Use of Off-line Dynamic Programming for Efficient Image Interpretation

Ramana Isukapalli
101 Crawfords Corner Road
Lucent Technologies
Holmdel, NJ 07733 USA
risukapalli@lucent.com

Russell Greiner
Department of Computing Science
University of Alberta
Edmonton, AB T6G 2E8 Canada
greiner@cs.ualberta.ca

Abstract

An *interpretation system* finds the likely mappings from portions of an image to real-world objects. An *interpretation policy* specifies when to apply which imaging operator, to which portion of the image, during every stage of interpretation. Earlier results compared a number of policies, and demonstrated that policies that select operators which maximize the information gain per cost, worked most effectively. However, those policies are *myopic* — they rank the operators based only on their *immediate* rewards. This can lead to inferior overall results: it may be better to use a relatively expensive operator first, if that operator provides information that will significantly reduce the cost of the subsequent operators.

This suggests using some lookahead process to compute the quality for operators *non-myopically*. Unfortunately, this is prohibitively expensive for most domains, especially for domains that have a large number of complex states. We therefore use ideas from reinforcement learning to compute the *utility* of each operator sequence. In particular, our system first uses dynamic programming, over *abstract* simplifications of interpretation states, to precompute the utility of each relevant sequence. It does this off-line, over a training sample of images. At run time, our interpretation system uses these estimates to decide when to use which imaging operator. Our empirical results, in the challenging real-world domain of face recognition, demonstrate that this approach works more effectively than myopic approaches.

1 Introduction

Interpretation is the process of finding the likely mapping from portions of an image to real-world objects. It is the basis for a number of imaging tasks, including recognition ("is objectX in the image?") and identification ("which object is in the image?"), as well as several forms of tracking ("find all moving objects of typeX in this sequence of images"), etc. [PL95; HR96]. It is important that an interpretation system (*IS*) be *efficient* as well as *accurate*. Any *IS* should have access to an inventory of "imaging operators" — like edge detectors, region growers, corner locators, etc., each of which, when applied to any portion of the image, returns meaningful tokens (like circles, regions of the same color, etc.). An "interpretation policy" specifies which operator to apply to which portion of the image, during each step of the interpretation process. Such policies must, of course, specify the details: perhaps by specifying exactly which bottom-up operators to use, and over what portion of the image, if and when to switch from bottom-up to top-down, which aspects of the model to seek, etc.

Our earlier work [IG01b] considered various types of policies, towards demonstrating that an "information theoretic policy", which selects operators that maximize the information gain per unit cost of the imaging operator, work more effectively than others. However, the [IG01b] policies evaluate each operator $o(\cdot)$ *myopically* — *i.e.*, independent of the cost and effectiveness of subsequent operators that would be applied, after performing this $o(\cdot)$. To see why this is problematic, assume the task is to determine whether there is an airplane in an image, by seeking the various parts of an airplane — *e.g.*, the fuselage, wings, engine pods, tailpiece, etc. Now consider the o_{ep} operator that detects and locates the engine pods. As the engine pods are small and often partially-occluded, o_{ep} is probably expensive. However, once these parts have been located, we expect to find the associated wings very easily, and then the remaining parts required to identify the entire airplane. A myopic policy, which evaluates an operator based only on its immediate cost, would miss this connection, and so would probably prefer a cheaper operator over the expensive o_{ep}. A better policy would consider "operator interactions" (here, relating o_{ep} to (say) the "wing finder operator") when deciding which operator to apply. The data in Section 5 (related to the complex task of face recognition) shows that such non-myopic policies can be both more accurate, and more efficient.

Of course, non-myopic policies must use some type of lookahead to evaluate the quality of its operators; this can be combinatorially expensive to compute. We address this concern by (1) dealing with an abstract version of the "interpretation state space", and then by (2) using dynamic programming techniques over this abstracted space, pre-computing many relevant "utility" [SB98] values *off-line* [BDB00]. In particular, our system computes the utility of imaging operator sequences in each state s encountered, based on data from a set of training examples. It produces a *policy*, which maps each state s to the operator o_s^* that appears to be the most promising, incorporating this lookahead. At run-time, our system finds the best matching state and applies the operator associated with that situation. Our empirical results, in the domain of face recognition, show that such policies work effectively — better than the best results obtained by any of the earlier myopic systems.

Section 2 presents relevant related work, to help frame our contributions. Section 3 discusses the salient features of interpretation strategies. Section 4 then presents our approach, of using reinforcement learning in image interpretation, within the domain of face recognition, using operators that each correspond to types of eigenfeatures. Section 5 provides empirical results that support our claims.

2 Related work

We produce an interpretation of an image by applying a sequence of operators, where each operator maps the current state (typically a partial interpretation) to a new state. As the effects of an operator may depend on information not explicitly contained in the state description, this mapping is stochastic. Moreover, each operator has a cost, and the state associated with the final interpretation has a "quality" (*i.e.*, its accuracy as an interpretation). As such, we view this image-interpretation task as a "Markov Decision Problem" (MDP). This means we can apply the host of reinforcement learning techniques [SB98] to this problem. This specific application area, of image interpretation, follows the pioneering work of Draper [BDB00], which shows that dynamic control policies can outperform hand-coded policies. We extend their work by addressing and exploiting the issue of operator interactions and by doing a systematic analysis of the cost and accuracy tradeoffs in face recognition.

While we could encode each state as the entire image, this would be too unwieldy. Boutilier et al. [BDH99] survey several types of representations in planning problems and discuss ways to exploit them to ease the computational cost of policies or plans. Their work focuses on abstraction, aggregation and decomposition techniques. We use abstractions to reduce the size of each state, and hence of the state space we are exploring; moreover, we are not concerned with the general planning problem. Our system is similar to the "forest inventory management system" [BDL02]. In their system, the output of some operators provide the input to some other operator, while in our system the result of one operator is used to narrow the search for some other operator. Moreover, our system can exploit the structure specific to our task to bound the lookahead depth.

3 Framework

As suggested above, our overall objective is to produce an *effective* interpretation policy — *e.g.*, one that *efficiently* returns a *sufficiently accurate* interpretation, where accuracy and efficiency are each measured with respect to the underlying task and the distribution of images that will be encountered. This section makes this framework more precise. Subsection 3.1 specifies our particular task; Subsection 3.2 lists the strategies we will evaluate; Subsection 3.3 outlines our performance domain, face recognition; and Subsection 3.4 describes the specific operators we will use.

3.1 Input to the Interpretation System

We assume that our interpretation system "*IS*" is given the following information:

⋆ The **distribution** D of images that the IS will encounter, encoded in terms of the distribution of objects and views that will be seen, etc. For our face recognition task, this corresponds to the distribution of all people the system will see, which varies over race, gender and age, as well as poses and sizes. We approximate this using the images given in the training set. See Figure 1.

⋆ The **task** $T = \langle \mathcal{V};\, C_{max},\, P_{min} \rangle$ includes two parts: First, $\mathcal{V}$ specifies the objects that the IS should seek, and what IS should return. Second, the task specification also provides the "evaluation criteria" for any policy, which is based on both the expected "accuracy" (P_{min}) and the maximum interpretation "cost" (C_{max}), which the IS should not exceed.

Here, $\mathcal{V}$ specifies our task is to identify the person from his/her given test image (wrt the people included in the training set), subject to the accuracy P_{min} and cost requirements C_{max}.

⋆ The **set of possible "operators"** $O = \{o_i\}$ includes (say) various edge detectors, region growers, graph matchers, etc. For each operator o_i, we must specify

- its input and output,
- its "effectiveness", which specifies the accuracy of the output, as a function of the input.
- its "cost", as a function of (the size of) its input and parameter setting.

We will use a specific set of operators in our face recognition task; we describe these in detail in Section 3.4.

Borrowing from the MDP literature, we view the main input to each operator as its "state". As we are dealing with scenes, we could use the entire pixel image as (part of) the state. However, for reasons of efficiency, we will often use an *abstracted* view $\mathcal{F}(\,s\,)$ of (our interpretation of the) the scene s. Section 3.4 presents the specific abstraction we are using.

3.2 Strategies

Strategy INFOGAIN**:** selects the operator that provides the largest information gain (per unit cost) at each step. This myopic strategy first computes the expected information gain $EIG(o)$ for each possible operator and argument combination o, as well as the cost $c(o)$. It then executes the operator that maximizes their ratio, $o^* = \text{argmax}_o\{EIG(o)/c(o)\}$.

We focus on this strategy as a number of earlier empirical studies have demonstrated that it was the best of all of the (myopic) strategies considered, across a number of task-contexts and domains, including simple blocks world [IG01a], car recognition (identifying the make and model of a car based on its tail light assembly [IG01a]), as well as the face recognition system considered here [IG01b].

Strategy BESTSEQ**:** selects an operator o^*_{BS} that appears to be the most promising in the current abstract state $\mathcal{F}(\,s\,)$, as given by the utility $o^*_{BS} = \pi^*(\mathcal{F}(\,s\,))$, where π^* is the (optimal) mapping from state to actions. See Section 4 for details.

3.3 Face recognition task

We investigate the efficiency and accuracy of the strategies listed above in the domain of face recognition [TP91; PMS94; PWHR98; EC97]. This section briefly discusses the prominent "eigenface" technique of face recognition that forms the basis of our approach; then presents our framework, describing the representation and the operators we use to identify faces; and finally presents our face interpretation algorithm, for identifying a person from a given image of his/her face.

Eigenface and EigenFeature Method: Many of today's face recognition systems use *Principal Component Analysis* (PCA) [TP91]: Given a set of training images of faces, the system first forms the covariance matrix C of the images, then computes the k main eigenvectors of C ("eigenfaces"). Every training face h_i is then projected into this coordinate space ("facespace"), producing a vector $\Omega_i = [\omega_1^{(i)}, \omega_2^{(i)}, \ldots, \omega_k^{(i)}]$.

During recognition, the test face h_{test} is similarly projected into the facespace, producing the vector Ω_{test}, which is then compared with each of the training faces. The best matching training face is taken to be the interpretation [TP91].

Following [PMS94], we extend this method to recognize facial features — eyes, nose, mouth, etc. — which we then use to help identify the individual in a given test image. We partition the training data into two sets, $T = \{h_{1,i}\}$ for constructing the eigenfeatures and $S = \{h_{2,i}\}$ for collecting statistics, where each set contains at least one face of each of the n people. (Feature regions for the training data T were extracted using the operators explained in 3.4 and verified manually for correctness). Letting $\mathrm{id}(h)$ denote the person whose face is given by h, we have $\mathrm{id}(h_{1,i}) = i = \mathrm{id}(h_{2,i})$ for $i = 1..n$; each remaining $h_{1,j}$ and $h_{2,j}$ ($j > n$) also maps to $1..n$; *i.e.*, $\mathrm{id}(h_{1,j}), \mathrm{id}(h_{2,k}) \in \{1, \ldots, n\}$

We use PCA on the mouth regions of each T image, to produce a set of eigenvectors; here *eigen-mouths*. For each face image h_i, let $\Omega_i^{(m)}$ be the "feature space" encoding of h_i's mouth-region. We will later compare the feature space encoding $\Omega_{test}^{(m)}$ of a new image h_{test} against these $\{\Omega_i^{(m)}\}$ vectors, with the assumption that $\Omega_{test}^{(m)} \approx \Omega_i^{(m)}$ suggests that h_{test} is really person i — *i.e.*, finding that $\|\Omega_{test}^{(m)} - \Omega_i^{(m)}\|$ is small should suggest that $\mathrm{id}(h_{test}) = i$. (Note $\|\cdot\|$ refers to the L_2 norm, aka Euclidean distance.) To quantify how strong this belief should be, we compute $M = |T| \times |S|$ values $\{d_{i,j}\}$ where each $d_{i,j} = \|\Omega_{1,i}^{(m)} - \Omega_{2,j}^{(m)}\|$ is the Euclidean distance between the "eigen-mouth encodings" of T's $h_{1,i}$ and S's $h_{2,j}$. Using the T training data, we can learn a mapping from these values to probabilities $P(\, \|\Omega_{test}^{(m)} - \Omega_i^{(m)}\| \,|\, \mathrm{id}(h_{test}) = i\,)$, which we can use to estimate $P(\, \Omega_{test}^{(m)} \,|\, \mathrm{id}(h_{test}) = i\,)$. (See [IG01b] for details.)

We compute similar estimates for the other facial features, such as nose (n), left eye (le) and right eye (re). We then use the Naïve-Bayes assumption [DH73] (that features are independent, given a specific person) [1] to compute a cumulative "face probability" from these "feature probabilities": the values $\|\Omega_{test}^{(f)} - \Omega_i^{(f)}\|$ for each feature f, corresponding to a set of $P(\, \Omega_{test}^{(m)} \,|\, \mathrm{id}(h_{test}) = i\,)$ values (one for each individual i). We then compute

$$
\begin{aligned}
& P(\, \mathrm{id}(h_{test}) = i \,|\, \Omega_{test}^{(m)},\, \Omega_{test}^{(le)},\, \Omega_{test}^{(n)}\,) \\
&= \alpha \cdot P(\, \mathrm{id}(h_{test}) = i\,) \cdot \\
&\quad P(\, \Omega_{test}^{(m)}, \Omega_{test}^{(le)}, \Omega_{test}^{(n)} \,|\, \mathrm{id}(h_{test}) = i\,) \qquad (1) \\
&= \alpha \cdot P(\, \mathrm{id}(h_{test}) = i\,) \cdot P(\, \Omega_{test}^{(m)} \,|\, \mathrm{id}(h_{test}) = i\,) \cdot \\
&\quad P(\, \Omega_{test}^{(le)} \,|\, \mathrm{id}(h_{test}) = i\,) \cdot P(\, \Omega_{test}^{(n)} \,|\, \mathrm{id}(h_{test}) = i\,)
\end{aligned}
$$

where α is a scaling constant.

3.4 Operators

We use four classes of **operators**, $O = \{\, o_{le}(k,\ell,r),\;\; o_{re}(k,\ell,r),\;\; o_n(k,\ell,r),\;\; o_m(k,\ell,r)\,\}$ to detect respectively "left eye", "right eye", "nose" and "mouth". Each specific operator also takes several parameters: $k \in \{25, 30, 35, 40, 45\}$ specifies the number of eigen-vectors being considered; the other parameters $\ell = \langle \ell_x, \ell_y \rangle$ and $r = \langle r_x, r_y \rangle$ specify the space this operator will sweep, looking for this feature: It will look in the rectangle $\langle \ell_x \pm r_x,\; \ell_y \pm r_y \rangle$; *i.e.*, sweep $r_x \times r_y$ pixels, centered at $\langle \ell_x,\, \ell_y \rangle$.

Each instantiated operator $o(\cdot) \in O$ takes as input the image of a test face h_{test}, and returns a probabilistic distribution over the individuals. It has three subtasks: SubTask#1 locates the feature $h_{test}^{(f)}$ from within the entire face h_{test}. Here we use a simple template matching technique in which we search in the fixed region r of size $r_x \times r_y$ pixels, centered at pixel ℓ. SubTask#2 then projects the relevant region of the test image $h_{test}^{(f)}$ into the feature space — computing $\Omega_{test}^{(f)}$ of dimension k. SubTask#3 uses this $\Omega_{test}^{(f)}$ to compute first the values $d_{i,test} = \|\Omega_{test}^{(f)} - \Omega_i^{(f)}\|$ for each person i and then to compute the probability $P(\, \mathrm{id}(h_{test}) = i \,|\, \Omega_{test}^{(f)}\,)$ for each person i. We use Equation 1 to update the distribution when considering the 2nd and subsequent features; see [IG01b]. For each eigenspace dimension k, we empirically determined the cost (in milliseconds) of the four operators —

$$
\begin{aligned}
C(o_{le}(k,r,\ell)) &= \;\;9.25\,k + 0.25\,|r| - 446.00 \\
C(o_{re}(k,r,\ell)) &= 10.35\,k + 0.27\,|r| - 454.27 \\
C(o_n(k,r,\ell)) &= \;\;9.95\,k + 0.23\,|r| - 486.51 \\
C(o_m(k,r,\ell)) &= \;\;14.4\,k + 0.26\,|r| - 308.44
\end{aligned}
$$

where $|r| = r_x \times r_y$ is the size of the range. While increasing the dimensionality k of the feature space should improve the accuracy of the result, here we see explicitly how this will increase the cost.

4 Use of Dynamic Programming

This section briefly overviews MDPs, presents "state abstraction" in face recognition and shows how dynamic programming can be used to compute the utility of operators in abstracted states. We also discuss operator interaction and describe how it can be exploited in face recognition.

4.1 Markov Decision Problem

A *Markov Decision Problem* can be described as a 4-tuple $\langle \mathcal{S}, A, M, R \rangle$ where $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ is a finite set

[1] Of course, this assumption is almost assuredly false in our situation. However, this classification has been found to work well in practice [Mit97].

of states, $A = \{a_1, a_2, \ldots, a_m\}$ is a finite set of actions, $M: \mathcal{S} \times A \times \mathcal{S} \to [0,1]$ is the state transition probability function ($M^a_{s,s'} = P(s' \mid s, a)$ is the probability that taking action a in situation s leads to being in state s') and $R: \mathcal{S} \times A \to \Re$ is the reward an agent gets for taking an action $a \in A$ in state $s \in \mathcal{S}$. The *Markov property* holds if the transition from state s_i to s_j using action a depends only on s_i and not the previous history. A *policy* $\pi : \mathcal{S} \to A$ is a mapping from states to actions. For any policy π, we can define a utility function U_π such that

$$U_\pi(s) \quad = \quad \max_a \left\{ R(s,a) \; + \; \sum_{s'} M^a_{s,s'} \times U_\pi(s') \right\}$$

which corresponds to the expected cumulative rewards of executing the apparently-optimal action a in state s, then following policy π after that.

Given an MDP, we naturally seek an optimal policy π^* — *i.e.*, a policy that produces the *optimal* cumulative reward for each state, $U^*(s)$. Dynamic programming provides a way to compute this optimal policy, by computing the utilities of the best actions. Of course, given the values of $U^*(\cdot)$, the optimal action at each state s is simply $\pi^*(s) = \mathrm{argmax}_a\{R(s,a) + \sum_{s'} M^a_{s,s'} \times U^*(s')\}$.

This can be challenging in the general setting, where sequences of actions can map one state to itself; much of the work in Reinforcement Learning [SB98] is designed to address these issues. In our current case, however, we will see there is a partial order on the states, meaning no sequence of actions can map a state to itself. Here, we can use dynamic programming to compute the optimal utility for each "final step", then use these values to compute the optimal action (and utility) for each penultimate state, and so forth.

4.2 State abstraction

An MDP involves "states". In our face recognition task, any attempt to define states in terms of the pixel values of an image would be problematic, as there will be far too many states to enumerate. Following [BDB00; BDL02], we use the notion of "abstract states", which basically redefines an original state in a much more compact form, using only the certain aspects of the state. Since we recognize a person using the features, we define an abstraction function

$$\mathcal{F}(\,s\,) \;=\; \langle C, \vec{P},\; x_{le}, y_{le},\; x_{re}, y_{re},\; x_n, y_n,\; x_m, y_m \rangle \quad (2)$$

where s is the actual complex state as present in the image and $\langle x_i, y_i \rangle$ denotes the location (center) of feature f_i (left eye, right eye, nose or mouth) in the image. (Here C is the total cost we have spent so far, and $\vec{P} = \langle P(\,\mathrm{id}() = x \mid \vec{E}\,)\rangle_x$ is the current posterior distribution over the possible faces, based on the current evidence $\vec{E}$.) The location of some feature f_i may not yet be known in an image; here, we use the value of $\perp$ for both x_i and y_i. Further, we say two abstractions are "δ-equivalent", written $\mathcal{F}(\,s_1\,) \approx_\delta \mathcal{F}(\,s_2\,)$, iff

- s_1 and s_2 have located the same set of features — *i.e.*, $x_i(s_1) = \perp$ iff $x_i(s_2) = \perp$ and
- $\displaystyle\sum_{i: x_i(s_1) \neq \perp} \|x_i(s_1) - x_i(s_2)\| + \|y_i(s_1) - y_i(s_2)\| \;\le\; \delta$ — *i.e.*, when the distance (in pixels) between the centers of feature f_i in s_1 and s_2 is small ($\le \delta$).

As we are using normalized images that contain only faces, denoting the distance in absolute pixel values is not an issue. In general, δ is a small, predefined constant; we used 10.[2]

The result of abstraction is that a large number of complex states can be described by a small number of compact state descriptions. Of course, the same abstract state can represent multiple states — $\mathcal{F}(\,s_1\,) = \mathcal{F}(\,s_2\,)$ when $s_1 \neq s_2$.

We use a lookahead algorithm to compute the U-values. Since we consider only four features, we need a lookahead of only depth four. Moreover, we can do this during the training phase. We will use the reward function

$$R(s,a) = \begin{cases} \vec{P}(\,\mathrm{id}() = x^* \mid \vec{E}\,) & \text{if } a = \text{Stop} \\ -\alpha \cdot t(a) & \text{otherwise} \end{cases} \qquad (3)$$

that penalizes each operator a by α times the time $t(a)$ it required (in seconds) as well as a positive score for obtaining the correct interpretation (here x^* represents that correct identity of the person in the image.) We used $\alpha = 0.2$. During the training phase, our system finds the optimal operator sequence — *i.e.*, the one that has the maximum $U'(s) = U(\mathcal{F}(\,s\,))$ value, based on the abstracted state $\mathcal{F}(\,s\,)$.

4.3 Dynamic Programming

This section shows how to compute $U(s)$ utility values.

Tree expansion: We use four classes of operators $O = \{\, o_{le}(k,\ell,r),\;\; o_{re}(k,\ell,r),\;\; o_n(k,\ell,r),\;\; o_m(k,\ell,r)\,\}$, each with 5 different values of $k \in \{25, 30, 35, 40, 45\}$, for a total of 20 operators (see Section 3.4). The parameters $\ell = \langle \ell_x, \ell_y \rangle$ and $r = \langle r_x, r_y \rangle$ specify the rectangular area where the operator will search. Our system does not have to search over their values; instead, it directly computes their values from the locations of the features $\langle f_1, f_2, \ldots, f_i \rangle$ ($i < 4$) we have already detected; see Section 4.4.

Our system expands the operator tree exhaustively using a depth-first search. Of course, the depth of the tree is at most 4 (not 20), as we only consider at most one instance of each operator along any path.

Computing $U(\,s\,)$: We compute the various utility values using a dynamic programming approach:

(i) We first apply every sequence of 4 different operators to the original image, to produce the set of all possible $s^{(4)}$ leaf states. Then set $U(s^{(4)}) \;=\; P(\,\mathrm{id}() = x^* \mid \vec{E}\,)$, where x^* is the correct interpretation, and $\vec{E}$ is the observed locations of the various features.

(ii) Now consider each state $s^{(3)}$ that involves some set of 3 operators. For each, we can consider all $5 + 1$ possible actions: either `Stop`, or apply the remaining operator (with one of the 5 possible k values). We can trivially compute the utility of each option: as $P(\,x^* \mid \vec{E}'\,)$ for the `Stop` action, and $-\alpha\, t(a) \;+\; U(s^{(4)})$ if the action a takes time $t(a)$ and produces the state $s^{(4)}$. We set $U(\,s^{(3)}\,)$ to be the largest of these 6 values, and $\pi_{BS}(\,s^{(3)}\,)$ to the action which produced this largest values.

[2] For this value of δ, we found that there were about 1600 entries of $\mathcal{F}(\,s\,)$ totally.

Figure 1: Training images (top); test images (bottom)

After computing $U(s^{(3)})$ for all "depth-3" states, we then recur, to deal with depth-2 states, and so forth.

Of course, these states here actually refer only to the abstracted state $\mathcal{F}(s)$. Moreover, we "bin" these values: Suppose we have encountered some abstracted state $\mathcal{F}(s_1)$. If we later find a state s_2, where $\mathcal{F}(s_1) \approx_\delta \mathcal{F}(s_2)$, and determine $U(\mathcal{F}(s_1)) < U(\mathcal{F}(s_2))$, we would then reset the value of $U(\mathcal{F}(s_1))$ and $\pi_{BS}(\mathcal{F}(s_1))$ to be the values found for $\mathcal{F}(s_2)$.

Retrieving the most promising operator during interpretation: This entire procedure of computing the optimal policy π_{BS} is done off-line, during the training phase. During interpretation (performance phase), in any state s_i we (i) find the "nearest neighbour" $\mathcal{F}(s_j)$, such that the sum of the distances between the corresponding features in $\mathcal{F}(s_i)$ and $\mathcal{F}(s_j)$ is the minimum over all possible entries; and (ii) return the operator $\pi_{BS}(\mathcal{F}(s_j))$, which is either "`Stop`", or an instantiated operator — say $o_n(25,\ \langle 50, 66\rangle,\ \langle 60, 52\rangle)$. If not-`Stop`, the run-time system then executes this operator to locate another feature, which produces a new state (updating the total cost and posterior distribution as well). It then determines the next action to perform, and so forth.

There is one place where our system might perform differently from what the policy dictates: As we are maintaining the *actual* cost so far, and the *actual* posterior distribution over interpretations, our run-time system will actually terminate if either the actual cost has exceeded our specs, or if the highest probability is above the minimal acceptable value.

Notice, given our specific set-up — *e.g.*, only 4 types of operators that can only be executed once, and which always succeed, etc. — the policy obtained can be viewed as a "straight-line" policy: seek one specific feature, than another, until achieving some termination condition; see Figure 2 for an example. In general, this basic dynamic programming approach could produce more complex policies.

4.4 Operator dependencies

Our face recognition system will exploit a certain type of operator interactions, *viz.*, using the result of one operator to simplify a subsequent one. That is, after detecting some feature (say the left eye), we expect to have some idea where to find the other features (*e.g.*, the right eye). We model this as a linear function, mapping from the location of the left eye to the expected location of the right eye. If the left eye was

$\pi_{BS}(\ \langle C,\ \vec{P},\ \langle x_{le}, y_{le}, x_{re}, y_{re}, x_n, y_n, x_m, y_m\rangle\rangle\ )$
If $C \geq C_{max} \quad \vee \quad P_{min} \leq \max_x P(\,\mathrm{id}() = x \mid \vec{E}\,)$
Then Return(`Stop`)
Else Case $\langle x_{le}, y_{le}, x_{re}, y_{re}, x_n, y_n, x_m, y_m\rangle$ of

⋆ $\langle\ \perp,\ \perp,\ \perp,\ \perp,\ \perp,\ \perp,\ \perp,\ \perp\rangle$:
Return($o_{le}(25;\ \langle 31, 52\rangle \pm \langle 21, 22\rangle)$)

⋆ $\langle x_{le}, y_{le},\ \perp,\ \perp,\ \perp,\ \perp,\ \perp,\ \perp\rangle$:
Return($o_{re}(25;\ \langle x_{le} + 46, y_{le} + 8\rangle \pm \langle 25, 17\rangle)$)

⋆ $\langle x_{le}, y_{le}, x_{re}, y_{re},\ \perp,\ \perp,\ \perp,\ \perp\rangle$:
if $\langle x_{le}, y_{le}, x_{re}, y_{re}\rangle \approx_\delta \langle 16, 43, 50, 38\rangle$
then Return($o_n(30;\ \langle x_{re} - 6, y_{re} + 22\rangle \pm \langle 24, 22\rangle)$)
else Return($o_n(25;\ \langle x_{re} - 6, y_{re} + 22\rangle \pm \langle 24, 22\rangle)$)

⋆ $\langle x_{le}, y_{le}, x_{re}, y_{re}, x_n, y_n,\ \perp,\ \perp\rangle$:
if $\langle x_{le}, y_{le}, x_{re}, y_{re}, x_n, y_n\rangle \approx_\delta \langle 19, 41, 52, 41, 33, 51\rangle$
then Return($o_m(30;\ \langle x_n + 14, y_n + 30\rangle \pm \langle 31, 19\rangle)$)
else Return($o_m(25;\ \langle x_n + 14, y_n + 30\rangle \pm \langle 31, 18\rangle)$)

⋆ Return(`Stop`)

Figure 2: Policy π_{BS} learned by BESTSEQ, for $\langle C_{max} = \infty, P_{min} = 0.9\rangle$

detected at $\langle x_{le}, y_{le}\rangle$, then the expected location of the right eye would be the window $\langle x_{le} + 46, y_{le} + 8\rangle \pm \langle 25, 17\rangle$. We also model the "variance" — *i.e.*, the size of the search window around this expected position. To be more precise, let us assume that the search window for some feature f_1 is of size $r_{x1} \times r_{y1}$ pixels. After detecting feature f_2, we instead search for f_1 in a smaller region $r'_{x1} \times r'_{y1}$, centered at the location that we computed from f_2's location, using that linear function.

After observing several features (say nose and left eye), we will have several estimates for the location of the current feature (here right eye). Here we take the smallest bounding box and use that as the search area for locating the current feature. This "factoring" means we need only consider a set of 4×3 transformations (of each of the four feature versus the other three features), rather than deal with all possible subsets.

To make these ideas more concrete: Initially, given no other information, we would look for the right eye in the region $\langle 67, 52\rangle \pm \langle 25, 22\rangle$. However, after finding the left eye at location $\ell_{le} = \langle 25, 50\rangle$ our system knows that, if asked to look for the right eye, it should search in the region $\langle 71, 58\rangle \pm \langle 25, 17\rangle$. (We are not committing to looking for the right eye at this time; just indicating where to search, if requested.) Notice this region is different from the one we would consider if we had not located the left eye; moreover, if we also knew that the nose was at location $\ell_n = \langle 30, 50\rangle$ (as well as left eye), we would use a yet more refined region — here $\langle 61, 55\rangle \pm \langle 15, 12\rangle$.

4.5 Image interpretation policies

During interpretation, INFOGAIN and (the policy produced by) BESTSEQ each iteratively select an operator $o(k) \in O$. (Recall that the values of r and ℓ are determined from the context (of other detected features); we therefore do not need to specify them here.) INFOGAIN chooses an instantiated operator $o = o_{IG}(k_{IG})$ that has the maximum $EIG(o)/c(o)$ value. BESTSEQ selects an operator $o^*_{BS} = \pi_{BS}(\mathcal{F}(s))$

where s is the current state and π_{BS} is the (optimal) mapping from state to actions. In each case, the operator is applied to the appropriate region in the given test face image and the distribution is updated... until one face is identified with sufficiently high probability ($> P_{min}$) or the system fails (by exhausting all the possible operators, or cost $> C_{max}$).

5 Experiments

We used face images of 102 different people, each 92×112 pixels. We assigned 187 images to the set T, another 187 images to S and used another 333 as test images.[3] As shown in Figure 1, the faces are more or less in the same pose (facing front), with some small variation in size and orientation.[4] We considered all 20 operators based on the four features listed above and $k \in \{25, 30, 35, 40, 45\}$ for each feature. The BESTSEQ interpretation policy decides precisely what operator $o_i(k, \ell, r)$ to apply on which portion of the image, given each state (which here corresponds to a specific set of information gathered); see Figure 2.

Basic Experiment: We set $P_{min} = 0.9$ and $C_{max} = \infty$ (*i.e.*, no upper limit on identification cost).[5] In each "set-up", we assigned a random probability to each person (corresponding to drawing a sample with replacement). On each run, we picked one face randomly from the test set as the target, and identified it using each of the policies. We repeated this for a total of 30 runs per set-up, then changed the probability distribution and repeated the entire process again, for a total of 15 set-ups. The accuracy of identification for INFOGAIN and BESTSEQ are 89.78% and 90.44%, respectively. (Recall both were required to obtain at least $P_{min} = 90\%$.) The average cost of identification was 1.045 ± 0.038 and 0.717 ± 0.030 seconds[6] for INFOGAIN and BESTSEQ respectively. Hence, we see BESTSEQ took much less time than INFOGAIN to obtain this level of accuracy.

The first two rows of Table 1 shows how often INFOGAIN (resp., BESTSEQ) used one (resp., two, three, four) operators before terminating, over these 450 runs. We observe the INFOGAIN applied slightly more operators than BESTSEQ on the average. More importantly, as BESTSEQ can exploit simple operator interactions, the search area it uses for the subsequent operators can be narrower than INFOGAIN's. These two factors result in a much lower cost for BESTSEQ.

Bounding the Cost: In many situations we need to impose a hard restriction on the total cost; we therefore considered various values of C_{max} in the range [0.25–2.0] seconds. We then picked one face randomly from the test set, and identified the test image for each of these maximal costs, using INFOGAIN and BESTSEQ.

Table 1: How often BESTSEQ (INFOGAIN) used k Operators

	C_{max}	1	2	3	4	ave
IG	∞	7.11	57.56	10.00	25.33	2.54
BS	∞ ($P_{min} = 0.9$)	8.89	56.00	15.33	19.79	2.46
BS	2.00	11.33	64.89	3.78	20.00	2.32
BS	1.50	11.33	64.89	3.78	20.00	2.32
BS	1.00	11.33	64.89	5.33	19.11	2.34
BS	0.50	65.33	34.67	0.00	0.00	1.35
BS	0.25	100.0	0.00	0.00	0.00	1.00

BS = BESTSEQ; IG = INFOGAIN

As always, we terminate whenever the probability of any person exceeds P_{min} or if the cost exceeds C_{max}, returning the most likely interpretation.

We repeated this experiment for a total of 15 set-ups (each with a different distribution over the people) and with 30 random runs (target face images) per set-up. We found that 74.29% of the policies produced matched the policy shown in Figure 2.

Figure 3(a) plots the accuracy (the percentage of correct identifications) for the policies found for various values of C_{max}. In general, the policies produced by BESTSEQ had better accuracy than the ones produced by INFOGAIN. BESTSEQ found different policies for different costs. The bottom 5 rows of Table 1 show how often BESTSEQ used one (resp., two, three, four) operators for different values of C_{max} before terminating.

For low cost values (under 0.5 seconds), BESTSEQ performs much better than INFOGAIN (accuracy of ≈65.33%, versus ≈1.78%). As BESTSEQ was able to exploit operator interactions. it was able to apply a second (and probably a third) operator by using a smaller sweep area (ℓ, r) within the allotted time. Since INFOGAIN does not exploit this, it needed significantly more time.

Varying the Minimum Accuracy: In this experiment, we varied P_{min} from 0.1 to 0.9. For each of these values, we chose a face randomly from the test set as the target and identified it using each of the two policies. During the process, the first person i in the training set for which $P(\,\mathrm{id}(h) = i \,|\, \cdot) > P_{min}$ is returned (or if cost $> C_{max}$, the most probable face is returned). We repeated this for 30 different faces (runs) per set-up, and repeated the entire process for a total of 15 different set-ups. About 70.4% of these policies matched Figure 2.

We evaluated the results in two different ways. First, Figure 3(b) compares the percentage of wrong identifications of each policy, for each P_{min} value. We see no significant difference in error between BESTSEQ and INFOGAIN. The second graph, Figure 3(c), compares the average cost of each policy, for each P_{min} value. Here we see that BESTSEQ has much lower cost than INFOGAIN. Again, this is because BESTSEQ can use operator interactions to sweep a narrower region.

[3] We assume that any test face-image belongs to one of the people in the training set, but probably with a different facial expression or in a slightly different view, and perhaps with some external features not in the training image (like glasses, hat, etc.), or vice versa.

[4] (1) These faces were downloaded from the web sites `whitechapel.media.mit.edu` and `www.cam-orl.co.uk`. (2) This work assumes the head has already been located and normalized; if not, we can use standard techniques [TP91] first.

[5] The other part of the task, $\mathcal{V}$, is "identifying people"; this is true for all of the experiments discussed here.

[6] All the experiments reported in this paper were run on a Pentium 866 MHz. PC with 128 MB RAM running Linux OS 2.2.19-12.

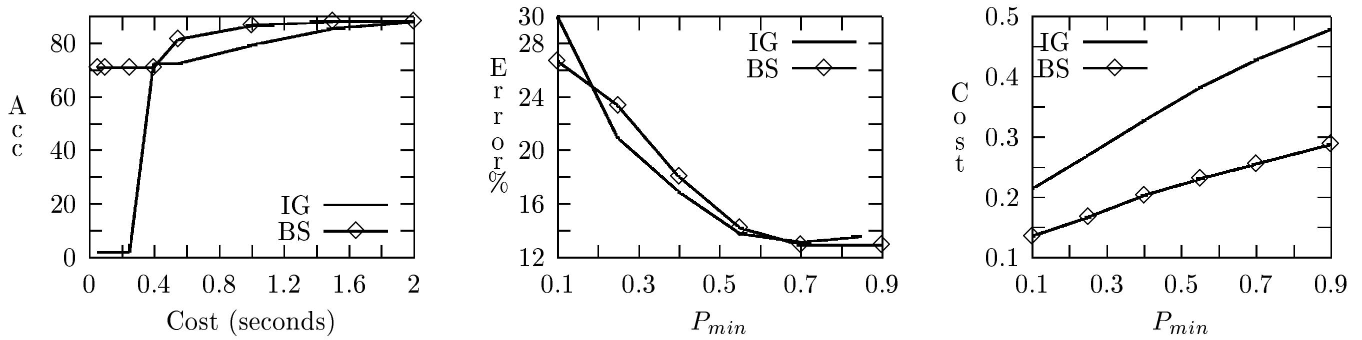

Figure 3: (a) Cost vs. Accuracy (b) Min. Accuracy vs. Error (c) Min. Accuracy vs. Cost

6 Conclusions

Future Work: This paper explored one type of operator interaction: where the output of one operator can be used to help a subsequent operator — here by reducing the time it will require to search. Moreover, our operators were relatively simple, as they always succeeded (or at least, think that they succeed), and they could be run in any order. As a consequence, our policy was quite simple — just straight-line sequence of operators.

There are many ways to extend our analysis. First, it might be useful to allow each operator to return not just a position, but also other information, such as confidence (which would become part of the state). The best policy, then, could use that information when deciding on the proper future action to take. If the abstracted state also included other information about the image, such as average intensity, the policy could use that information as well. This could result in a policy that was more complicated and, presumably, more accurate.

Finally, it would be interesting to consider operators that have some explicit dependency — *e.g.*, one cannot run the "connect edgel" operator unless we have already run some "produce edgel" operator. These precedence constraints would add yet other challenges to our framework.

Contributions: This paper shows how dynamic programming can be used to build efficient interpretation systems. We argue that computing the *utility* of operator sequences, which incorporate the benefits of operator interactions, can play a significant role towards building efficient interpretation systems. However, in several real world situations, such systems may have (i) to deal with complex and unwieldy states, and (ii) to explore the operator space to find the most promising operator sequence, which is expensive. We addressed the first issue by using simplified abstract states in face recognition, and addressed the second by using an off-line limited lookahead search to find the most promising operator sequence.

We framed the image interpretation task as a Markov Decision Problem and used ideas from reinforcement learning to compute the utility of imaging operators *non-myopically* over a finite lookahead depth in an operator space. We built an interpretation system that uses these concepts and compared its performance with a successful myopic system in face recognition. Our results show that the former has a much better performance in various experiments that address the cost and accuracy tradeoffs.

Acknowledgments

The authors gratefully acknowledges the generous support from Canada's Natural Science and Engineering Research Council and the Alberta Ingenuity Centre for Machine Learning.

References

[BDB00] J. Bins B. Draper and K. Baek. Adore: Adaptive object recognition. In *Videre, 1(4)*, pages 86–99, 2000.

[BDH99] C Boutilier, T Dean, and S Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence*, 1999.

[BDL02] V Bulitko, B Draper, and I. Levner. Mr adore control policies. Technical report, Dept. of Computing Science, University of Alberta, 2002.

[DH73] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

[EC97] K Etemad and R Chellappa. Discriminant analysis for recognition of human faces. *Journal of Optical Society of America*, 1997.

[HR96] R Huang and S Russell. Object identification: A bayesian analysis with application to traffic surveillance. *Artificial Intelligence*, 1996.

[IG01a] R. Isukapalli and R. Greiner. Efficient car recognition policies. In *ICRA*, pages 2134–2139, Seoul, 2001.

[IG01b] R. Isukapalli and R. Greiner. Efficient interpretation policies. In *IJCAI*, Seattle, 2001.

[Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[PL95] A R Pope and D Lowe. Learning object recognition models from images. In *Early Visual Learning*, 1995.

[PMS94] A Pentland, B Moghaddam, and T Starner. View-based and modular eigenspaces for face recognition. In *IEEE CVPR*, 1994.

[PWHR98] P Phillips, H Wechsler, J Huang, and P Rauss. The feret database and evaluation procedure for face recognition algorithms. *Image and Vision Computing*, 1998.

[SB98] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press (A Bradford Book), Cambridge, MA, 1998.

[TP91] M Turk and A Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 1991.