Dynamic Processing Allocation in Video

Daozheng Chen, Mustafa Bilgic, Lise Getoor, and David Jacobs
Department of Computer Science
University of Maryland, College Park

Abstract

Large stores of digital video pose severe computational challenges to existing video analysis algorithms. In applying these algorithms, users must often trade-off processing speed for accuracy, as many sophisticated and effective algorithms require large computational resources that make it impractical to apply them throughout long videos. One can save considerable effort by applying these expensive algorithms sparingly, directing their application using the results of more limited processing. We show how to do this for retrospective video analysis by modeling a video using a chain graphical model and performing inference both to analyze the video and to direct processing. To accomplish this, we develop a new algorithm to direct processing. This algorithm approximates the optimal solution efficiently. We apply our algorithm to problems in background subtraction and face detection and show in experiments that this leads to significant improvements over baseline algorithms.

Index Terms

Video processing, resource allocation, graphical models, optimization, background subtraction, face detection, dynamic programming

I. Introduction

New technology is giving rise to large stores of digital video. Their size has increased much faster than the computational resources needed to effectively process them. At the same time, as we develop increasingly sophisticated and effective vision algorithms, these also demand greater computational resources. Consequently, it is important to develop strategies for applying vision algorithms with greater efficiency to video data.

The scope of the problems we face is evident. Surveillance systems can contain thousands of cameras and a large amount of video. Real time processing of huge data sets is extremely challenging; retrospective or forensic analysis creates even greater problems when one must rapidly examine hours or days of video from thousands of cameras. For example, British police were required to examine 80,000 CCTV tapes from a network of 25,000 cameras [1] to discover the image of a bomber after the terrorist attack in London in 2005 [2]. Automatic processing is needed to speed up this analysis, but one cannot hope to process all available video in such cases; it is essential to direct processing to portions of video most likely to be informative.

In this paper, we develop a new method for controlling processing, so that available resources are directed at the most relevant portions of the video. In our proposed approach, we initially perform some inexpensive processing of a video by applying a cheap but less accurate algorithm combined with sparse application of a more expensive and accurate algorithm. We then use an inference algorithm to determine to which frames we should apply further expensive processing.

Our first contribution is to combine information from cheap and expensive features, using a graphical model for video. This is a second-order Markov model with a node for each frame, and a state variable that indicates whether this frame is relevant to our current query. For example, the state might indicate whether the frame contains a visible face. Each state has two potential observations. The first observation is always given; it is obtained by running a cheap algorithm on all frames. For example, cheap background subtraction might provide a clue as to whether people are currently visible. The second observations is only obtained if a more expensive and accurate algorithm is applied to that frame (in this example, a face detector). As in a Hidden Markov Model (HMM), each observation directly depends on the current state.

In addition, in our model each observation directly depends on the previous observation. This captures the phenomenon that errors made by an algorithm are often correlated from one state to the next. This model allows us to effectively combine information from cheap and expensive algorithms to improve performance.

Our second and primary contribution is a new algorithm that uses this model to determine where in a video to apply the expensive algorithm. We build on prior work by Krause and Guestrin [3] that shows that one can use a dynamic programming algorithm to determine the optimal places at which to make observations in a first-order Markov chain. While this work is readily extended to our graphical model, it requires $O(B^2n^3)$ computation time, where n is the number of nodes in the Markov chain, and B is the number of places at which we will apply the expensive algorithm. In our setting n is the number of frames in the video and B is also $O(n)$, so this algorithm is not practical for video analysis.

We solve this problem with a new algorithm that produces an approximately optimal answer efficiently. More precisely, we make an additional assumption about the convexity of the reward from observations as the budget increases, a law of diminishing returns that we show is valid in practice. Then, we show that by applying part of the total budget to make observations at a uniform step size, we can find a batch allocation of observations that will be at least as good as the optimal allocation, and that requires a modest amount of computation. This batch allocation makes use of rewards computed by Krause and Guestrin's algorithm, applied to small sections of the video.

Our approach is quite general, and can be applied to a wide range of scenarios in which multiple algorithms are combined into a single system. Our final contribution is to experimentally demonstrate the value of this algorithm in two very different vision tasks: background subtraction and face detection. First, to perform background subtraction efficiently we combine a very cheap background subtraction algorithm that uses frame differencing [4] with a more expensive background subtraction algorithm using Gaussian mixture models [5]. Second, we use background subtraction to trigger face detection. We show that our algorithm can be used to significantly improve performance of systems that combine these algorithms. In particular, our inference algorithm requires the idealized assumption that the expensive algorithm determines the state perfectly. This is never true in practice, but we show that in real-life settings this assumption holds sufficiently well that our algorithm produces excellent results. We also show that a relatively poor cheap algorithm can be effectively combined with an expensive algorithm to improve system performance.

The organization of our paper is as follows. In Section II we discuss related work. In particular, we describe a dynamic programming algorithm [3] that determines the optimal place to make observations. We then describe our new algorithm in the context of Markov Chains in section III. Then, we introduce our graphical model for video analysis and describe how to apply the new algorithm to this model in section IV. In section V, we show experiments in video processing.

II. Prior Work

In this section we first review background subtraction and face detection algorithms that are used in our experiments. Next, we describe work on video processing that deals with issues of resource constraint. We then discuss work on feature and label acquisition. Finally, we describe the algorithm by Krause and Guestrin [3] in detail.

A. Background Subtraction

Background subtraction is a technique to detect moving objects in video, usually taken by static cameras. This typically involves building a background model to classify whether a pixel is in the background or not. We describe two methods in detail since our experiments use these two techniques.

In frame differencing (FD), the background model of the frame at time t, f_t, is the frame in the previous time step, f_{t-1} (Jain and Nagel [4]). Given a threshold, Th,

$$|f_t - f_{t-1}| > Th \tag{1}$$

gives the foreground region of f_t. This technique may not be able to identify the interior pixel of a large and uniformly-colored moving object. However, it requires very little computation.

Stauffer and Grimson [6] use a mixture of Gaussians (MoG) for the background model of each pixel. Let $\vec{x}$ be the value of the pixels, with the assumption that each dimension of $\vec{x}$ is independent. The probability distribution of $\vec{x}$ is modeled as

$$P(\vec{x}) = \sum_{k=1}^{M} \omega_k N(\vec{\mu}_k, \sigma_k I), \tag{2}$$

where M is the number of Gaussian components, ω_k is the weight of the kth component, $N(\vec{\mu}_k, \sigma_k I)$ is a Gaussian distribution with mean $\vec{\mu}_k$ and covariance matrix $\sigma_k I$, where I is the identity matrix. This model is for both foreground and background pixel values. With the assumption that higher and more compact distribution is more likely to be the background, MoG selects components whose ratio between its peak value and standard deviation is greater than a certain threshold. Finally, it uses pixel values from a reasonable period of recent time to update the model parameters. A pixel value is assigned to a component by an on-line K-means approximation and the parameters, $\omega_k, \vec{\mu}_k$, and σ_k are updated by some online cumulative average formulas. This method is much more sophisticated than the FD method, and requires significantly more computational resources. Zivkovic [5] improves this work by using recursive equations that can also simultaneously select the appropriate number of components in the mixture model for each pixel. We call this method the improved adaptive Gaussian mixture model (IAGMM), and we use it in our experiments.

B. Face Detection

Face detection refers to the problem of determining whether an image contains faces, and determining the location and extent of each faces. Yang et al. [7] provide a comprehensive survey of various face detection methods. Of these methods, we describe the work by Viola and Jones [8] in detail because our experiment uses this technique.

Viola and Jones introduce a now widely used object detection scheme, applying it first to face detection. With integral images for fast computation, they use a set of feature that are similar to the coefficients in Haar wavelet transforms. They then construct classifiers by selecting a small number of important features using Adaboost. Finally, the scheme detects faces inside an image region by applying classifiers in a cascade. At each level of the cascade, one uses a classifier with a very low false negative rate, although false positives might be high. Subsequent classifiers are run only when previous classifiers indicate a positive result. One can think of this as a methodology for combining classification algorithms, which is also our topic. However, while very effective, this approach requires a variety of cheap classifiers with a range of performance characteristics, and is hard to apply when one wishes to combine the results of a few specific algorithms. Lienhart and Maydt [9] extend this work by adding an efficient set of 45° rotated features to the original feature set and by using a new post-optimization procedure for a given boosted classifier. Their work shows significantly lower false alarm rate, and we use this method to detect faces in our experiment.

C. Computational Performance in Video Processing

There is a vast literature on video processing. Performance is often an issue, as many effective algorithms are too slow to run in real-time, and even fast algorithms may require enormous amounts of time when used to perform retrospective analysis of large quantities of previously taken video. One common strategy is to run cheap, lower level algorithms such as motion detectors to determine when something interesting might be happening. When these detect motion, higher level algorithms are then deployed. In a variant of this, Krishna et al. [10] propose an algorithm switching approach to handle background subtraction. The system starts by processing each frame with a uni-modal background subtraction algorithm. When

the system shows poor segmentation quality, it switches to use the MoG model. The system considers when to apply the expensive algorithm in real time, and it simulates the trigger of algorithm switch from an external source, such as a motion sensor or a light sensor.

Barotti et al. [11] present a closely related work that uses algorithm switching to handle lighting changes and solve bootstrapping problems in motion detection. When the system detects sudden global illumination variation, the motion detection switches from background subtraction using a single Gaussian to FD. The latter algorithm requires smaller computational resources, adapts quickly to lighting changes, and does not need any bootstrapping. The system then re-initializes the single Gaussian model to handle the bootstrapping problem, and returns to the original configuration when initialization completes. This work uses the advantages of the cheap algorithm to handle the drawbacks of the expensive algorithm in a particular application.

An alternative approach is to use special-purpose hardware or high performance computing to run expensive algorithms efficiently. A number of researchers consider using graphics hardware to accelerate computer vision algorithms. For example, real-time stereo algorithms suitable for implementation on Graphics Processing Units (GPU) are proposed in [12], [13], and [14]. Yang and Welch [15], and Griesser et al. [16] introduce algorithms to do background subtraction in real time using a GPU. Heymann et al. [17] and Sinha et al. [18] use a GPU to detect and track image feature points in real time. Fung et al. [19] introduce openVIDIA which provides a library and API for using single or multiple GPUs to accelerate computer vision and image processing. Besides GPUs, there are many other hardware solutions to efficient computation of expensive algorithms. For instance, implementations of image feature detectors using a Field-programmable gate array (FPGA) are discussed in [20] and [21]. Seinstra et al. [22] introduce Parallel-Horus which allows researchers in multimedia content analysis to implement sequential programs in Grid computing environment.

In general, while these methods are of interest, they are largely orthogonal to our concerns in this paper. In particular, if we can gain greater efficiency by directing processing of expensive algorithms in a video, these can be readily combined with hardware solutions that speed up these algorithms.

D. Feature and Label Acquisition

In closely related work, the machine learning community has looked at the problem of determining which features to acquire in order to correctly classify instances. In this setup, instances are described by a set of features each of which has an associated acquisition cost, and a total budget limits feature acquisition. Some example strategies are [23], [24]. The biggest difference between this line of work and ours is that the feature-value acquisition community treats each instance as independent; the decisions for one instance are made independent of the decisions made for the other instances. However, in our case, the information we want to extract in nearby frames of video is highly correlated, and we should be able to do better if we take these correlations into account.

Another related area of work is label acquisition: instead of obtaining features, we can query an oracle to determine an instance's label directly. Given a network of instances, such as a sequence of frames, a network of friends, etc, acquiring the label for an instance helps in correctly classifying the rest of the network. The question is then which instances should be queried in order to get the best performance on the remaining ones. Rattigan et al. [25] queries the instances that are structurally important, e.g. highly connected instances, central instances, etc. Bilgic and Getoor [26] build a classifier that can predict which instances might be misclassified and query a central instance only if it is predicted as misclassified. Even though these methods have been quite successful in practice, they are heuristic approaches and have no theoretical guarantees (partly because they are applicable to general networks). However, for the class of chain graphical models such as HMMs, Krause and Guestrin [3] show how to solve the label acquisition problem optimally. We describe their work next.

E. Optimal Observation Plans

Krause and Guestrin [3] present an optimal algorithm for selecting observations for the class of chain graphical models. Since we build on this method, we now describe it in some detail, although we consider a special case of their work that is suitable to our problem. A set of random variables $S = \{X_1, \cdots, X_n\}$ forms a chain graphical model if whenever $i < j < k$, X_i is conditionally independent of X_k given X_j. For example, consider a HMM unrolled for n time slices. Then the n hidden state variables form a chain graphical model. Suppose that for each of these variables, it is possible to observe its hidden state at a fixed cost. This corresponds, in our problem, to the supposition that an expensive algorithm is extremely accurate, and reveals the hidden state. They also suppose that observing variable X_j has a local reward R_j. In particular, the expected local reward is

$$R_j(X_j|O) \triangleq \sum_o P(O = o) R_j(P(X_j|O = o)), \tag{3}$$

which is an expectation taken over all possible values o for the observed variables O. Many different functions, such as entropy, are applicable for this local reward. They describe the max-marginal:

$$R_j(P^{max}(X_j|O)) = \sum_o P(o)[P^{max}(x_j^*|o) - P^{max}(\bar{x}_j|o)], \tag{4}$$

where x_j is the value of state variable X_j, $x^* = argmax_{x_j} P^{max}(x_j|o)$, and $\bar{x} = argmax_{x_j \neq x^*} P^{max}(x_j|o)$. This reward is very useful for classification purposes [3], and we use it in all examples and experiment in this paper. In our case, since we have a two-class problem, this reward is equivalent to considering the expected number of correct classifications.

Assume that there is a fixed budget B for selecting observations, the formulation of the optimization problem is:

Select observations to maximize

$$J(O) = R(O) = \sum_j R_j(X_j|O) \tag{5}$$

Subject to

$$b \leq B \tag{6}$$

where j is the index over the state variables S, b is the number of observed state variables O, and B is the total budget for the whole chain. Also, observations can include variables at any time step in the chain since we consider processing a video after it is recorded. Krause and Guestrin refer to this as the *smoothing* problem.

In addition, the conditional independence property in the chain graphical model simplifies the local reward. With this property, the local reward is simply $R(X_j|O) = R(X_j|X_j)$ in the case that $X_j \in O$. In the case that X_j is not in O, we have $R(X_j|O) = R(X_j|O_j)$, where O_j is the subset of O containing the closest ancestor and descendant of X_j in O. This is the key insight for efficiently solving this optimization problem.

Furthermore, they consider both a conditional planning version of this problem, in which the best observation is made and then the optimal next observation is computed, and this is repeated k times, and a batch version of the problem, in which one decides on the locations of the best observations with a total cost of k first, and then makes these observations. We only consider the conditional planning variant here since it in general produces the best performance.

Krause and Guestrin [3] solve this problem by noting that once an observation is made, it splits the problem of determining future observations into conditionally independent components before and after the observations. This allows for a dynamic programming solution. They define a value, $J_{a:b}(x_a, x_b; k)$

which denotes the reward produced by the optimal plan with a budget of k over the interval from variables X_a to X_b, given that these variables have been observed to have states x_a and x_b. Then they note that $J_{a:b}(x_a, x_b; k)$ can be recursively computed given the value of $J_{c:d}(x_c, x_d; l)$ for all $a \leq c \leq d \leq b$ and $l < k$. For the case of smoothing and conditional planning, the recursive formula is

$$J_{a:b}(x_a, x_b; k) = \max\{J_{a:b}(x_a, x_b; 0), \max_{a<j<b}\{\sum_{x_j} P(X_j = x_j | X_a = x_a, X_b = x_b)\{$$
$$R_j(X_j | X_j = x_j) + \max_{0 \leq l \leq k-1}[J_{a:j}(x_a, x_j; l) + J_{j:b}(x_j, x_b; k - l - 1)]\}\}\}, \quad (7)$$

where the base case is

$$J_{a:b}(x_a, x_b; 0) = \sum_{j=a+1}^{b-1} R_j(X_j | X_a = x_a, X_b = x_b). \quad (8)$$

To handle the case that X_a and X_b are not always observed in advance, the algorithm adds two independent dummy variables, X_{a-1} and X_{b+1}, which have no reward and observation cost but have default states, to the head and tail positions of the chain. Thus the optimal reward for a chain of n variables with a budget of B is computed as $J_{0:n+1}(x_0, x_{n+1}; B)$. Assuming that the probabilities to evaluate local rewards have been computed and computing local rewards takes constant time, we conclude from Theorem 2 in [3] that the complexity of this algorithm is

$$O(B^2 n^3). \quad (9)$$

III. Efficient Observation Plans for Chain Graphical Models

We now present a novel algorithm to find observation plans that is efficient enough to apply to problems with very large values of n. While the new algorithm only approximates the optimal one, it is much faster for domains such as ours. In the next sections, we will show how this algorithm can be applied to video processing. Our algorithm first uses part of the total budget to make uniform observations with a certain step size. This splits the Markov chain into M consecutive intervals where the first and last variables of each interval are observed. Because of the Markov assumption of the chain model, this breaks our problem up into a series of smaller problems of the same size. These problems are not independent, however, since the remaining budget must be parceled out between all these intervals. We show that, with an additional, reasonable assumption, this can be done optimally. Once the budget is allocated to intervals, observations can be allocated within intervals using the optimal algorithm of Krause and Guestrin. The main novel insight of this approach is that rather than just allocating observations to optimize reward, in practice it may make sense to allocate some observations in a way that makes the inference algorithm more tractable.

This leads to Algorithm 1, Dynamic Processing Allocation (DPA), which is something of a hybrid between conditional and batch allocation. Such a strategy uses a budget of size $B' + B''$ and is guaranteed

Algorithm 1 Dynamic Processing Allocation (DPA)

1) We use B' observations to make uniform observations to break the chain model into consecutive disjoint intervals separated by the observed variables;
2) We allocate the remaining budget B'' to these intervals in a batch mode;
3) We use the optimal observation algorithm in section II-E to determine the observation locations within each interval in a conditional mode.

to perform at least as well as an optimal batch algorithm that uses a budget of size B''. In practice, we expect to do much better than this optimal batch algorithm, since the observations we use to break the problem into intervals also provide very useful information, and because we can use an optimal conditional plan within each interval, which can be much better than the optimal batch plan.

In step 2 of the algorithm, which allocates the budget between intervals, we consider maximizing the sum of optimal reward across all intervals. That is, let k_i be the budget allocated to interval i and $J_i(k_i)$

be the reward of the optimal plan for interval i with a budget of k_i. We want to find $k_1, k_2, \cdots$, and k_M such that they maximize

$$\sum_{i=1}^{M} J_i(k_i), \tag{10}$$

subject to

$$\sum_{i=1}^{M} k_i = B''. \tag{11}$$

To do this optimization, we observe that for each interval, the optimal reward typically forms a convex curve as the budget increases. That is, for internal i, the plot of $J_i(k_i)$ against k_i as k_i increases is convex in general. This means that we suppose function $J_i(k_i)$ exhibits submodularity, which is a kind of law of diminishing returns property: Adding additional observations helps less if many observations have already been made and helps more if only a few observations have been made [27]. We denote these kinds of curves as Reward-Budget (RB) curves, and the assumption that these curves are convex will allow us to optimally allocate our budget. We will experimentally verify that this assumption is reasonable.

We now introduce an algorithm to maximize the sum of reward increments for each interval. Let N_i be the maximum budget allowed for interval i, typically the number of unobserved state variables in the interval. We define the reward increment with budget k as

$$\Delta J_i(k) \equiv J_i(k) - J_i(k-1), \tag{12}$$

where $k = 1, \cdots, N_i$. Given the convexity assumption, we have

$$\Delta J_i(k) \leq \Delta J_i(k-1) \tag{13}$$

for all possible ks, and we denote this as the convexity property. With this notation, we describe Algorithm 2, batch budget allocation. In practice, most intervals are allocated small budgets, so it is wasteful to compute the entire RB curve for all these intervals. We therefore can use Algorithm 3, incremental budget allocation, to do the optimization and save running time.

Algorithm 2 Batch budget allocation

1) Compute the reward increment $\Delta J_i(k)$ for $i = 1, 2, \cdots, M$ with all possible ks;
2) Sort all these reward increments in descending order. Then the budget for each interval is determined by the number of increments it has in the top B positions of the sorted list.

Algorithm 3 Incremental budget allocation

1) Initialize the budget of each interval to be zero;
2) Compute the reward increment $\Delta J_i(1)$ for $i = 1, 2, \cdots, M$;
3) Select the highest increment, and add one to the budget of the corresponding interval I;
4) If the total budget for the whole chain has been used up, terminate and use the current budget allocation for each interval as the final budget allocation;
5) If not, compute the next reward increment for interval I, and use it to replace the current reward increment for this interval. Go back to step 4.

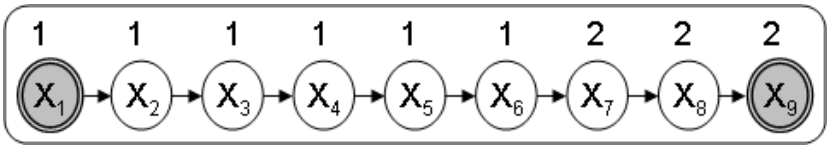


Fig. 1. The example chain graphical model. Because we assume that X_1 and X_9 are observed in advance, we use gray nodes with double-line boundaries to highlight this. The state of a node can be 1 or 2, and we show the actual state on the top of each node.

A. An Example

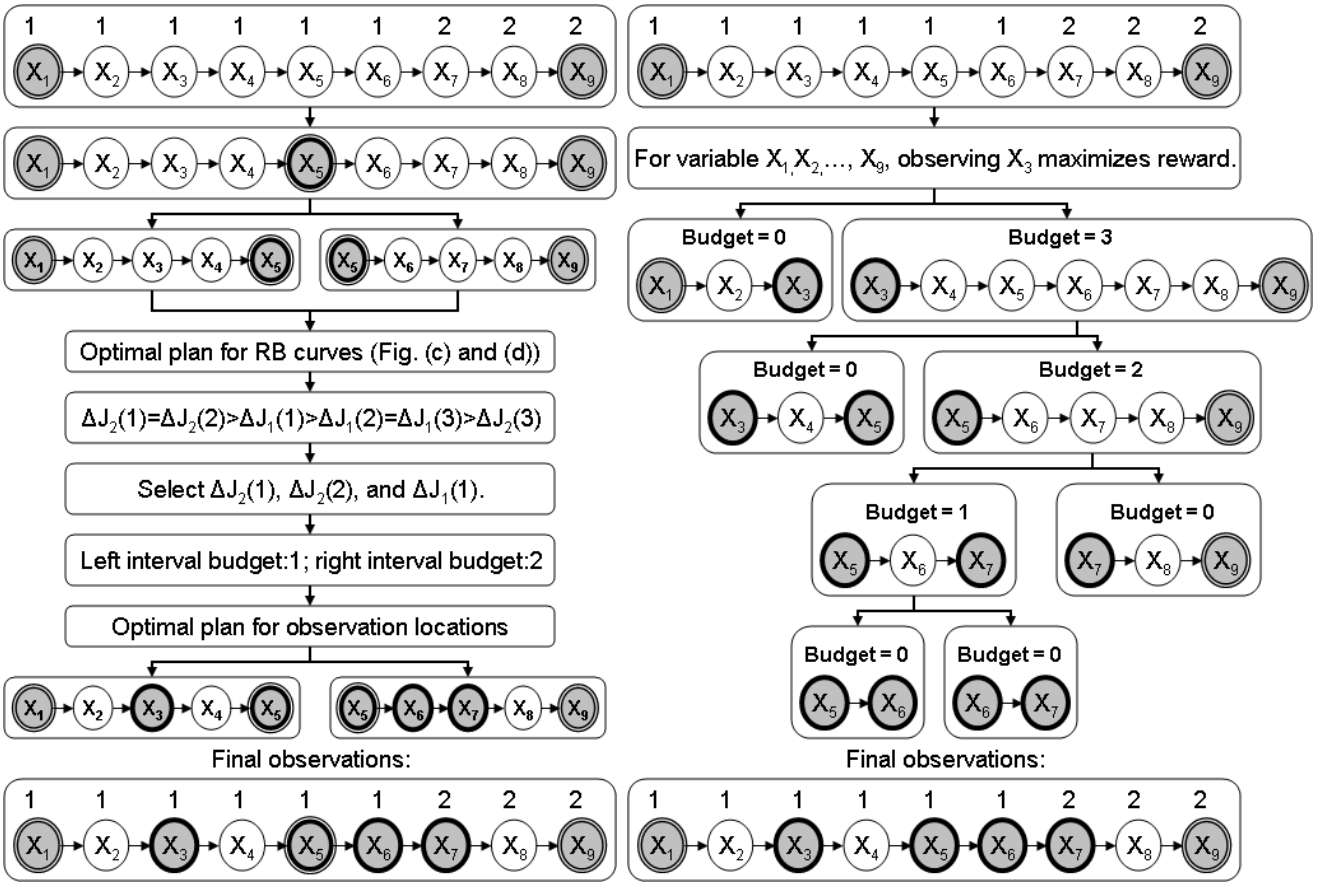
We use an example to illustrate how DPA works and how the optimal algorithm is different from ours. Consider a chain graphical model with nine state variables whose value can be either 1 or 2. The prior probability of being in each state is 0.5. The transition probability of switching from one state to the other is 0.2. Figure 1 shows this model and it displays the states on the top of each variable. For simplicity, we assume the first and last states are known in advance. Fig. 2 shows how DPA with batch budget allocation and how the optimal algorithm determine the observation locations with a budget of 4.

First, we note that for this example, it is most likely that the initial states have a value of 1, and that at some point in the chain there is a single transition from 1 to 2. To correctly determine the state values, the main goal is to find the location of this transition. It is also possible that there are really three transitions from 1 to 2 to 1 to 2, and a secondary goal will be to check on that. Next, we note that the optimal conditional plan is able to perform a binary search to find such transitions. The optimal strategy for this situation involves first determining the value of X_3. If this state is 1, then the remaining budget is sufficient to allow a binary search to be performed on states X_4-X_8, to find the transition from 1 to 2. This strategy therefore guarantees that the transition from 1 to 2 will be found, and maximizes the chances that any additional transitions will be found.

Suppose instead we run DPA, with a budget of 4. In this case, the algorithm must begin by determining the state of X_5, to break the problem into two equal parts. When this state is found to be 1, the algorithm then allocates its budget between these two subsequences. To allocate the remaining budget to each interval, it computes the RB curves up to its maximum budget for both intervals, as shown in Fig. 2(c) and 2(d). Notice that both curves satisfy the convexity property. In addition, the reward increment under a small budget for the right interval is higher than those for the left interval. This is because the state of the first and last nodes for the right interval indicate a state transition in between. As a result, the right interval obtains a higher budget. In fact, DPA allocates two observations to the right side of the chain, which is enough to perform a binary search for the state transition, and one observation to the left side. This final observation on the left side is more likely to find something interesting than if allocated to the right side, once the binary search has occurred.

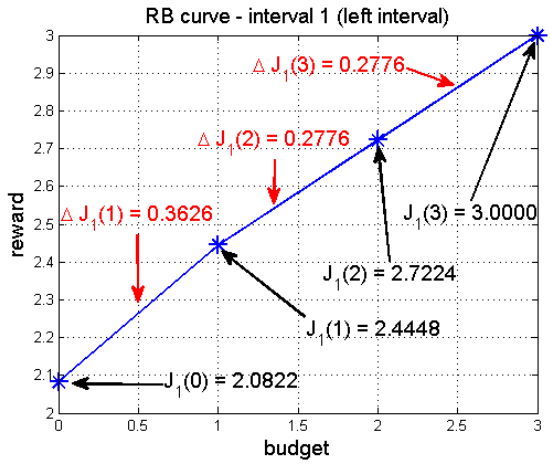
Notice that DPA is not optimal in two ways. First, an optimal set of observations may not include X_5. Second, allocating one observation to the left subsequence and two to the right subsequence may not be optimal; future observations could determine that a different allocation would be better. On the other hand, in the optimal algorithm shown in Fig. 2(b), notice that the initial observation X_3 is determined after computation and comparison of expect reward of observing $X_1, \cdots, X_9$ with all possible budget distribution. This is a considerable amount of computation. In DPA, this interval is split into two shorter intervals, and reward can be more cheaply computed for each subchain separately.

Finally, we consider how incremental budget allocation works in this example in detail. After the observation of X_5, the chain is split into two intervals as shown in Fig. 2(a) and the remaining budget becomes 3. The incremental algorithm then initializes the budget for both intervals to be 0 and computes $J_1(0), J_1(1), J_2(0)$, and $J_2(1)$ for $\Delta J_1(1)$ and $\Delta J_2(1)$. Since $\Delta J_1(1) = 0.3626 < \Delta J_2(1) = 1.0000$, it increases the budget for the right interval by 1. The remaining budget becomes 2 and it computes $J_2(2)$ for $\Delta J_2(2)$. After this, it again increases the budget for the right interval by 1 because $\Delta J_1(1) < \Delta J_2(2) = 1.0000$. The remaining budget becomes 1 and it computes $J_2(3)$ for $\Delta J_2(3)$. After this, because $\Delta J_1(1) > \Delta J_2(3) = 0.1176$, it increases the budget for the left interval by 1. The remaining budget

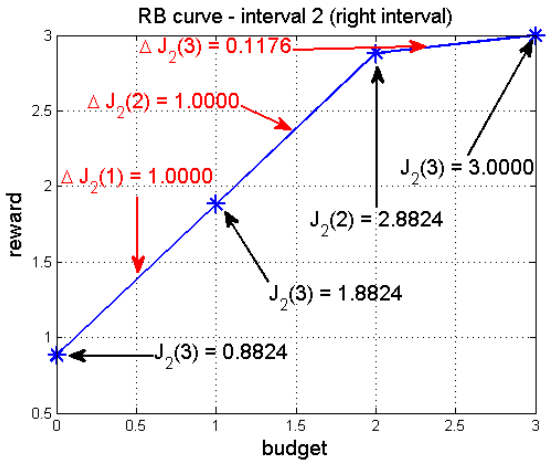


(a) DPA with batch budget allocation

(b) Optimal algorithm



(c) RB curve - interval 1 (left interval)



(d) RB curve - interval 2 (right interval)

Fig. 2. (a) shows how DPA with batch budget allocation determines the observation locations. We use gray color to highlight observed nodes. The interval size is 5, so the initial observation is X_5, which is highlighted by a double-line boundary with a thick inner line. (c) and (d) show the RB curves for left and right intervals respectively. The observation locations for each of these two intervals are determined by the optimal algorithm, and are highlighted by a single-line thick boundary. (b) shows how the optimal algorithm determines observations locations, which are also highlighted by a single-line thick boundary. It shows how the chain is split into sub-chains by sequential observations.

becomes 0 and the algorithm terminates with budget of 1 for left interval and 2 for the right interval. Notice that compared with batch allocation, incremental allocation saves computation by not computing $J_1(2)$ and $J_1(3)$.

B. Proof of Correctness

We now provide a proof that batch budget allocation is correct; it follows directly that incremental budget allocation must also be correct.

We use the following new notations, definitions, and facts. With a budget of B, we let $\hat{k}_1^B, \hat{k}_2^B, \cdots, \hat{k}_M^B$ be an optimal budget allocation, where M indicates the number of subsequences among which we must divide the budget. Let $\bar{k}_1^B, \bar{k}_2^B, \cdots, \bar{k}_M^B$ be the budget allocation by the algorithm. Denote the sorted list rewards gained by an additional observation as Z and the sum of the top B reward increments in Z as ΔL. In addition, we define $\Delta J_i(0) \equiv 0$ to handle the case that some interval has a budget of 0. Then we define $\Delta \hat{J}_i^B \equiv \sum_{j=0}^{\hat{k}_i^B} \Delta J_i(j)$, which means $\Delta \hat{J}_i^B$ is the sum of all reward increments for interval i with a budget of $\hat{k}_i^B$. Similarly, we define $\Delta \bar{J}_i^B \equiv \sum_{j=0}^{\bar{k}_i^B} \Delta J_i(j)$. Furthermore, since the algorithm pick only top B reward increments from Z to distribute budget, it is clear that $\sum_{i=1}^M \bar{k}_i^B = B$. Therefore, to prove the correctness, we only need to show that $\sum_{i=1}^M J_i(\bar{k}_i^B) = \sum_{i=1}^M J_i(\hat{k}_i^B)$. We now use Lemma 1, Lemma 2, and Theorem 1 to prove this.

Lemma 1:

$$\sum_{i=1}^{M} \Delta \bar{J}_i^B = \Delta L. \tag{14}$$

Proof: For an interval i, by the procedure of the algorithm, there must be $\bar{k}_i^B$ reward increments from interval i in ΔL. Let the sum of these increments be ΔL_i. In case that no such increment exists, we let $\Delta L_i \equiv 0$. Suppose $\Delta \bar{J}_i^B \neq \Delta L_i$. By convexity property, we know that ΔL_i must include some reward increment $\Delta J_i(x)$ such that $x > \bar{k}_i^B$ and $\Delta J_i(x) < \Delta J_i(\bar{k}_i^B) \leq \Delta J_i(\bar{k}_i^B - 1) \leq \cdots \leq \Delta J_i(1)$. Thus, there must be at least $\bar{k}_i^B + 1$ reward increments in top B positions from interval i. But this conflicts with the fact that the top B positions only contain $\bar{k}_i^B$ such increments. Thus, it can only be that $\Delta \bar{J}_i^B = \Delta L_i$. Finally, because $\sum_{i=1}^M \bar{k}_i^B = B$, we have $\sum_{i=1}^M \Delta L_i = \Delta L$. Therefore, $\sum_{i=1}^M \Delta \bar{J}_i^B = \sum_{i=1}^M \Delta L_i = \Delta L$. ■

Lemma 2:

$$\sum_{i=1}^{M} \Delta \bar{J}_i^B \geq \sum_{i=1}^{M} \Delta \hat{J}_i^B. \tag{15}$$

Proof: For any reward increment which is outside the top B positions of Z, we know it must be less than or equal to any reward increment within the top B positions because Z is sorted in descending order. Therefore, by Lemma 1, $\sum_{i=1}^M \Delta \bar{J}_i^B = \Delta L$ must be greater than or equal to the sum of any B reward increments from Z. Because $\sum_{i=1}^M \hat{k}_i^B = B$ and list Z contains the all possible reward increments from all intervals, we know that $\sum_{i=1}^M \Delta \hat{J}_i^B$ is also the sum of B reward increments from Z. Therefore, $\sum_{i=1}^M \Delta \bar{J}_i^B \geq \sum_{i=1}^M \Delta \hat{J}_i^B$. ■

By the definition of $\Delta J_i(k)$, we know that $J_i(k) = \Delta J_i(k) + J_i(k-1)$. Using induction, it is trivial to show that $J_i(k) = \sum_{j=1}^k \Delta J_i(j) + J_i(0)$. Because we define $\Delta J_i(0) \equiv 0$, then

$$J_i(k) = \sum_{j=0}^{k} \Delta J_i(j) + J_i(0). \tag{16}$$

With this formula, we can finally prove the correctness in following theorem.

Theorem 1:

$$\sum_{i=1}^{M} J_i(\bar{k}_i^B) = \sum_{i=1}^{M} J_i(\hat{k}_i^B). \tag{17}$$

Proof: By equation (16), we have

$$\sum_{i=1}^{M} J_i(\bar{k}_i^B) = \sum_{i=1}^{M}[\sum_{j=0}^{\bar{k}_i^B} \Delta J_i(j) + J_i(0)] = \sum_{i=1}^{M}[\Delta \bar{J}_i^B + J_i(0)] = \sum_{i=1}^{M} \Delta \bar{J}_i^B + \sum_{i=1}^{M} J_i(0). \tag{18}$$

Similarly,

$$\sum_{i=1}^{M} J_i(\hat{k}_i^B) = \sum_{i=1}^{M} \Delta \hat{J}_i^B + \sum_{i=1}^{M} J_i(0). \tag{19}$$

Then by lemma 2, it follows that $\sum_{i=1}^{M} J_i(\bar{k}_i^B) \geq \sum_{i=1}^{M} J_i(\hat{k}_i^B)$. Finally, because the budget allocation, $\hat{k}_1^B, \hat{k}_2^B, \cdots, \hat{k}_M^B$, is optimal, we have $\sum_{i=1}^{M} J_i(\bar{k}_i^B) = \sum_{i=1}^{M} J_i(\hat{k}_i^B)$. ■

C. Complexity Analysis

We can now determine the complexity of DPA. Here we consider the cost of determining where to make additional observations, where the cost of making each observation is always the cost of running the expensive algorithm. We introduce the following notation. Let the total budget be $B = B' + B''$ and let ϵ be a number less than 1 such that

$$\frac{n}{M} = \frac{1}{\epsilon}. \tag{20}$$

Since each interval has the same length, this is $\frac{1}{\epsilon}$. In addition, because the first step of the algorithm uses B' budget to make observations at a uniform step size, we have $B' = M = \epsilon n$ (we choose ϵ so that ϵn is an integer).

The first step of DPA takes $O(B') = O(\epsilon n)$ running time, since these observations are at fixed locations with uniform step size. For the second step, using batch budget allocation, we need to compute the RB curve for each interval up to its maximum budget, which is in $O(\frac{1}{\epsilon})$. Following the asymptotic result of (9), the running time to compute these curves is

$$O(M \cdot (\frac{1}{\epsilon})^2 (\frac{1}{\epsilon})^3) = O(\epsilon n \cdot (\frac{1}{\epsilon})^5) = O(\frac{1}{\epsilon^4} n). \tag{21}$$

Since producing the sorted list and picking the reward in the top positions takes $O(nlog(n))$ time, the total running time is

$$O(\frac{1}{\epsilon^4} n + nlog(n)). \tag{22}$$

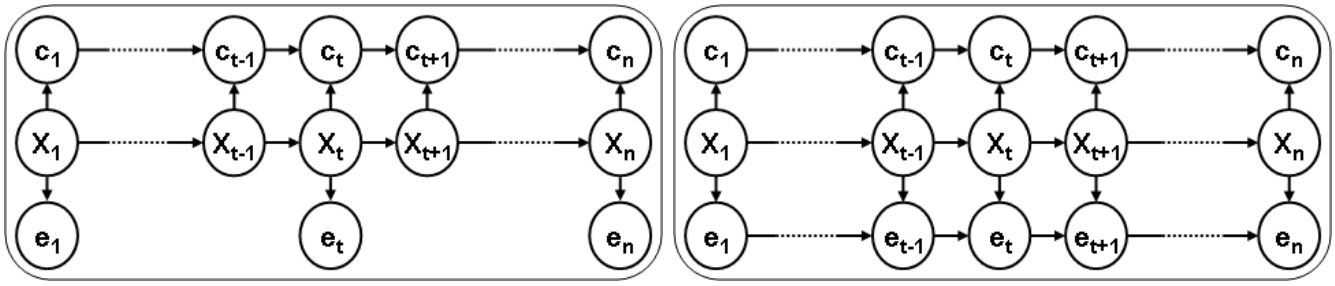
Using incremental budget allocation, we only need to compute the RB curve for each interval up to its allocated budget. Thus, the complexity is

$$O(\sum_{j=1}^{M} (\tilde{k}_j^2 (\frac{1}{\epsilon})^3)) = O(\frac{1}{\epsilon^3} \sum_{j=1}^{M} \tilde{k}_j^2), \tag{23}$$

where $\tilde{k}_j$ is the allocated budget for interval j. Because $\sum_{j=1}^{M} \tilde{k}_j = B''$ and $\tilde{k}_j \leq \frac{1}{\epsilon}$ for all j, $\sum_{j=1}^{M} \tilde{k}_j^2$ reaches its maximum by letting as many intervals have budget $\frac{1}{\epsilon}$ as possible. For each of the other intervals, it either has 0 budget or has the remaining of the total budget. As a result, the number of intervals that have nonzero budget is $\lceil B''/\frac{1}{\epsilon} \rceil = O(B''/\frac{1}{\epsilon}) = O(\epsilon B'')$. Thus, we further conclude that

$$O(\frac{1}{\epsilon^3} \sum_{j=1}^{M} \tilde{k}_j^2) = O(\frac{1}{\epsilon^3} \cdot \epsilon B'' \cdot (\frac{1}{\epsilon})^2) = O(\frac{1}{\epsilon^4} B'') \tag{24}$$

In addition, we must also maintain a sorted list of the incremental rewards from computing RB curves for each interval as budget increases. Using data structures such as a binary heap [28], we can build an



(a) (b)

Fig. 3. Markov models for video frame sequences. (a) The model we use when expensive feature is not available at every frame; (b) The model we use when all frames have expensive features.

initial sorted list in $O(Mlog(M)) = O(B'log(B'))$ time, and then add and remove B'' increments to that list in $O(B''log(B'))$ time. This gives a total running time of:

$$O(\frac{1}{\epsilon^4}B'' + (B' + B'')log(B') = O(\frac{1}{\epsilon^4}B + Blog(B)). \tag{25}$$

Finally, the last step involves using the optimal algorithm for each interval to determine the additional sampling locations. This takes $O(B'')$ running time given dynamic programming memory tables for each interval in the budget allocation stage. However, storing all the tables for each interval require a large amount of memory space. Alternatively, we can rerun the optimal algorithm for each interval using the allocated budget, which gives an additional run time of $O(\frac{1}{\epsilon^4}n)$ for batch allocation, and $O(\frac{1}{\epsilon^4}B'')$ for incremental allocation. Finally, the total running time is

$$O(\epsilon n + \frac{1}{\epsilon^4}n + nlog(n) + \frac{1}{\epsilon^4}n) = O(\epsilon n + \frac{2}{\epsilon^4}n + nlog(n)) = O(\frac{1}{\epsilon^4}n + nlog(n)) \tag{26}$$

for batch budget allocation, and is

$$O(B' + \frac{1}{\epsilon^4}B + Blog(B) + \frac{1}{\epsilon^4}B) = O(B + \frac{2}{\epsilon^4}B + Blog(B)) = O(\frac{1}{\epsilon^4}B + Blog(B)) \tag{27}$$

for incremental budget allocation. This is a vast improvement over $O(B^2n^3)$ for the optimal algorithm. In our experiments, we show that this computation is quite practical.

IV. Data Fusion for frame sequences

In this section, we first describe how we model the frame sequence from a video as a graphical model for data fusion in section IV-A. This model is a Markov chain which emits cheap and expensive features. Therefore, we can apply DPA to this model and section IV-B describes this. Finally, we discuss on how to run the forward-backward algorithm for this model in section IV-C.

A. A Graphical Model for Frame Sequences

The graphical model is a Markov model, where each frame of video corresponds to a node. Each node contains a state variable that represents the property we wish to infer, such as whether a face, or a moving object, is present. Each node can emit two observable quantities, corresponding to cheap and expensive features extracted from the frame. This is similar to a hidden Markov model (HMM), but here we also model dependencies between observations. In our model, the value of a cheap feature at time t is not conditionally independent of the rest of the model given the state at time t, but is also dependent on the cheap feature at times $t-1$ and $t+1$. We do this to capture the fact that when a feature makes an error in one frame, it is quite likely to make a similar error at an adjacent frame. (Note if we assume conditional

independence among features at different time steps, it results in less accurate inference performance). This model can be considered a type of autoregressive hidden Markov model [29], or conditional random field [30].

However, we typically have expensive features for a small fraction of frames, so it is not trivial to model the dependency between them. In addition, we assume the expensive feature is accurate when predicting the state. Therefore, we do not model the dependency between expensive features, and make it only depend on the state. However, in the case that we have an expensive feature at every frame, we model the dependency the same as with cheap features. We illustrate these two cases in Figure 3. State variables are labeled "X", cheap observations are labeled "c", and expensive observations are labeled "e". They all have numbered subscripts indicating their time steps.

B. Applying DPA

We have described DPA for the case of simple, chain graphical models. However, it is straightforward to apply it to the model in previous section, since it has a chain structure and the same conditional independence property. Specifically, to apply DPA to our model we first treat the chain formed by the state variables in our model as the chain graphical model. Second, assuming that the expensive feature is accurate in predicting the state, we consider the places to reveal states in the chain as the locations to sample expensive feature. And we use the predicted states to compute the optimal reward described in section. Finally, cheap feature also provides useful information. Thus, we make the recursive formulas for computing the optimal reward be conditioned on applicable cheap features. Denoting $c_{a:b}$ as the cheap feature over the interval from variable X_a and X_b, the formula (7) and (8) for computing the optimal reward become

$$J_{a:b}(x_a, x_b; k) = \max\{J_{a:b}(x_a, x_b; 0), \max_{a<j<b}\{\sum_{x_j} P(X_j = x_j | X_a = x_a, X_b = x_b, c_{a:b})\{ \\ R_j(X_j | X_j = x_j, c_{a:b}) + \max_{0\le l\le k-1}[J_{a:j}(x_a, x_j; l) + J_{j:b}(x_j, x_b; k-l-1)]\}\}\}, \tag{28}$$

where the base case is

$$J_{a:b}(x_a, x_b; 0) = \sum_{j=a+1}^{b-1} R_j(X_j | X_a = x_a, X_b = x_b, c_{a:b}). \tag{29}$$

C. Forward-backward Algorithm

When using formula (28) and (29) to determine where to sample expensive features, and using our model to do inference to predict the state of each frame, we need to determine the probability distribution of each state based on observations. We can use the Forward-Backward algorithm [31] to do this. Using our model, this algorithm works a little differently from the standard version, due to dependencies in the observations, and we describe it below. The derivation uses the ideas in [29] and [32].

Let N be the number of states, and denote individual states as $S = \{s_1, s_2, \cdots, s_N\}$. Let the sequence have T time slices. Also let cheap feature observations from time 1 to T be $c_1, c_2, \cdots, c_T$ respectively, and expensive feature observations from time 1 to T be $e_1, e_2, \cdots, e_T$. Let f_i be $\{c_i, e_i\}$ at time i and O be all the observations. Similarly, we use X_t to denote the state variable at time t. Then,

$$\begin{aligned} P(X_t = s_i | O) &= \frac{P(O, X_t = s_i)}{P(O)} = \frac{P(f_{1\cdots T}, X_t = s_i)}{P(O)} \\ &= \frac{P(f_{t+1\cdots T} | X_t = s_i, f_{1\cdots t}) P(X_t = s_i, f_{1\cdots t})}{P(O)} \\ &\propto P(f_{t+1\cdots T} | X_t = s_i, f_t) P(X_t = s_i, f_{1\cdots t}). \end{aligned} \tag{30}$$

Given the forward variable defined as $\alpha_t(i) = P(X_t = s_i, f_{1\cdots t})$ and the backward variable as $\beta_t(i) = P(f_{t+1\cdots T}|X_t = s_i, f_t)$, we have

$$P(X_t = s_i|O) \propto \alpha_t(i) \cdot \beta_t(i). \tag{31}$$

To compute $\alpha_t(i)$, we have

$$\begin{aligned}
\alpha_t(i) &= P(X_t = s_i, f_{1\cdots t}) = P(X_t = s_i, f_t, f_{1\cdots t-1}) \\
&= P(X_t = s_i, f_t|f_{1\cdots t-1})P(f_{1\cdots t-1}) \\
&= \sum_{j=1}^{N} P(X_t = s_i, f_t|X_{t-1} = s_j, f_{1\cdots t-1})P(X_{t-1} = s_j|f_{1\cdots t-1})P(f_{1\cdots t-1}) \\
&= \sum_{j=1}^{N} P(f_t|X_{t-1} = s_j, f_{1\cdots t-1}, X_t = s_i)P(X_t = s_i|X_{t-1} = s_j, f_{1\cdots t-1})P(X_{t-1} = s_j, f_{1\cdots t-1}) \\
&= \sum_{j=1}^{N} P(f_t|f_{1\cdots t-1}, X_t = s_i)P(X_t = s_i|X_{t-1} = s_j)P(X_{t-1} = s_j, f_{1\cdots t-1}) \\
&= \sum_{j=1}^{N} P(f_t|f_{1\cdots t-1}, X_t = s_i)P(X_t = s_i|X_{t-1} = s_j)\alpha_{t-1}(j),
\end{aligned} \tag{32}$$

where $P(f_t|f_{1\cdots t-1}, X_t = s_i) = P(c_t|c_{t-1}, X_t = s_i)P(e_t|X_t = s_i)$ for the left model and $P(f_t|f_{1\cdots t-1}, X_t = s_i) = P(c_t|c_{t-1}, X_t = s_i)P(e_t|e_{t-1}, X_t = s_i)$ for the right model in Fig. 3. This gives the recursive formula to compute the forward variable, where the base condition is the same as that in [31]. The only difference from the standard forward algorithm is that the probability of the observation needs to be conditioned on the observation in the previous time step.

To compute $\beta_t(i)$, we have

$$\begin{aligned}
\beta_t(i) &= P(f_{t+1\cdots T}|X_t = s_i, f_t) = \sum_{j=1}^{N} P(f_{t+2\cdots T}, X_{t+1} = s_j, f_{t+1}|X_t = s_i, f_t) \\
&= \sum_{j=1}^{N} P(f_{t+2\cdots T}|X_{t+1} = s_j, f_{t+1}, X_t = i, f_t)P(X_{t+1} = s_j, f_{t+1}|X_t = s_i, f_t) \\
&= \sum_{j=1}^{N} P(f_{t+2\cdots T}|X_{t+1} = s_j, f_{t+1})P(f_{t+1}|X_{t+1} = s_j, f_t)P(X_{t+1} = s_j|X_t = s_i) \\
&= \sum_{j=1}^{N} \beta_{t+1}(j)P(X_{t+1} = s_j|X_t = s_i)P(f_{t+1}|X_{t+1} = s_j, f_t),
\end{aligned} \tag{33}$$

where $P(f_{t+1}|X_{t+1} = s_j, f_t) = P(c_{t+1}|X_{t+1} = s_j, c_t)P(e_{t+1}|X_{t+1} = s_j)$ for the left model and $P(f_{t+1}|X_{t+1} = s_j, f_t) = P(c_{t+1}|X_{t+1} = s_j, c_t)P(e_{t+1}|X_{t+1} = s_j, e_t)$ for the right model in Fig. 3. Again, the only difference from the standard backward algorithm is that probability of the observation is conditioned on the observation in the previous time step.

V. Experiments

We now apply DPA to two vision tasks involving motion detection and face detection. Our main goal is to show that inference can be used to efficiently allocate processing in two very different tasks. We begin by first describing some common characteristics of our experiments in the next section. We then present the results for the two tasks in section V-B and V-C. Section V-D and V-E discuss how the expensive and cheap features can affect DPA. Finally, section V-F discusses the running time.

A. General Experiment Setup

For DPA, given a video with n frames and a budget of $B = B' + B''$, we first use B' budget to run the expensive algorithm uniformly with a step size of 20, while also running the cheap algorithm on all frames. Then we use the batch budget allocation to distribute the remaining budget B'' among intervals, which are separated by frames whose state is predicted by the expensive feature. This step gives us a budget for each interval, and we use the optimal algorithm to determine where to observe expensive features within each interval. For convenience of description, we describe budget in term of its percentage of the total number of frames.

One assumption in this budget allocation method is that the reward curve as the budget increases is convex for the subsections. To evaluate this assumption, we compute the RB curve for each interval up to its maximum budget. For those curves which are not strictly convex, the difference between two consecutive reward increments must be positive at some points. We pick the highest such value as the nonconvexity measure of a RB curve, and find that this measure is typically very small. So we approximate each nonconvex RB curve with a convex curve, which yields a good approximation. Section V-B and V-C give more details on the nonconvexity measure.

When running DPA, we also assume that the expensive algorithm always correctly reveals the state of a frame. This is not really true, so we expect that occasional errors made by an expensive algorithm will cause us to allocate resources suboptimally. However, since our inferences will in general be correct, we still expect DPA to produce decisions that are effective overall.

Finally, when all observations have been made, we predict the state of each frame using the inference model in Fig. 3(a) since expensive features are not available in every frame. We compared our results to several baseline algorithms. Competing algorithms are always provided the same total budget for a fair comparison.

- The first baseline method is *uniform sampling*, which runs the expensive algorithm at a uniform step size. This method is in essence equivalent to running the expensive algorithm at a lower frame rate.
- The second baseline method is *most-relevant sampling*. In this sampling method, we first run the cheap algorithm at each frame and perform inference using the model in Fig. 3(a) to obtain the conditional probabilities of the state variables at each frame. We then run the expensive algorithm on the frames that are most likely to satisfy our query. This is equivalent to using the cheap algorithm to prune the least interesting frames.
- The third and last baseline method is *most-uncertain sampling*. Similar to the most-relevant sampling method, we again run the cheap algorithm at each frame and then perform inference to obtain conditional probabilities. Then, we run the expensive algorithm on frames that have the greatest uncertainty. Uncertainty can be measured in many ways. For example, we can use entropy of the conditional probability, or we can use 1 - maximum conditional probability. For binary classification tasks such as ours here, both choices produce an identical ranking. The motivation behind this sampling strategy is that we want to invest resources at locations we are most uncertain about.
- Finally, to calibrate the performance of algorithms on different tasks, we compared to an idealized method, *ceiling sampling*, in which we run the cheap and expensive algorithms at all the frames, i.e, the budget is equal to 100%. We use Fig. 3(b) to model the frame sequence because the expensive feature is available everywhere. This method should provide an upper bound on the performance of any method given that the model fits the data well enough.

To compare these methods, we used 4-fold cross validation on each video, using three-quarters of the data to train the model and one quarter to evaluate the model. We recorded each video in 30 frame per second. We then uniformly sampled 3 frames per second to generate the training and testing sequences, and this is a reasonable frame rate to experiment on for real world surveillance videos. By beginning sampling at different locations, we produced 10 different sequences for training and also for testing. All ten sequences are used as training data. We also use all ten for testing, helping to smooth the results a bit. The performance measure we used was the 11-point average precision of the precision recall (PR) curves

[33]. That is, we take the average precision for 11 uniformly spaced levels of recall. This is averaged over all 40 testing sequences from the 4 folds.

For the first three sampling methods, we ran the cheap algorithm at each frame and ran the expensive algorithm only at the sampled locations. For ceiling sampling, we ran the expensive algorithm on all frames. We combined the cheap and expensive features and performed inference using the models illustrated in Fig. 3. We used the conditional probabilities for each frame's state variable to generate PR curves. The methods differed only in their choice of where to apply the expensive algorithm, and use the same algorithm to do inference. To compare the sampling methods under varying conditions, we varied the total budget from 5% to 25% of n. We next provide more details about the experiments for both tasks.

B. Motion Detection

We first evaluated these algorithms in a simple background subtraction task. We collected three half hour videos at thirty frames per second, for a total of $n \approx 55,000$ per video, with each frame at 240×320 resolution. We hand-labeled each frame as "interesting" if it contained a moving object, such as a person or car, "uninteresting" otherwise.

As a cheap algorithm, we used FD [4] and we used IAGMM [5] as the expensive algorithm. Using the first algorithm, the feature was the number of foreground pixels in a frame after applying a threshold of 10. This avoided postprocessing, saving a significant amount of computational resources. It is an interesting question for future work to determine how best to build a background model suitable for the expensive algorithm based on sparsely sampled frames. However, in this experiment we wish to focus on the effectiveness of our algorithm in directing application of IAGMM. Therefore, we build a background model using all recent frames and then apply IAGMM only at sampled locations. After applying the IAGMM, we performed an opening and a closing morphological operation. We then extracted the largest connected component to generate features, which were discretized to run the experiment. We had used 8 different features, which included the area of the component, the width of its Bounding Box, and the diameter of a circle with the same area as the component. They all produced similar performance, and we chose the area of the component to show results. Figure 4 shows some frame examples and the corresponding output by FD and IAGMM algorithms.

Next, we tested our convexity assumption on these videos, since DPA assumes that the reward curves were convex. Table I shows the results. We can see that over a half of all intervals produce convex reward curves, while most of the non-convex ones have very small concavities.

TABLE I

CONVEXITY OF RB CURVES FOR THE MOTION DETECTION TASKS

Video	**Video 1**	**Video 2**	**Video 3**
Number of intervals[b]	2720	2720	2720
Convex (%)	56.21	67.39	59.93
Convex or $\approx$ convex[b](%)	94.34	97.28	94.45
Median of the rest[c]	0.0468	0.0181	0.0834

[a] The total number of intervals over the 40 testing sequences.
[b] RB curves which are not convex but with nonconvexity measure not greater than 0.01.
[c] The median of nonconvexity measure for those intervals which are neither convex or $\approx$ convex.

We show 11-point average precision results on the three videos in Fig. 5. In all three videos, our method outperforms the baseline methods. We also observe from the plots that uniform sampling outperforms both most-relevant sampling and most-uncertain sampling. We postulate that the reason may be that the cheap algorithm does not produce high quality features and so decisions based purely on the cheap algorithm are unreliable. In these videos, FD faces difficulties because leaves often move in the background.

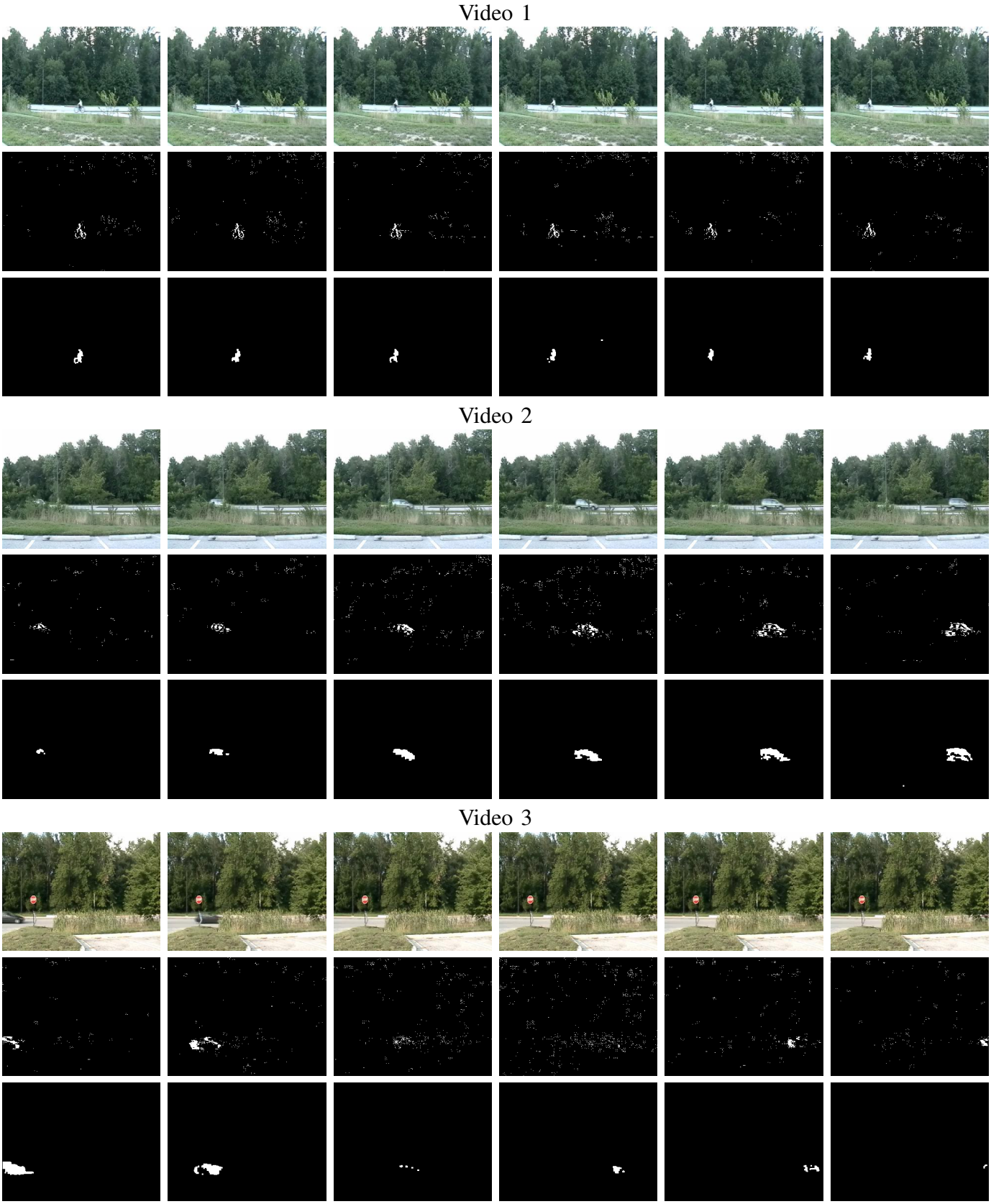


Fig. 4. Frame examples from video 1, 2, and 3 for the motion detection task and their corresponding output from the cheap and expensive algorithms. Within each video, the first row shows the original frame, the second and the third rows show the corresponding output from FD and IAGMM respectively. FD tends to produce more noise for foreground pixels which affects accuracy of features. IAGMM gives much fewer false positives, and in general produce better feature for detection.

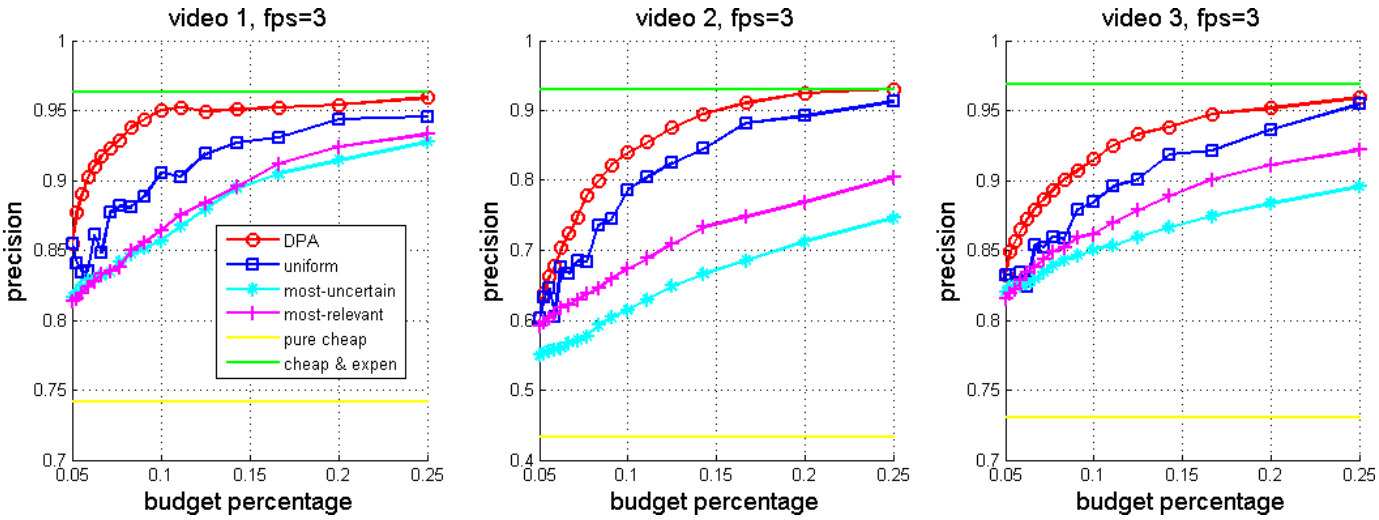


Fig. 5. 11-point average precision values for the background subtraction task.

C. Face Detection

Next, we applied our approach to the problem of identifying frames containing a face. As with the last task, we collected three half-hour videos. We hand-labeled each frame as "interesting" if there is a frontal or profile face in it and labeled it as "uninteresting" otherwise.

For the cheap algorithm, we used IAGMM with the area of the largest connected component as a feature since it is relatively good at detecting the motion of a human and is still computationally cheap compared to the face detector. The expensive algorithm was the face detection algorithm based on OpenCV [34], using the scheme in [9]. We used both frontal and profile face detectors and the expensive feature was a binary indicator of whether the detectors found a face. Fig. 6 shows frame examples from these three videos.

TABLE II

CONVEXITY OF RB CURVES FOR THE FACE DETECTION TASK.

Video	**video 4**	**video 5**	**video 6**
Number of subsections	2720	2840	2800
Convex (%)	80.22	66.16	68.61
Convex or $\approx$ convex (%)	98.16	98.45	99.07
Rest median	0.0970	0.1014	0.1014

We again first measured the convexity of the reward curves for the face detection videos. The results are given in Table II. Again, the reward curves are largely convex. We show the 11-point average precision results on the three videos in Figure 7. Our method outperforms the baseline methods in two videos, video 4 and 5, under all budget percentages. However, in video 6, our method has no advantage over uniform sampling when the budget is small, but as the budget increases, the advantage of our method becomes clear.

D. Accuracy of Expensive Features

The accuracy of the expensive feature can cause variations in performance of DPA, since our decisions are based on the assumption that the expensive feature is very accurate. To see this, we measured the prediction error rates of the expensive feature on all six videos. Remember that our method is guaranteed to be optimal when the expensive feature determines the correct state perfectly, and our graphical model

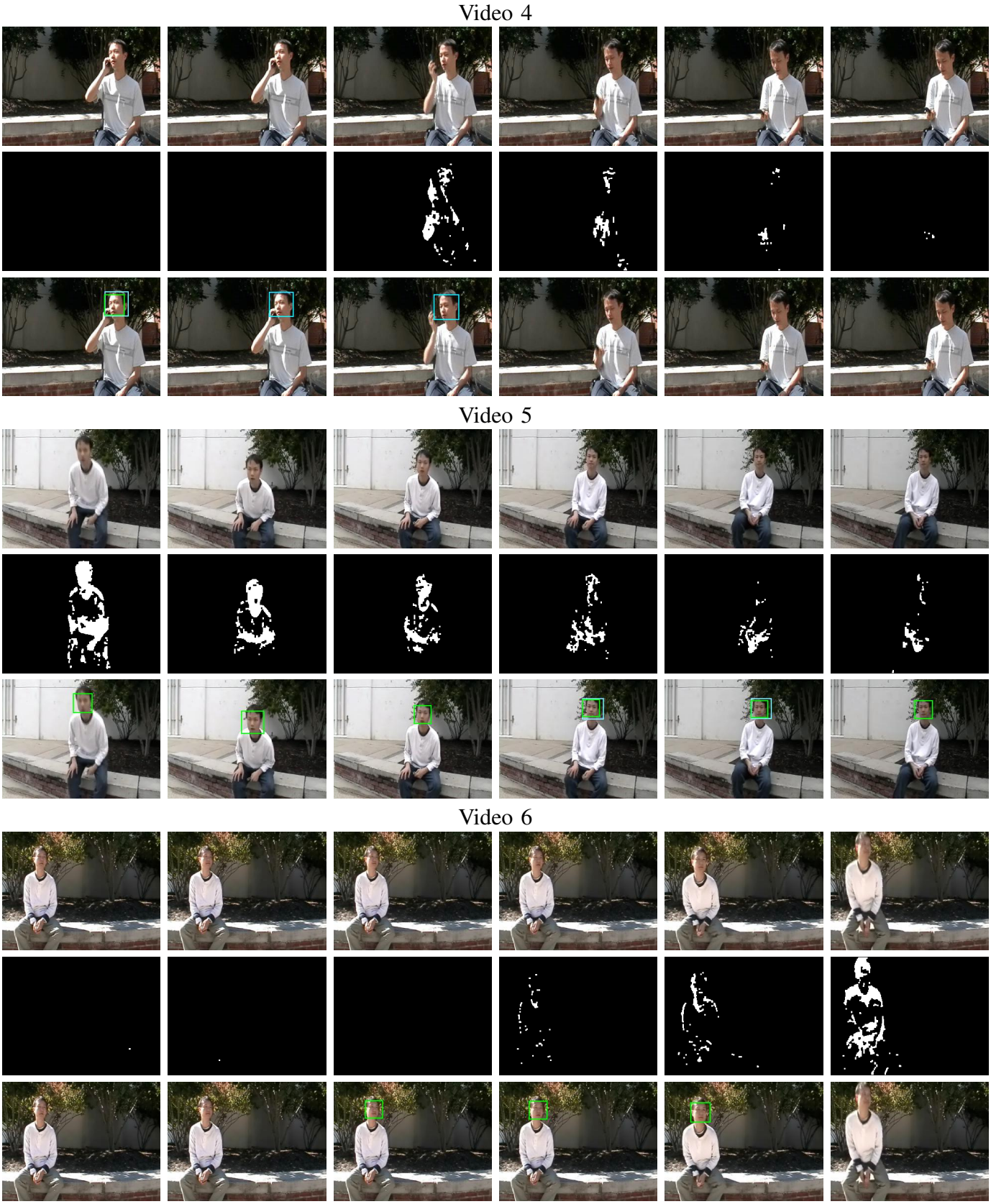


Fig. 6. Example frames from video 4, 5, and 6 for the face detection task and their corresponding output from the cheap and expensive algorithms. Within each video, the first row shows the original frame, the second and the third rows show the corresponding output from IAGMM and face detectors respectively. IAGMM produces information about whether moving objects, such as humans, are present or not, while face detectors give explicit knowledge about presence of faces.

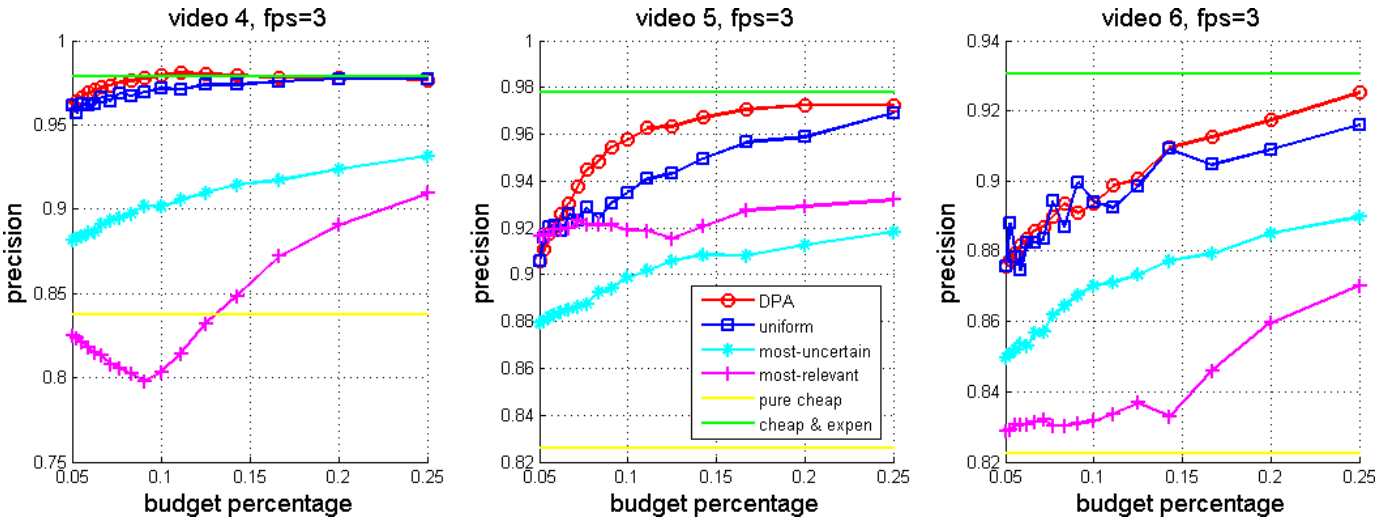


Fig. 7. 11-point average precision values for the face detection task.

correctly models the domain. The prediction error rates of the expensive feature in the six videos are .0153, .0421, .0395, .0587, .0830, and .2328 respectively. Note that the error rate is highest in the sixth video, the video on which our method has no clear advantage over uniform sampling when given a small budget.

E. Usefulness of Cheap Features

Ideally, combining both features to guide DPA to determine the expensive feature sampling locations should give better result than purely using expensive feature. However, we observe that when the model does not fit the dependency properties of the domain, combining both features can actually hurt performance of DPA. To show this, we replace the cheap feature with a constant synthetic feature. Under this setting, the sampling locations and inference only depends on the expensive features. We compare performance of these two cases. Following the similar setup above, we observe that combining both features has a clear advantage over only using the expensive feature. Fig. 8 shows the result.

However, we do observe that when we vary the frame rates, purely using expensive features can have better performance than combining both features. Fig. 9 shows the performance when the frame rate is 30, 6, 2, and 1. When the frame rate is 30 or 6 frame per second, combining both features has no advantage over using purely expensive features. In particular, using purely expensive features has better performance for the first three videos for background subtraction task under 30 frames per second. However, when the frame rate is 2 or 1 frame per second, the advantage of combining both feature becomes clear again. This is very likely due to the issue that how well our model fits the data. When the frame rate becomes higher and higher, the dependency between the cheap features is stronger and stronger, and our model cannot capture this well enough. As a result, the cheap feature is overconfident in some locations, and suppresses the correct decisions by the expensive features. Using a better model that captures this dependency adequately is one way to deal with this problem, however it adds significant computational complexity. Furthermore, real world videos, such as surveillance videos, need to run for a long time in general. A small frame rate is more suitable for storage and power issues [35]. Combining both features is good in this case. At the same time, we stress that in situations in which cheap features are not helpful, DPA will still provide significant benefits, using only expensive features to control processing.

F. Running Time

Finally, we discuss the processing overhead of DPA. Without considering the feature generation time, which is application dependent, the main processing time of DPA is at the stage of determining expensive

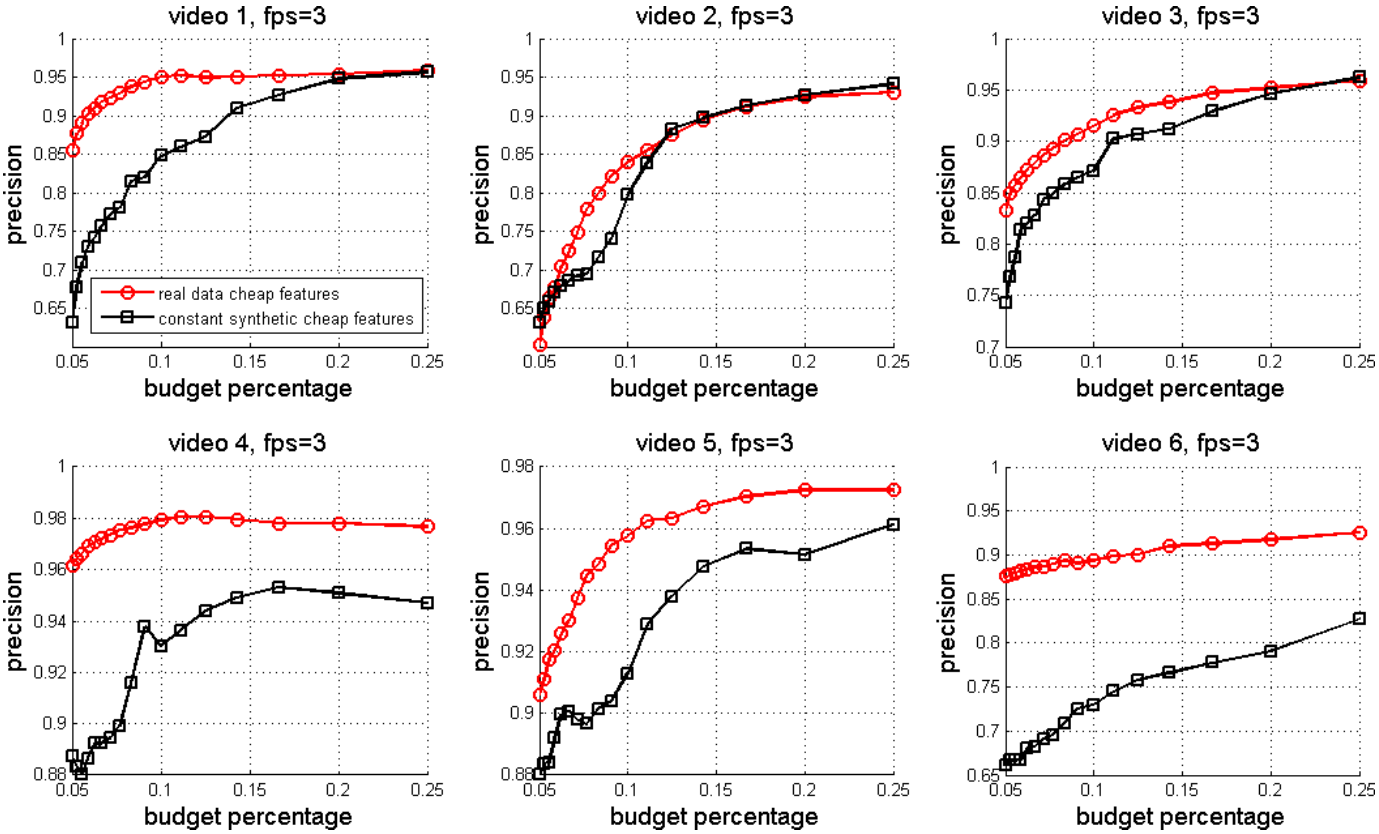


Fig. 8. Comparison of using cheap features from real data and constant synthetic cheap features under frame rate of 3 frame per second.

features sampling locations. With unoptimized MATLAB and Java code and without considering the feature generation time, DPA requires an average of 7.6 seconds to handle one-fourth of a half-hour video when the budget is 5% at 3 frames per second. And it needs about 9.7 seconds when the budget is 25%. This overhead is small compared with the time to record the video. However, when the frame rate is 30 frames per second, the average running time becomes 76.3 seconds when the budget is 5% and 97.0 seconds when the budget is 25%. Both the length of a testing sequence and the running time in the later case is 10 times longer than those in former case. With a fixed interval size, the running time grows linearly as the testing frame sequence grows. However, we see that the computational overhead is relatively small compared with the video recording time. For real world applications, in which the frame rate needs to be small to save computational resource, DPA will in particular be useful.

VI. Conclusion

Our main goal has been to design inference algorithms that can be used to direct video processing. This allows us to replace simplistic methods such as reducing the frame rate with principled decisions that carry theoretical performance guarantees. We believe that this is a quite general framework that can be applied to many video processing tasks and may be extended in the future to more complex graphical structures.

To this end, we have made two more detailed contributions. First, we propose a graphical model that maps onto a video frame sequence and allows us to combine features from expensive and cheap algorithms to do inference. We show that in practical situations, there is much to be gained by this combination. Second, we have shown how to build on an existing algorithm that was designed for short chains to create an algorithm that runs efficiently on long video sequences. Specifically, we show that by applying an expensive algorithm in some extra locations, we can control additional applications of the algorithm

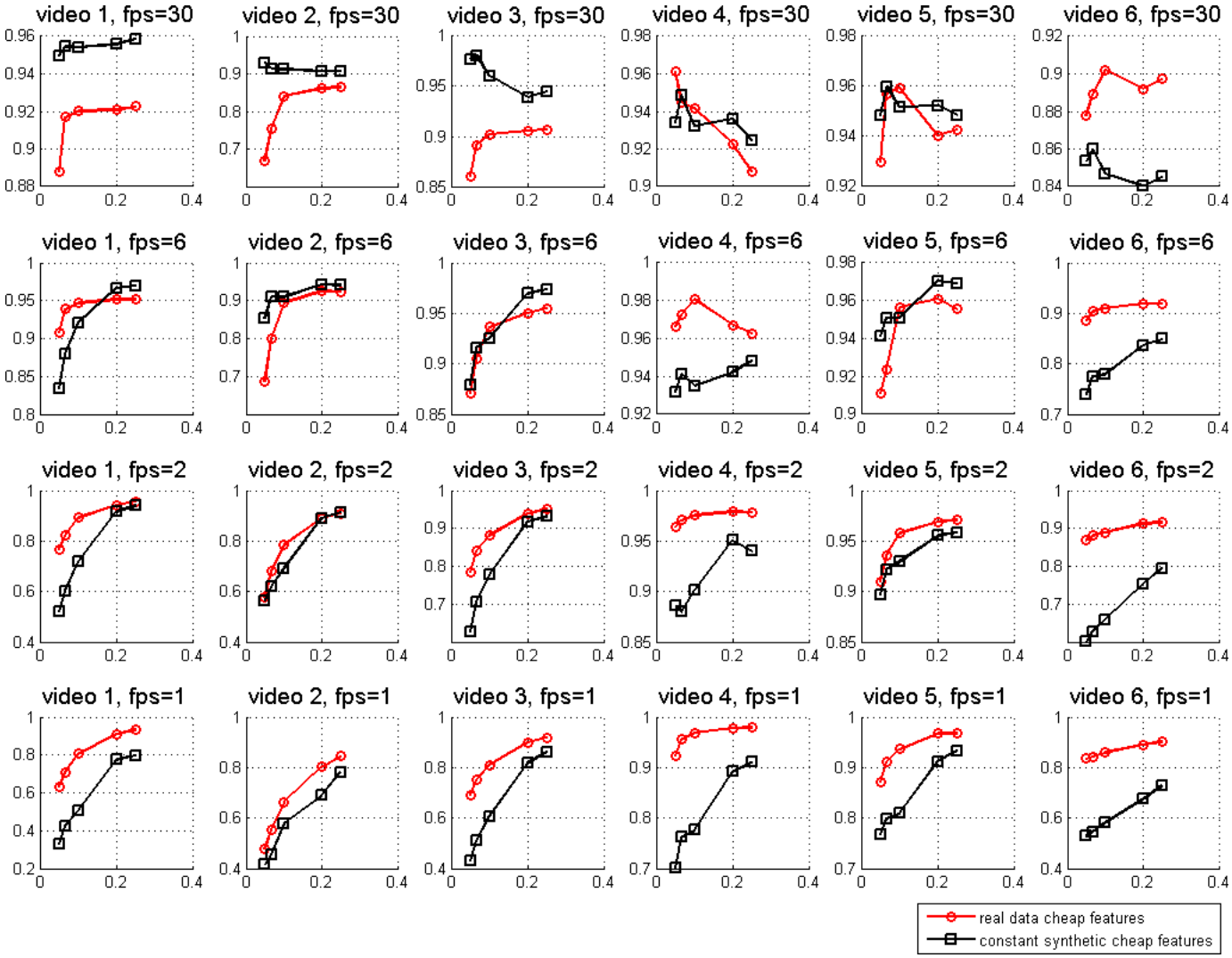


Fig. 9. Comparison of using cheap features from real data and constant synthetic cheap features under frame rate of 30, 15, 5, and 1 frame per second. From top to bottom, each row shows the results for all videos under a frame rate, from 30 to 1 frame per second.

with significantly smaller running time. Experiments with two concrete video processing tasks, low-level background subtraction and the higher level task of face detection, show that these can be mapped onto our framework. The effectiveness of DPA's inference algorithm in these tasks illustrates the potential of our approach for general video processing.

REFERENCES

[1] L. Telindus Surveillance Systems, "http://www.telindussurveillance-us.com/," 2005.
[2] N. Dean, "Bombers staged dry run before london attacks," The Independent, online edition, 20 Sept. 2005.
[3] A. Krause and C. Guestrin, "Optimal nonmyopic value of information in graphical models - efficient algorithms and theoretical limits," in *International Joint Conference on Artificial Intelligence*, 2005.
[4] R. Jain and H. Nagel, "On the analysis of accumulative difference pictures from image sequences of real world scenes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 1, pp. 206–213, 1979.
[5] Z. Zivkovic, "Improved adaptive gaussian mixture model for background subtraction," in *International Conference on Pattern Recognition*, 2004.
[6] C. Stauffer and W. Grimson, "Adaptive background mixture models for real-time tracking," *IEEE Conference on Computer Vision and Pattern Recognition*, 1999.
[7] M. hsuan Yang, D. J. Kriegman, S. Member, and N. Ahuja, "Detecting faces in images: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 34–58, 2002.
[8] P. Viola and M. Jones, "Robust real-time object detection," *International Journal of Computer Vision*, vol. 57, pp. 137–154, 2002.

[9] R. Lienhart and J. Maydt, "An extended set of haar-like features for rapid object detection," in *IEEE International Conference on Image Processing*, 2002.

[10] R. Krishna, K. McCusker, and N. O'Connor, "Optimising resource allocation for background modeling using algorithm switching," in *ACM/IEEE International Conference on Distributed Smart Cameras*, 2008.

[11] S. Barotti, L. Lombardi, and P. Lombardi, "Multi-module switching and fusion for robust video surveillance," in *International Conference on Image Analysis and Processing*, 2003.

[12] R. Yang and M. Pollefeys, "Multi-resolution real-time stereo on commodity graphics hardware," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2003.

[13] C. Zach, K. Karner, and H. Bischof, "Hierarchical disparity estimation with programmable 3d hardware," in *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2004.

[14] J. Woetzel and R. Koch, "Real-time multi-stereo depth estimation on GPU with approximative discontinuity handling," in *European Conference on Visual Media Production*, 2004.

[15] R. Yang and G. Welch, "Fast image segmentation and smoothing using commodity graphics hardware," *Journal of Graphics Tools*, vol. 7, pp. 91–100, 2002.

[16] A. Griesser, S. D. Roeck, A. Neubeck, and L. V. Gool, "GPU-based foreground-background segmentation using an extended colinearity criterion," in *International Fall Workshop on Vision, Modeling, and Visualization*, 2005.

[17] S. Heymann, K. Maller, A. Smolic, B. Froehlich, and T. Wiegand, "SIFT implementation and optimization for general-purpose GPU," in *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2007.

[18] S. N. Sinha, J. michael Frahm, M. Pollefeys, and Y. Genc, "GPU-based video feature tracking and matching," in *Workshop on Edge Computing Using New Commodity Architectures*, 2006.

[19] J. Fung and S. Mann, "OpenVIDIA: parallel GPU computer vision," in *Annual ACM International Conference on Multimedia*, 2005.

[20] C. Cabani and W. J. MacLean, "Implementation of an affine-covariant feature detector in field-programmable gate arrays," in *International Conference on Computer Vision Systems*, 2007.

[21] S. Se, T. Barfoot, and P. Jasiobedzki, "Visual motion estimation and terrain modeling for planetary rovers," in *International Symposium on Artificial Intelligence for Robotics and Automation in Space*, 2005.

[22] F. J. Seinstra, J.-M. Geusebroek, D. Koelma, C. G. Snoek, M. Worring, and A. W. Smeulders, "High-performance distributed image and video content analysis with parallel-horus," *IEEE Multimedia*, vol. 14, pp. 64–75, 2007.

[23] V. Bayer-Zubek, "Learning diagnostic policies from examples by systematic search," in *Conference on Uncertainty in Artificial Intelligence*, 2004.

[24] P. D. Turney, "Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm," *Journal of Artificial Intelligence Research*, vol. 2, pp. 369–409, 1995.

[25] M. Rattigan, M. Maier, and D. Jensen, "Exploiting network structure for active inference in collective classification," in *ICDM Workshop on Mining Graphs and Complex Structures*, 2007.

[26] M. Bilgic and L. Getoor, "Effective label acquisition for collective classification," in *International Conference on Knowledge Discovery and Data mining*, 2008.

[27] A. Krause, B. McMahan, C. Guestrin, and A. Gupta, "Robust submodular observation selection," *Journal of Machine Learning Research*, vol. 9, pp. 2761–2801, 2008.

[28] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*. Cambridge, MA, USA: The MIT Press, 2009.

[29] K. P. Murphy, "Dynamic bayesian networks: Representation, inference and learning," Ph.D. dissertation, UC Berkeley, Computer Science Division, 2002.

[30] C. Sutton and A. McCallum, "An introduction to conditional random fields for relational learning," in *Introduction to Statistical Relational Learning*, L. Getoor and B. Taskar, Eds. Cambridge, MA, USA: MIT Press, 2007, pp. 93–127.

[31] L. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, pp. 257–286, 1989.

[32] T. Fletcher, "Switching autoregressive hidden markov model," Department of Computer Science, University College London, Tech. Rep., 2009.

[33] C. D. Manning, P. Raghavan, and H. Schtze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[34] Intel, "Opencv open source computer vision library," http://www.intel.com/technology/computing/opencv/index.htm.

[35] C. Loy, T. Xiang, and S. Gong, "Multi-camera activity correlation analysis," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.