Learning in Worlds with Objects

Leslie Pack Kaelbling
Tim Oates
Natalia Hernandez G.
Sarah Finney
Artificial Intelligence Laboratory
Massachusetts Institute of Technology

December 13, 2000

This work is sponsored by NTT and by the Office of Naval Research.

1 Introduction

As we attempt to build systems that learn in environments made up of collections of objects with varying properties and relations, it is highly likely that we will have to develop new learning techniques. In this research note, we explore some of the representational and algorithmic issues involved in learning about objects, and outline our first experiments.

We'll assume that we are building an agent that is embedded in its environment. The agent can take actions that change the underlying state of the environment, and can make observations of the environment's state. In general, the observations will not reveal the true state of the environment: they will be noisy, and many underlying environmental states will look the same. The agent also has a special scalar input signal, called reward, which is correlated with the underlying state of the environment. The agent's goal is to act in such a way as to gain a large amount of reward over time. It will have to learn about its environment in order to learn how to behave in a valuable way.

In all of this work, the goal will be to learn to carry out some externally rewarded task, by taking actions in the environment based on perception. Although we will try to specify the learning task in a relatively representation-independent way, we believe that the choice of internal representation and associated learning algorithms will have a huge effect on the efficiency, and even possibility, of the learning process.

2 Simple Example Domain

Consider a simple blocks-world domain, in which there are multiple blocks, stacked in piles on a table. The blocks are made of different materials and are painted different colors.

We'll assume the blocks are stacked in orderly piles, so that each block is either on the table or on a single other block. The agent can perceive all of the blocks, their colors and materials, and their spatial relations. It can take action by picking up any clear block (a block is clear if it doesn't have another block on top of it) and putting it down on any other clear block or on the table.

Within this general domain, the agent is given a task by rewarding it whenever it puts the blocks in front of it into the appropriate configuration. We use the term "task" to describe the kinds of block configurations for which the agent is rewarded and "initial state" to describe an initial configuration of objects (this may also be called a "problem instance"). Possible tasks include:

- Pick up a green block.
- Cause every green block to have a red block on top of it.
- Make a copy of the stack of blocks that is on the far left of the table.

Our first goal will be to learn general strategies for solving individual tasks from all (or many) initial configurations. Even within a single task, we can investigate "transfer" of knowledge to see whether learning to solve one initial state will allow the agent to solve other, similar, initial states more quickly.

We are also interested in the more general question of how learning to solve a particular task can speed learning of other tasks in the same domain.

(Similar domain used by Whitehead and Ballard; Baum).

3 Perception and Action

Most of the work on reinforcement learning methods assumes that the agent has complete, perfect perception of its environment state. In the simple blocks world described above, that might not be implausible; but in any kind of realistic domain, it is. You can't see everything that is going on in your building; there are occlusions and distant objects. Furthermore, if you *could* see everything, you would still have to focus your attention on some small subset of the available information.

Ideas of attention, both physical attention due to moving gaze and implicit attention to areas within a fixed visual field, have been captured in work on active perception. The idea is that an agent makes observations of its environments through a fairly narrow

channel, such as a visual fovea, and can take action to move the channel around. This converts the perceptual problem from one of dealing with a vast amount of data in parallel to a sequential control problem that explicitly selects the data required.

Ullman did early work on characterizing a basic set of visual primitives and describing their aggregation into *visual routines*. Chapman took these ideas and applied them in the visual system for a video-game player. We'll use a similar set of visual primitives in thinking about the blocks world.

One important idea is that of a *visual marker*. The agent is assumed to be able to keep track of a small number (say 5) of objects in its visual field. It does so by "putting a marker" on an object. Having a marker on an object allows it to be the subject of visual queries, such as "What is the color of the marked object?" or "Is marked object 1 above marked object 2?" Markers can be placed on objects by relative motion: "move marker 2 to the top of the stack" or "move marker 3 one square to the left". Markers can also be placed on objects via *visual search*. For simple image properties, called "pop-out" properties, this search can be done very quickly in the human visual system (cite Triesman). Thus, we can have as a visual primitive "move marker 3 to a red object" or "move marker 4 to a vertically-oriented object". More complex image properties require a slower, more serial, search process. We may include them in our visual repertoire, but perhaps model their use with an additional time cost.

With just a small set of visual markers, the amount of information available in a single observation is fairly small; but by moving the markers around, it is possible, over time, to gain a fairly complete picture of the environment. This kind of partial observability requires the agent's strategy to have memory, allowing it to combine observations over time into an internal "view" of the current situation. It also requires a much more sophisticated set of learning techniques, which we will discuss in more detail later in this document.

Visual markers can also play a role in specifying parameters for actions. Thus, a grasp action might always grasp the object that marker1 is on; or the go-to action might cause a mobile robot to visually servo toward a marked target.

We close this section with two formulations of active perception in the simple blocks-world domain. The first has a "wide" view, giving immediate access to all properties and relations of marked objects; the second has a "narrow" view, requiring explicit queries.

Wide view The agent has a fixed set of k markers, each of which can be on a block, on the table, or unused. The agent can immediately observe all properties of all marked objects, and all relations between marked objects. With two perceptible properties (color and material) and one relation (on), that makes $2k + k^2$ observables. Also, the agent will have some way of knowing whether any two markers are on the same object, and of knowing when various operations have failed (for instance, **find** will fail if there are no blocks with the desired property).

The actions available are:

- **pickup** the object that marker 1 is on (if it is clear)
- **putdown** the currently held object on the object that marker 2 is on (if it is clear)
- **move-marker**(i, *dir*) for each marker i and each direction (up, down, left, right), move the marker one unit (it's not yet clear what a unit is, exactly) in that direction
- **move-to-top**(i) for each marker i, move it to the block that's on the the top of the stack of blocks containing the block the marker is currently on
- **find**(i, *property*) for a given marker i and visual property of a block, put the marker on some block with that property. Properties will include colors and materials.[1]

Focused view The agent has a fixed set of k markers, plus a privileged marker called the focus. Most operations can only be done on the focus; but relations can be observed between the focus and other markers, and the focus can be easily moved to the location of one of the other markers. The actions available are:

- **pickup** the object that the focus is on (if it is clear)
- **putdown** the currently held object on the object that the focus is on (if it is clear)
- **move-focus**(*dir*) for each each direction (up, down, left, right), move the marker one unit (it's not yet clear what a unit is, exactly) in that direction
- **move-to-top** move the focus to the block that's on the the top of the stack of blocks containing the block the focus is currently on
- **find**(*property*) for a given visual property of a block, put the focus on some block with that property. Properties will include colors and materials.[2]
- **mark**(i) put marker i on the block that the focus is on
- **focus**(i) put the focus on the block that marker i is on.

The agent can observe all of the properties of the object that is in focus, and the relations between the focused object and each of the marked objects. With two perceptible properties and one relation, that makes $2 + \mathrm{k}$ observables.

[1] There are some interesting questions having to do with systematicity. In order to do explicit visual search, or enumeration of all the red blocks, for example, it might be nice to restrict this action to a region of the image, in order to rule out some objects already accounted for.

[2] There are some interesting questions having to do with systematicity. In order to do explicit visual search, or enumeration of all the red blocks, for example, it might be nice to restrict this action to a region of the image, in order to rule out some objects already accounted for.

Narrow view In this model, the agent still has a fixed set of k markers. However, no properties or relations are immediately observable; the agent must explicitly query to find anything out. Thus, the observation consists of a single bit (or small number of bits, to allow the query to be, for example, the color of a particular block and the answer to be one of a small set of possibilities).

The available actions include all of the previous actions, as well as the following query actions:

- **query-color**(i) ask the color of the block that marker i is on
- **query-material**(i) ask the material of the block that marker i is on
- **query-on**(i,j) ask whether the block that marker i is on is on top of the block that marker j is on

We would add additional actions for any other properties or relations that the agent might wish to observe in the environment.

Wide vs narrow The question of whether to have a wide vs narrow view is open. With a wide view (and many markers), the agent needs less memory because it can "see" more of the situation at once. In addition, its action space is fairly constrained. The limiting case of a wide view is to have a marker on every single block. Then, the problems have to do with too large an observation space. With a narrow view, there is a tiny observation space, but a huge action space and a requirement for lots of memory. One of our main early experimental efforts will be to examine the ramifications of a choice of view.

The focused view was motivated most directly by Ullman's idea of visual markers and seems to correspond well to natural foveated visual systems.

4 What to Learn?

An agent embedded in a complex environment, trying to act so as to gain reward, can learn knowledge about its environment in different forms. There are trade-offs involved in learning the different forms of knowledge.

World dynamics The most general thing an agent can learn is a model of the world dynamics. That is, a predictive model that maps a state of the world and an action of the agent into a new state of the world (or a probability distribution over new states of the world). Such a model can be used as the basis for planning (calculating which action to take given the current situation and a goal), and may be completely independent of the agent's particular reward function. Thus, knowledge gained about the general workings of the environment when learning to do one task may be directly applied to another task.

When the environment is completely observable, this learning problem is not (conceptually) too difficult. When it is partially observable, learning the world dynamics is equivalent to learning a complex finite-state automaton or hidden Markov model (HMM).

Policy When the agent's reward function is fixed, it is not necessary to learn a model of the world dynamics. Although the world dynamics, coupled with the reward function, entail an optimal behavior for the agent, it may be possible to simply learn the policy directly. In a completely observable environment, a policy is a mapping from states of the world to actions (or a distribution over actions); in a partially observable environment, it is typically some kind of finite state machine that takes in observations and generates actions.

In general, it is hard to learn policies directly. In completely observable environments, it is much easier to learn a value function and derive a policy from it, as described below. In partially observable environments, it remains difficult to learn policies directly, but there are fewer good alternative strategies.

The one general-purpose policy-learning method is gradient-descent. If the general structure of the policy is given as a parametrized model, in which the probabilities of taking actions are differentiable functions of the parameters, then the agent can adjust those parameters based on its experience in the world. These methods seem to work well for simple policy classes, but may have serious problems with local optima in larger policy classes.

An advantage of learning a policy is that, once learned, a policy may be executed very quickly, with no deliberation required to choose an action. A disadvantage is that there is usually no knowledge transfer. Having learned a policy for one task, it is not easy to use it to advantage in learning a policy for another task.

Value function In completely observable environments, it is often advantageous to learn a *value function* rather than a policy. A value function maps states of the environment to a measure of the long-term reward available from that state (assuming the agent acts optimally thereafter). Given a value function, it is easy to compute the optimal policy, which is to take the action that leads to the highest expected value. Basic reinforcement-learning methods, such as Q-learning, can effectively learn the value function from experience.

In many cases, though, we find the basic reinforcement-learning methods to be too slow (especially in terms of the amount of data required). Sutton's Dyna architecture couples world-model learning with reinforcement learning to accelerate learning. It has the desirable side-effect of learning a world model, which can be used later if the task changes.

In partially observable environments, value functions are not defined over observations.

One set of learning methods (for example, McCallum's UTree) attempts to partially reconstruct the state of the environment based on the history of observations and then to apply standard reinforcement learning to this space. We find this technique to be particularly attractive and will discuss it further below.

(Cite standard RL paper; also Stuart Russell paper (IJCAI89?))

5 Representational Strategies

We need to choose a methods for representing the agent's observations, actions, and internal representations of states of the world; and to build on those representations to encode the knowledge that the agent learns about the world over time. The choice of representation may

- leave out some detail; this may be useful if the detail left out is irrelevant to the task;
- give opportunities for generalization (make learning easier in terms of number of examples required);
- make learning computationally easier or harder; or
- make planning computationally easier or harder.

5.1 Propositional vs First-Order

A state of the simple blocks world is a configuration of the blocks, including their colors, materials, and support relationships. Let us assume that individual blocks of the same material and color are completely interchangeable; there is no reason to distinguish particular individuals.[3]

Consider a domain with n blocks, each of which can be one of c colors and m materials, and which can be on any of the other blocks (this is not possible, because any block may have at most one other block on top of it, but this restriction doesn't change the order-of-magnitude calculations we will make here). There are about c^n possible color configurations, m^n possible material configurations, and n^n possible support relations, yielding $(cmn)^n$ possible complete configurations.[4]

[3] Note that this would not be true if, for instance, we could put things inside the boxes; then it might be of crucial importance which green plastic box contains my house key.

[4] Actually, there may be significantly fewer if we factor in the interchangeability properly.

Atomic representation The most naive representational strategy would assign an atom to each possible configuration. We could speak of configuration 5434, for example, leading to configuration 10334 after taking a particular action. This representation is simple in the sense that a single number can represent an entire state. Because it is completely unstructured, however, it doesn't give any leverage for generalization; there is no reasonable notion of two states being similar, based on this representation. In a domain with ten blocks, three colors, and two materials, there would be on the order of 10^{17} possible states; it would require the square of that to store a transition model.

Propositional representation The next level of representational structure would be to encode a state of the world using a set of Boolean-valued variables corresponding to propositions that are true or false of the current configuration of the world. Example propositions would include: **block10-is-green**, **block2-is-plastic**, and **block3-is-in-block5**.[5] A particular state of the world would be a vector of bits, one corresponding to each possible primitive proposition that could be true in the world. In this domain, we would have approximately nc color propositions (for each block and each color, there would be a proposition asserting that the block had that color[6]), nm material propositions, and n^2 support propositions. So, a world state would be described using about $n(n+m+c)$ bits. In our ten-block example, that comes to 150 bits.[7]

Propositional representations afford much opportunity for generalization; the distance between states can be measured in terms of the number of bits that are different in their encodings. Learning algorithms such as neural networks and decision trees take advantage of propositional structure in the encoding of objects. It might be possible to learn, for instance, that any state in which all of the blocks are green has high reward, independent of the values of the stacking propositions. This knowledge could be stored much more compactly and learned from many fewer examples in the propositional rather than the atomic representation.

In many cases, the number of bits that are on (corresponding to the number of propositions that are true) is significantly fewer than the number that are off. Consider a situation in which only ten percent of the bits are on. In that case, it might be more space efficient to simply encode which propositions are true. In our ten-block example, that would mean that 15 propositions are true. If there are 150 propositions, then it will

[5] How does this work when blocks don't have identity? We have to give the blocks arbitrary names, which is very unsatisfactory.

[6] Technically we really only need to have $c-1$ such propositions, since if a block doesn't have one of the first $c-1$ colors, then it must have the remaining color; furthermore, it is impossible that more than one color bit for a particular block would be on, since a block cannot be two colors at the same time; that constraint is not directly encoded in this representation.

[7] You can see the inefficiency in this encoding, because there are $2^{150} = 10^{45}$ possible assignments to the bit vector, but only 10^{17} possible states. Inefficiency isn't always a bad thing, though; it can sometimes make generalization easier.

take about 7 bits to name each proposition, requiring 105 bits for the entire representation. As the number of propositions increases, the savings from using this representation can increase significantly.

Relational representation In the propositional representation, only part of the structure is revealed. That the propositions are derived from properties and relations among underlying objects is lost. In a relational representation, we retain the idea of objects. Thus, rather than having a fixed bit to stand for **block4-is-on-block2**, we would encode that fact relationally as **on(block4,block2)**. A state is typically encoded by listing the true (or known) propositions, but this time describing the propositions using their relational structure.

Such a representation affords even more opportunity for generalization. We might be able to learn and represent much more efficiently that, for example, states in which any block is on top of block2 are rewarding.

We can get further generalization if we allow quantification in our representation. So far, we have had to name the individual blocks and haven't been able to directly take advantage of the idea, described at the beginning of this section, of interchangeability. If we allow individual states to be described using existential quantification over the objects, then we don't have to name them. We can say

$$\exists x, y . \mathbf{red}(x) \wedge \mathbf{blue}(y) \wedge \mathbf{on}(x, y)$$

to describe a situation in which a red block is on a blue block, generalizing over which particular red and blue blocks are on one another.

More discussion here

5.2 Deictic Representations

From the perspective of opportunity to generalize, relational representations seem like the best choice. However, we have much more experience and facility with learning and with probabilistic representation in propositional representations. We may be able to have the best of both worlds by taking advantage of *deictic* or *indexical-functional* representations.

A deictic expression is one that "points" to something; its meaning is relative to the agent that uses or utters it and the context in which it is used or uttered. "Here," "now," and "the book that is in front of me" are examples of deictic expressions in natural language.

Two important classes of deictic representations are derived directly from perception and action relations between the agent and objects in the world. An agent with directed perception can sensibly speak of (or think about) **the-object-I-am-fixated-on**. And agent that can pick things up can name **the-object-I-am-holding**. The objects that are

designated by these expressions can be directly "gotten at" by perception. It should be easy to answer the question of whether **the-object-I-am-fixated-on** is red.

Given a few primitive deictic names, such as those suggested above, we can make compound deictic expressions using directly perceptible relations. So, for example, we might speak of

$$\textbf{the-object-on-top-of(the-object-I-am-fixated-on)}$$

or

$$\textbf{the-color-of(the-object-to-the-left-of(the-object-I-am-fixated-on))} \ .$$

We can evaluate predicates, such as

$$\textbf{bigger-than(the-object-I-am-fixated-on, the-object-to-the-left-of(the-object-I-am-fixated-on))} \ .$$

One crucial property of these expressions is that they are *perceptually effective*; that is, that the agent can always foveate (or mark) them by taking a sequence of perceptual actions that is derived directly from the expression. It is for the agent to test for equality of two of these expressions by: following the first expression, putting a marker on the resulting object, following the second expression, putting a marker on the resulting object, and then testing to see whether the two markers are on the same object.

Another important property of these deictic representations is that they implicitly perform a great deal of generalization. It is true, for instance, that **holding(the-object-I-am-holding)** no matter what particular object I am holding. Using deictic representations, we have a method for naming, and for generalizing over, objects without resorting to *ad hoc* names like **block5**.

Deixis can also be used in the action space. Rather than have an action like **pickup**(*block*) that takes the particular block to be picked up as a parameter, we might instead have a single action, **pickup** that always picks up the block that is currently fixated (or that has the "action" marker on it). In such a situation, we might be able to learn the rule:

$$\textbf{red(the-block-I-am-fixated-on)} \wedge \textbf{pickup} \rightarrow \textit{next} \ \ \textbf{red(the-block-I-am-holding)}$$

Although it has no quantifiers, it expresses a universal property of all blocks.[8]

So far, we have been illustrating deixis in the context of relational representations. It can also be used in propositional representations, to limit the number of objects (and therefore propositions) under consideration, and to do some generalization for us. Thus, if we

[8] This example illustrates a number of other questions we'll have to deal with eventually. We happen to know that, in fact, the block I am holding now is *the same block* as the one that I was fixated on a minute ago. How we can learn that is not clear. It requires a strong inductive leap. Even if we can't learn that, we'd like to learn the second-order fact that all of the properties of the block I'm holding now held of the block I was fixated on a minute ago.

have an active vision system with k markers, we might build propositional representation with bits for each property of each marked object and for each relation between marked objects. With 5 markers, in our 10-block example world, we would have $5*(5+2+3) = 50$ bits in our perceptual vector. This representation captures the interchangeability of blocks of the same color and material. It also makes some other distinctions imperceptible, at least directly. Thus, to learn in a domain with this observation space, we need to use techniques suited to partial observability.

Include some story about symbol grounding?

(Cite Agre and Chapman; Whitehead and Ballard; Ballard et al.; Ullman (Visual Routines); Scott Benson)

5.3 Partial Observability

When the agent is unable to observe the entire true state of the world at each time step, it must remember something about its history of actions and observations in order to behave effectively in the world. There are two ways to think about this aggregation: histories and belief states.

In the history-based model, the agent really does remember all, or as much of its history as it can. It may be selective about what it remembers, but the knowledge is represented in its historical context: two time steps ago I was looking at a red block, or the last block I was holding was green.

An alternative view is to aggregate the perceptual information into a *belief state*, which encodes what the agent currently knows about the state of the world. The fact that two time steps ago I was looking at a red block may actually mean that there is a red block to the left of my hand.

These two styles of representation may be equivalent at an information-theoretic level; howver, the choice between them seems to have important consequences for learning and planning..

5.4 The Role of Probability

An independent representational question is whether, and, if so, how, to represent the uncertainty in the domain. There may be uncertainty (noise) in the agent's perception, in the effects of its actions, or in the dynamics of the environment, independent of the agent's actions.

Markov models capture uncertain world dynamics with atomic state representations. Dynamic Bayesian networks do the same for propositional representations. There is some initial work on probabilistic relational models (cite Koller, Pfeffer), but these models are not yet rich enough to capture probabilistic world dynamics we're interested in. Look, again, at probabilistic logic programs (Muggleton, Cussins, Poole, Bacchus).

These models allow representation of any probabilistic world dynamics; to the extent that there are independence relations in the propositional and relational cases, the models can be represented compactly. However, there is considerable overhead in learning and working with such models.

On the other hand, strictly logical models have serious problems with inconsistency when applied to real environments.

One idea would be to learn and use a model that is a deterministic idealization of the true stochastic world model, and to somehow degrade gracefully when the idealization is violated. Some kinds of default logic (Reiter, etc.) or theories of infinitesimal probability (Pearl, Goldszmidt) might be appropriate.

6 Learning Algorithms

In this section we consider which learning algorithms would be appropriate for learning policies (possibly via a value function) and world models, based on the choice of representation of observations.

6.1 Policy learning

First, we'll consider the completely observable case.

When the states of the world are represented atomically, something like Q-learning or SARSA is appropriate.

Most current research in reinforcement learning addresses the problem of learning in a propositional representation. The goal is to learn a value function, mapping state of the world into long-term values, but to do so in a compact way that affords generalization. Two main approaches are *neurodynamic programming* (NDP) and decision-tree methods (why not nearest neighbor?). In NDP, the value function is stored in a feed-forward neural network. An alternative is to incrementally build a decision tree, with values stored at the leaves. An advantage of this method is that it has a direct extension to the partially observable case; in addition, it makes clear which of the propositional attributes are important. It might be of interest to combine these approaches into regression trees, using a decision tree to divide the space into regions that are fit with linear regression models.

There has been very little work on relational reinforcement learning. The most obvious idea is to adapt methods from inductive logic programming to the task. Cite Djeroski and deRaedt (ICML 98).

The question of most interest to us, for now, is how to do reinforcement learning with a deictic representation, which is by its very nature partially observable. There are two general strategies for learning in partially observable environments. The first is to learn policies directly, typically by performing local search in the space of policies. The second

is to try to reconstruct the underlying world state from histories of actions and observations, then to do standard propositional RL on a representation of the reconstructed state. We feel that this second approach will be more generally applicable, and offers more opportunity for between-task transfer. We will begin by using McCallum's UTREE algorithm, which is an extension of decision-tree learning that also considers properties of historic actions and observations.

Add a section here or elsewhere on POMDPs more generally. Dual views of the information state as observation histories or probability distributions over world states. Difficulty of optimal control. Availability of reasonable heuristics.

6.2 World model

atomic model learning: tables and counters

propositional model learning: drescher, 2TBN, association rule learning methods

relational model learning: ILP, PRMs

deictic: drescher?, association rules, propositional method over utree props

7 First Experiments

Although many people have argued effectively for the utility of deictic representation on intuitive grounds, we know of no empirical demonstrations of its value in learning. Our first experiments are aimed at understanding the utility of deictic representation in reinforcement learning.

We will use a simulated agent in the simple blocks world domain outlined above. The agent will have a single goal, such as to put a green block on top of every red block.

We will compare the following learning algorithms and representations in that domain:

- Complete propositional representation and NDP
- Complete propositional representation and G-learning
- Deictic propositional representation and UTREE

Our conjecture is that the deictic representation will allow the system to learn much more efficiently (in terms of the number of learning examples required), even though it is solving an ostensibly harder (partially observable) problem. We believe the UTREE algorithm is the most straightforward option for learning in this type of partially observable domain. Its direct analogue for completely observable domains is G-learning, a decision-tree method for RL, so that is a natural choice for the complete representation experiment.

However, NDP has become the default choice for propositional RL, so we will compare its performance as well.[9]

8 Future Work

Transfer between tasks by learning a model driven by distinctions made by u-tree on the first task.

Selectively generating deictic expressions (a la Benson).

Build up to reinforcement learning and model learning and planning in relational deictic representation.

[9] G-learning and NDP were compared once before on a very simple domain (cite Chapman and Kaelbling), but we think it will be informative to do it again here.