The Thing That We Tried Didn't Work Very Well: Deictic Representation in Reinforcement Learning

Sarah Finney SJF@AI.MIT.EDU
Natalia H. Gardiol NHG@AI.MIT.EDU
Leslie Pack Kaelbling LPK@AI.MIT.EDU
Massachusetts Institute of Technology, Artificial Intelligence Lab, 200 Tech. Sq., Cambridge, MA 02139

Tim Oates OATES@CSEE.UMBC.EDU
Dept of Computer Science & Electrical Engineering, Univ. of Maryland Baltimore County, Baltimore, MD 21250

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. Introduction

Real-world domains involve objects: things like chairs, tables, cups, and people. Yet most current machine learning algorithms require the world to be represented as a vector of attributes. How should we apply our learning algorithms in domains with objects? 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 (Muggleton & De Raedt, 1994), and specifically by Dzeroski et al (Dzeroski et al., 2001) 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 that has been successful in the planning world (Kautz & Selman, 1992) is to *propositionalize* what is essentially a relational domain. That is, to make an attribute vector with a single Boolean attribute for each possible instance of the properties and relations in the domain.

There are some fairly serious potential problems with such a representation, including the fact that it does not give much basis for generalization over objects. Additionally, the number of bits to be considered grows with the number of objects in the world, even if the task to be accomplished does not become more complicated. An alternative to this full-propositional representation is to create a *deictic*-propositional representation that, intuitively, affords more possibility for appropriate generalization.

The word deictic was introduced into the artificial intelligence vernacular by Agre and Chapman (Agre & Chapman, 1987), who were building on Ullman's work on visual routines (Ullman, 1984). 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. The primary motivation for the use of deictic representation is that it avoids the arbitrary naming of objects, naturally grounding them in agent-centric terms (Ballard et al., 1997). Deictic representations have the potential to bridge the gap between relational and propositional representations, allowing much of the generalization afforded by FOL representations yet remaining amenable to solution (even in the face of uncertainty) by existing algorithms.

Our starting hypothesis was that deictic representations would ameliorate the problems of the full-propositional case. First, we hoped to achieve some passive generalization through use of the markers. 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*. Second, since the size of the deictic observation space only grows with the number of attentional markers, our agent should be able to perform a task in domains with varying numbers of objects more easily than the full-propositional agent. Last, we hoped that our deictic agent would gain an advantage from having its attention restricted to aspects of the world that play a role in its current activity.

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 (McCallum, 1995b), partial observability is a two-edged sword: it may help learning by obscuring irrelevant distinctions as well as hinder it by obscuring relevant ones.

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

2. Experiment Domain

Our learning agent exists in a simulated blocks world. It must learn to pick up a green block by first removing any blocks covering it. This problem domain was introduced by Whitehead and Ballard (Whitehead & Ballard, 1991) in their early work on deixis in relational domains. They developed the Lion algorithm to deal with the domain's partial observability. McCallum (McCallum, 1995a) showed that this partial observability could be directly handled by keeping a short history of observations.

The experiments described in this section differ from previous empirical work with deictic representations in two important ways. First, our goal was to compare the utility of the different representations, rather than to evaluate or develop a learning algorithm tailored for one representation. Second, we have not tuned the perceptual features, actions, or training paradigm to the task but instead developed a set of perceptual features and actions that seemed reasonable for an agent that might be given an arbitrary task in a blocks world.

2.1 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 moves its attention somewhere else, the block becomes *the-block-that-I-was-looking-at*.

For our experiments, we developed two flavors of deictic representation. In the first case, called "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 a marker bound to any block that is above, below, left of, or right of the focus. The second case, called "wide" deixis, receives perceptual information (color and identities of any adjacent markers) for each marked block, not just the focused block. The action set for both deictic agents is:

- *move-focus(direction)*: The focus cannot be moved beyond the top of the stack or 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.

2.2 Full-Propositional Representation

In the fully observable propositional case, arbitrary names are assigned to each block. The agent can perceive a block's color, the location of the block, and the name of any block under it. 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.

2.3 Comparing State and Action Spaces

Propositional representations yield large observation spaces and full observability; deictic representations yield small observation spaces and partial observability. We examine the concrete implications of this next.

Our experiments used two different blocks-world starting configurations (Figure 1). The number of distinct[1] block arrangements is 12 in the *blocks1* setup and 60 in *blocks2*. The underlying state space in the full-propositional case depends on all the ways to name the blocks. This yields 5760 ground states in *blocks1* and 172,800 in *blocks2*. The size of the observation space, however, outpaces the number of ground states dramatically: roughly 10 billion in *blocks1* and roughly 3 trillion in *blocks2*.[2]

The underlying state space in the deictic case depends on possible marker locations rather than on block names. This gives a total of 1200 ground states in *blocks1* and 8640 in *blocks2*. The size of the observation space, however, is constant in both domains: the focused deictic observation space is 512, and the wide deictic observation space is 4096 (Finney et al., 2002).

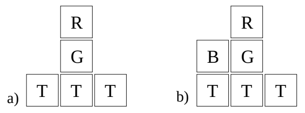

Figure 1. The two blocks-world configurations: a) *blocks1* and b) *blocks2*.

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

3. Choice of Learning 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.

[1]Blocks of the same color are interchangeable.

[2]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.

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 chose Q-learning with a neural-network function approximator (known as neuro-dynamic programming (Bertsekas & Tsitsiklis, 1996), 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 (Chapman & Kaelbling, 1991) and McCallum's U-Tree algorithm (McCallum, 1995b). After some initial experiments with U-Tree, we settled on using a modified version of the simpler G algorithm.

3.1 Neuro-Dynamic Programming

Our implementation of NDP used a two-layer back-propagation neural network for each action. The input to the network was a vector containing the current observation plus some number of previous observations and actions. The output of each network was the estimated Q-value for that action in the state represented by the input vector. As has been observed by others (Tsitsiklis & Van Roy, 1997), we found that SARSA led to more stable results than Q-learning because of the partial observability of the domain.

3.2 G Algorithm

The G algorithm uses a tree structure to determine which elements of the state space are important for predicting reward. A leaf in the tree is the agent's internal representation for a state, and it corresponds to a series of perceptual distinctions useful for predicting reward. Each leaf determines the agent's policy by estimating Q-values for the possible outgoing actions. The tree is initialized with a root node that makes no state distinctions. The root has a fringe of nodes beneath it, where each fringe node represents a possible further distinction. Statistics are kept in the root node and the fringe nodes about reward received during the agent's lifetime. A statistical test determines whether any of the distinctions in the fringe are worth adding permanently to the tree. If a distinction is found to be useful, the corresponding fringe nodes become official leaves, and a new fringe set is created beneath each new leaf node.

At this level of description, the G algorithm is essentially the same as U-Tree. One major distinction between them is 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. The G algorithm makes each new splitting choice based on direct estimates of the Q values from new data. U-Tree uses a non-parametric statistical test (the Kolmogorov-Smirnov test) for splitting nodes, which is more robust than the original test used by G—the version of the G algorithm used in this work uses that test as well. See the technical report (Finney et al., 2002) for a discussion of the differences between G and U-Tree, and the way in which the G algorithm of this paper differs from the original.

4. Experiment Outcomes

We conducted a set of learning experiments in the blocks-world environments shown in Figure 1. The task in both cases was to pick up the green block. The agent received a reward whenever it succeeded at the task, a penalty if it took an action that failed (e.g., attempted to move its hand off the edge of the world, or attempted to pick up the table), and a smaller penalty for each step otherwise. The agent used an ϵ-greedy exploration strategy, with $\epsilon = 0.10$.

The left plot in Figure 2 shows the results from running the NDP algorithm in the *blocks1* domain. The graph shows the scaled total reward[3] received in a testing trial plotted against the number of training steps: at the end of each set of 200 training steps, the state of the learning algorithm was frozen and the agent took a 100-step testing trial during which the total accumulated reward was measured; exploration was not turned off during testing. Each curve for *blocks1* is averaged over 10 experiments, and for *blocks2* over five experiments.

From the graph, we see that the deictic representations did not immediately show the edge we anticipated. We expected, then, to see them gain an advantage with the addition of a distractor block, as in *blocks2*. The results are shown on the right side of Figure 2. Rather than surpassing, or even approaching, the performance of the full-propositional agent, the deictic agents performed worse than before.

Clearly, by adding additional blocks yet retaining the same observation space, we were aggravating the partial observability for the deictic agents. Since selectively using history is a way to manage partial observability, we tried it. Figure 3 shows the results of using G in the two domains. While the deictic agents certainly learn faster than any of the agents learned using NDP, the deictic agents with G never learn the task as *well* as the full-propositional agent does with NDP. Furthermore, the full-propositional agent was never able to get off the ground with G.

[3]For each of the three representations, the reward total was scaled by the maximum reward achievable by the optimal policy.

5. Discussion

Because the goal of our work was to understand the characteristics of these learning approaches, rather than to build a particular working demonstration, we continued with a program of experimentation aimed at elucidating our counter-intuitive results.

5.1 On Deictic and Full-Propositional Representations in NDP

We initially reasoned that the deictic agent had more trouble learning in NDP because the action sequence it needed to learn was longer than the full-propositional agent's. The optimal policy for the deictic agent is to start with a *focus-on(green)* action, then to move the focus up (until the top of the stack is reached), then to pick up the top block and move it to the side. This sequence should repeat until the green block is uncovered and picked up. In both blocks-world setups, this requires a sequence of nine actions. In the full-propositional case, the optimal policy is tedious but shorter. It goes roughly as follows: if *block-1* is green and clear, then the pick up *block-1*; otherwise, if *block-2* is green and clear, then pick up *block-2*; etc. If there is no block that is green and clear, then if *block-1* is on top of a green block and is clear, pick up *block-1*; etc. In both blocks-world setups, the optimal policy requires a sequence of four actions. While the actual policy is short, the same ideas have to represented over and over for each assignment of names to blocks.

Experiments with an extended task requiring the full-propositional agent to take roughly the same number of steps as the deictic agents required on the original task showed that the length of the action sequence was not what made learning difficult (Finney et al., 2002). It seems that the problem with the deictic agent was not the number of actions required but how easily the agent's progress towards the goal could be disrupted by exploratory actions. Whenever the agent erroneously moved the focus, it "lost its place" and essentially had to begin the whole action sequence again.

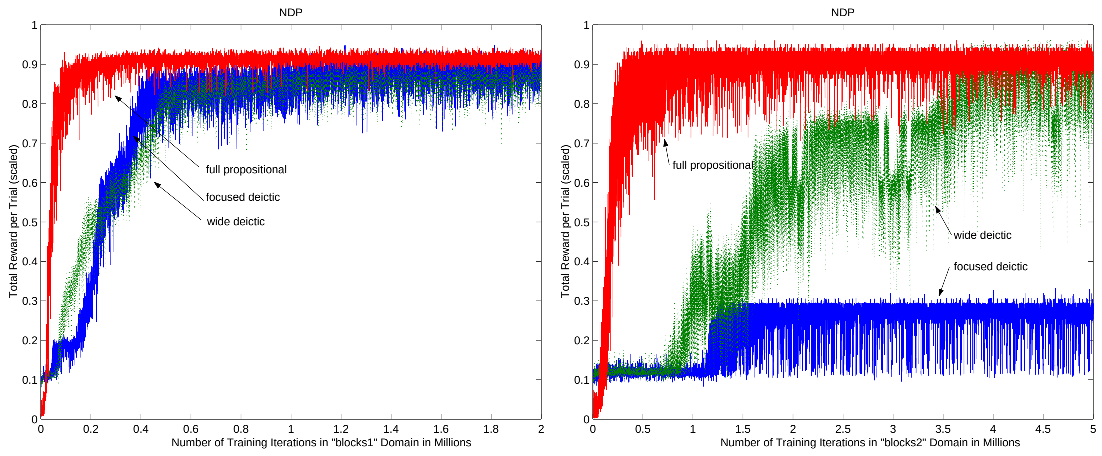

Figure 2. Learning curves for NDP in a) the *blocks1* domain and b) the *blocks2* domain.

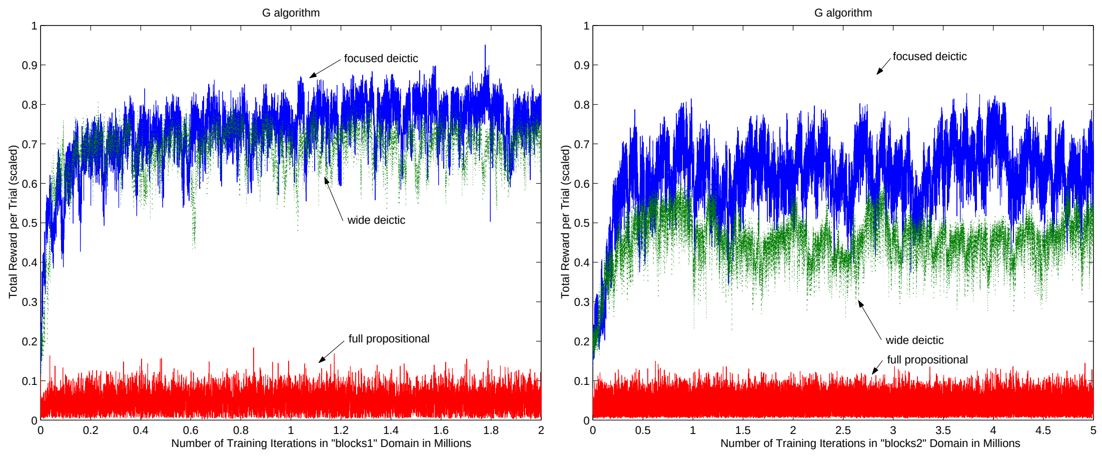

Figure 3. Learning curves for G algorithm in both domains.

In the deictic action set, the focus location is crucial—the behavior of the agent's actions is highly dependent on it. Yet, the none of the marker locations are observable by the agent; indeed, the number of ways to place the markers grows with the number of distractor blocks. The dependence of the *pick-up()*, *marker-to-focus(marker)*, and *focus-to-marker(marker)* actions on the focus location means that it is very easy for the agent to thwart itself by unwittingly moving its focus to useless parts of the state space. This "distractability" reduces the effectiveness of using exploration to make learning progress.

To analyze the exploration problem, we created a modified action set. In this new action set, the *pick-up()* action automatically picks up the block at the top of the stack pointed to by the focus, and the *marker-to-focus(marker)* and *focus-to-marker(marker)* actions were removed. Otherwise, the action set was the same as the original action set.[4] The implication of changing the *pick-up()* action in this way is that the action is now more likely to result in a successful pickup, since the agent cannot even try to pick up blocks that are not clear (i.e., the blocks in the middle of a tall stack). The two other actions were removed for two reasons: first, we found that the agent never used either action once it had learned to complete the task successfully, so they were merely providing distractions and aggravating the exploration problem; second, those distractions were often particularly expensive to the agent since they tended to move the agent's attention to irrelevant parts of the state space. As we shall see, the modified action set rendered exploration much more effective in pointing the agent towards the goal.

To compare the effects on exploration of the original and modified action sets, we measured, for each representation, the number of steps required by a random agent to stumble upon a solution. This metric, the

[4]Interestingly, this modified action set is very similar to the set used by McCallum in his blocks-world experiments (McCallum, 1995a).

mean time-to-goal, was plotted as a function of the number of distractor blocks. Figure 4 shows the result of this experiment. It is dramatically clear from the figure that the modified deictic action set makes it much easier to achieve the goal via a random walk; with the modified actions, exploration in the deictic system scales in the same way as in the propositional system. Furthermore, learning experiments with the modified action set show the deictic agents able to learn in *blocks2* as expected (Finney et al., 2002).

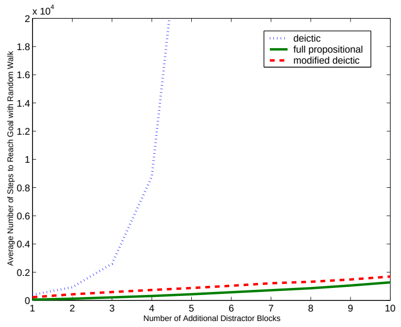

Figure 4. The mean time-to-goal for different action sets plotted against the number of distractor blocks.

It is clear that naïvely chosen deictic state and action sets not only fail to exhibit the advantages we expected to find, but introduce new challenges to learning that must be overcome in order to make effective use of such representations. We conclude that it is possible to tailor the action set to the task so that a deictic representation is more feasible, but the flexibility of such an action set is obviously more limited. It seems that an action set that includes the ability for the agent to control its own attentional focus inherently increases the difficulty of the exploration problem by allowing it to easily spend a lot of time exploring a useless part of the state space. In addition, the fact that there is information about the domain stored implicitly in the location of the focus (and potentially the markers) means that exploratory actions that "lose focus" make it very hard for the agent to succeed initially.

Another strategy for dealing with the increased exploration difficulty is to guide exploration. This strategy turned out to be very important for McCallum in his blocks-world task (McCallum, 1995a), where the exploration was guided by a human. It is clear that investigating exploration strategies is an important direction for future work.

5.2 On NDP and G

The common wisdom is that function approximators like neural nets are appropriate for problems in which all of the input attributes are relevant to some small degree, and that decision trees are appropriate when the function is well represented in terms of an unknown subset of the input features. In the full-propositional representation, any of the bits could be important, so it seems reasonable that NDP worked well. In the deictic representation, we included many historical observations into the input vector, not knowing which ones might be relevant to the problem. In this situation, we might expect the tree-growing algorithms to be better: they should build a representation that reveals only enough of the hidden state to do the job.

Our surprising results with the G algorithm seem to be primarily caused by the trees growing much larger than expected; they grew without reaching a natural limiting state. To avoid running out of memory, we had to add an arbitrary cap on the size of the trees.

While the deictic agents initially learn faster than with NDP, they stopped making progress upon reaching the tree-size cap, and therefore never completely learn the task. Similarly, the full-propositional agent made no progress at all before the tree reached its maximum size. Our intuition for this was the following: tree splits are made throughout the course of learning, which means that the structure of the tree is required to have room enough to represent not only the value function for a particular policy, but for all policies that are followed during the course of learning. To verify this intuition, we moved to a simpler and more easily visualized maze domain.

5.2.1 Unnecessary Distinctions

We will use a very small maze example to illustrate why the trees grow without bound. The maze is shown in Figure 5. The agent only observes whether there is a wall in each cardinal direction, so this simple maze 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 an observation and may move in one of the four directions. The agent is rewarded for reaching the goal, penalized for each step otherwise, and penalized slightly more for attempting to move into a wall.

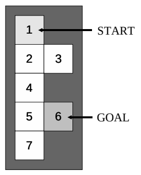

Figure 5. A simple partially observable maze-world.

In one trial, the G algorithm made the following sequence of distinctions. 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 6 shows the tree at this point during the learning process. Clearly this policy is not yet optimal, since we have distinguished states *2* and *5*.

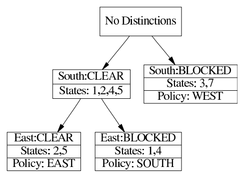

Figure 6. Example G tree after first two splits.

Intriguingly, the next distinction is made on the previous action under the node for states *3* and *7*, as shown in Figure 7. At first glance, this looks like a useless distinction to make.

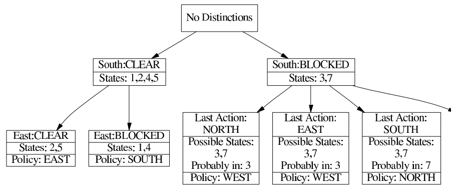

Figure 7. Example G tree after the first three splits.

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 when the agent visits state *3*, it has generally just performed an east action. However 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 7. Intermediate policies seem to lead the algorithm to make more distinctions in an attempt to fully represent the value function under each policy. 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 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. This problem of G growing unreasonably large trees in POMDPs seems very difficult to address. Note that this problem will almost certainly be exhibited by U-Tree, as well. 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.

5.2.2 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.

6. 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.

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 (Williams, 1992; Jaakkola et al., 1994), 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 (Meuleau et al., 1999), 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 not entirely on deixis; 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 (Dzeroski et al., 2001), showing that relational representations can be used to get appropriate generalization in complex completely observable environments.

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 pursue 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.

References

Agre, P. E., & Chapman, D. (1987). Pengi: An implementation of a theory of activity. *Proceedings of the Sixth National Conference on Artificial Intelligence.*

Ballard, D. H., Hayhoe, M. M., Pook, P. K., & Rao, R. P. (1997). Deictic codes for the embodiment of cognition. *Behavioral and Brain Sciences*, *20*.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.

Chapman, D., & Kaelbling, L. P. (1991). Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. *Proceedings of the International Joint Conference on Artificial Intelligence.*

Dzeroski, S., De Raedt, L., & Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, *43*.

Finney, S., Gardiol, N. H., Kaelbling, L. P., & Oates, T. (2002). *Learning with deictic representations* (Technical Report (forthcoming)). A.I. Lab, MIT, Cambridge, MA.

Jaakkola, T., Singh, S., & Jordan, M. (1994). Reinforcement learning algorithm for partially observable Markov decision problems. *Advances in Neural Information Processing Systems 7.*

Kautz, H. A., & Selman, B. (1992). Planning as satisfiability. *10th European Conference on Artificial Intelligence.*

McCallum, A. K. (1995a). Instance-based utile distinctions for reinforcement learning with hidden state. *12th International Conference on Machine Learning.*

McCallum, A. K. (1995b). *Reinforcement learning with selective perception and hidden state*. Doctoral dissertation, University of Rochester, Rochester, New York.

Meuleau, N., Peshkin, L., Kim, K.-E., & Pack, K. L. (1999). Learning finite-state controllers for partially observable environments. *15th Conference on Uncertainty in Artificial Intelligence.*

Muggleton, S., & De Raedt, L. (1994). Inductive logic programming: Theory and methods. *Journal of Logic Programming.*

Tsitsiklis, J. N., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control.*

Ullman, S. (1984). Visual routines. *Cognition*, *18*.

Whitehead, S., & Ballard, D. H. (1991). Learning to perceive and act by trial and error. *Machine Learning*, *7*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*.