Exploratory Search of Long Surveillance Videos

Greg Castañón and Venkatesh Saligrama
Boston University
Boston, MA, USA
gdc1@bu.edu, srv@bu.edu

André Louis Caron and Pierre-Marc Jodoin
Université de Sherbrooke
Sherbrooke, Quebec, Canada
andre.louis.caron@usherbrooke.ca,
pierre-marc.jodoin@usherbrooke.ca

ABSTRACT

We present a fast and flexible content-based retrieval method for surveillance video. Designing a video search robust to uncertain activity duration, high variability in object shapes and scene content is challenging. We propose a two-step approach to video search. First, local features are inserted into an inverted index using locality-sensitive hashing (LSH). Second, we utilize a novel dynamic programming (DP) approach to robustify against temporal distortion, limited obscuration and imperfect queries. DP exploits causality to assemble the local features stored in the index into a video segment which matches the query video. Pre-processing of archival video is performed in real-time, and retrieval speed scales as a function of the number of matches rather than video length. We derive bounds on the rate of false positives, demonstrate the effectiveness of the approach for counting, motion pattern recognition and abandoned object applications using seven challenging video datasets and compare with existing work.[1]

Categories and Subject Descriptors

I.4.8 [**Computing Methodologies**]: Image Processing and Computer Vision—*Scene Analysis*; I.5.4 [**Computing Methodologies**]: Pattern Recognition—*Applications*

General Terms

Algorithms

Keywords

Exploratory search, Video search, Dynamic programming, Surveillance

1. INTRODUCTION

Surveillance video camera networks are increasingly ubiquitous, providing pervasive information gathering capabilities. In many applications, surveillance video archives are used for forensic purposes to gather evidence after the fact. This calls for content-based retrieval of video data matching user defined queries with robustness to clutter and distortion.

Exploratory search [22] is a specialization of information-seeking developed to address such situations. It describes the activity of attempting to obtain information through a combination of querying and collection browsing. Development of exploratory search systems requires addressing two critical aspects: video browsing and content-based retrieval. The focus of this paper is the latter. We present a method for fast content-based retrieval adapted to characteristics of surveillance videos. The most challenging of these are:

1.) Data lifetime: since video is constantly streamed, there is a perpetual renewal of video data. This calls for a model that can be updated incrementally as video data is made available. The model must also scale well with the temporal mass of the video.

2.) Unpredictable queries: the nature of queries depends on the field of view of the camera, the scene itself and the type of events being observed. The system should support queries of different nature that can retrieve both recurrent events such as people entering a store and infrequent events such as abandoned objects and cars performing U-turns.

3.) Unpredictable event duration: events are unstructured. They start anytime, vary in length, and overlap with other events. The system is nonetheless expected to return complete events regardless of their duration and whether or not other events occur simultaneously.

4.) Clutter and occlusions: Tracking and tagging objects in urban videos is challenging due to occlusions and clutter; especially when real-time performance is required.

1.1 Related Work

Previous approaches to image and video-based retrieval are based on summarization and scene understanding. Summarization methods are loosely related to search. They attempt to reduce the video to its relevant sections, typically dividing the video into "shots" [17, 15] by locating and annotating key frames corresponding to scene transitions or an absence of motion. Unfortunately, such scene transitions are infrequent in surveillance video, rendering most summarization approaches unsuitable for exploratory search in complex scenes, the focus of this paper.

Scene-understanding methods have been suggested for exploratory search [10, 19, 25]. These methods in general focus on classifying observed activities in terms of activities learned from a training video. For instance, activities observed in public areas often follow some basic rules (such as traffic lights, highways, building entries, etc), which are learned during the training phase. Common topic modeling techniques include

[1]This research was supported by ONR grant N000141010477, NGA grant HM1582-09-1-0037, NSF grant CCF-0905541, and DHS grant 2008-ST-061- ED0001

MM'12, October 29–November 2, 2012, Nara, Japan.

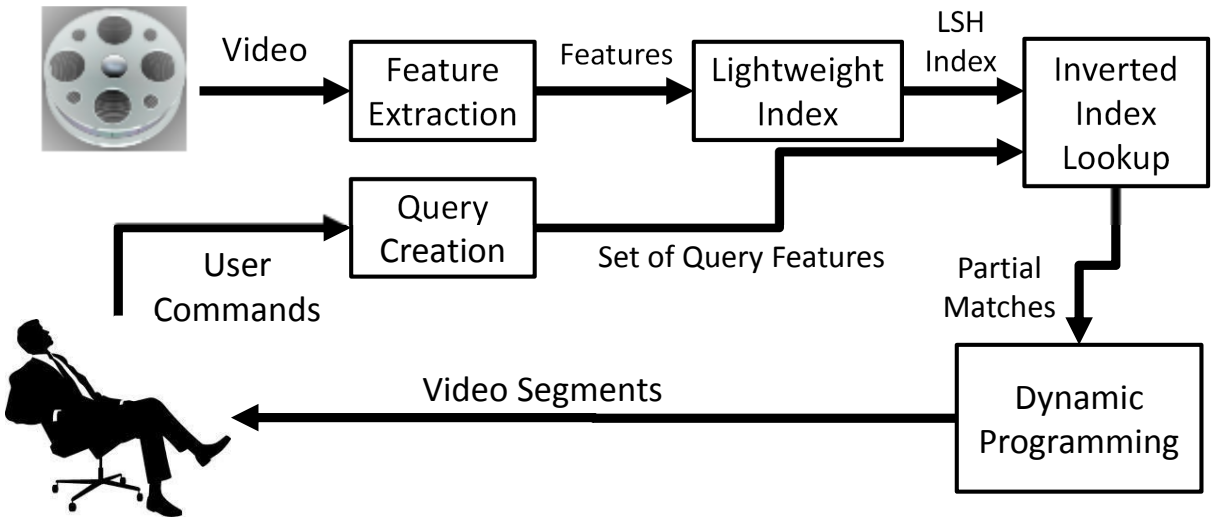

Figure 1: From streaming video, local features for each document are computed and inserted into a fuzzy, lightweight index. A user inputs a query, and partial matches (features which are close to parts of the query) are inserted into a dynamic programming (DP) algorithm. The algorithm extracts the set of video segments which best matches the query.

HMMs [13, 10], Bayesian networks [23], context free grammars [20], and other graphical models [11, 14, 21]. We point out that techniques that require global behavior understanding often rely on tracking [13, 11] while those devoted to isolated action recognition rely on low-level features [7, 26]. The classification stage can then be used to retrieve pre-defined patterns of activity [10]).

Scene understanding techniques which focus on global explanations operate at a competitive disadvantage in search: the preponderance of clutter (requirement four) in surveillance video makes the training step of scene understanding prohibitively difficult. Second, since these techniques often focus on understanding recurrent activities, they are unsuited for retrieving infrequent events - this can be a problem, given that queries are unpredictable (requirement two). Finally, the training step in scene understanding can be prohibitively expensive, violating requirement three, large data lifetimes.

Unlike summarization approaches, we extract a full set of features; this is mandatory, as we have no a-priori knowledge of what query will be asked. Unlike scene understanding techniques, we have no training step; this would be incompatible with the data lifetimes and magnitudes of the corpus. Instead, we develop an approach based on exploiting temporal orders on simple features, which allows us to find arbitrary queries quickly while maintaining low false alarm rates. We demonstrate a substantial improvement over scene-understanding methods such as [23, 10] on a number of datasets in Section 5.

2. OVERVIEW

To address the challenges of exploratory search, we utilize a two-step process that first reduces the problem to the relevant data, and then reasons intelligently over that data. This process is shown in Fig. 1. As data streams in, video is pre-processed to extract relevant features - activity, object size, color, persistence and motion. These low-level features are hashed into a fuzzy, light-weight lookup table by means of LSH [8]. LSH accounts for spatial variability and reduces the search space for a user's query to the set of relevant partial (local) matches.

The second step is a search engine optimization which reasons over the partial matches to produce *full matches*; segments of video which fit the entire query pattern, as opposed to part of it. This optimization operates from the advantageous standpoint of having only to reason over the partial matches, which are the relevant subset of the video. In surveillance video, where a long time can pass without relevant action, this dramatically reduces the workload of the optimization algorithm.

We then present two solutions for finding full matches. The first is a greedy approach which flattens the query in time. The second, a novel dynamic programming (DP) approach, exploits the causal ordering of component actions that makeup a query. DP reasons over the set of partial matches and finds the best full match. We also rigorously establish theoretical guarantees (Theorem 4.1). We show that the number of false positives in a surveillance environment - as actions become more complex – the probability of random actions mimicking them goes to zero.

3. LIGHTWEIGHT INDEX

Surveillance video frequently contains long periods of inactivity and irrelevance. In this section we describe an approach for creating a lightweight index with local features in order to reduce complexity for the search engine.

3.1 Feature extraction

3.1.1 Structure

For the purpose of feature extraction, a video is considered to be a spatio-temporal volume of size $H \times W \times F$ where $H \times W$ is the image size in pixels and F the total number of frames in the video. The video is divided along the temporal axis into contiguous *documents* each containing A frames. As shown in Fig. 2, each frame is divided into tiles of size $B \times B$. An *atom* is formed by grouping the same tile over A frames. These values vary depending on the size of the video - for our videos, we chose B equal to 8 or 16 pixels, and A equal to 15 or 30 frames, depending on frame rate. As the video streams in, features are extracted from each frame. Whenever a document is created, each atom n is assigned a set of features (see Sec. 3.1.2) describing the dynamic content over that region.

We construct a pyramidal structure to robustify our algorithm to

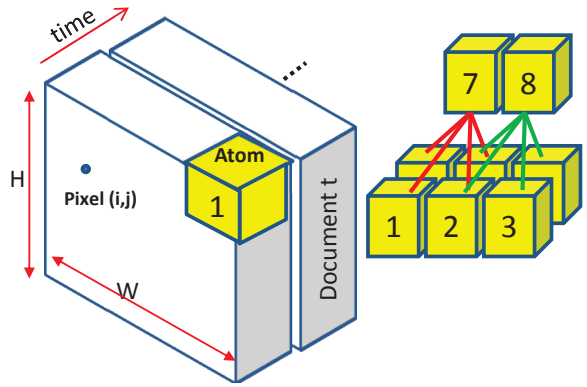

Figure 2: (Left) Given an $W \times H \times F$ video, *documents* are non-overlapping video clips each containing A frames. Each of the frames are divided into *tiles* of size $B \times B$. Tiles form an *atom* when aggregated together over A frames. (Right) Atoms are grouped into two-level trees - every adjacent set of four atoms is aggregated into a parent, forming a set of partially overlapping trees.

location and size variability in detected features. Each element of the pyramid has four children. This structure is made of k-level trees, each containing $M = \sum_{l=1}^{k} l^2$ nodes (Fig. 2). In this approach, a document containing $U \times V$ atoms will be assigned $(U - k + 1) \times (V - k + 1)$ partially overlapping trees to be indexed. For instance, in Figure 2, we draw a depth two tree on a document which is 2×3 atoms, resulting in two overlapping trees, each containing $M = 5$ nodes.

Each node of a tree is assigned a feature vector obtained by aggregating the feature vector of its children. Let n be a non-leaf node and a, b, c, d its four children. The aggregation process can be formalized as

$$\vec{x}_f^{(i)} = \psi_f \left(\vec{x}_f^{(a)}, \vec{x}_f^{(b)}, \vec{x}_f^{(c)}, \vec{x}_f^{(d)} \right),$$

where ψ_f is an aggregation operator for feature f. Sec. 3.1.2 presents more details on the aggregation operator. Given that several features are extracted for each atom, aggregating a group of $k \times k$ atoms results in a set of feature trees $\{\text{tree}_f\}$, one for each feature f. Given that a k-level tree contains M nodes, each tree_f contains a list of M feature instances, namely $\text{tree}_f = \left\{ \vec{x}_f^{(i)} \right\}$ where i stands for the ith node in the tree.

3.1.2 Features

As reported in the literature [12, 18, 24], atom features can be of any kind such as color, object shape, object motion, tracks, etc. We chose to use local processing due to the computational efficiency which makes it better suited to the constant data renewal constraint and real-time feature extraction. Because our focus is surveillance video, we assume a stable camera. To be effective on a moving or zooming camera, motion and zoom compensation would have to be applied in advance of feature extraction. Feature extraction computes a single value for each atom, which is aggregated into feature trees. Our method uses the following five features:

(1) Activity x_a: Activity is detected using a basic background subtraction method [5]. The initial background is estimated using a median of the first 500 frames. Then, the background is updated using the running average method. At the leaf level, x_a contains the proportion of active pixels within the atom. Aggregation for non-leaf nodes in feature trees, ψ_a, is the mean of the four children.

(2) Object Size x_s: Objects are detected using connected components analysis of the binary activity mask obtained from background subtraction [5]. Object size is the total number of active pixels covered by the connected component. The aggregation operator ψ_s for non-leaf nodes in feature trees is the median of non-zero children. Whenever all four children have a zero object size, the aggregation operator returns zero.

(3) Color $\vec{x}_c$: Color is obtained by computing the quantized histogram over every active pixel in the atom. RGB pixels are then converted to the HSL color space. Hue, saturation and luminance are quantized into 8, 4 and 4 bins respectively. The aggregation operator ψ_a for non-leaf nodes in feature trees is the bin-wise sum of histograms. In order to keep relative track of proportions during aggregation, histograms are not normalized at this stage.

(4) Persistence x_p: Persistence is a detector for newly static objects. It is computed by accumulating the binary activity mask obtained from background subtraction over time. Objects that become idle for a long periods of time thus get a large persistence measure. The aggregation operator ψ_p for non-leaf nodes in feature trees is the maximum of the four children.

(5) Motion $\vec{x}_m$: Motion vectors are extracted using Horn and Schunck's optical flow method [6]. Motion is quantized into 8 directions and an extra "idle" bin is used for flow vectors with low magnitude. $\vec{x}_m$ thus contains a 9-bin motion histogram. The aggregation operator ψ_m for non-leaf nodes in feature trees is a bin-wise sum of histograms. In order to keep relative track of proportions during aggregation, histograms are not normalized at this stage.

As mentioned previously, these motion features are extracted while the video streams in. Whenever a document is created, its atoms are assigned 5 descriptors, namely $\{x_a, x_s, \vec{x}_c, x_p, \vec{x}_m\}$. These descriptors are then assembled to form the 5 feature trees $\{\text{tree}_a\}, \{\text{tree}_s\}, \{\text{tree}_c\}, \{\text{tree}_p\}, \{\text{tree}_m\}$. These feature trees are the basis for the indexing scheme presented in section 3.2. After feature trees are indexed, all extracted feature content is discarded, ensuring a lightweight representation. It is worth noting that these features are intentionally simple. This speeds up feature extraction and indexing while being robust to small distortions because of the coarse nature of the features. While motion can be sensitive to poorly-compensated camera motion or zoom, and color can be sensitive to illumination changes, the other features have been shown to relatively robust to these effects [5]. In addition, we leverage on the dynamic programming in section 4.2.2 to limit false activity detection.

3.2 Indexing & Hashing

When a document is created, features are aggregated into $(U - k + 1) \times (V - k + 1)$ k-level trees. Each tree is made of 5 feature trees, namely $\{\text{tree}_a\}, \{\text{tree}_s\}, \{\text{tree}_c\}, \{\text{tree}_p\}, \{\text{tree}_m\}$. To index a given feature tree tree_f efficiently, our method uses an inverted index for content retrieval. Inverted index schemes, which map content to a location in a database, are popular in content-based retrieval because they allow extremely fast lookup in very large document databases. For video, the goal is to store the document number t and the spatial position (u, v) in the database based on the content of tree_f. This is done with a mapping function which converts "tree_f" to an entry in the database where (t, u, v) is stored. Two trees with similar contents, therefore, should be mapped to proximate locations in the index; by retrieving all entries which are near a query tree, we can recover the locations in the video of all features trees that are similar.

This mapping and retrieval can be made for which update and lookup exhibit flat performance ($O(1)$ complexity). Because similar content at different times is mapped to the same bin, the

time required to find the indices of matching trees does not scale with the length of the video. Obviously, the total retrieval time must scale linearly with the number of matching trees, but this means that the run-time of the retrieval process scales only with the amount of data which is relevant. In videos where the query represents an infrequently-performed action, this optimization yields an immense improvement in runtime.

Hashing: A hash-based inverted index uses a function to map content to an entry in the index. This is done with a hashing function h such that $h : \text{tree}_f \rightarrow j$, where j is a hash table bucket number. Usually, hash functions attempt to distribute content uniformly over the hash space by minimizing the probability of collisions between two non-equal entries:

$$\vec{x} \neq \vec{y} \implies P\left\{h(\vec{x}) = h(\vec{y})\right\} \approx 0.$$

However, in a motion feature space, descriptors for two similar events are never exactly equal. Moreover, it is unlikely that a user query can be translated to feature vectors with sufficient accuracy for such a strict hash function.

As a solution, we resort to a locality-sensitive hashing (LSH) [8] technique. LSH is a technique for approximation of nearest-neighbor search. In contrast to most hashing techniques, LSH attempts to cluster similar vectors by maximizing the probability of collisions for descriptors within a certain distance of each other:

$$\vec{x} \approx \vec{y} \implies P\left\{h(\vec{x}) = h(\vec{y})\right\} \gg 0.$$

If feature trees are close in Euclidian distance (the element-wise square of the distances between node values in the two trees is small), then the probability of them having the same hash code is high. Because our feature trees contain M real-valued variables, LSH functions can be drawn from the p-stable family:

$$h_{\vec{a},b,r}(\text{tree}_f) = \left\lfloor \frac{\vec{a} \cdot \text{tree}_f + b}{r} \right\rfloor,$$

where $\vec{a}$ is a M-dimensional vector with random components drawn from a stable distribution, b is a random scalar drawn from a stable distribution and r is an application-dependent parameter. Intuitively, $\vec{a}$ represents a random projection, an alignment offset b and a radius r controlling the probability of collision inside the radius.

Indices are built and searched independently for each feature. Thus, the database is made of five indices I_f, one for each feature f. Each index I_f is composed of a set of n hash tables $\{T_{f,i}\}$, $\forall i = 1, \ldots, n$. Each hash table is associated its own hash function $H_{f,i}$ drawn from the p-stable family $h_{\vec{a},b,r}$. The parameter r can be adjusted to relax or sharpen matches. In our implementation, r is fixed for a given feature. The random parameters $\vec{a}$ and b ensure projections from the different hash functions complement each other.

Given a feature tree tree_f with hash code $H_{f,i}(\text{tree}_f) = j$, $T_{f,i}[j, u, v]$ denotes the set of document numbers $\{t\}$ such that feature trees at (t, u, v) have similar content. Lookup in the index I_f consists of taking the union of document numbers returned by lookup in all tables $\{T_{f,i}\}$:

$$I(\text{tree}_f, u, v) = \cup_{i=1}^{n} T_{f,i}\left[H_{f,i}(\text{tree}_f), u, v\right].$$

Fig. 3 illustrates several feature trees partitioned into groups, where trees in the same group have been given the same hashing key. For a given video, we plotted the content of four of the most occupied buckets for the motion feature tree_m. As one can see, the trees associated to similar motion patterns in various parts of the scene have been coherently hashed into similar buckets.

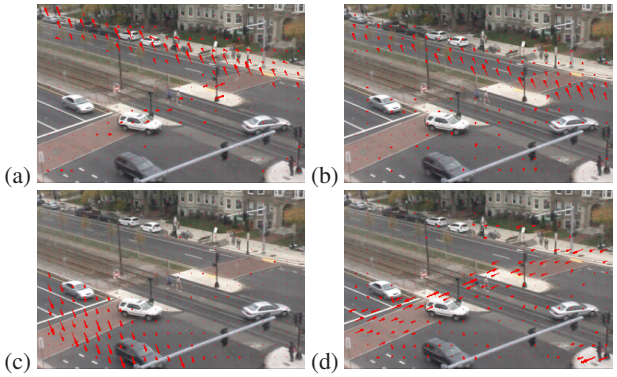

Figure 3: Contents of four buckets of a hash table for the *motion* feature. Arrows size is proportional to the number of hits at the specified location across the entire video. The hash buckets are associated to activity (a) side walk (b) upper side of the street (c) lower side of the street (d) crosswalk.

3.2.1 *Data structure*

As described previously, the inverted index stores in the same bucket the spatio-temporal position $\{(t, u, v)\}$ of all trees whose content is similar. As shown in Fig. 4, each bucket is a $(U - k + 1) \times (V - k + 1)$ matrix (see Sec. 3.1.1) whose cells contain a list of document numbers $\{t\}$. In that way, having an (u, v) matrix lookup located right after the hashing lookup ensures a retrieval time of $O(1)$.

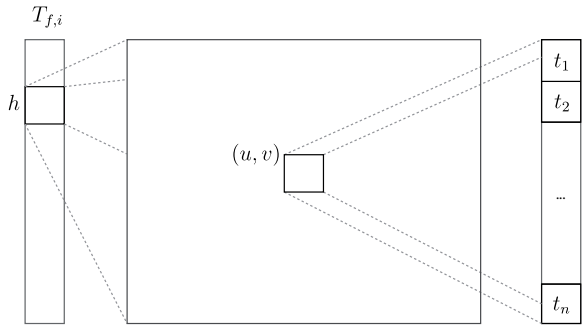

Figure 4: Hash table structure. For a given tree tree_f with $H_{f,i}(\text{tree}_f) = j$, lookup $T_{f,i}[j, u, v]$ is performed in two steps: (1) fetch the bucket at position j and (2) fetch the list of document numbers at position (u, v) in the bucket.

Lightweight Storage: In contrast to approaches relying on a distance metric, such as K-nearest neighbor search, the hash-based index representation does not require storage of feature descriptors. $T_{f,i}$ contains only document numbers $\{t\}$, which are stored in 4-byte variables. Our features, being local and primitive, are dependent on foreground content. As such, both our indexing times and storage scale linearly with the amount of foreground content in the video. This is a useful feature for surveillance video, which can have persistently inactive spaces or times throughout a video. For example, the size of the index for a 5 hour video with an activity rate of 2.5% is only 150kb while the input color video requires almost 7Gb (compressed).

Building Lookup Table: As video streams in, the index for each feature is updated by a simple and fast procedure. After extraction of feature f for document t is completed, the extracted features are grouped into trees, as described in Sec. 3.1. Then, I_f is updated with the mapping $\text{tree}_f \rightarrow (t, u, v)$ for each tree position (u, v)

covering an area with significant activity. This is repeated for each feature f.

4. SEARCH ENGINE

We showed in Sec. 3 how to extract low-level features from a video sequence, bundle them into trees and index them for $O(1)$ content-based retrieval. Here we explain how to use feature index for high-level search.

4.1 Queries

In video search without exemplars, it's essential to provide a straightforward way for a user to input a query. For our purposes, a query is defined as an ordered set of *action components* (for simple queries, a single component frequently suffices), each of which is a set of feature trees. To create a query, the user types in the number of action components, and is then presented with a GUI, shown in Figure 5 containing the first frame of the video to search for each of the action components. The user then selects the type of feature he wishes to search for, and draws the region of interest (ROI) that he wishes to find it in. These regions and the features (directions of motion, in this case) are shown in Fig.8 as green areas and red arrows, respectively.

As an example, to describe a u-turn, the user might describe three action components: one detecting motion approaching an intersection, one detecting motion turning around, and one detecting motion leaving the intersection. Likewise, a man hopping a subway might be represented by somebody approaching the subway, jumping up, and then continuing past the subway.

Figure 5: The query creation GUI provides a straightforward way to construct queries. The user draws each action component (shown in blue), and can additionally specify features.

Because our features (activity, size, color, persistence, and motion) are semantically meaningful, this method of input is a relatively accessible way to define a query. After the features and the ROI are selected, a feature tree $tree_f$ is created to represent that action component. Note that this formulation provides the user with a way to produce a complex query vocabulary. A single component could describe a search for "small red stationary objects" or 'large objects moving to the right". While our claims to simplicity are theoretical, they are also supported by anecdotal evidence. People unfamiliar with the system are able to create their own queries in under a minute after a brief explanation of the tools.

After the action component has been input, the set of feature trees is extracted from it using the formulation in Section 3.1. These feature trees is used to query the inverted index of Section 3.2 to produce a set of documents and locations called *partial matches* $M(q)$ that contained similar trees to query q. In the case where the query contains multiple feature trees, the set of matching locations is the intersection of the matches for individual trees. Fig.8 presents 10 different queries with their ROI.

4.2 Full matches

Search in a surveillance video requires more than partial matches. Activities in a video are inherently complex and show significant variability in time duration. For instance, a fast car taking a U-turn will span across fewer documents and generate different motion features than a slow moving car. Also, due to the limited size of a document (typically between 30 to 100 frames), partial matches may only correspond to a portion of the requested event. For example, partial matches in a document t may only correspond to the beginning of a U-turn. The results expected by a user are so-called *full maches*, i.e. video segments $[t, t+\Delta]$ containing one or more documents ($\Delta > 0$). For example, the video segment $R = \{t, t+1, t+2\}$ corresponds to a full U-turn match when documents $t, t+1, t+2$ contain the beginning, the middle and the end of the U-turn. Given a query q and a set of partial matches $M(q)$, a full match starting at time τ is defined as

$$R_{q,\tau}(\Delta) = \{(u,v) | (t,u,v) \in M(q),\ \forall t \in [\tau, \tau+\Delta]\}\,. \qquad (1)$$

Thus, $R_{q,\tau}(\Delta)$ contains the set of distinct coordinates of trees partially matching q in the video segment $[\tau, \tau+\Delta]$.

We propose two algorithms to identify these full matches from the set of partial matches. The first is a greedy optimization procedure based on the total number of partial matches in a document that does not exploit the temporal ordering of a query. The second approach (Sec. 4.2.2), uses dynamic programming to exploit temporal structure of the of a query's action components.

4.2.1 *Full matches using a greedy algorithm*

The main difficulty in identifying full matches comes with the inherent variability between the query and the target. This includes time-scaling, false detections and other local spatio-temporal deformations. Consequently, we are faced with the problem of finding which documents to fuse into a full match given a set of partial matches. We formulate it in terms of the following optimization problem:

$$\Delta^* = \arg\max_{\Delta>0} v_{q,\tau}(\Delta). \qquad (2)$$

where q is the query, τ is a starting point and Δ the length of the retrieved video segment. The value function $v_{q,\tau}(\Delta)$ maps the set of partial matches in the interval $[\tau, \tau+\Delta]$ to some large number when the partial matches fit q well and to a small value when they do not. To determine the optimal length of a video segment starting at time τ we maximize the above expression over Δ.

While many value functions are viable, depending upon user preference, a simple and effective $v_{q,\tau}(\Delta)$ is:

$$v_{q,\tau}(\Delta) = \quad |R_{q,\tau}(\Delta)| - \lambda\tau, \qquad (3)$$

where $R_{q,\tau}(\Delta)$ is defined by Eq. (1) and $|R_{q,\tau}(\Delta)|$ is the total number of distinct matching locations found in the interval $[\tau, \tau+\Delta]$. The value function is time-discounted since $R_{q,\tau}(\Delta)$ is increasing in Δ (by definition, $R_{q,\tau}(\Delta) \subseteq R_{q,\tau}(\Delta+1)$). The parameter λ is a time-scale parameter and loosely controls size of retrieved video segment.

We can determine Δ by a simple and fast greedy algorithm. The algorithm finds a set of non-overlapping video segments and a natural ranking based the value function provided above. As will be shown in Sec. 5, Eq. (3) produces compact video segments while keeping low false positives and negatives rates.

It may seem strange that such a simple value function provides accurate results over a wide range of queries and videos. Intuitively, this is because the more complex a query is, the less likely it is to be generated by unassociated actions. We state this formally in theorem 4.1.

Theorem 4.1 *Suppose that we have a random video: we sample independently across time and at each instant uniformly from the set of all trees with replacement. Suppose the query q consists of $|q|$ distinct trees and the random video has $R_{q,\tau}(\Delta)$ matches. For $\Delta = \gamma|q|$ the probability that $\log(v_{q,\tau}(\Delta)) \geq \alpha|q|$ for some $\alpha \in (0,\ 1)$ is smaller than $O(1/|q|^2)$.*

This result suggests that the false alarm probability resulting from an algorithm that is based on thresholding $v_{q,\tau}(\Delta)$ is small. This result is relevant because we expect $\log(v_{q,\tau}(\Delta))$ for video segments that match the query to have a value larger than $\alpha|q|$ for some α when $\Delta = \Omega(|q|)$.

PROOF. For simplicity we only consider each document to contain a single random tree drawn from among $|\mathcal{H}|$ trees. The general result for a fixed number of trees follows in an identical manner. We compute the value of random video of length τ, i.e.,

$$P\left\{v_{q,0}(\Delta) \geq \exp(\alpha|q|)\right\} = P\left\{|R_{q,0}(\Delta)| \geq \alpha|q| + \gamma\right\},$$

where we substituted $\frac{\Delta}{\lambda} = \gamma$ and taken logarithms on both sides to obtain the second equation. Let, Δ_j be the number of documents before we see a new document among the set q after just having seen the $j-1$th new document. This corresponds to the inter-arrival time between the $(j-1)$th and jth document. Given our single tree random model we note that $R_{q,0}(\Delta)$ is a counting process in Δ and so we have the following equivalence,

$$R_{q,0}(\Delta) \geq \ell \iff \sum_{j=1}^{\ell} \Delta_j \leq \Delta.$$

Thus we are now left to determine the $P\left\{\sum_{j=1}^{\ell} \Delta_j \leq \Delta\right\}$ where $\ell = \alpha|q| + \gamma$. Next, $\Delta_1, \Delta_2, \ldots$ are independent geometric random variables. The jth random variable Δ_j has a geometric distribution with parameter $p_i = \frac{|q|-j}{|\mathcal{H}|}$. Using these facts we can determine the mean value and variance of the sum using linearity of expectations. Specifically, it turns out that

$$E\left(\sum_{j=1}^{\ell} \Delta_j\right) = \sum_{j=1}^{\ell} \frac{1}{p_i};\ Var\left(\sum_{j=1}^{\ell} \Delta_j\right) = \sum_{j=1}^{\ell} \frac{1-p_i}{p_i^2}.$$

Upon computation the mean turns out to be $O(|\mathcal{H}|)$, while the variance turns out to be $O(|\mathcal{H}|^2/|q|^2)$. We now apply Chebyshev inequality to conclude $P\left\{v_{q,0}(\Delta) \geq \exp(\alpha|q|)\right\} \leq O(1/|q|^2)$, which establishes the result. □

4.2.2 *Full matches with Dynamic Programming(DP)*

We can improve upon the performance of the greedy algorithm in Sec. 4.2.1 by exploiting the order of the action components. For example, in the example of a man hopping a subway turnstile, he has to approach the turnstile from the wrong direction, hop over it, and continue. For a car taking a u-turn, it has to approach the intersection, turn across it, and depart the way it came. While these component actions have many valid configurations when examined independently, there is only one order in which they represent the desired full action. This is illustrated in Fig. 6.

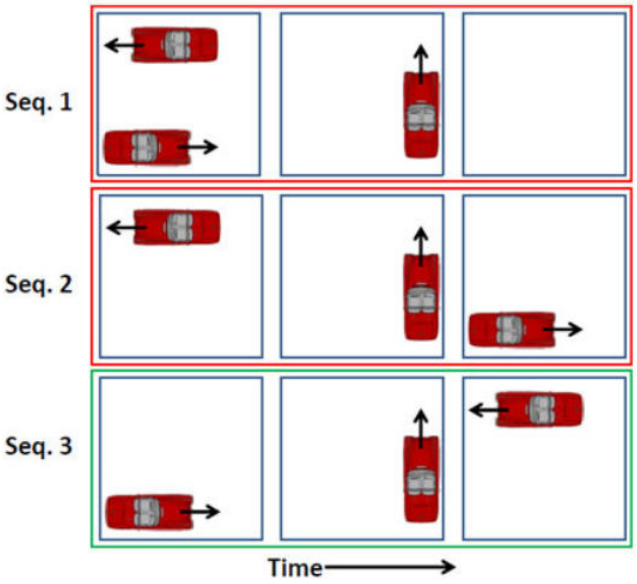

Figure 6: Three sequences of actions. All have equivalent values of $R_{Q,\tau}(\Delta)$; only the third row is valid U-turn.

In the greedy optimization of Section 4.2.1, we ignore the time-ordering that a set of queries contains. The value function $R_{q,\tau}(\Delta)$ does not differentiate between the sequences of actions *(Forward, Left, Back)* and the sequence of actions *(Back, Forward, Left)*. Intuitively, this should hurt performance - false matches are more likely to appear than were we to exploit this causality.

After a user uses the GUI described in Section 4.1 to create a query containing a set of N action components, we search through the inverted index to retrieve N sets of matching locations within the video, one for each action component.

After this pre-processing step, we adapt the Smith-Waterman dynamic programming algorithm [16] , originally developed for gene sequencing, to determine the best set of partial matches in the query. Our algorithm, described in Algorithm 1, operates over the set of matches $m_{\tau,\alpha}$ which contains matches in document α to action component τ to recover a query that has been distorted. For the purposes of our videos, we consider three types of distortion, namely insertion, deletion and continuation.

(1) Insertion covers documents where we believe the query is happening but there are no partial matches. This can happen if unrelated documents are inserted, if there is a pause in the execution of the activity, or if the activity is obscured.

(2) Deletion covers documents where sections of the query are missing. If a deletion has occurred, one (or more) of the action components in the query will not be present in the video. Deletions can happen because of obscuration, or simply because somebody does not perform one component of the full action.

(3) Continuation covers documents where we continue to see an action component which we have already seen. This is important because of time distortion; a single action component does not necessarily occur in multiple consecutive documents before the next action component is reached.

As described in Algorithm 1, to search for a query with $|q|$ action components in a video with N documents, our dynamic programming approach creates an $N \times |q|$ matrix, V, which is

Algorithm 1 Dynamic programming (DP) algorithm

procedure SEARCH(m, W, T)
  $V \leftarrow 0$; $paths \leftarrow \emptyset$; $\tau \leftarrow 1$; $\alpha \leftarrow 1$
  while $\tau \leq$ number of documents **do**
    while $\alpha \leq$ number of action components **do**
      $$V_{\tau,\alpha} \leftarrow \max \begin{cases} 0 \\ (V_{\tau-1,\alpha-1} + W_{match}) * m_{\tau,\alpha} \\ (V_{\tau-1,\alpha} + W_{cont}) * m_{\tau,\alpha} \\ (V_{\tau-1,\alpha} + W_{delete}) * (1 - m_{\tau,\alpha}) \\ (V_{\tau,\alpha-1} + W_{insert}) * (1 - m_{\tau,\alpha}) \end{cases}$$
      Let (a, b) be the index which was used to generate the maximum value
      if $V_{(\tau,\alpha)} > 0$ **then**
        $paths_{\tau,\alpha} \leftarrow paths_{a,b} \cup (\tau, \alpha)$
      else
        $paths_{\tau,\alpha} \leftarrow paths_{a,b}$
      end if
      $\alpha \leftarrow \alpha + 1$
    end while
    $\tau \leftarrow \tau + 1$
  end while
  $Matches \leftarrow \emptyset$
  while $max(V) > T$ **do**
    Let (a, b) be the index of V containing the maximum value
    $Matches \leftarrow Matches \cup paths_{a,b}$
    for $\tau, \alpha \in paths_{a,b}$ **do**
      $V_{\tau,\alpha} = 0$
    end for
  end while
  Return $Matches$, the set of paths above threshold T
end procedure

filled out from the top left to the bottom right. A path through the matrix is defined as a set of adjacent (by an 8-neighborhood) matrix element, where each element of the path represents a hypothetical assignment of an action component taking place in a document. A path containing element $V_{a,b}$ would indicate that that path believed that action component b occured in document a.

As the matrix is filled out, each element chooses to append itself to the best possible preceding path - which, by definition, ends immediately above it, immediately to the left of it, or both. It stores the resulting value, and a pointer to the preceding element, in the value matrix V. When the matrix is fully filled out, the optimal path can be found by starting at the maximal value in the matrix and tracing backwards. In order to find multiple matches, we repeatedly find the maximal value in V, the optimal path associated with it, set the values along that path to zero, and repeat until the maximum value in V is below a threshold T. An example of this matrix, with paths overlaid, is shown in figure 7.

For a given penalty on each type of distortion W_I, W_D, W_C (corresponding to insertion, deletion, and continuation) and a given bonus for each match, W_M, the DP algorithm (Algorithm 1) is guaranteed to find the set of partial matches which maximizes the sum of the penalties and bonuses over an interval of time. For our queries, we were relatively certain that elements of the query would not be obscured, but we were uncertain about our detection rate on features and how long an event would take. Thus, we set $W_I = -2$, $W_D = -10$, $W_C = 1$, and $W_M = 8$. These values preclude optimal solutions which involve deletions, look for longer sequences that match, and are relatively robust to missed detection.

We note that because it reasons over specific partial matches, our dynamic programming approach also finds the locations in the video segments where the event occurs, but this is not exploited in the results of this paper.

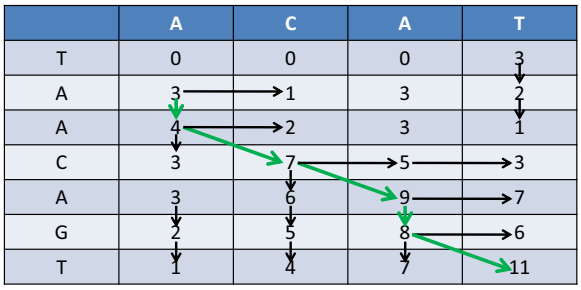

Figure 7: An example of the V matrix. The query is actions A, C, A, T, and the seven documents in the video corpus each contains a single action, T, A, A, C, A, G, T. The values for W_I, W_D, W_C and W_M are $-1, -2, 1$ and 3 respectively. The optimal path, A,A,C,A,G,T, involves an insertion, a continuation and a deletion. It is found by tracing backwards from the maximal element, valued at 11.

5. EXPERIMENTAL RESULTS

5.1 Datasets

In order to evaluate performance of the two-step problem formulation, and the DP approach in particular, we initially tested our two-step approach on seven surveillance videos (see table 1 and Fig. 8). These videos were selected to test the application of this basic approach to multiple domains, as well as to provide a basis for comparison to other algorithms. The *Winter driveway*, *U-Turn* and *Abandoned object* sequences were shot by us, *PETS* and *Parked-vehicle* and *MIT-traffic* come from known databases [3, 1], *MIT-traffic* was made available to us by Wang *et al.* [2]; *Subway* from Adam *et al.* [4].

As listed in Table 1, we tested different queries to recover moving objects based on their color, size, direction, activity and persistence. We queried for rare and sometimes anomalous events (cat in the snow, illegal U-turns, abandoned objects and people passing turnstile in reverse) as well as usual events (pedestrian counting, car turning at a street light, and car parking). Some videos featured events at a distance *MIT-traffic*, while others featured people moving close to the camera *Subway*. We searched for objects, animals, people, and vehicles. Given these queries, we watched the videos and created a ground-truth list for each task.

5.2 HDP Comparison

For the purposes of comparison, we employed high-level search functions based on scene understanding techniques using Hierarchical Dirichelet Processes (HDP) [23, 10]. We chose this, because unlike [11, 9], it does not require tracking, which can be difficult to do in complex scenes and is computationally prohibitive. At each iteration, the HDP-based learning algorithm assigns each document to one or more high-level activities. This classification is used as input to the next training iteration. Xiang *et al.* [23] propose a search algorithm that uses learned topics as high-level semantic queries. The search algorithm is based on the classification outputs from the final HDP training iteration. We compare our method to this HDP-based search algorithm.

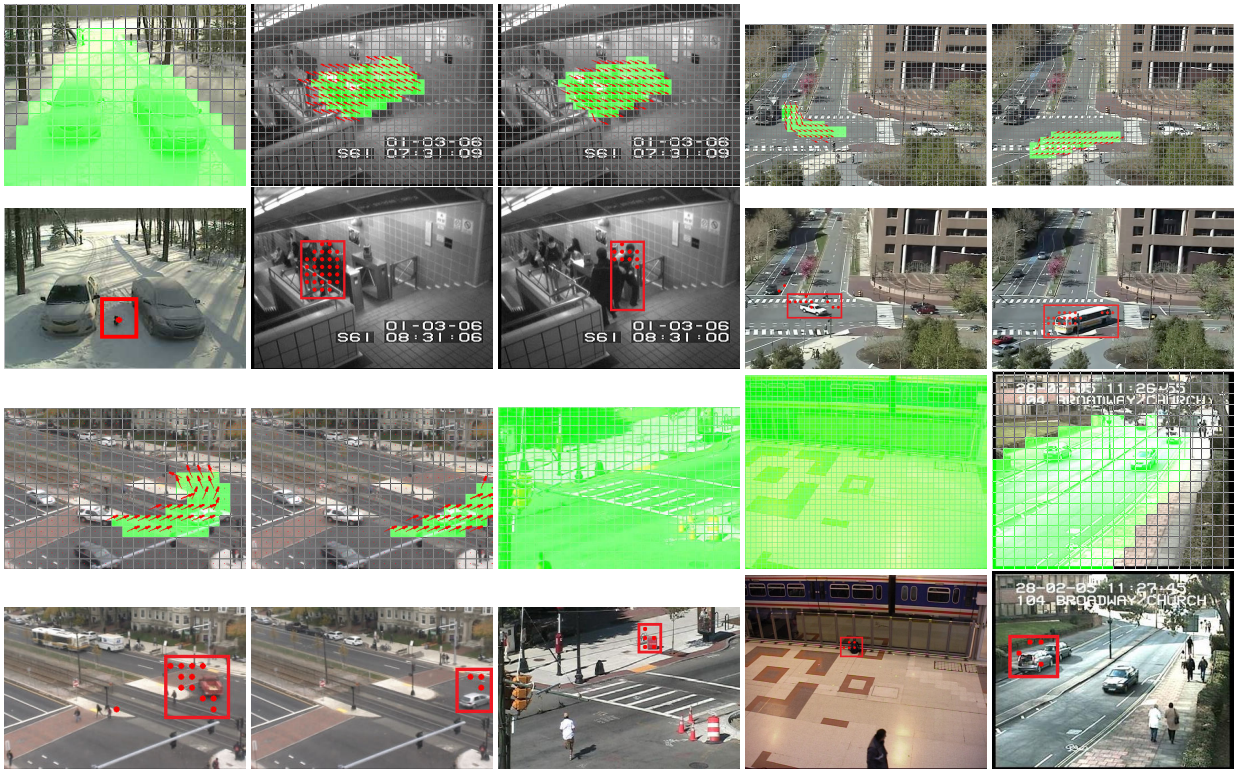

Figure 8: Screen shots of the ten tasks. These images show the search queries (with green ROI) and a retrieve frame (with a red rectangle). The red dots correspond to the tree whose profile fit the query.

Task	Video	Search query	Features	Video size	Index size
1	Winter driveway	black cat appearance	color and size	6.55 GB	147 KB
2	Subway	people passing turnstiles	motion	2.75 GB	2.3 MB
3	Subway	people hopping turnstiles	motion	2.75 GB	2.3 MB
4	MIT Traffic	cars turning left	motion	10.3 GB	42 MB
5	MIT Traffic	cars turning right	motion	10.3 GB	42 MB
6	U-turn	cars making U-turn	motion	1.97 GB	13.7 MB
7	U-turn	cars turning left, no U	direction	1.97 GB	13.7 MB
8	Abandoned object	abandoned objects	size and persistence	682 MB	2.6 MB
9	Abandoned object	abandoned objects	size, persistence and color	682 MB	2.6 MB
10	PETS	abandoned objects	size and persistence	1.01 GB	5.63 KB
11	Parked-vehicle	parked vehicles	size and persistence		

Table 1: Tasks' number, videos, search query, associate features, video size and index size. Tasks 1, 8, 9, 10, and 11 use compound search operators. The index size can be several orders of magnitude smaller than raw video. Our use of primitive local features implies that index times and index size are both proportional to the number of foreground objects in the video. Consequently, index size tends to be a good surrogate for indexing times.

Queries are specified as the ideal classification distribution and the search algorithm compares each document's distribution over the learned topics against this ideal distribution. Comparison is performed using the relative entropy (Kullback-Leibler divergence) between the two distributions. The Kullback-Leibler divergence gives a measure of distance between the query q and the distribution p_j for document j over the K topics:

$$D(q, p_j) = \sum_{k=1}^{K} q(k) \log \frac{q(k)}{p_j(k)}.$$

The query q is created by looking at the ideal documents and assigning to q a uniform distribution over the topics present in them. The search procedure evaluates $D(q, p_j)$ for each document j and ranks the documents in order of increasing divergence.

5.3 Eleven Tasks

Once we had defined the queries, we watched all of the videos and created a ground-truth list for each task. Ground truth was obtained manually by noting the range of frames containing each of the expected results. Comparison is obtained by computing the intersection of the ranges of frames returned by the search procedure to the range of frames in the ground truth. Events are counted manually by viewing the output results. An event is marked as detected if it appears in the output video and at least 1 partial match hits objects appearing in the event. The greedy-optimization results are displayed in Fig. 8 and summarized in table 2. The comparable results for HDP are summarized below in table 2.

The "Ground truth" column of table 2 indicates the true number of events which exist in the dataset. The "Greedy

Task	Video	Duration (minutes)	Ground Truth (events)	Greedy True (events found)	HDP [10] True (events found)	Greedy False (events found)	HDP [10] False (events found)	Runtime (seconds)
1	Winter driveway	253	3	2	–	1	–	7.5
2	Subway	79	117	116	114	1	121	0.3
3	Subway	79	13	11	1	2	33	3.0
4	MIT Traffic	92	66	61	6	5	58	0.4
5	MIT Traffic	92	148	135	54	13	118	0.5
6	U-turn	3.4	8	8	6	0	23	1.2
7	U-turn	3.4	6	5	4	1	14	0.6
8	Abandoned object	13.8	2	2	–	0	–	4.8
9	Abandoned object	13.8	2	2	–	0	–	13.3
10	PETS	7.1	4	4	–	0	–	20.2
11	Parked-vehicle	32	14	14	–	0	–	12.3

Table 2: Results for the elevens tasks using greedy optimization and HDP labels. Crossed-out rows correspond to queries for which there was no corresponding topic in the HDP search.

True" column indicates the number of correct detections (true positives) for the Greedy algorithm and "HDP True" the number of correct detections (true positives) for the HDP-based search [21]. Likewise, the "Greedy False" and "HDP False" indicate the number of false alarms that were found for those eleven tasks.

Table 2 demonstrates the robustness of the two-step method in a wide-array of search applications, outperforming the HDP baseline significantly in detection and false alarm rate. The figures in the table are given for results of total length approximately equal to that of the ground truth. As can be seen from the figures in the table, the absolute detection rate is strong.

In table 2, we learn that HDP-search deals well with search of recurring large-scale activities and poorly otherwise. While several queries could not be executed because of a lack of topics that could be used to model the query, the results nonetheless demonstrate some of the shortcomings of the algorithm. The HDP search scales linearly with the number of documents, an undesirable quality with large datasets. Further, the cost of the training phase is prohibitive (approximately 2 days for the "subway" sequence) and must be paid again every time thatmore video data is included.

5.4 Dynamic Programming

Of the eleven tasks described in table 1, only tasks two to seven had temporal structure which could be exploited through dynamic programming. As it turned out, the features used in these tasks were purely motion. In order to demonstrate the potential gain from exploiting this structure, we chose tasks three, four and six and performed dynamic programming using the full query, as well as greedy and HDP search algorithms. The ROC curves for those three scenarios are provided in Fig 9, contrasting dynamic programming with HDP and Greedy Optimization.

Fig. 9 demonstrates the type of improvement that can be attained by the two-step approach to search. The ROC curves for LSH-based greedy optimization dominate the HDP curves, and there is clear improvement from employing time-ordering with DP. These improvements come as no surprise - HDP is doing a global search, attempting to create a topic for each action. LSH does a compelling local search which is fast and produces low false alarm rate. This is largely due to the global nature of HDP - the topics it discovers are more likely to be common events, so infrequent events such as u-turns and turnstile-hopping pose a problem. Note that the gap between local and global searches narrow on the MIT-traffic dataset, where the event being found (left turns at a light) is a relatively common occurrence with enough repetition to fill a topic model.

5.5 Discussion

This approach represents a fundamentally different way of approaching the video search problem. Rather than relying on an abundance of training data or finely-tuned features to differentiate actions of interest from noise, we rely on simple features and causality. In addition to the clear benefits in terms of a run-time which scales sub-linearly with the length of the video corpus, the simple features and hashing approach render the approach robust to user error as well as poor-quality video. The results of section 5.4 demonstrate clearly that causality and temporal structure can be powerful tools to reduce false alarms. Another added benefit is how the algorithm scales with query complexity. Whereas algorithms such as topic modeling or a feature-based matching suffer as queries becomes more complex due to efforts to characterize the query, the two-step approach becomes more successful - the more action components in a query, the more likely it is to differentiate itself from noise. There is, of course, non-temporal structure that we have yet to exploit. Spatial positioning of queries, such as "The second action component must occur to the northeast of the first", or "The second action component must be near the first" is a simple attribute which may further differentiate queries of interest from background noise. This is not to say that the approach is not without its limitations. It requires that the activity being described contain discrete states, each of which is describable by a simple feature vocabulary. Complex actions like sign language or actions which are to fast or too small to be identified at the atom level will be difficult to search for.

6. CONCLUSION

We presented a method that summarizes the dynamic content of a surveillance video in a way that is compatible with arbitrary user-defined queries. We divide the video into documents each containing a series of atoms. These atoms are grouped together into trees all containing a feature list. These features describing the size, the color, the direction and the persistence of the moving objects. The coordinates of these trees are then stored into a hash-table based feature index. Hash functions group trees whose content is similar. In this way, search becomes a simple lookup as user-specified queries are converted into a table index. Our method has many advantages. First, because of the indexing strategy, our search engine has a complexity of $O(1)$ for finding partial matches. Second, the index requires only minimal storage. Third, our method requires only local processing, making it suitable for facing constant data renewal. This makes it simple to implement and easy to adjust. Fourth, our method summarizes the entire

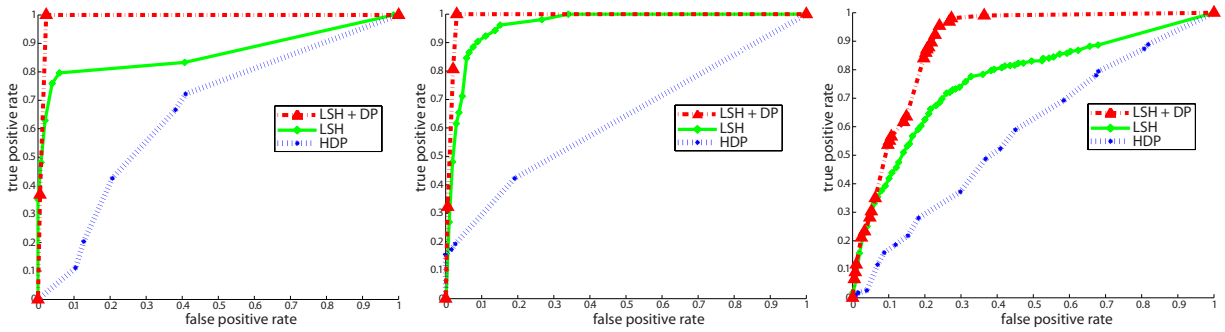

Figure 9: ROC curves for the U-turn, subway and traffic datasets. Both our greedy search and DP significantly outperform scene-understanding methods such as HDP methods [23, 10].

video, not just the predominant modes of activity: it can retrieve any combination of rare, abnormal and recurrent activities.

7. REFERENCES

[1] i-lids. computervision.wikia.com/wiki/I-LIDS.

[2] Mit traffic. people.csail.mit.edu/xgwang/HBM.html.

[3] Pets 2006. http://ftp.pets.rdg.ac.uk/.

[4] A. Adam, E. Rivlin, I. Shimshoni, and D. Reinitz. Robust real-time unusual event detection using multiple fixed-location monitors. *IEEE Trans. Pattern Anal. Machine Intell.*, 30(3):555–560, 2008.

[5] Y. Benezeth, P.-M. Jodoin, B. Emile, H. Laurent, and C. Rosenberger. Comparative study of background subtraction algorithms. *J. of Elec. Imaging*, 19(3):1–12, 2010.

[6] B.Horn and B. Schunck. Determining optical flow. *Artif. Intell.*, 17(1-3):185–203, 1981.

[7] Q. Dong, Y. Wu, and Z. Hu. Pointwise motion image (PMI): A novel motion representation and its applications to abnormality detection and behavior recognition. *IEEE Trans. Circuits Syst. Video Technol.*, 19(3):407–416, 2009.

[8] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. Int. Conf. on Very Large Data Bases*, pages 518–529, 1999.

[9] W. Hu, X. Xiao, Z. Fu, D. Xie, T. Tan, and S. Maybank. A system for learning statistical motion patterns. *IEEE Trans. Pattern Anal. Machine Intell.*, 28(9):1450–1464, 2006.

[10] D. Kuettel, M. Breitenstein, L. Gool, and V. Ferrari. What's going on? discovering spatio-temporal dependencies in dynamic scenes. In *Proc. IEEE Conf. Computer Vision Pattern Recognition*, pages 1951–1958, 2010.

[11] G. Medioni, I. Cohen, F. Bremond, S. Hongeng, and R. Nevatia. Event detection and analysis from video streams. *IEEE Trans. Pattern Anal. Machine Intell.*, 23(8):873–889, 2001.

[12] J. Meessen, M. Coulanges, X. Desurmont, and J.-F. Delaigle. Content-based retrieval of video surveillance scenes. In *MRCS*, pages 785–792, 2006.

[13] I. Pruteanu-Malinici and L. Carin. Infinite hidden markov models for unusual-event detection in video. *IEEE Trans. Image Process.*, 17(5):811–821, 2008.

[14] C. Simon, J. Meessen, and C. DeVleeschouwer. Visual event recognition using decision trees. *Multimedia Tools and Applications*, 2009.

[15] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proc. IEEE Int. Conf. Computer Vision*, volume 2, pages 1470–1477, 2003.

[16] T. Smith and M. Waterman. Identification of common molecular subsequences. *J. of Molecular Biology*, 147, 1981.

[17] X. Song and G. Fan. Joint key-frame extraction and object segmentation for content-based video analysis. *IEEE Trans. Circuits Syst. Video Technol.*, 16(7):904–914, 2006.

[18] E. Stringa and C. Regazzoni. Content-based retrieval and real time detection from video sequences acquired by surveillance systems. In *Proc. IEEE Int. Conf. Image Processing*, pages 138–142, 1998.

[19] Y.-L. Tian, A. Hampapur, L. Brown, R. Feris, M. Lu, A. Senior, C.-F. Shu, and Y. Zhai. Event detection, query, and retrieval for video surveillance. In Z. Ma, editor, *Artificial Intelligence for Maximizing Content Based Image Retrieval*. Information Science Reference; 1 edition, 2008.

[20] H. Veeraraghavan, N. Papanikolopoulos, and P. Schrater. Learning dynamic event descriptions in image sequences. In *Proc. IEEE Conf. Computer Vision Pattern Recognition*, pages 1–6, 2007.

[21] X. Wang, X. Ma, and E.Grimson. Unsupervised activity perception in crowded and complicated scenes using hierarchical bayesian models. *IEEE Trans. Pattern Anal. Machine Intell.*, 31(3):539–555, 2009.

[22] R. W. White and R. A. Roth. *Exploratory Search: Beyond the Query-Response Paradigm*. Morgan-Claypool, VT, U.S.A., 2009.

[23] T. Xiang and S. Gong. Video behavior profiling for anomaly detection. *IEEE Trans. Pattern Anal. Machine Intell.*, 30(5):893–908, 2008.

[24] Y. Yang, B. Lovell, and F. Dadgostar. Content-based video retrieval (cbvr) system for cctv surveillance videos. In *Proc of Dig. Img. Comp. Tech. and App.*, pages 183–187, 2009.

[25] C. Yeo, P. Ahammad, K. Ramchandran, and S. Sastr. High speed action recognition and localization in compressed domain videos. *IEEE Trans. Circuits Syst. Video Technol.*, 18(8):1006 – 1015, 2008.

[26] A. Yilmaz and M. Shah. A differential geometric approach to representing the human actions. *Comput. Vis. Image Und.*, 109(3):335–351, 2008.