[11] Nicolas Meuleau, Leonid Peshkin, Kee-Eung Kim, and Leslie Pack Kaelbling. Learning finite-state controllers for partially observable environments. In *15th Conference on Uncertainty in Artificial Intelligence*, 1999.

[12] T. J. Perkins and M. D. Pendrith. On the existence of fixed points for q-learning and sarsa in partially observable domains. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 490–497, 2002.

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, Cambridge, Massachusetts, 1998.

[14] Gerald J. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8, 1992.

[15] Shimon Ullman. Visual routines. *Cognition*, 18, 1984.

[16] Steven Whitehead and Dana H. Ballard. Learning to perceive and act by trial and error. *Machine Learning*, 7, 1991.

[17] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8, 1992.