Modeling Object Recognition as a Markov Decision Process

Bruce A. Draper
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
bdraper@cs.umass.edu

Abstract

The field of computer vision has made significant advances over the past twenty years, yet we still have not developed a theoretical or practical understanding of how the many components of vision are combined into coherent, functioning systems. As a result, there are few applications of computer vision technology in the real world, even though the library of available computer vision techniques keeps growing. This paper models the control of visual procedures as a Markov Decision Problem, and presents a version of the Schema Learning System (SLS) that uses this model to assemble object recognition programs from existing computer vision algorithms. An example of SLS learning to recognize rooftops in aerial images is presented.

1 Introduction

The field of computer vision has made significant advances over the past twenty years. There are now, for example, good and improving algorithms for camera calibration, edge and line extraction (straight and curved), stereo analysis, tracking, depth from motion (two-frame and multi-frame), shape recovery, and 3D pose determination. Unfortunately, we have not yet developed a theoretical or practical understanding of how these components are combined into functioning vision systems. As a result, there are few applications of advanced computer vision technology in the real world, even though the library of available algorithms keeps growing. What we are missing is the ability to select, parameterize and sequence algorithms in order to perform specific tasks.

The problem, at least in part, is that the most effective combinations of algorithms are often object or task dependent. Some objects, for example, have distinct colors that can be used to focus attention on particular parts of an image, while others have easily identifiable substructures, repetitive textures, or other properties that help us to recognize them and place them in space. Unfortunately, the specific features (and the techniques needed to identify them) vary from object to object and context to context, so that many objects require specialized recognition strategies. Similarly, the user's task (or goal) may change the constraints on a recognition problem in terms of the accuracy required or the time allowed, or even the type of information to be extracted from the image. This again calls for specialized recognition strategies, and discourages the wide-spread use of computer vision technology because successful vision systems must be redesigned and/or hand-tuned for each new application.

This paper presents a framework, based both on traditional search techniques and the mathematics of Markov Decision Processes (MDPs), for learning to select, parameterize and sequence visual procedures to achieve specific goals. In this framework, an off-line learning system is trained to produce a control policy for selecting and executing visual procedures. The control policy is closed-loop, meaning that at each step of the process the next visual procedure is selected (according to the policy) based on the results of previous visual procedures. To demonstrate this framework, we have built a new version of the Schema Learning System (SLS), which we demonstrate by learning a control policy for recognizing buildings in aerial images.

2 Previous Work

Recently, a small set of researchers have sought to build systems that learn control strategies for object recognition and/or target recognition tasks. Draper [2] and Roberts & Brown [5], for example, have built systems that learn to recognize objects from examples, while Peng et al [4] proposes a system for learning to parameterize a complex image segmentation algorithm. Although still in the early stages, such technology has the potential to automatically assemble interpretation strategies from libraries of available visual procedures;

unfortunately, the strategies learned by these techniques may still be difficult to analyze and/or validate.

This paper presents a new system that learns recognition strategies by modeling the contol of visual procedures as a Markov Decision Problem (MDP). Although the traditional control-theoretic techniques for solving MDPs (e.g. Dynamic Programming) require a more detailed process model than is generally available for computer vision applications, we believe that recent advances in reinforcement learning and function approximation [6, 7] make it possible to learn near-optimal control policies for image understanding. In principle, we should be able to transform the task of constructing a new vision application to one of training the system with a set of representative input-output examples, and the new version of SLS presented here is meant to demonstrate this capability.

3 Learning to Recognize Buildings

The RADIUS project [3] is an interesting example of both the importance of image understanding technology and the problems with it. Current U.S. military doctrine is to achieve dominant battlefield awareness by digitizing and interpreting ever-increasing numbers of images. Unfortunately, the number of image analysts available to interpret this data is expected to remain the same or even decrease, implying that image analysts will have to become far more productive, presumably by automating or semi-automating portions of their job.

To this end, the RADIUS project has sought to develop image understanding tools to automate analysis tasks, such as (2D) building detection, (3D) building reconstruction, change detection and road detection. As part of this program, several universities have developed new and original algorithms for achieving all or parts of these tasks. Because of the practical nature of the RADIUS project, however, these universities have had to craft not just isolated algorithms, but complete, functioning systems. Although many of the underlying algorithms are generally useful, changes in the problem statement – such as new image domains and/or new types of sensors – have often meant that the overall system had to be retuned, if not overhauled entirely, in order to work on the new task.

This is exactly the type of problem that SLS is meant to address, so we have adopted the RADIUS data and task statements as a test domain for SLS. This paper presents some early results of a major experiment in using SLS to accomplish RADIUS-project tasks such as building detection and reconstruction. The first of these tasks (described below) is to recognize the image positions of rectangular roof surfaces in aerial photographs. Several other universities have previously addressed this problem by developing hand-crafted strategies for finding rooftops; our goal is to automatically learn equivalent (or better) strategies based on the same types of information.

4 Markov Decision Problems

Control engineers have long modeled the control of discrete processes as a Markov Decision Problem (MDP). Although even a brief tutorial on MDPs is beyond the scope of this article, MDPs can be pictured in terms of systems (similar to finite state machines) having a discrete set of states, and making discrete transitions between those states as a result of actions. Initially, an MDP system starts in state s_0; in response to an action (call it a_0), the system is advanced to a new state (call it s_1); the system is also given a reward (or penalty) for making the transition from state s_0 to state s_1. The next action a_1, applied at state s_1, advances the system to state s_2 at time t_2 (and gives it another reward or penalty), and so on until the system reaches a terminal state. The goal is to select a sequence of states and actions $s_0, a_0, s_1, a_1, ..., s_n, a_n$ that maximizes the total reward of reaching a terminal state.

Three features of the MDP formalism are particularly important. One is that MDPs are *stochastic*; each action a_i has a probability function associated with it that gives the probability of transitioning to state s_k from state s_j (written as $P(s_j|a_i, s_k)$). A second is the critical role of the reward function. The goal of the system is to maximize the expected total reward over time, so the reward function determines what the system learns from examples. In this paper, we will use a function that gives a reward of one for a goal-level hypothesis that exactly matches the "true" answer, declining to zero for a goal-level hypothesis that does not correspond to the true answer at all. However, in future studies we hope to use functions that trade off between accuracy and time (i.e. cost). Finally, the third feature is that the solution to a Markov Decision Problem is a *control policy*, often expressed as a table, that maps actions onto states so as to maximize the expected total reward.

Underlying the MDP formalism are the $V^\pi(s)$ and $Q^\pi(s, a)$ functions. Intuitively, $V^\pi(s)$ is the expected reward from starting in state s and following control policy π until a terminal state is reached; the $V^\pi(s)$ function is also sometimes called the state value function. $Q^\pi(s, a)$ is the expected reward from starting in state s, applying action a, and then following control policy π thereafter until a terminal state is reached; the Q function is sometimes called the state/action value function. The basis of Dynammic Programming algorithms is that given a process model, they compute $V^*(s)$ or $Q^*(s, a)$ for every state or state/action pair, where $*$ is the optimal policy. The $Q^*(s,a)$ function can then be used to explicitly generate the optimal policy table by selecting, for every state s, the action a with the highest $Q^*(s,a)$ value.

5 Object Recognition as an MDP

In this paper, we propose to assemble computer vision algorithms into working special-purpose computer vision systems, by using neural networks to compute an optimistic approximation to the $Q^*(s,a)$ functions in a manner similar to reinforcement learning. We then use these Q functions as the heuristic functions in A^* search, creating a system that can be analyzed in control theoretic terms, but which is capable of backtracking if necessary during the search process (something traditional reinforcement learning systems are not capable of). Before considering how A^* plays a role, however, let us first describe how object recognition can be modeled as an MDP.

5.1 Visual Procedures as MDP actions

Not surprisingly, visual procedures are the actions of the MDP. The states of the system correspond to the data items (called *tokens*) produced by visual procedures. For example, in the rooftop recognition scenario, the initial state of the system corresponds to an image. If the action selected is an edge operator, then it will produce a set of edges which becomes the new state of the system. Thus the edge operator action transitions the system from the initial image state to the edge state.

5.2 State Spaces

Clearly, not all sets of edges are the same; neither are all images, polygons, etc. In order to learn sophisticated control policies, it is necessary to distinguish good (high quality) data from bad. Unfortunately, it is not obvious how to divide any given level of representation into a discrete set of states a-priori. However, MDPs can be defined over infinite state spaces. In this formulation, the control policy becomes a function that maps points in the (now infinite) state space onto actions, and the $V^\pi(s)$ and $Q^\pi(s,a)$ functions similarly range over infinite state spaces. In particular, in SLS we define a state space for every level of representation, so that an action maps points in one space onto points in another space, depending on the level of representation of its input and output. To return to the edge operator as an example, it maps points in image space onto points in edge (set) space.

Each level of representation therefore has a state space associated with it. This space is determined by the set of measurable features predefined for that representation. For example, in the current experiment images have four measurable features: average intensity, standard deviation, edginess and speckle. The image state space is therefore defined as a four dimensional space with each feature as one dimension. Any particular image is represented as a point in this space. As another example, one of the representation used heavily in the rooftop experiments is the parallel line pair. The space of parallel line pairs is five dimensional, corresponding to the five features that we have predefined: relative angle, extent of overlap, average edge contrast, average intensity in the direction of a shadow prediction, and the variance of pixel intensities between the lines.

Mathematically, this is a clean formulation of the problem. In practice, it only works if we can learn estimate for the $Q^*(s,a)$ or $V^*(s)$ functions over these infinite state spaces from a finite number of samples. Inspired by Tesauro [6] and Zhang and Dietterich [7], we use back-propagation neural networks to learn approximations to the $Q^*(s,a)$ function for each action, as described below in Section 5.4. The control policies learned by SLS are therefore represented as a set of neural networks, each approximating the Q function for a state/action pair. At run-time, the best action to apply to a state (i.e. data token) is the action with the highest estimated Q value given its feature vector.

5.3 Backtracking and A^* Search

As a first pass, SLS can therefore be visualized as a system that begins with a data token s_0, typically an image. SLS evaluates the function $Q(s_0,a)$ for every action a that can be applied to token (and state) s_0, and selects the action with the highest estimated total reward. The action (a_0) is then applied to data s_0, creating a new data token s_1. SLS then selects the action with the highest estimate for $Q(s_1,a)$ and applies it, creating state s_2. This process repeats itself, creating a sequence of states and actions $s_0, a_0, s_1, a_1, ...$, until a data token at the target level of representation (for example, 2D roof hypothesis) is created. Such a token represents a terminal state for the system.

The problem with this simplified model of object recognition is that many computer vision procedures – particularly matching procedures – do not produce a single data item as output. One of the visual procedures in the library for recognizing rooftops, for example, is the procedure that finds parallel line pairs in a set of line segments. Depending on the number of line pairs found, the matching procedure may produce zero, one or (more likely) many pair tokens. Consequently, when this action is applied to a data state s_i, it may produce many (or no) possible successor states s_j.

SLS therefore maintains a sorted state/action queue. When an action produces multiple results $s_{j1}, ..., s_{jn}$, it evaluates the Q function for each new state/action pair, and creates a state/action queue sorted by the estimated Q values. It then applies the highest rated state/action pair, producing zero, one or more new data states, which are then added to the state/action queue.

The state/action queue does more than just allow SLS to accomodate visual procedures that return an indeterminate number of arguments. It also allows SLS to backtrack during the interpretation process, if necessary. When SLS selects an action, it does so because the estimated reward for that action/state pair is high (meaning that it expects it to lead to a good goal-level hypothesis). Sometimes, however, this estimate is erroneous, and the selected action creates low-quality data. In this case, SLS will not pursue the bad hypothesis. Since new action/state pairs made from the bad hypothesis will have lower Q estimates than some of the unexecuted state/action pairs already on the queue, SLS will backtrack to try one of these previously unexecuted actions. In essence, unlike most MDP applications, SLS is maintaining a complete search tree, and using the Q function as a heuristic to select which nodes to expand.

5.4 Computing Q and V

There are several reinforcement learning algorithms for estimating Q or V without a-priori process models. These algorithms build successively better approximations of Q and V based on experience gained by running the system; Tesauro [6] and Zhang and Dietterich [7] have used these techniques with neural network function approximators to learn Q values for systems with continuous state spaces. One problem with these techniques is that they have to execute tens of thousands of actions to converge to good Q and V estimates – even more when continuous state spaces are used. In an application domain in which each action may take a minute or more to execute, we do not have enough time to directly apply these techniques.

Fortunately, very few sequences of visual procedures are very long; in general, it only takes between five and ten visual prodecures to get from an image to an object hypothesis. As a result, although the search trees associated with object recognition problems have high branching factors, they are typically not very deep. It is possible to do an exhaustive search of a limited number of training images and use this data to estimate Q.

In particular, for every data instance created from a training image during exhaustive search, we can compute the reward of the best goal-level token that can be created from it. These reward values are samples of a function $Q^{opt}(s,a)$, which represents the best reward that can be computed starting from a state/action pair. $Q^{opt}(s,a)$ should not be confused with $Q^*(s,a)$, the estimated reward for the optimal control policy. It may not be possible to select the best action at every state because of ambiguity in the feature vectors, so $Q^{opt}(s,a)$ is an *optimistic estimate* of $Q^*(s,a)$ with the property that $Q^{opt}(s,a) \geq Q^*(s,a)\ \forall s,a$. In fact, as the data features become more discriminating, $Q^{opt}(s,a)$ approaches $Q^*(s,a)$ from above. Most importantly, $Q^{opt}(s,a)$ can be computed from far fewer training samples than $Q^*(s,a)$, because each $Q^{opt}(s,a)$ is independent of every other, allowing the neural nets to be trained seperately.

6 Experiment – Detecting Rooftops

6.1 Data and Training Signal

To test the principles above, we assigned SLS the task of finding rectangular roof surfaces in aerial images. On each trial, the system is given an image containing one or sometimes two buildings, the angle of the sun, and a set of 3D line segments computed as described in [1]. The height of the ground plane is also known, and is used to filter the initial set of line segments. This corresponds roughly to the data available to the hand-crafted system described in [1]. The training signal is provided by a reward function that assigns a score between zero and one to rectangles according to how well they match line segments projected from a hand-built 3D model of the scene.

6.2 The Visual Procedure Library

The visual procedure library (as depicted in http://vis-www.cs.umass.edu/projects/SLS3D.html) contains the visual procedures and representations available to SLS for this experiment. The levels of representation correspond to images, sets of 3D line segments, parallel line pairs, corners (i.e. vertices of orthogonal lines), line groups and polygons. Each representation has a predefined set of features, as described in Section 5.2.

The visual procedures in the library are meant to approximate some of the techniques being used by researchers in the RADIUS project. The procedure for computing 3D line segments is described in [1]. The pairwise grouping procedure locates parallel, colinear and orthogonal line pairs, using information about the camera pose (available for all RADIUS images) to remove the effects of perspective distortion. Another, recursive grouping procedure finds sets of interrelated lines, while a Graph Matching procedure searches for rectangles of related line pairs. In addition, given a pair

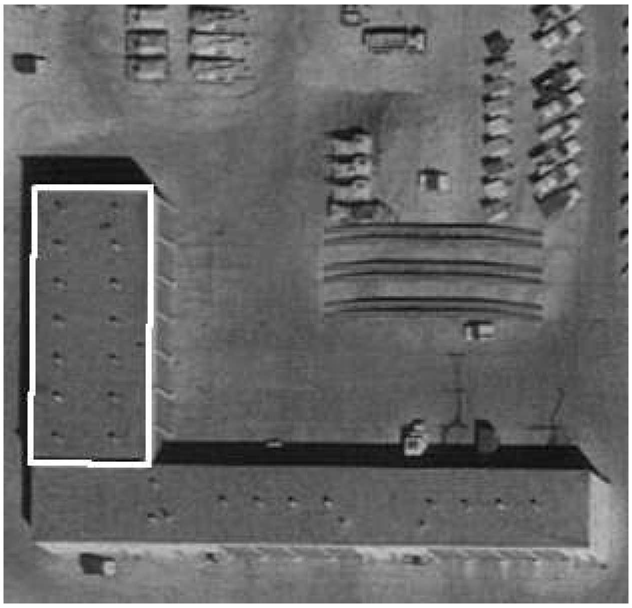

Figure 1. One of the ten aerial images of buildings at Ft. Hood, and the roof hypothesis (shown in white) SLS learned to find in it.

of parallel or orthogonal line segments, the system has a visual procedure for searching top-down for evidence of additional lines to complete a rectangle. As a measure of the complexity of the visual procedure library, there are about three hundred polygons that can be extracted from most images using some combination of the procedures and representations in the library, of which five to ten will correspond reasonably well to roof surfaces.

6.3 Detecting Roofs

SLS was tested on a set of ten images of Ft. Hood, Texas, using a "leave one out" methodology. On each trial, SLS learned a control policy from nine images, and the policy was applied to the tenth. This was repeated ten times, each time holding a different image out of the training set for testing. Figure 1 shows one of the rooftops extracted by SLS; the other nine can be found at http://vis-www.cs.umass.edu/projects/SLS3D.html. Over ten trials, SLS found a perceptually good approximation to a rooftop every time.

The contol policies learned by SLS did not always take a straight path to the solution. Although they generally prefered finding corners to parallel lines (and always prefered looking to complete the reactangle top-down, rather than using the graph matcher), they would often select one corner as being the most promising, use it to generate a polygon hypothesis, and then decide that the polygon was not as good as it thought it would be. The system would then backtrack, find the next most promising corner, and try again. In general, the system created ten to twenty polygon hypotheses (out of several hundred possible ones) before finding the polygon it finally selected to be the rooftop.

7 Conclusions

Over the last twenty years, computer vision researchers have developed algorithms for solving many visual subproblems, such as edge extraction, stereo analysis, and pose determination. We seek to understand the science and practice of combining these algorithms into special-purpose vision system, by casting object recognition as a Markov Decision Problem. This requires defining state spaces for object recognition and computing approximations to the Q and V functions; it also requires building a system capable of integrating many different algorithms. The Schema Learning System (SLS) is a first pass at such an algorithm and system.

References

[1] R. Collins, et al. "UMass Progress in 3D Building Model Acquisition," *Image Understanding Workshop,* 1996.

[2] B. Draper. "Learning Control Strategies for Object Recognition," in *Symbolic Visual Learning,* Ikeuchi and Veloso (eds.), Oxford University Press, 1996.

[3] D.Gerson and S.Wood, "RADIUS Phase II – The RADIUS Testbed System," *Image Understanding Workshop,* Monterey, CA, November 1994.

[4] J. Peng, B. Bhanu and R. Dutta. *Delayed Reinforcement Learning for Closed-Loop Object Recognition,* University of California (Riverside), 1995.

[5] B. Roberts and C. Brown. "Adaptive Configuration and Control in an ATR System," *Image Understanding Workshop,* Monterey, CA, 1994.

[6] G. Tesauro. "Temporal Difference Learning and TD-Gammon" *Communications of the ACM,* 38(3):58-68

[7] W. Zhang and T. Dietterich. "A Reinforcement Learning Approach to Job-Shop Scheduling," *Int. Joint Conference on Artificial Intelligence,* 1995.