2016

Exploratory search through large video corpora

BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Dissertation

EXPLORATORY SEARCH THROUGH LARGE VIDEO CORPORA

by

GREGORY D. CASTAÑÓN

B.S., Washington University in St. Louis, 2005

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2016

Approved by

First Reader

Venkatesh Saligrama, Ph.D.
Professor of Electrical and Computer Engineering
Professor of Systems Engineering

Second Reader

Prakash Ishwar, Ph.D.
Associate Professor of Electrical and Computer Engineering
Associate Professor of Systems Engineering

Third Reader

Janusz Konrad, Ph.D.
Professor of Electrical and Computer Engineering

Fourth Reader

Stanley E. Sclaroff, Ph.D.
Associate Dean of Faculty, Mathematical and Computer Sciences
Professor of Computer Science

Fifth Reader

Brian Kulis, Ph.D.
Assistant Professor of Electrical and Computer Engineering

cui dono lepidum novum libellum
arida modo pumice expolitum
Corneli tibi namque tu solebas
meas esse aliquid putare nugas
iam tum cum ausus es unus Italorum
omne aevum tribus explicare cartis
doctis Iuppiter et laboriosis
quare habe tibi quidquid hoc libelli
qualecumque quod o patrona virgo
plus uno maneat perenne saeclo *Catullus I*

Acknowledgments

I would like to thank my advisor, Venkatesh Saligrama, for his patience and guidance. Additionally, I would like to thank all of my wonderful colleagues at BU, both professors and students, who have provided insight, input and advice. This work would not be at all possible without them, and I am grateful for the opportunity to work with such gifted minds. I would like to thank my friends for their support through the process. Finally, I would like to thank my family for encouraging me and chipping in to help when needed.

EXPLORATORY SEARCH THROUGH LARGE VIDEO CORPORA

GREGORY D. CASTAÑÓN

Boston University, College of Engineering, 2016

Major Professor: Venkatesh Saligrama
Professor of Electrical and Computer Engineering

ABSTRACT

Activity retrieval is a growing field in electrical engineering that specializes in the search and retrieval of relevant activities and events in video corpora. With the affordability and popularity of cameras for government, personal and retail use, the quantity of available video data is rapidly outscaling our ability to reason over it. Towards the end of empowering users to navigate and interact with the contents of these video corpora, we propose a framework for exploratory search that emphasizes activity structure and search space reduction over complex feature representations.

Exploratory search is a user driven process wherein a person provides a system with a query describing the activity, event, or object he is interested in finding. Typically, this description takes the implicit form of one or more exemplar videos, but it can also involve an explicit description. The system returns candidate matches, followed by query refinement and iteration. System performance is judged by the run-time of the system and the precision/recall curve of of the query matches returned.

Scaling is one of the primary challenges in video search. From vast web-video archives like youtube (1 billion videos and counting) to the 30 million active surveillance cameras shooting an estimated 4 billion hours of footage every week in the United States, trying to find a set of matches can be like looking for a needle in a haystack. Our goal is to create

an efficient archival representation of video corpora that can be calculated in real-time as video streams in, and then enables a user to quickly get a set of results that match.

First, we design a system for rapidly identifying simple queries in large-scale video corpora. Instead of focusing on feature design, our system focuses on the spatiotemporal relationships between those features as a means of disambiguating an activity of interest from background. We define a semantic feature vocabulary of concepts that are both readily extracted from video and easily understood by an operator. As data streams in, features are hashed to an inverted index and retrieved in constant time after the system is presented with a user's query.

We take a zero-shot approach to exploratory search: the user manually assembles vocabulary elements like color, speed, size and type into a graph. Given that information, we perform an initial downsampling of the archived data, and design a novel dynamic programming approach based on genome-sequencing to search for similar patterns. Experimental results indicate that this approach outperforms other methods for detecting activities in surveillance video datasets.

Second, we address the problem of representing complex activities that take place over long spans of space and time. Subgraph and graph matching methods have seen limited use in exploratory search because both problems are provably NP-hard. In this work, we render these problems computationally tractable by identifying the maximally discriminative spanning tree (MDST), and using dynamic programming to optimally reduce the archive data based on a custom algorithm for tree-matching in attributed relational graphs. We demonstrate the efficacy of this approach on popular surveillance video datasets in several modalities.

Finally, we design an approach for successive search space reduction in subgraph matching problems. Given a query graph and archival data, our algorithm iteratively selects spanning trees from the query graph that optimize the expected search space reduction at each

step until the archive converges. We use this approach to efficiently reason over video surveillance datasets, simulated data, as well as large graphs of protein data.

Contents

List of Tables

List of Figures

List of Abbreviations

ARG	.....	Attributed Relational Graph
BP	.....	Belief Propagation
CCTV	.....	Closed Circuit Television
DP	.....	Dynamic Programming
GMM	.....	Gaussian Mixture Model
GT	.....	Ground Truth
GUI	.....	Graphical User Interface
HDP	.....	Hierarchical Dirichlet Process
HMM	.....	Hidden Markov Model
HOG	.....	Histogram of Oriented Gradients
KLT	.....	Kanade - Lucas - Tomasi
LSH	.....	Locality-Sensitive Hashing
MDSM	.....	Maximally Discriminative Subgraph Matching
MDST	.....	Maximally Discriminative Spanning Tree
MIT	.....	Massachusetts Institute of Technology
MST	.....	Minimum Spanning Tree
NP-Hard	.....	Non-deterministic Polynomial time hard
UAV	.....	Unmanned Aerial Vehicle
ROC	.....	Receiver Operating Characteristic
ROI	.....	Region of Interest
ST-AOG	.....	Spatio-Temporal And/Or Graph
STIP	.....	Spatio-Temporal Interest Points

Chapter 1

Introduction

1.1 Introduction

As cameras become cheaper and more ubiquitous, algorithms that reason over large video corpora are becoming increasingly vital. Faced with an overwhelming amount of information, users seek answers to both specific ('What's moving?') and abstract ('What's important?') questions about video. With nearly 1 billion hours of video on a single website like youtube, a web browser might be interested in automatically generating tags for video [Siersdorfer et al., 2009, Wang et al., 2012] to allow easier browsing. In retail, a security guard looking at a dozen monitors might want an algorithm to tell him what's moving [Barnich and Van Droogenbroeck, 2011, Chen et al., 2012], or what's unusual [Li et al., 2014]. A person browsing through family videos might be interested in finding all instances of a certain person [Li et al., 2002], a particular activity like running or jumping [Oneata et al., 2013, Sadanand and Corso, 2012]. A law enforcement officer presented with a wealth of street surveillance might be interested in higher-level activities, such as who is jaywalking, running a red light, or loading objects into a car [Zhu et al., 2013, Kwak et al., 2013, Castañón et al., 2015].

Of these manifold types of visual reasoning, we are primarily interested in semantic video search: looking for things in a video that have clear semantic meanings. Semantically-meaningful queries differ from concepts like cornerpoint features, textures, and integral images [Mita et al., 2005] because they are easily described by a person. Examples of semantic concepts are objects and activities; everybody knows what a basketball or jogging

Figure 1·1: Whether for identifying people in a crowd or figuring out what's worth looking at in surveillance footage, video search is a necessary part of many large video systems.

should look like, with few exceptions. There are two primary families of algorithms that reason over semantic concepts: classification and search engines.

A classification engine is typically provided with a set of videos of different classes and expected to label each of the videos with the correct class. In images and video, class tends to correspond to the object or activity occuring in the image or video. In many modern datasets [Schüldt et al., 2004, Yao et al., 2011, Blank et al., 2005, Rodriguez et al., 2008, Soomro et al., 2012, Marszałek et al., 2009] this is not a multilabel problem because each video in the corpus has only one correct label. Classification engines can learn their classes from training sets of exemplar videos [Weinland et al., 2007, Laptev et al., 2008, Oneata et al., 2013, Sadanand and Corso, 2012], human input, or a combination of both [Felzenszwalb et al., 2010, Raptis et al., 2012, Raptis and Sigal, 2013, Wang and Mori, 2009, Wang and Mori, 2011]. A classification engine is typically provided with a set of videos of different classes and expected to label each of the videos with the correct class. In images and video, class tends to correspond to the object or activity occuring in the image or video. In many modern datasets [Schüldt et al., 2004, Yao et al., 2011, Blank et al., 2005, Rodriguez et al., 2008, Soomro et al., 2012, Marszałek et al., 2009] this is not a multilabel problem because each video in the corpus has only one correct label. Classification engines can learn their classes from training sets of exemplar videos [Weinland et al., 2007, Laptev

et al., 2008, Oneata et al., 2013, Sadanand and Corso, 2012], human input, or a combination of both [Felzenszwalb et al., 2010, Raptis et al., 2012, Raptis and Sigal, 2013, Wang and Mori, 2009, Wang and Mori, 2011].

Once a classifier is trained, it can be applied in parallel to each video in a corpus. While training time can be prohibitive, in a real-world scenario, model training happens offline. Thus, the primary challenge of classification problems is to learn a model of sufficient fidelity to differentiate between the classes it trains on.

In contrast, search engines operate in two modalities. First comes the archival modality, where the engine is presented with the video corpus and has to assemble it in a format to be reasoned over quickly. Second is the search modality, where the search engine is provided a query activity or object and has to identify all locations in the video corpus where that query occurs.

Challenges of search include:

- *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, as well as one which scales well with the temporal extent of the video.

- *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, such as identifying abandoned objects and finding instances of cars performing U-turns.

- *Unpredictable event duration*: events in video are ill-structured. Events start at any moment, vary in length and overlap with other events. The system is nonetheless expected to return complete events whatever their duration may be and regardless of whether other events occur simultaneously.

- *Clutter and occlusions*: urban scenes are subject to high number of occlusions. Tracking and tagging objects in video is still a challenge, especially when real-time performance is required.

1.1.1 Representation and Models

Of course, efficient representation of large video corpora has been studied in other parts of the computer vision community. Videos and images contain a tremendous amount of redundant information, a fact exploited in both image [Taubman and Marcellin, 2012, Taubman, 2000] and video [Le Gall, 1991] compression. Since the earliest days of computer vision, approaches [Lowe, 1999, Li et al., 2002, Efros and Berg, 2003] have tried to find ways to capture the minimum essential information required from a video to accomplish a particular task. Many relevant approaches are compared in [Mikolajczyk and Schmid, 2005] for the purposes of local interest region detection. Of these, the Scale-Invariant Feature Transform [Lowe, 1999], a descriptor of interest points in images, was the earliest popular feature for representing objects in videos. Another popular feature is Haar-like features [Mita et al., 2005, Zhang and Viola, 2008], based on wavelets, that were used extensively in face detection [Viola and Jones, 2004].

Moving from images to videos introduces time as a third dimension. Videos tend to contain more redundant information than images, as an immobile background (with unchanging recording conditions) yields the same set of pixels at every point in time. Thus, video feature description tends to focus on areas of motion, with the assumption that the things of interest in video move. Some approaches [Scovanner et al., 2007] directly extend the SIFT framework, while others extract pyramids of integral images [Dollar et al., 2014, Dollár et al., 2009] to classify local patches.

Acquiring these models for video corpora has been the subject of much research over the last 30 years. Exemplar-based approaches [Tianmin Shu et al., 2015, Çeliktutan et al., 2013] are provided sets of training examples for each potential object or activity to be de-

tected in the corpus. However, it has generaly been recognized that the number of training examples cannot match the increasing complexity of classifiers and activities [Ryoo and Aggarwal, 2010, Kwak et al., 2013].

In the face of increased query complexity, user-driven queries have become a popular way to access more complex models. Early works [Chang et al., 1998, Zhong and Chang, 1999] focused on enabling users to draw simple motions, while later approaches [Hu et al., 2013] allowed sketches for object detection in videos.

These approaches present a good-starting point for a system which aims to efficiently represent large video corpora for fast search. Most of these features have reached the point where they can be reasonably extracted in real-time from streaming video, and their memory footprint is far less than that of the raw video itself.

1.1.2 Classification and Detection

In the search problem, we ask a system to find all the examples of a particular query. For certain datasets [Yao et al., 2011, Soomro et al., 2012, Schüldt et al., 2004, Marszałek et al., 2009, Blank et al., 2005, Rodriguez et al., 2008], where each element of the corpus (video or image) contains exactly one object or activity, such as jumping rope or running, search can be a specialization of object or activity classification. Ample work has been done building video models and distance metrics between them, to answer the question 'Do these contain the same thing?'. The earliest of these models are histogram models [Laptev et al., 2008, Oneata et al., 2013, Sadanand and Corso, 2012], which accumulate local features into a histogrammed representation of the video in question and use statistical distance measures like earth-mover's distance [Rubner et al., 2000] to compare distributions. Other approaches track spatio-temporal interest points through the video and compare trajectories [Ikizler and Forsyth, 2008, Wang et al., 2013, Wang and Schmid, 2013, Zhang et al., 2014a]. Dense trajectory approaches measure local optical flow and directly compare motion via correlation, but tend to be too computationally expensive to perform on large video corpora

in real time.

Given our stated desire to scale well with the archive video corpus size, we are particularly interested in graphical models of activity. These models were originally used to search for objects in images [Felzenszwalb et al., 2010, Tianmin Shu et al., 2015]. Each image was represented with an attributed, relational graph (ARG), with each node representing a local feature, and each edge encoding the spatial relationship between nodes. In classification, images are compared via graph matching [Riesen and Bunke, 2009, Berretti et al., 2007, Cho et al., 2010], which produces an assignment of nodes from one image's graph to the other's. The distance metric for this association typically takes the form of a sum of kernel functions between matched node and edge pairs.

Graphical representations of video have also been popularized for activity retrieval in particular video corpora. Specifically, when each video in a corpus contains a single activity, graph-matching approaches [Çeliktutan et al., 2013, Ma et al., 2013] have proven to be effective in comparing activities.

Unfortunately, while in many datasets each element of the corpus has one and only one activity happening, this is not true in some real-world applications. In video surveillance, the corpus can be a single extremely long video. This has lead to a number of efforts within the search community to reduce this problem to classification, where it can be solved easier. In news video or television, the video can be divided into 'shots', [Song and Fan, 2006,Sivic and Zisserman, 2003, Sujatha and Mudenagudi, 2011], with each shot having a single topic. However, for generic video corpora without a-priori knowledge, there has only proven to be a single reliable way to perform pattern recognition for search: to run a sliding window across each video.

Sliding window approaches are an extremely popular way to transform a search problem into a classification problem, and they see extensive use in both object [Viola and Jones, 2001, Philbin et al., 2007, Sivic and Zisserman, 2003, Johnson et al., 2015] and activity de-

tection [Gaur et al., 2011, Lin et al., 2014]. If a query has a maximum size in space-time, a sliding window approach will attempt to segment a video into overlapping space-time boxes of that size, and independently classify each of the boxes.

In certain circumstances, applying sliding windows over every video in a corpus to detect activities can be effective. In movies or youtube videos with narrow field of view, an activity can be expected to take up the whole view, so the sliding window only has to slide over time. Likewise, those videos are more likely to contain only one thing going on within any window because of the limited view.

In more generic video modalities, particularly surveillance video [Oh et al., 2011, Ferryman, 2006], the assumptions that enable effective sliding window search fall apart. First, surveillance video corpora are extremely large, rendering a sweeping approach unappealing. Second, the amount of background activity is generally far larger than the activity that matches the query. Finally, the maximum size of a query often spans a significant amount of space and time, so any given window that contains the query activity also likely contains a significant amount of background activity.

Subgraph matching has been used in a diverse array of fields to identify sub-structures of larger graphs. Sub-graph and graph matching are also commonly used in image search applications [Christmas et al., 1995, De la Torre, 2012] as well as image duplicate detection [Zhang and Chang, 2004]. In image search and image duplicate detection, the goal is the same: to match the graph extracted from a query image to the graphs extracted from each image in a corpus. Subgraph matching is used to allow these algorithms to be robust to small amounts of clutter. These problems differ significantly from ours - because the corpus is divisible into a series of relatively small graphs, one for each image, it can be solved efficiently by a series of subgraph matching problems.

In our problem, we are looking at an extremely asymmetric subgraph matching problem with a dense archive graph containing millions of nodes and a query graph containing up

to a dozen nodes. Algorithms for subgraph matching in this context are more often used in bioinformatics [Bonnici et al., 2013,Sun et al., 2012,Zhang et al., 2010,Koutra et al., 2011], a field where query graphs generally represent proteins or amino acids to be searched for in a larger dataset. Because subgraph matching is provably NP-hard [Ullmann, 1976], efficient algorithms [Sun et al., 2012, Zhang et al., 2010] focus on search space reduction before actually solving the subgraph matching problem.

1.1.3 Overview and Notation

A video search system operates in two modalities: archival and search. During the archival step, we process each video in a video corpus and extract features from a pre-defined feature vocabulary $\mathcal{F}_v$. These features are all local - they are associated with a certain area or location in space/time in a specific video. We store these locations in an inverted index by feature and feature value. So, if the system needs to find all places in a video corpus where we found the color 'red', it would go to the color index and look in the 'red' bin, which would contain all location/video pairs where that color was found.

In our later systems in Chapters 4 and 5, we also focus on the relationships between these features. For each pair of locations that might potentially satisfy a relationship in our relationship vocabulary $\mathcal{F}_e$, we check to see if that relationship exists. If so, we hash the pair of locations that satisfy the relationship to an inverted index based on the relationship they satisfy. As an example, in the 'same as' bin, we would have all pairs of locations where we believe the same object is at both locations.

At the end of our archival step, we have a bunch of indices, some for local features and some for relationships between them. These indices implicitly represent an archive graph $A = G(V(A), E(A))$. In this graph, each node $v \in V(A)$ is the collection of features at a specific spatiotemporal location in a given video of the corpus. An edge $e \in E(A)$ exists between two nodes if a relationship has been found between them.

The query step begins when a user wants to find a particular activity in the video cor-

pus. He assembles features into nodes ('a large, red, moving object') and then adds edges containing relationships. The result is the query graph $Q = G(V(Q), E(Q)$. In search, our goal is to find the set of subgraphs of A which best match the query graph Q. Alternately, our goal is to find a set of matchings $\mathcal{M}$ where $m \in \mathcal{M} : V(Q) \rightarrow V(A)$.

We rank these matchings according to a score function $S(m, Q, A)$ which incorporates distances between nodes $d_v(v_a, v_i)$, distances between edges $d_e((v_a, v_b), (v_i, v_j))$, and also penalties for failing to find a matching for an edge entirely. Finally, we return in descending order the clipped video segments corresponding to the set of matchings for which $S(m, Q, A)$ is larger than threshold τ.

1.1.4 Our Contributions

In this dissertation, we describe a functional system that can process and archive large video corpora in real-time. Later, when provided a query by a user, our system is able to find and rank matches to the query in a matter of seconds and display these results. The novel components of this system are:

1. **Inverted indices for efficient archive downsampling.** In Chapter 3 we introduce an inverted hashing scheme for simple features. In 4, we use these and other indexing techniques to dramatically downsample a video corpus to the set of features that are potentially relevant to a given query. This allows us to efficiently reason over large video corpora without prior knowledge and without that each corpus being subdivided into small videos.

2. **Sub-graph matching in video search.** In Chapter 3, we introduce temporal relationships on simple features to find a wide variety of user-driven queries using a novel dynamic programming approach. In Chapter 4, we expand this approach to include spatial relationships and search for arbitrary graphs in large videos. In particular, we use a subgraph matching approach to render our method agnostic to background

noise. Other approaches use bipartite matching [Lin et al., 2014] or require a tree-based query [Tianmin Shu et al., 2015], and are thus unable to represent activities with a similar degree of structural complexity.

3. **Tree-matching for space-downsampling in video search.** We introduce a novel method for successive search-space reduction based on selecting the Maximally Discriminative Spanning Tree in Chapter 4. In Chapter 5, we extend this method to iteratively reduce the search space based on the statistics of the dataset. This approach significantly outperforms contemporary algorithms for search space reduction in subgraph matching like random trees.

Chapter 2

Related Work

In this work, we propose a fully operational video surveillance system that utilizes a number of pre-existing approaches, as well as comparing to others. While we cite these works directly, for purposes of clarity we explain the most important of them in this chapter.

2.1 Locality Sensitive Hashing

In order to be able to quickly return a set of matches to a given search, we hash features to inverted indices. Features like object type, are hashed via regular hash tables because there is no notion of distance between types. However, for continuous features like motion direction, object size, and persistence or activity, we use Locality-Sensitive Hashing [Datar et al., 2004] (LSH) such that a query to the inverted index returns (with high probability) all locations that have a feature within radius r of the query. This has the advantage of mitigating, to some extent, user and feature extraction errors that cause a mismatch between what the user expects and what is stored in the archive.

For our implementation of Locality Sensitive Hashing, we have three parameters: collision radius r, dimensionality of the feature data d and number of tables N. Our goal is to choose a set of hash functions $H_1, ..., H_N$ where $H_i : R^d \rightarrow \mathbb{Z}$. Following the approach in [Datar et al., 2004], we use Algorithm 1.

This creates N hash functions for a given feature type, like color, with dimensionality d. As video is processed, we get measurements of features, paired with spatiotemporal locations ($\{x, y, t\}$ for video). For example, we might see the color red in the first frame

Algorithm 1 LSH Function Creation

1: **procedure** CREATE LSH FUNCTIONS(r, d, N)
2: **for all** $i \in \{1, ..., N\}$ **do** ▷ For each table
3: Draw each element of $\mathbf{a} \sim N(0, 1)$ ▷ Each element of $\mathbf{a}$ is drawn from a univariate normal distribution
4: Draw $b \sim unif(0, r)$ ▷ The offset b is drawn uniformly from the range $[0, r]$
5: $H_i(\mathbf{x}) \leftarrow \lfloor \frac{\mathbf{a} \cdot \mathbf{x} + b}{r} \rfloor$
6: **end for**
7: **end procedure**

in the upper-left hand corner. For each of our N hash functions, we compute $H_i(\text{red})$, and store the spatiotemporal location in that bin.

When somebody asks for all locations with a color, like purple, we compute $H_i(\text{purple})$ for each of our hash functions, and look at the contents of those N bins. If a given location appears in more than τ_{color} percent of the bins, we consider that location to contain the color purple.

We independently create sets of hash tables for each feature type in a dataset.

2.2 Smith-Waterman Algorithm

Some of our work in Chapter 3 extends the Smith-Waterman algorithm for dynamic programming. This algorithm takes in two strings of characters and attempts to compute the match that is minimally distorted. The model for distortion is that the string in a is present in b, except that characters in a may be missing ("deletion"), and that in the middle of an otherwise perfectly good match there may be extraneous characters in b ("insertion"). We are given fixed penalties w_k and w_l for insertion and deletion, respectively. Second, between each element $a_i \in \mathbf{a}$ and $b_j \in \mathbf{j}$ is a score $s(i, j)$ indicating how well they match.

Given these quantities, our goal is to pick a starting point t where a_0 maps to b_t, and an optimal series of insertions and deletions to maximize the sum of the matching scores minus the penalties. To achieve this, Smith and Waterman set up a matrix $H \in n \times m$, as

follows:

$$H(i,0) = 0 \quad \forall i = 1...n \tag{2.1}$$

$$H(0,j) = 0 \quad \forall j = 1...m \tag{2.2}$$

$$H(i,j) = max(0, H(i-1,j-1) + s(a_i, b_j), H(i-1,j) + w_k, H(i,j-1) + w_l) \tag{2.3}$$

Most importantly, when calculating this matrix, the Smith-Waterman algorithm keeps a pointer to which of the four options (representing a new start, an element happening correctly, a deletion or an insertion) were chosen at each element $H(i,j)$.

To compute the optimal match, we use backtracking. The Smith-Waterman algorithm selects the maximum element of H and then iteratively asks "Which element generated the maximum value that we used?". This backtracking process produces a series of matches, insertions and deletions corresponding to the optimal match.

2.3 Sliding Window Approaches

A number of approaches [Lin et al., 2014,Tianmin Shu et al., 2015] utilize sliding windows to reduce the space over which they have to search. In [Lin et al., 2014], the goal is to match a textual description of an activity to the KITTI [Geiger et al., 2012] dataset. This dataset contains recordings of a Volkswagon-mounted camera driving around Karlsruhe, Germany. In [Lin et al., 2014], a small subset of this dataset is used: 21 30-second video clips are used - 13 for training their set of concepts, and eight as a search corpus. The selection of eight short videos as a search corpus can be viewed as a specialization of a 30-second sliding window. A complete sliding window approach for queries with 30 second maximum length would be to take every time interval $[t_0, t_0 + 30]$ in the video corpus and consider that an independent video as a result. Presumably for the sake of computational

complexity, [Lin et al., 2014] elect to only look at eight such videos.

For each video, they solve a bipartite matching problem. Given a textual query like "A car is moving forwards", they convert it to a set of features $u \in U$, and have a set of features $v \in V$ for each video segment. They compute a matching score h_{uv} between feature u and feature v based on appearance and motion, and compute a global assignment score $S\mathbf{y}$ and assignment vector $\mathbf{y} \in \{0,1\}^{U \times V}$:

$$S(\mathbf{y}) = \sum_{uv} h_{uv} y_{uv} \tag{2.4}$$

$$\max_{\mathbf{y}} S(\mathbf{y}) \tag{2.5}$$

subject to the constraints that y assign each element of U to one or less elements of V. Videos are ranked for the user in descending order of their global assignment score.

Compared to our approach, this method does not take advantage of relationships between features - it is looking to find a single optimal match for every feature in the query. However, common to a lot of sliding window approaches, by using a relatively small sliding window in time (30 seconds), they do implicitly enforce that all the features must occur within 30 seconds of one another, which is a sort of relationship. When features alone are sufficient to differentiate the activity in question, and the activity has limited spatiotemporal extent, this approach can be effective.

[Tianmin Shu et al., 2015] use a similar approach to dividing an aerial events dataset. Using combinations of ground truth, tracks, and manual object annotations, they partition tracks into non-overlapping sets. For each event, they have learned a spatio-temporal and/or graph (ST-AOG) that is a series of trees (in space) linearly sequenced in time. For each partitioned set of tracks, they slide a window of length T, and at each window solve a dynamic programming problem that assigns detections in the tracks present in that window to elements of the graph. Videos are classified as the event that has the best match within

their windows.

This method does take into account relationships between features. However, the acceptable structure for an input is limited to trees because of the necessity of solving via dynamic programming. Further, it assumes that moving objects within the data can be clearly partitioned into disjoint sets. In more realistic surveillance datasets, this is not the case - objects are continually moving.

In this work we aim to produce a system that takes in arbitrary graphs (expressed by our vocabulary) and makes few assumptions about the video corpora that we are searching over. We do not assume that the dataset is partitionable, nor that our input graph has any particular structure. Unlike the above approaches, we also do not learn models from exemplar video. In our work, we compare to a feature-accumulation approach like [Lin et al., 2014] to show the utility of incorporating relationships between features, and to a sliding window approach later. In our experiments, we merge spatiotemporally overlapping windows when appropriate to attain video intervals for users.

Chapter 3

Dynamic Programming for Activity Search

In this chapter, we present a content-based retrieval method for long surveillance videos both for wide-area (Airborne) as well as near-field imagery (CCTV). Our goal is to retrieve video segments, with a focus on detecting objects moving on routes, that match user-defined events of interest. The sheer size and remote locations where surveillance videos are acquired, necessitates highly compressed representations that are also meaningful for supporting user-defined queries. To address these challenges, we archive long-surveillance video through lightweight processing based on low-level local spatio-temporal extraction of motion and object features. These are then hashed into an inverted index using locality-sensitive hashing (LSH). This local approach allows for query flexibility as well as leads to significant gains in compression. Our second task is to extract partial matches to the user-created query and assembles them into full matches using Dynamic Programming (DP). DP exploits causality to assemble the indexed low level features into a video segment which matches the query route. We examine CCTV and Airborne footage, whose low contrast makes motion extraction more difficult. We generate robust motion estimates for Airborne data using a tracklets generation algorithm while we use Horn and Schunck approach to generate motion estimates for CCTV. Our approach handles long routes, low contrasts and occlusion. We derive bounds on the rate of false positives and demonstrate the effectiveness of the approach for counting objects, motion pattern recognition and abandoned object applications.

3.1 Introduction

Video surveillance camera networks are increasingly common, generating thousands of hours of archived video every day. This data is rarely processed in real-time and primarily used for scene investigation purposes to gather evidence after events take place. In military applications UAVs produce terabytes of wide area imagery in real-time at remote/hostile locations. Both of these cases necessitate maintaining highly compressed searchable representations that are local to the user but yet sufficiently informative and flexible to handle a wide range of queries. While compression is in general lossy from the perspective of video reconstruction it is actually desirable from the perspective of search since it not only reduces the search space but it also leverages the fact that for a specific query most data is irrelevant and pre-processing procedures such as video summarization is often unnecessary and inefficient. Consequently, we need techniques that are memory and run-time efficient to scale with large data sets, and be able to retrieve video segments matching user defined queries with robustness to common problems, such as low frame-rate, low contrast and occlusion.

Some of the main challenges of this problem are:

1.) Data lifetime: The model must scale with the growing size of video data.

The continuous stream of video data requires a model that can handle this growing temporal size. This requires periodic processing of the new data as it streams in the system.

2.) Unpredictable events:

Some events are rare, yet they could be of great interest. For instance a person dropping a bag and continuing walking, or a car violating a traffic light are examples of not so common events. Yet their detection is valuable as it could help in forcing the law and preventing terrorist attacks. The system should be able to support such events.

3.) Clutter and Low Contrast:

Clutter and low contrast often generate motion errors. This will have a direct impact on

processing temporal queries. Low contrast is common in airborne footage while clutter is frequent in videos of urban areas.

4.) Unpredictable event duration:

Events vary in duration, co-occur with each other and could start and end anytime. Accurate estimation of an event time window is important. Such task is made more difficult in regions of high clutter and poor contrast.

This chapter extends our method for content-based retrieval [Castañón and Caron, 2012] to the domain of airborne surveillance. In [Castañón and Caron, 2012] we presented a query-driven approach where the user draws routes manually. The algorithm then retrieves video segments containing objects that moved in the user supplied routes. Our technique consists of two steps: The first extracts low level features and hashes them into an inverted index using locality-sensitive hashing (LSH). The second stage extracts partial matches and assembles them into full matches using Dynamic Programming (DP).

While [Castañón and Caron, 2012] dealt with CCTV data, in this chapter we handle motion patterns in aerial view images shot from a UAV (airborne data). Handling such data is more challenging than handling CCTV due to their lower frame rate and lower contrast (see Fig. 3·1). As a result certain features such as optical flow information (extracted using Horn and Schunck technique [Horn and Schunck, 1981]) are rendered highly noisy. Hence for airborne data we first generate tracklets, using ideas from [Pitié et al., 2005, Baugh and Kokaram, 2010], and then generate motion estimates from those tracklets. We use these motion estimates in combination with our previous two-step process to retrieve routes generated by different objects (such as cars and human) while handling long routes, low-contrast, occlusion, and outperforming current techniques.

This algorithm extends our preliminary work in [Castañón and Caron, 2012] to videos collected via airborne surveillance. Airborne video differs significantly from standard surveillance video: the resolution and contrast of objects of interest is extremely low, and

(a)

(b)

Figure 3·1: Top: Examples of CCTV footage. Bottom: Example of Airborne footage. Blue and green boxes show zoomed-on regions. Objects of interests (cars in this figure) usually have much lower contrast in Airborne footage (see red boxes) over CCTV footage. This makes processing Airborne footage more challenging than CCTV.

the frame-rate is extremely low. This poses a number of problems to our standard approach - due to low frame-rate, objects are effectively obscured for significant periods of an action, as the camera is not taking a picture. Likewise, low resolution, low contrast and low frame rate make estimation of optical flow unstable. To address these issues, we perform significant filtering work on the raw video both to identify targets and suppress camera artifacts. We also develop a rudimentary tracker which operates effectively in harsh conditions. We use the results of this tracker to create a low-level feature set for exploitation by the approach described in [Castañón and Caron, 2012]. Note that we are not interested in obtaining long-term (i.e. perfect) tracks; instead we are interested in tracklets that are good enough for our retrieval problem.

Our approach differs significantly from current approaches [Kuettel et al., 2010, Xiang and Gong, 2008]. In this chapter we assume that there is no prior knowledge on the queries and hence we first extract a full set of low-level features. In addition unlike scene understanding techniques [Kuettel et al., 2010, Xiang and Gong, 2008], we do not incorporate a training step as this would scale unfavorably with the magnitude of the video corpus. Instead, our technique exploits temporal orders on simple features, which in turn allows examining arbitrary queries and maintains a low false rate. We show significant improvement over scene-understanding methods [Kuettel et al., 2010], both in terms of search accuracy and computational complexity. We also show how tracklets are better suited for airborne footage than optical flow. Results for CCTV data are also included.

3.2 Overview

Our procedure consists of two main stages. The first reduces the problem to the examined data while the second reasons over that data. Fig. 3·2 shows this procedure in more detail. Low level features are continuously extracted as data streams in. Here features as activity, object size, color, persistence and motion are used. For CCTV footage motion is

estimated using the Horn and Shunck approach [Horn and Schunck, 1981], while for Airborne footage motion is estimated using tracklets. LSH [Gionis et al., 1999] is then used to hash the extracted low level features into a fuzzy, light-weight lookup table. LSH reduces the spatial variability of examined features and reduces the queries search space into a set of *partial (local) matches* .

The second step of our search algorithm optimizes over the partial matches in order to generate *full matches*. Here video segments of partial matches are combined to fit the examined query. Our search approach simplifies the problem from examining the entire video corpus into reasoning over the partial matches that are relevant to our query. This feature is advantageous especially in examining surveillance videos where hours of video may not be of interest to an observer. Hence our approach reduces the search workload considerably.

We examine two approaches for generating full matches. The first is a greedy approach based on exhaustive examination of partial matches. The second approach exploits temporal ordering of component actions. Here Dynamic Programming (DP) processes partial matches and finds the optimal full match for a given query. We present two DP versions, one that uses optical-flow (for CCTV) and another that uses tracklets (for Airborne). Our technique outperforms current approaches and we show that increased action complexity drives false positives to zero.

3.3 Feature extraction

In this section we explain how to extract relatively basic features. In our previous work [Castañón and Caron, 2012], we observe that the retrieval process is not strongly feature dependent, and use coarse features both for the purposes of robustness and speed of computation to address problems of data lifetime. For purposes of completeness, we present that extraction process in Section 3.3.2.

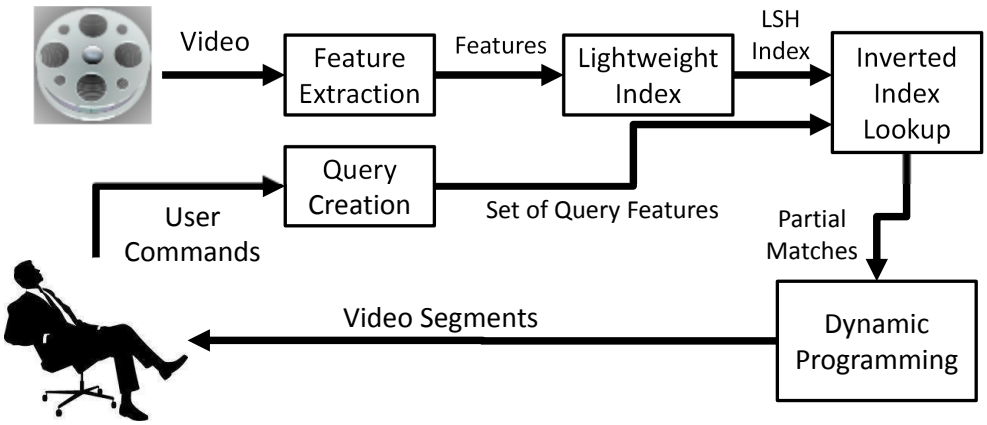

Figure 3·2: Our video search framework. As data streams in features are extracted and inserted into a lightweight index. The user defines query of interest and partial matches are generated through inverted lookup index. Partial matches are then combined into full matches by means of Dynamic Programing. The final output is a video segment matching the user query.

Feature extraction for airborne video is significantly more difficult. As the video is low frame-rate, low-contrast and colorless we focus on motion features. Due to the lack of image quality, histograms of optical flow are inaccurate - as such, we perform short-term tracking and derive motion information based on these tracklets. This process in described in Section 3.3.3.

For our system to work, we need a spatio-temporal structure to summarize local motion features. Although we could use 3D atoms (3D spatio-temporal block of video) as in [Wang et al., 2009], we found empirically that this structure is very sensitive to slight variations in the dynamics and shape of moving objects. For example, a car running through a 3D atom would leave a different signature than if it were slightly shifted and went through half that atom and half a neighboring atom. Instead, we use a hierarchical structure which aggregates information from neighboring atoms. This structure is described in the following section.

3.3.1 Structure

In this chapter, we treat a video as a spatiotemporal volume of size $H \times W \times F$ where $H \times W$ is the pixel image size and F denotes the total frames in the video. The video is temporally partitioned into consecutive *documents* containing A frames. We illustrate in Fig. 3·3 how to tile a given frame of video using $B \times B$ squares of pixels. We form *atoms* by amalgamating consecutive tiles over A frames. Depending on video size and frame rate, for CCTV we chose B to be 8 or 16 pixels and A to be 15 or 30 frames. For Airborne footage however we used $B = 20$ and $A = 1$. A much smaller value of A is used here to capture the fine-detailed nature of tracklets. As video is streaming, we extract features in real-time, assigning a set of features (see Sec. 3.3.2 and Sec. 3.3.3) to each atom n to describe its content.

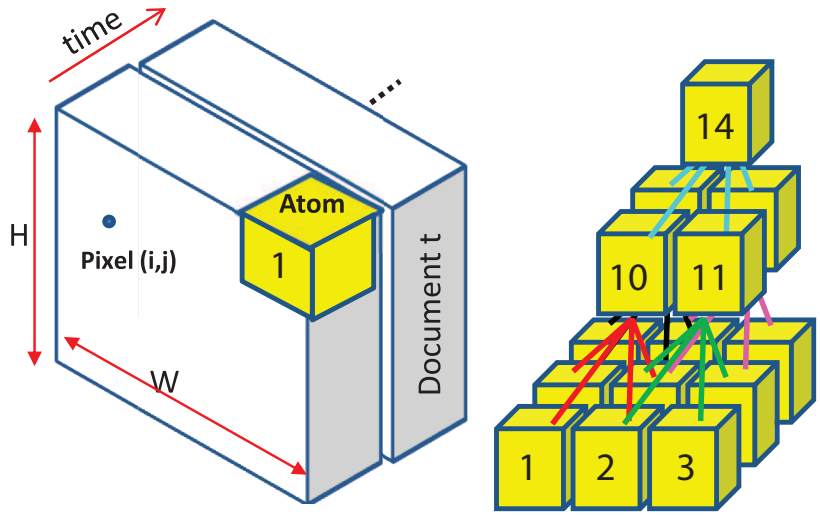

Figure 3·3: (Left) A video of size $W \times H \times F$ divided into *Documents* t. Each *document* contains non-overlapping A frames, and each frame is divided into tiles of size $B \times B$. Temporal aggregation of tiles over A frames generate an *atom*. (Right) Tree structure of atoms. Here every set of four adjacent atoms are linked to the same parent. This forms a set of partially overlapping trees.

In order to render our algorithm robust to location and size variability over sets of de-

tected features, we create a spatial pyramid structure. For this implementation, we chose a quaternary pyramid (tree) where each element has 4 children, all spatially adjacent. Because the pyramids overlap, a k-level pyramid contains $M = \sum_{l=1}^{k} l^2$ nodes, as seen in Fig. 3·3. Given this formulation, a document with $U \times V$ atoms will be partitioned into $(U - k + 1) \times (V - k + 1)$ partially overlapping pyramids for indexing. As an example, we construct a depth-3 pyramid tree in Figure 3·3 to aggregate 3×3 atoms into $M = 14$ nodes.

We compute the feature vector for each node of the tree by an aggregation operator over each of its child nodes. Given node n and its four children a, b, c, d, we formalize aggregation in Eq. (3.3.1).

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

where ψ_f denotes the aggregator over relevant features. The output of this aggregator is $\mathbf{x}_f^{(n)}$, a concatenation of all features. Sec. 3.3.2 and Sec. 3.3.3 explain the aggregation process in further detail. For each atom we extract a number of features, so the aggregation of a group of $k \times k$ atoms yields $\{\mathrm{tree}_f\}$ one for feature tree for each feature f. Each tree_f contains a list of M feature instances, namely $\mathrm{tree}_f = \left\{ \mathbf{x}_f^{(n)} \right\}$, because there are M nodes in every k-level tree.

3.3.2 Feature Extraction for CCTV footage

Existing work [Meessen et al., 2006, Stringa and Regazzoni, 1998, Yang et al., 2009] has shown features such as color, object shape, object motion, and tracks to be effective at the atom level. Because of computational limitations and the constant data renewal inherent to the surveillance problem, we chose to exploit local processing to enable real-time feature extraction. We assume a stable camera (common in surveillance, though an unstable camera could utilize existing stabilization options) and compute a single feature

value for each atom. Note that this feature value does not require any understanding of how many objects are in the atom or what it is composed of - they are a a simple amalgamation of values computed at the pixel level. In our current implementation, we exploit five features:
(1) Activity x_a: this feature is associated with moving objects that we detect with a background subtraction method [Benezeth et al., 2010]. The background is computed using a temporal median filter over 500 frames of the video and subsequently updated using a running average. For every atom, we compute the proportion x_a of pixels that are both not part of the background and have sufficient motion magnitude. The aggregator is defined as the mean activity of the four children.
(2) Size x_s: In order to detect motion, we create a binary activity mask differentiating activity from background. We then perform connected component analysis to identify moving objects, with the number of pixels in an object yielding the size. The aggregator for size is the median of the non-zero children, with a default value of zero for all-zero children.
(3) Color $\mathbf{x}_c$: We calculate color by computing a histogram over an atom's non-background pixels in 8/4/4 quantized HSL color space. The aggregator is the bin-wise sum of the children's histograms. Because proportions are important, we do not normalize during this stage.
(4) Persistence x_p: The persistence measure identifies objects that are not part of the background, but stay in one place for some time. To compute it, we accumulate the binary activity mask from background subtraction over time, and measure the percentage of pixels that have been active for longer than a threshold. Non-background objects which are immobile thus obtain a long persistence measure. The aggregator for persistence is the maximum value among the four children.
(5) Motion $\mathbf{x}_m$: In order to compute aggregate motion in an atom, we quantize pixel-level optical flow, extracted using Horn and Schunck's method, into 8 cardinal directions. We utilize an extra "idle" bin to denote an absence of significant motion (low magnitude flow),

yielding a 9-bin histogram of motion. The aggregator for motion is the bin-wise sum of the motion histograms for the four children. We note that, like color, we store un-normalized histograms to maintain relative proportion.

We extract all of these features from a real-time streaming video. Upon creation, we assign 5 descriptors to each atom, $\{x_a, x_s, \mathbf{x}_c, x_p, \mathbf{x}_m\}$. We assemble an atom's descriptors into 5 feature trees $\{\text{tree}_a\}$, $\{\text{tree}_s\}$, $\{\text{tree}_c\}$, $\{\text{tree}_p\}$, $\{\text{tree}_m\}$, which form the foundation of the our approach to indexing in Sec. 3.4. After indexing, we discard the feature content; the actual value of the features is implicitly encoded in our lightweight index. Note that the choice for the specific aggregation functions (mean, max, and median) came after empirical validations.

3.3.3 Feature Extraction for Airborne Footage

First, we register airborne images onto a common reference frame. Then, in order to be robust to issues of low contrast and frame rate, we extract tracklets and derive features directly from them. Our tracklet extraction process uses a Viterbi-style algorithm with ideas from Pitie et al. [Pitié et al., 2005] and Baugh et al. [Baugh and Kokaram, 2010]. At each frame we generate a set of candidates and update existing tracklets with these candidates based on distance in feature and position space.

We are supplied with Airborne footage stabilized with respect to the global camera motion. In order to identify moving objects (cars and people) we use frame differencing. Consecutive frames are used for cars, and frames spaced by 10 for slower-moving humans. We use a very small detection threshold to ensure all false negatives are eliminated. Cars are detected if a frame difference of more than 15 gray-scale levels is observed. A smaller threshold of 10 is used for humans detection as they have lower contrast. Some detection errors often occur around borders of trees and objects (see Fig. 3·4, red region). Other artifacts are caused due to local motions as the ones generated by the moving sea waves (see Fig.3·4, blue region). Such artifacts are removed through filtering by size. Here we

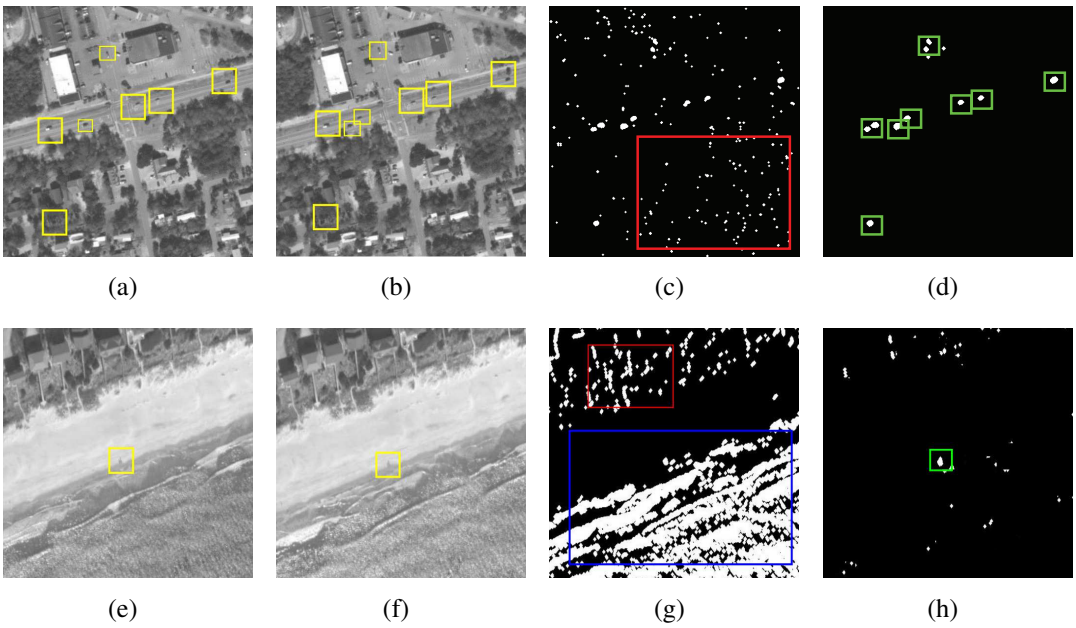

Figure 3·4: First row, from left: Two consecutive frames showing cars (shown in yellow) moving near a junction, (c) Frame subtraction and (d) result after applying morphological opening. The final result (d) is void of noisy data shown in red in (c). Second row, from left: Two frames showing a group of people (shown in yellow) walking near a beach, (g) detection after applying morphological opening and (h) detected candidates of size more than 150 pixels in green.

first detect connected bodies and remove any body that contains more than 150 pixels.

An example of candidate selection with artifacts removal is shown in Fig.3·4. Note that filtering by size eliminates the vast majority of false alarms. This is evident by examining the blue and red rectangles of Fig. 3·4 before and after filtering (third and fourth columns respectively). Quantitative results show that this filtering reduces false alarms by 72.3% and 96.5% for cars and humans respectively. This is taken as the average over processing 1000 frames for each of Task 12 and 13 (see Table 3.1). It is worth noting that our algorithm only detects moving vehicles and humans. Stopped vehicles are dealt with by our dynamic programming algorithm which we introduce in Section 3.5.2.

Given a set of detection candidates, our goal is to associate them over time for the purposes of computing tracklets. To do so, we set up and solve a Viterbi Trellis as described

in [Baugh and Kokaram, 2010]. Given a set of detection candidates in frame n and $n+1$, we estimate the temporal matching cost of each detection pair:

$$\begin{aligned}\mathcal{C}_{l,m} &= \Lambda_1\|p_l - p_m\| + \Lambda_2|z_l - z_m| \\ &+ \Lambda_3 f_3(s_l, s_m) + \Lambda_4 f_4(c_l, c_m) + \Lambda_5 f_5(p_l, p_m). \end{aligned} \tag{3.1}$$

Here l and m denote the examined candidate of frame n and $n+1$ respectively. $(p\ s\ z\ c)$ are the position, shape, size and color of each examined candidate. $\Lambda_1 ... \Lambda_5$ are tunable weights to emphasize the importance of each term. Position is defined as the geo-registered location, shape is the binary mask of the occupied pixels and size is the number of pixels in the examined mask. The purpose of the five elements of Eq. (3.1) is to ensure that the target: (1) has not moved too far between n and $n+1$, (2) is roughly the same size, (3) the same shape, (4) the same color and (5) it is roughly where we expect it to be given its previous motion and location.

s_l and s_m are binary detection masks for the targets under examination, where $f_3(s_l, s_m)$ is the mean absolute error between both masks. Given the color measurements of the examined candidates c_l and c_m, we fit a Gaussian mixture model (GMM) for c_l being $G(c_l)$ [Bouman et al., 1997]. We then estimate $f_4(c_l, c_m)$ as the error of generating c_m from $G(c_l)$ [Bouman et al., 1997]. To calculate $f_5(p_l, p_m)$ we first estimate the expected location of l in the next frame $(n+1)$ given its current location in frame n. The expected location is estimated using a 3 frame window and a straight line, constant acceleration model. $f_5(p_l, p_m)$ is then taken as the absolute difference between the expected location of l in $n+1$ and the actual location of m.

Given the matching costs for all possible l and m pairs, we approximate the two-dimensional assignment problem by greedy assignment for computational efficiency. That is we match the pairs with least matching cost first, remove them, match the next pair, remove them and keep doing so until all candidates in frame n are matched with candidates

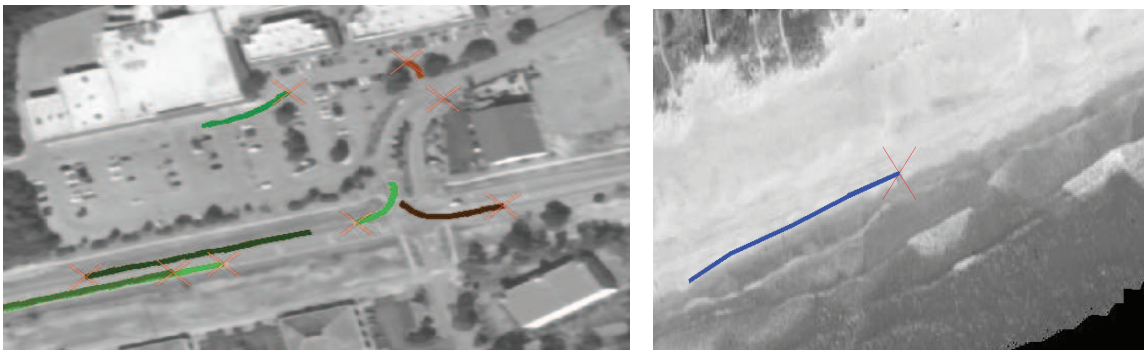

Figure 3·5: Examples of tracklets generated by the technique discussed in Sec. 3.3.3. Here tracks start with a red cross and their tails have different color. The example of the left shows tracklets generated for cars while the example on the right shows tracklet generated for a group of people walking on the beach.

in frame $n + 1$. Unmatched candidates can begin tracks if we find matches for them in subsequent frames, while tracks which are unmatched for enough frames are terminated. Note that this is a one-to-one assignment process hence no candidate in either frame n or $n + 1$ can have more than one match. This assignment, plus efficient computation of f_4, yield a simple tracking approach that can scale to large datasets.

Fig. 3·5 shows examples of tracklets generated by the technique discussed in this section. Here we show tracklets of cars (see left of Fig. 3·5) and for a group of people walking on the beach (see right of Fig. 3·5). In order to quantitatively evaluate tracklets, we selected the 4 longest streets in our videos and examined every tracklet associated to them. This was equivalent to examining 48 cars doing a full trip on their corresponding streets. We then computed the average angular error associated to each tracklet knowing that cars all move along those streets. This gave us an average error of 1.2 degree with a variance of 0.9.

Note that our tracklets generator assumes temporal consistency and hence it disregards false detections that are temporally inconsistent. This further removes detection artifacts. In closing, note that we do not need particularly good long-term tracks; we need tracks good enough for our retrieval problem.

3.4 Indexing & Hashing

We aggregate features at document creation into $(U-k+1)\times(V-k+1)$ k-level trees. As data streams in we construct 5 feature trees, namely $\{\mathrm{tree}_a\}$, $\{\mathrm{tree}_s\}$, $\{\mathrm{tree}_c\}$, $\{\mathrm{tree}_p\}$, and $\{\mathrm{tree}_m\}$. For Airborne data there is one feature tree, namely $\{\mathrm{tree}_d\}$. $\{\mathrm{tree}_d\}$ is a 1-level tree that stores the current-to-next frame displacements for all points of the generated tracks.

We use an inverted hash table to index a given feature tree tree_f efficiently for content retrieval. Inverted index schemes map content to an address in memory and are popular because they enable quick lookup in sizeable document storage. In the case of video we aim to hash the document number t and the location in space (u,v) of a given atom based on the contents of its feature tree $tree_f$. We accomplish this by mapping "tree_f" to a location in the index to store (t,u,v). Our goal is to store similar trees in nearby locations within the index. Thus we can retrieve similar feature trees by retrieving all stored elements proximate to a given query tree. Note that for Airborne data, in addition to storing (t,u,v) for each tree, we also store the track ID. This encourages results to be generated from as few tracks as possible and hence makes solution more robust to clutter.

The construction of this index is such that update and lookup have performance which does not scale with video complexity, but instead with the magnitude of the returned results because similar content across the video is mapped to the same hash bin. For a given query tree, the hash bin can be identified in constant ($O(1)$) time, and its contents extracted in time scaling linearly with the amount of content in the bin. When the query represents an action which does not commonly occur, or where action itself is sparse, this optimization yields significant performance improvements in runtime. This strong time scaling enables us to reason over video corpora with very large data lifetime at minimal cost to the search engine.

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 $h : \mathrm{tree}_f \rightarrow j$, where j is a hash table bucket number. We use a locality-sensitive hashing (LSH) function [Gionis et al., 1999] in this chapter. LSH approximates nearest-neighbor search in databases. To do this, LSH clusters similar vectors $\mathbf{x}$ and $\mathbf{y}$ by maximizing the likelihood that descriptors within a certain radius of each other are mapped to the same hash key:

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

If input vectors have a small Euclidean distance (calculated on an element-by-element basis between feature trees), their probability of hashing to the same bin is high. The feature values in our trees are real, and selected from M possible values, so we draw LSH functions from the p-stable family:

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

where $\mathbf{a}$ is a random M-dimensional vector, b is a random scalar and r is a parameter dependent on the application. M and b must both be drawn from stable distributions (univariate gaussian and uniform on the $[0, r]$ interval, in our case). Intuitively, $\mathbf{a}$ is a random direction for projection, b an offset and r a radius which controls the probability of collision for feature vectors within that radius of each other. Note that the collision radius r is an important parameter to set correctly - different hashed vectors have different lengths, so a different radius must be set for each table to ensure that the distance at which two things are considered similar is accurate.

We build and search indices I_f independently for each feature, with five for CCTV data and two for Airborne data, using one database per index of each feature f. Each index I_f is composed of a set of n hash tables $\{T_{f,i}\}$, $\forall i = 1, \ldots, n$, with each table given its own p-stable hash function $H_{f,i}$ drawn from $h_{\mathbf{a},b,r}$. So, specifically, a given atom is stored in $N \times 5$ locations - in each of the N tables for each of the five features that we compute. For

storage efficiency, we only store atoms where something is happening - high persistence, high activity, or high motion.

Due to the low-dimensionality of our features, n can be relatively small - in our experiments, we use three hash tables per feature. Depending on the scenario and features selected, the parameter r can be set to allow more or less fuzzy matches. We fix r for each feature in our implementation. The random parameters $\mathbf{a}$ and b ensure projections from the different hash functions complement each other.

Given a feature tree tree_f at location u, v with key $H_{f,i}(\mathrm{tree}_f) = j$, $T_{f,i}[j]$ denotes document numbers $\{t\}$ which contain similar feature trees, located at (t, u, v). In order to perform a lookup in index I_f, we take the union of document numbers returned by each individual lookup in the set of tables $\{T_{f,i}\}$:

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

Fig. 3·6 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.

Lightweight Storage: Unlike distance-metric dependent approaches, our hash-based index does not require explicit storage of feature descriptors. We store only tree coordinates $\{t, u, v\}$, which compress to 12-byte variables (we also store track ID for airborne footage). Because our features are local and primitive, they depend only on non-background content, making our indexing times and storage scale linearly with a video's foreground content. Given that surveillance video can involve a great deal of inactive space or inactive time (consider the surveillance within a retail store while it is closed), this is an incredibly useful feature. In a 5 hour video with an activity rate of 2.5%, we attain a compression ratio of

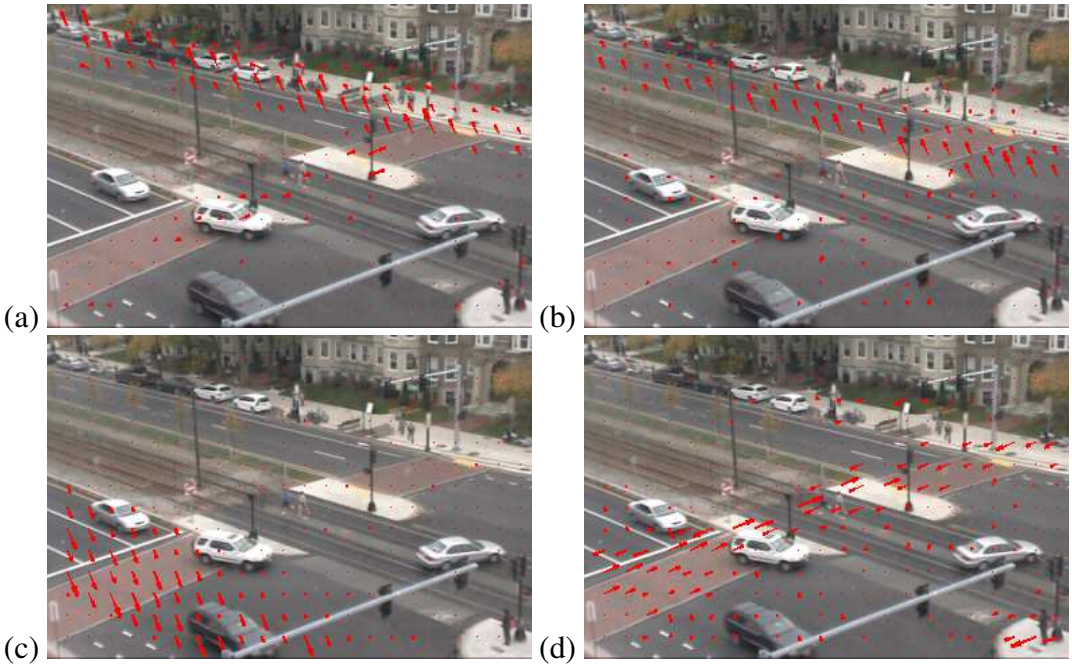

Figure 3·6: Motion features (in red) of four buckets of a hash table. Arrow sizes are proportional to the number of hits at the corresponding sites. Here the four buckets describe: (a) side walk (b) upper side of the street (c) lower side of the street (d) crosswalk.

50,000 to 1 over already compressed video.

Building Lookup Table: In order to accommodate real time video streaming, we iteratively update the index. After feature extraction, we group features into trees as described in Sec. 3.3 and, for each feature f, we update the index I_f by mapping $\mathrm{tree}_f \rightarrow (t, u, v)$ with an LSH function. Once this is done, the features of the current document are deleted and a new document is processed. Note that in order to address the issue of large video lifetime, a garbage collecting process can easily screen the hashing tables and eliminate every tuple whose time variable t is too old to prevent the tables from growing continuously.

3.5 Search engine

Previous sections explained how to extract low-level features from a video sequence, bun-

dle them into trees and index them for $O(1)$ content-based retrieval. In this section we discuss how to use feature index for high-level search. First, we allow users to directly input queries in the form of a sequence of *action components*. Second, we retrieve a set of *partial matches* to these action components in $O(1)$ time from the indices built in section 3.4. Third, we look for *full matches* to the set of action components. We establish a baseline for this search first by a greedy algorithm with an emphasis on colocation. We improve upon this algorithm by extending the Smith-Waterman algorithm for gene-sequence matching [Smith and Waterman, 1981] to accomodate events with duration. This improvement allows us to search for activities that are significantly distorted from the original query, including partial obscuration and time-warping.

3.5.1 Queries

In this chapter, we define a query as a set of ordered *action components*. A U-turn, for example, contains three such action components i.e. horizontal, vertical and then opposite horizontal (see Fig. 3·8). Simpler queries can be defined by one component i.e. car moving in one direction.

For ease of use, we created a GUI (see Fig. 3·7) to enable a user to input sequences of action components. The user is shown a background scene, and inputs components one after the other. Individual components are identical to the features automatically extracted from archive video and hashed in Section 3.3: motion, size, color, persistence and activity. Queries can be directly compared to archived features because they are described in the same vocabulary.

We allow the user to draw regions of interest (ROIs) that he expects these features to appear in (see Fig.3·10, green regions). In the case of motion, the user moves the mouse to indicate direction. In airborne video, if a user chooses to describe a more complex series of motions (a ”route”), we automatically segment the path they draw into components. After the user has drawn an action component, we treat the canvas as a single frame and divide it

Figure 3·7: A GUI for creating queries with instructions outlined from 1-5 (see red regions). Here the user draws queries of interest (in blue) and selects additional target properties (see step 3,4).

into atoms, as in Section 3.3, except that we do not aggregate over time as we only have one frame. We aggregate statistics for each atom in this activity component, and then perform pyramidal binning to generate a feature tree $tree_f$. The end result is that we follow an identical process to that described in Section 3.3, except that instead of having to calculate terms like motion direction and color, we provide them directly.

Note that this method of direct input is only feasible due to the simplicity and semantic saliency of our features. We ask a user to describe his query in terms of motion direction, size, persistence, activity and color; the same features that we automatically extract and

archive from video. While it would be difficult for a user to manually input corner-point features or high-dimensional vector representations, it is relatively easy to specify things such as “red”, “large blob”, and “right-moving”? for example. In our GUI, they would do this by moving their mouse to the right using a motion tool, and then drawing a red area they expected the object to be in. Users are also helped by the background image that they draw on. This gives them both a sense of the type of video that they are exploring, the brightness of the video, and also what sorts of motion may be reasonable to search for. This helps create action components that correlate strongly with archive video.

While each of the elements of the query is rather simple, the formulation as a whole empowers the user with a complex query vocabulary that grows combinatorially with the number of raw features he is allowed to use in query composition. A single component could describe a search for “small red stationary blob” or “large blob moving to the right”. Though our claims to simplicity are theoretical, in our experience a brief explanation is all that is required for people to begin promptly creating their own queries. The system has been used briefly by surveillance professionals and members of the armed forces with few issues.

Once we have a series of action components (each represented by a set of feature trees) we query the inverted index. The result of that query is a set of locations (u, v, t) whose feature tree matches the manually-input action components. These (u, v, t) location are called partial matches and referred to as $M(q)$. This process is robust to user error because of the fuzzy hashing scheme, which allows for approximate matches. It is thus possible that the query “Small, red, right-moving” might yield results that are small, reddish and moving generally to the right. Fig.3·10 presents 10 queries with their ROI.

Coarse features and fuzzy hashing render the query creation process robust to small errors in query definition. Larger issues, like failure to draw a certain query component in its entirety or an obscured drawing are reasoned over by the dynamic programming

algorithm described in Section 3.5.2.

3.5.2 Full matches

Partial matches, however, are insufficient to identify activities in surveillance video, which have higher temporal variance in their duration. As an example, a fast car taking a U-turn will cover fewer documents than a slow moving car as well as generating different motion features. Because documents are relatively limited in time (generally less than a second of real time), a given action component may span many documents, and thus many partial matches. A user expects entire matching sequences, which we refer to as *full matches*, i.e. video segments $[t, t + \Delta]$ containing one or more documents ($\Delta > 0$). 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. We define a full match starting at time τ, given a query q and a set of partial matches $M(q)$, as:

$$R_{q,\tau}(\Delta) = \{(u, v) | (t, u, v) \in M(q),\ \forall t \in [\tau, \tau + \Delta]\}\,. \tag{3.2}$$

Thus, the unique coordinates of partial matches to q in the video segment $[\tau, \tau + \Delta]$ are contained in $R_{q,\tau}(\Delta)$.

We have developed a pair of algorithms to find full matches given a 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. 3.5.2), uses dynamic programming to exploit temporal structure of the query action components.

Full matches using a greedy algorithm

The primary challenge in activity detection is the difference between an idealized query and the realization of that activity in video. Common issues include time-scaling, false detections and spatiotemporal distortion. Given a set of partial matches, our goal is to find

the span of time which is most likely to contain the desired query. We pose the following optimization to solve for this span: 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.

There is an infinite array of potential value functions; we choose a simple and practical $v_{q,\tau}(\Delta)$ given by:

$$v_{q,\tau}(\Delta) = \exp(|R_{q,\tau}(\Delta)| - \lambda\Delta), \tag{3.3}$$

where $R_{q,\tau}(\Delta)$ is defined by Eq. (3.2) and $|R_{q,\tau}(\Delta)|$ is the number of unique partial matches found in the interval $[\tau, \tau + \Delta]$. In order to limit the span of the event, we apply a penalty function, controlled by λ, to increase $R_{q,\tau}(\Delta)$ in Δ (by definition, $R_{q,\tau}(\Delta) \subseteq R_{q,\tau}(\Delta + 1)$).

We opt to solve for Δ via a greedy approach. Based on the above value function, the algorithm ranks non-overlapping video segments in descending order for the user. We demonstrate in Sec. 3.6 that Eq. (3.3) yields compact video segments while keeping low false positives and negatives rates.

Intuitively, this value function is simple and effective because the more complex a query is, the less likely it is to be generated by unassociated random actions. Specifically, given a query Q containing N components drawn from dictionary D, the probability $P(Q)$ of randomly drawing Q as sequence of video components within Δ frames is given by Eq. (3.4), using Stirling's approximation [Abramowitz et al., 1966].

$$P(Q) = \binom{|D|}{N} \frac{|D|^{\Delta - N}}{|D|^{\Delta}} = \binom{|D|}{N} |D|^{-N} \approx e^{-N \log \frac{|D|}{\Delta}} \tag{3.4}$$

The expression shows that if the dictionary is large relative to the time-interval Δ then

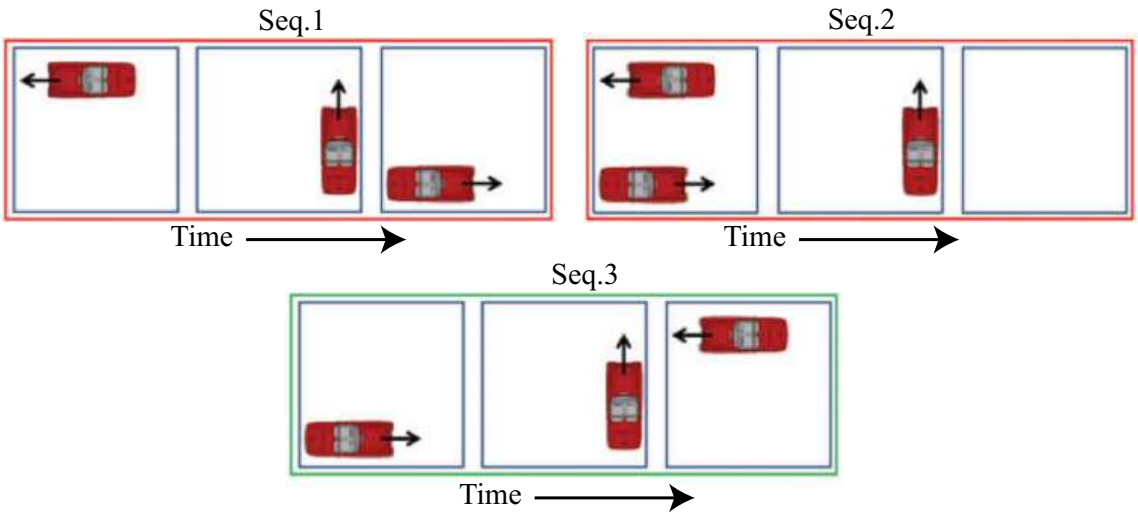

Figure 3·8: Searching for a U-turn. Despite three sequences of actions have same $R_{Q,\tau}(\Delta)$ values (see Seq. 1,2,3), yet only Seq.3 contains a valid U-turn.

the probability of randomly drawing the user defined query decreases exponentially with the number of query components. Nevertheless, we can significantly improve this bound if we also account for the order of the query components.

Full matches with dynamic programming (DP)

We can improve upon the performance of the greedy algorithm in Sec. 3.5.2 by exploiting the order of the action components as well as their colocation. When a car takes a u-turn, it has to approach an opportunity to turn, take that turn, and then proceed in the opposite direction. For a man hopping a subway turnstile, he must approach the turnstile from the wrong direction, navigate it, and depart in the wrong direction. Independently, these components occur commonly. Together, there is only one order in which they reproduce the query in full. This is illustrated in Fig. 3·8.

In the greedy optimization of Section 3.5.2, we ignore the time-ordering inherent to the query set. The value function $R_{q,\tau}(\Delta)$ is a set, not a list - it fails to differentiate between orderings of the same components such as *(Forward, Left, Back)* and*(Back, Forward, Left)*.

Intuitively, this should hurt performance - false positives diminish once we exploit causality. We quantify the improvement upon the performance bounds given in Eq. (3.4) in Eq. (3.5).

$$P(Q) = \frac{e^{-N \log \frac{|D|}{\Delta}}}{N!} \approx e^{-N \log(\frac{|D|N}{\Delta})} \tag{3.5}$$

It is worth highlighting the difference between the two expressions Eq. 3.4 and Eq. 3.5. In the latter expression we only need $|D|N$ to be sufficiently large relative to Δ as opposed to $|D| > \Delta$. Consequently, if we were to account for the order of query components, the probability of randomly matching the user defined query approaches zero even when Δ scales with the size of the dictionary.

Once a user has created a set of N action components using our GUI, and we have retrieved N sets of partial matches within the video, our next step is to use dynamic programming to exploit causality and retrieve the best matches. To accomplish this, we adapt the Smith-Waterman dynamic programming algorithm [Smith and Waterman, 1981], originally developed for gene sequencing, to determine which set of partial matches best matches the query. Our algorithm, described in Algorithm 2, reasons over the set of partial matches $m_{\tau,\alpha} \in M(q)$ 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 focus on three types of distortion, namely insertion, deletion and continuation.

(1) Insertion distortion occurs in documents where the query is happening but there are no partial matches. The most common causes for this are a pause in the activity (such as the stop in a u-turn that a stoplight or oncoming traffic might mandate) or obscuration.

(2) Deletion distortion occurs when part of the query is missing, and an entire action component is not present in the video. Deletions happen most frequently because of obscuration, but can also occur simply because one component of an action is not performed.

(3) Continuation distortion occurs when an action component takes more than one doc-

ument. This is where an action component "stretches" in time. This is by far the most common type of distortion, and allows our value function to account for significant temporal variability.

Algorithm 2 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**
      $$V_{\tau,\alpha} \leftarrow \max \begin{cases} 0 \\ (V_{\tau-1,\alpha-1} + W_M) * m_{\tau,\alpha} \\ (V_{\tau-1,\alpha} + W_C) * m_{\tau,\alpha} \\ (V_{\tau-1,\alpha} + W_D) * (1 - m_{\tau,\alpha}) \\ (V_{\tau,\alpha-1} + W_I) * (1 - m_{\tau,\alpha}) \end{cases}$$
      if Airborne Data **then**
        $V_{\tau,\alpha} \leftarrow V_{\tau,\alpha} + \mathbf{1}_{\phi_{\tau-1,\alpha-1}}(\phi_{\tau,\alpha}) * m_{\tau,\alpha} *$
        W_S
      end if
      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

In addition to the three above distortions, we introduce an extra term to handle the Airborne data. This term adds more score to a match if its Track ID is the same as the ID of the match in the previous document. In Algorithm 2, the track ID of an examined cell is $\phi_{\tau,\alpha}$ where $\mathbf{1}_{\phi_{\tau-1,\alpha-1}}(\phi_{\tau,\alpha})$ is an indicator function that returns 1 if $\phi_{\tau,\alpha} = \phi_{\tau-1,\alpha-1}$.

Our dynamic programming approach creates an $N \times |q|$ matrix V to search for a query with $|q|$ action components in a video with N documents. This matrix is filled out from top left to bottom right, from the beginning of the set of partial matches and earliest action component to the latest of each. A path through V is defined as a set of adjacent (by an 8-neighborhood) matrix elements, where each path element represents a hypothetical assignment of an action component taking place in a document. A path hypothesizing that action component b occurred in document a should contain element $V_{a,b}$.

As the matrix is filled out, each element examines its 3 neighbors (left, up-left, up) and the distortion that would have to occur in a hypothesis where this neighbor preceded the element. Then, the element selects the one with the best possible score and stores both the score and a pointer to the chosen element in the matrix of values, V. To find the optimal path given V, identify the maximal value in V and trace the path backwards to its origin - that is the best match. To iteratively find ranked, non-overlapping values, set the elements of V that were used in a given hypothesis to zero and repeat the search until the maximal value in V is below the retrieval score threshold T. We show an example matrix V with overlaid paths in figure 3·9.

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 2) 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 $W_I = -1, W_D = -2$,

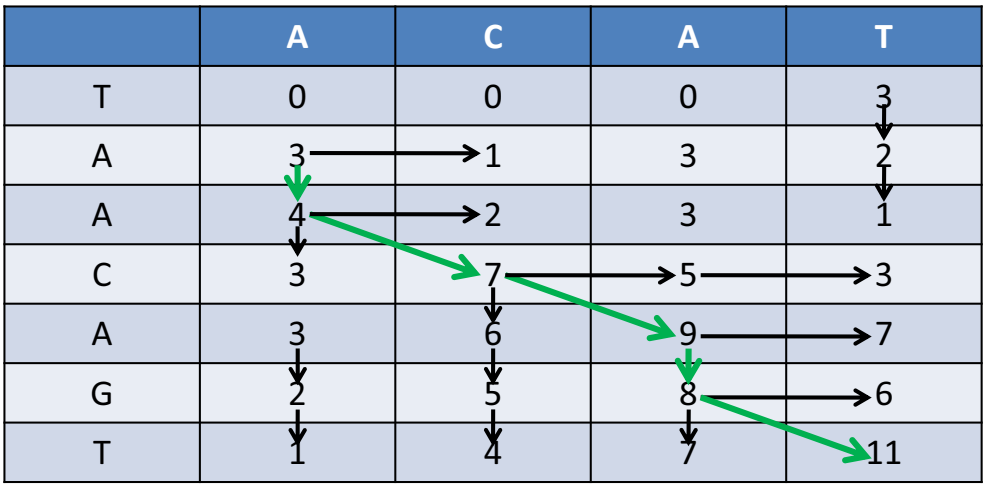

Figure 3·9: Example of the V matrix. The query is actions A, C, A, T, and the seven documents in the video corpus each contains a single action, T, A, A, C, A, G, T. The values for W_I, W_D, W_C and W_M are $-1, -2, 1$ and 3 in this example. The optimal path, A,A,C,A,G,T, involves an insertion, a continuation and a deletion. It is found by tracing backwards from the maximal element, valued at 11.

$W_C = 1$, and $W_M = 3$. These values favor longer sequences without deletions and are relatively robust to missed detection. For Airborne footage we also have a bonus W_S for a match generated by the same tracklet. In our experiments W_S is set to 5.

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 these results.

3.6 Experimental Results

3.6.1 Datasets

In order to gauge performances of our approach (with and without dynamic programming), we ran experiments on 11 CCTV videos and 2 airborne videos (see Table 3.1, Fig. 3·10 and Fig. 3·11). We selected these videos to test our method on various application contexts, as well as to provide comparison to other methods. The *Winter driveway*, *U-Turn* and *Aban-*

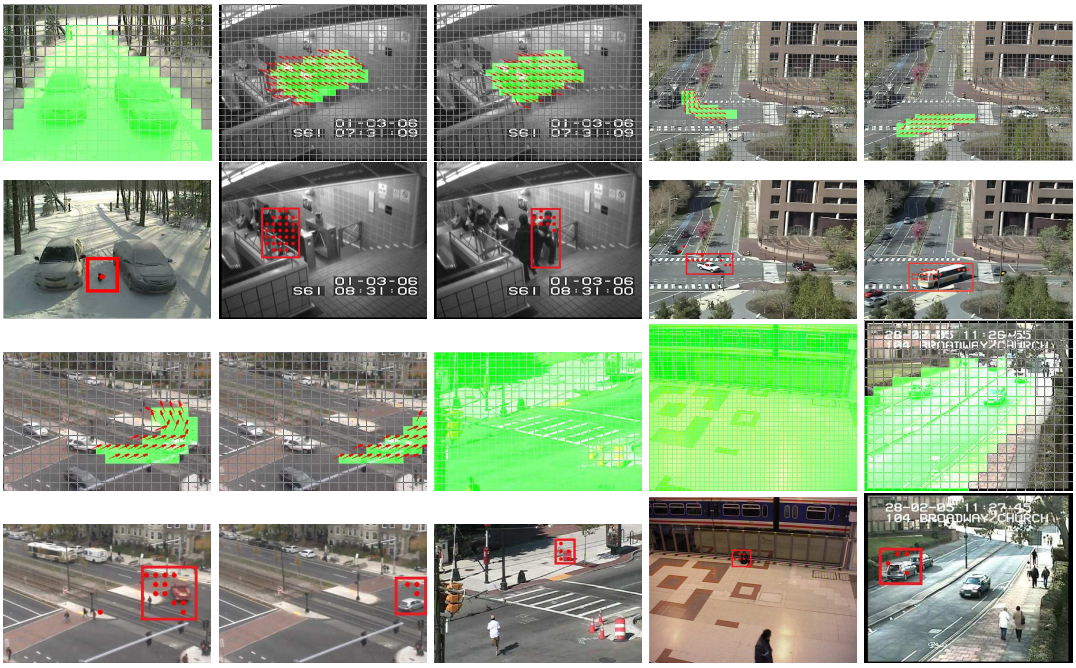

Figure 3·10: Results for ten tasks. For each task we show the examined query (in red arrows), ROI (shown in green) and the generated retrieval (bottom to query, in red rectangle). Here red dots are trees whose profile fit the query.

doned object were taken from our own dataset, *PETS* and *Parked-vehicle* and *MIT-traffic* come from known datasets [Ferryman, 2006, ILI,], *MIT-traffic* was provided to us by Wang *et al.* [Wang and Wang, 2011]; *Subway* from Adam *et al.* [Adam et al., 2008]. *Airborne* was given to us by the National Geospatial Agency (NGA). These datasets contain a diverse array of real-life scenarios. The subway dataset includes multiple partial occlusions, as people walk behind turnstiles and ticket booths while the U-turn datasets features a task performed at many different speeds. Some cars perform a continuous U-turn, while others stop halfway due to red lights. In airborne surveillance, people are so small as to be mainly identifiable through temporally consistent behavior, and cars are obscured by trees for up to 30% of activity duration. From low frame-rate and low contrast black and white airborne video to high framerate, color CCTV data, these datasets demonstrate the flexibility of our

Figure 3·11: Examples of the examined routes. Routes are shown in yellow/red arrows and they start from point X and end at point Y. Some of the routes undergo strong occlusion (see dashed yellow region, top row, first column, for route in second column) and others undergo many turns (see first row, second and last column).

algorithm. We tested different queries to recover moving objects based on various features (see Table 3.1 for more details). Query features include size, color, tracklets, activity, direction, and persistence. We queried for objects, animals, people, and vehicles moving along user-specified routes. We searched for infrequent and sometimes abnormal events (unexpected U-turns, animal in the snow, abandoned objects and people passing turnstile in reverse) as well as frequent events (car turning at a street corner, people counting, and vehicle parking). Some videos show events at a distance *MIT-traffic [Wang and Wang, 2011]*, while others show people walking and interacting together close to the camera *Subway*.

3.6.2 Examined Tasks

We examined 11 CCTV footage (see Fig.3·10 for examples) and 2 Airborne sequences (Fig.3·11). We defined queries and manually created a ground-truth list for each task consisting of ranges of frames. We obtained comparisons by computing the intersection of the ranges of frames returned by the search procedure to the range of frames in the ground truth. We marked an event as detected if it appears in the output video and at least 1 partial match hits objects appearing in the event. For airborne footage, we also require that the start and end frames be within 15 frames of the ground truth.

HDP Comparison

For the purposes of comparison, we implemented a scene understanding technique based on Hierarchical Dirichlet Processes (HDP) [Xiang and Gong, 2008, Kuettel et al., 2010]. 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.* [Xiang and Gong, 2008] 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.

Queries are specified as the ideal classification distribution and the search algorithm

Task	Video	Search query	Features	Duration	Video size	Index size
1	Winter driveway	black cat appearance	color and size	253	6.55 GB	147 KB
2	Subway	people passing turnstiles	motion	79	2.75 GB	2.3 MB
3	Subway	people hopping turnstiles	motion	79	2.75 GB	2.3 MB
4	MIT Traffic	cars turning left	motion	92	10.3 GB	42 MB
5	MIT Traffic	cars turning right	motion	92	10.3 GB	42 MB
6	U-turn	cars making U-turn	motion	3.4	1.97 GB	13.7 MB
7	U-turn	cars turning left, no U	motion	3.4	1.97 GB	13.7 MB
8	Abandoned object	abandoned objects	size and persistence	13.8	682 MB	2.6 MB
9	Abandoned object	abandoned objects	size, persistence and color	13.8	682 MB	2.6 MB
10	PETS	abandoned objects	size and persistence	7.1	1.01 GB	5.63 KB
11	Parked-vehicle	parked vehicles	size and persistence	32		
12	Intersection 1	cars moving	tracklets	13.8	1.16 GB	153 KB
13	Beach 1	people walking	tracklets	13.8	529 MB	65 KB

Table 3.1: Tasks' number, videos, search query, associate features, video duration (min.), video size and index size. Videos of Task 12 and 13 have a frame rate of 2 frames per second. Tasks 1, 8, 9, 10, 11, 12 and 13 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.

compares each document 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:

$$D(q, p_j) = \sum_{k=1}^{K} q(k) \log \frac{q(k)}{p_j(k)}. \tag{3.6}$$

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 evaluates $D(q, p_j)$ for each document j and ranks the documents in order of increasing divergence.

3.6.3 CCTV Results

Results for our greedy method are depicted in Fig. 3·10. Quantitative results for our method as well as HDP are summarized in Table 3.2. The "Ground truth"(GT) column of

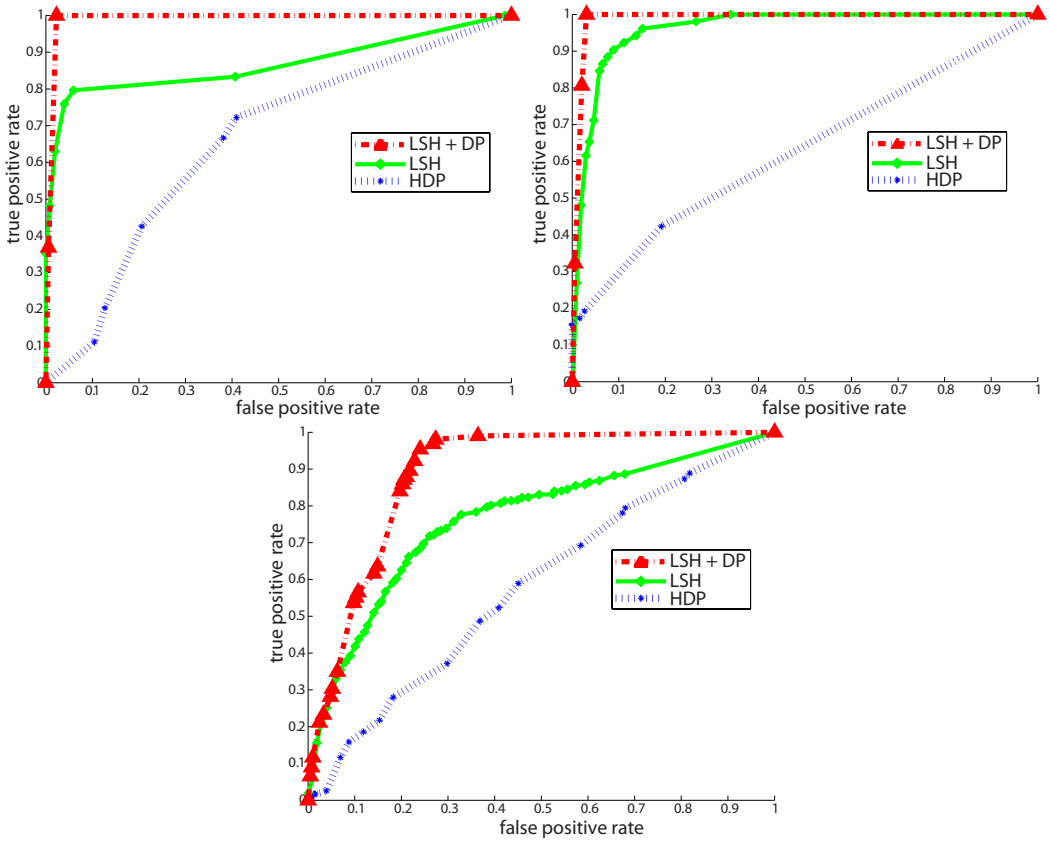

Figure 3·12: ROC curves for the U-turn, subway and MIT traffic datasets. Our methods significantly out-perform HDP [Xiang and Gong, 2008, Kuettel et al., 2010].

Table 3.2 indicates the exact number of events in the dataset. The "Greedy True" and "HDP True" columns indicate the number of true positives (correct detections) for our Greedy algorithm and for the HDP-based search [Wang et al., 2009]. Likewise, the "Greedy False" and "HDP False" indicate the number of false alarms found for those eleven tasks.

Table 3.2 underlines the robustness of our method over a large range of search applications, outperforming the HDP baseline on every aspect. As can be seen from the table, the absolute detection rate of our method is strong. One can also see that HDP-search deals well with search of recurring activities (as in task-1 Subway) and poorly otherwise (as in task-2 Subway). While some 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. Not only does the HDP search can only recover frequent events, its processing time scales linearly with the number of documents, an undesirable quality for large videos. Furthermore, the training phase is prohibitively slow (approximately 2 days for the "subway" sequence) and must be re-executed every time that more video frames are included.

Dynamic Programming

Two tasks in table 3.1 have a temporal structure which can be exploited through dynamic programming. In order to show how exploiting this structure can improve results, we selected tasks 4 and 6 and launched dynamic programming using the full query (algorithm 1) as well as greedy and HDP search algorithms. The ROC curves for these experiments are in Figure 3·12.

Figure 3·12 illustrates the kind of improvement that can be attained by our approach. As can be seen, dynamic programming has a much better true positive rate than HDP and greedy optimization. There is thus a clear improvement from employing time-ordering with DP. These improvements are unsurprising, as HDP optimizes a global criterion, attempting to create a topic for each action. LSH performs a local search which is fast and produces

low false alarm rate. Due to the global nature of HDP, the topics it discovers are more likely to be common events, so infrequent events such as abandoned objects and uturns pose a problem. Note that the difference between our methods and HDP-searches narrows on the MIT-traffic dataset (third plot), where the query (left turns at a street corner) is a common occurrence.

Task	Video	GT (events)	Greedy True (events found)	HDP [Kuettel et al., 2010] True (events found)	Greedy False (events found)	HDP [Kuettel et al., 2010] (events found)	Runtime (seconds)
1	Winter driveway	3	2	–	1	–	7.5
2	Subway	117	116	114	1	121	0.3
3	Subway	13	11	1	2	33	3.0
4	MIT Traffic	66	61	6	5	58	0.4
5	MIT Traffic	148	135	54	13	118	0.5
6	U-turn	8	8	6	0	23	1.2
7	U-turn	6	5	4	1	14	0.6
8	Abandoned object	2	2	–	0	–	4.8
9	Abandoned object	2	2	–	0	–	13.3
10	PETS	4	4	–	0	–	20.2
11	Parked-vehicle	14	14	–	0	–	12.3

Table 3.2: 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. Third row contains ground truth (GT).

3.6.4 Airborne Results

We compared our dynamic programming approach with tracklets (Tracklets+DP) to other methods including our dynamic programming method using the same motion features used for CCTV footage (ME+DP) as well as motion, size and activity features (MEB+DP), our dynamic programming method with a different tracking method (the KLT tracker) (KLT+DP) [Shi and Tomasi, 1994], and the HDP method [Kuettel et al., 2010]. We also compare against explicit track matching (Track matching) and a greedy method (Greedy). The greedy method is based on the total number of partial matches, without exploiting the temporal order of the query.

Fig. 3·13 shows the ROC for the task 12 and 13. Recall that the Retrieval Score Thresh-

old is the minimum path score (generated by Algorithm 1) required in order to declare this path as a correct search result. The points on Fig. 3·13 represent different Retrieval Score Threshold values for each technique. Those values are equally spaced between the lowest and highest retrieval scores generated by the results. As shown by the generated ROC, our technique (Tracklets+DP) outperforms all the other techniques quite noticeably. HDP [Kuettel et al., 2010] fails to handle infrequent events. This problem becomes more evident in Task 13 since the beach does not have much activity and hence most queries are considered infrequent.

The accuracy of our tracklets can be quantified by examining Fig. 3·13. Here our algorithm (see solid red) shows significant improvement over KLT (see solid black) even though dynamic programming was used in both methods. For cars the improvement accounts for 105% increase in true positives for a false positive rate of 0.2. For humans improvement is even higher, with 350% increase in true positives for a false positive rate of 0.2. The higher improvement for humans is expected as KLT is not robust to low contrast features. To quantify the improvements dynamic programming brings to the search problem, we compare our technique against 'Tracking Matching' (see Fig. 3·13, dashed black). Here our tracklets are used in both methods. However for 'Tracking Matching' we declare a tracklet as a match if the mean absolute error between the query track is within a threshold. Fig. 3·13 shows a significant improvement in true positives by 500% for both humans and cars at a false positive rate of 0.2 (see red and dashed black). This shows that dynamic programming is capable of significantly boosting search results despite any remaining detection errors.

In terms of computational complexity, HDP is the most expensive. Our approach takes an average of 2.9 seconds to process one frame of Task 12 and 4.32 seconds to process one frame of Task 13. Processing time is computed as the average over 2000 frames. All experiments are performed on an Intel(R) Core(TM) i7-3820 CPU @ 3.60 GHz 3.60GHz

with 16.GB RAM. All algorithms are written in an unoptimized MATLAB code except for the KLT extraction technique which is written in C++ by Shi et al. [Shi and Tomasi, 1994]. Note that our storage procedure spares the burden of having to store all the information. For instance, even though we are extracting as much as 4000 tracklets for task 12, the size of the generated hash table is around 8000 times smaller than the size of the actual original data (see Table 3.1).

3.6.5 Discussion

Our method offers a different way of approaching the problem of content-based video search. Instead of relying on training data and carefully-selected features to identify meaningful actions, we rely on simple low-level features and causality. Our method has the benefit of having a run-time that scales sub-linearly with the length of the video which is a key element when dealing with hours (if not days) of videos. Furthermore, the LSH function H(.) maps every feature tree extracted from a video document to a table entry index. This not only allows us to store (t,u,v) coordinates instead of the entire feature tree (which brings a huge advantage memory wise) but it also allows to map user-defined queries to the same table indices. In other words, the LSH function H(.) allows us to bridge the gap between spatio-temporal features such as persistence, optical flow, tracking, color and size and user-defined queries such as ”give me all big blobs moving right and then turning left”. LSH excels at the challenging task of converting user-defined queries into a representation compatible with computer-extracted features.

The results also demonstrate that causality and dynamic programming are useful tools to reduce false alarms, even when faced with low-quality video, time-warping, obscuration and confuser objects. This is a result of the formulation’s favorable scaling with query complexity. As a query adds more action components, it becomes less likely that it is accidentally created by noise. This property drives false alarm rate down even for relatively simple queries like U-turns and car routes.

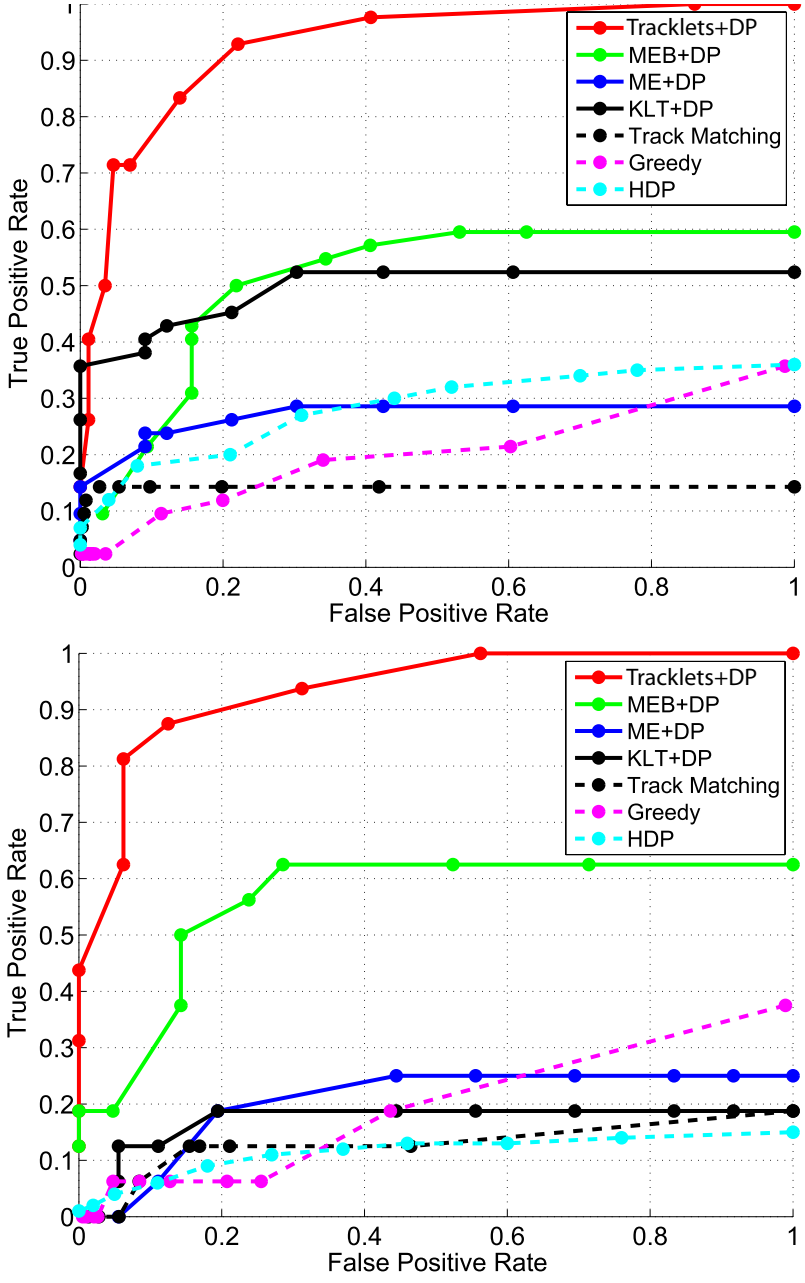

Figure 3·13: ROC curves for cars and humans in airborne data.

Our approach is also capable of exploiting non-temporal structures. Spatial positioning of queries, such as "The third action element occurs southwest of the second one" can differentiate relevant actions from background noise.

Of course, our method has limits. One limitation is that it requires each query to be made of a finite number of discrete states, each describable by a simple feature vocabulary. Complex actions like sign language or actions which are too quick or too small to be identified at the atom level will be difficult to search for.

3.7 Conclusion

In this chapter, we presented a content-based video retrieval method that archives the dynamic content of a surveillance video to allow for user-defined search queries.

As the video streams in, frames are grouped into documents (typically 30 frames per document) which are divided into atoms. These atoms are merged together into trees, each containing a collection of features. We used different features for different types of data. For CCTV footage we used color, size, persistence, and direction of the moving objects. For Airborne footage we used Viterbi-generated tracklets of examined objects. The spatio-temporal coordinates of the trees are then stored into a hash table. Thanks to the locality sensitive hashing function, trees with similar content have their coordinates stored in the same entry of the hash table. Search becomes a simple lookup because user-defined queries are also converted into a hash table index.

Our method has a number of advantages compared to other methods. First, since only the tree coordinates are stored (that is (t, u, v)) and not the features themselves, the hash table requires minimal storage. Second, local processing renders our method suitable for constant video renewal. Third, our method summarizes every dynamic aspect of the video, not just the predominant modes of activity. Our method can thus retrieve any combination of rare, abnormal and recurrent activities. Fourth, because of the indexing strategy, our

search engine has a complexity of $O(1)$ for finding partial matches.

Chapter 4

Zero-Shot Search in Video Corpora

In this chapter we develop an approach for video retrieval based on semantic graph queries. Unlike conventional approaches, our method does not require knowledge of the activity classes contained in the video. Instead, we propose a user-centric approach that models queries through the creation of sparse semantic graphs based on attributes and discriminative relationships. We then pose search as a ranked subgraph matching problem and leverage the fact that the attributes and relationships in the query have different levels of discriminability to filter out bad matches. Rather than solving the NP-hard exact subgraph matching problem, we propose a novel maximally discriminative spanning tree (MDST) as the relaxation of a given query graph, and then describe a matching algorithm that recovers matches to this discriminable tree in linear time using maximally discriminative subgraph matching (MDSM). We utilize MDST to minimize the number of possible matches to the original query while guaranteeing that the best matches are within this set. We test this algorithm on two large video datasets: the 35-GB Virat Ground dataset and a 1-TB aerial data collection from Yuma. These datasets yield graphs with 200,000 nodes and 1 million nodes, respectively, with an average degree of 5. Our approach finds complex queries in seconds while maintaining comparable precision and recall to slower current approaches.

4.1 Introduction

One of the main challenges of surveillance video search is to navigate a large corpus to quickly find matches to an a priori unknown query. Unlike typical approaches to search

[Dollár and Rabaud, 2005, Laptev, 2005, Niebles et al., 2008], we do not utilize exemplar videos or attempt to learn activity categories a priori. In video search, the *complex vocabulary of activities* renders activity model training both computationally infeasible and practically impossible due to a lack of training examples.

Our approach can be interpreted as zero-shot activity retrieval, a cousin of zero-shot learning [Palatucci and Pomerleau, 2009]. It does not rely on training data or exemplars, nor does it exploit multimedia contextual information such as text or audio since these are not frequently available for surveillance videos. We accomplish zero-shot retrieval by leveraging the fact that activities in certain surveillance videos can be characterized by a relatively small attribute vocabulary $\mathcal{A}$ and a limited relationship vocabulary $\mathcal{R}$ to organize them. Furthermore, this limited vocabulary of attributes and relationships can be easily understood by a user, autonomously extracted in real-time, and stored efficiently using several well-known hashing schemes. Rather than relying on complex attributes, we make use of the structural relationships of an activity to formulate a query Q as a semantic graph, associating attributes with vertices and relationships with edges. Our search algorithm then tries to match this graph to attributes and relationships in the video data using approximate subgraph matching.

Subgraph matching is an NP-hard [Ullmann, 1976] problem that is computationally infeasible on large datasets even when the subgraph is relatively small [Cordella et al., 2004]. Our approach is based on data reduction and in this sense is complementary to any large scale video search algorithm. First, we design coarse detectors for attributes and relationships to create a coarse graph C from the archive data, which precludes attributes and relationships that individually could not match vertices and edges in the query graph. The run-time for these detectors scales with $|C|$, the size of the coarse graph, as opposed to the size of the archive data. Second, we leverage the fact that not all attributes and relationships are equally discriminative, and that their discriminability can be calculated in

real time. We use this discriminability to calculate the *Maximally Discriminative Spanning Tree* (MDST) T^* for the query graph in $O(|E^q| \log(|V^q|))$ time, where V^q and E^q are the vertices and edges in the query graph, respectively. This is the spanning tree that is expected to yield the smallest possible coarse graph C. We maximize the discriminability of this tree by modeling the distributions of attributes within graphs and solving a spanning tree optimization. Third, we propose a new tree-matching algorithm that finds a set of solutions in C that match T^*. This process is shown in Figure 4·1, and is accomplished without the explicit construction of the coarse graph.

Intuitively, our approach is effective on large datasets with different fields of view and a priori unknown activities because we create simple models from semantic features instead of creating complex models based on high-dimensional features. Thus, while our models are less tuned than more complex models, they are flexible, easy to generate and have a high probability of detection. In addition, our approach takes advantage of the sparsity of surveillance queries. In order to find an activity, we do not have to model each detail - just a series of high-level features which differentiate it from other activities in the dataset. While sliding-window methods [Lin et al., 2014] struggle to model large-scale complex activities because of the amount of unrelated activity in a broad spatio-temporal window, our subgraph matching approach remains unaffected.

Our contributions are:

Zero-Shot Retrieval: We propose a zero-shot, user-centric approach to model acquisition through the creation of sparse semantic graphs based on attributes and discriminative relationships. This process requires no training data or training step, allowing for the description of novel classes.

Graph Representation: We introduce a graphical approach to query representation. While standard systems learn complete models for each activity, we provide a user with a series of simple semantic concepts and arrange them in a graph. In this graph, nodes

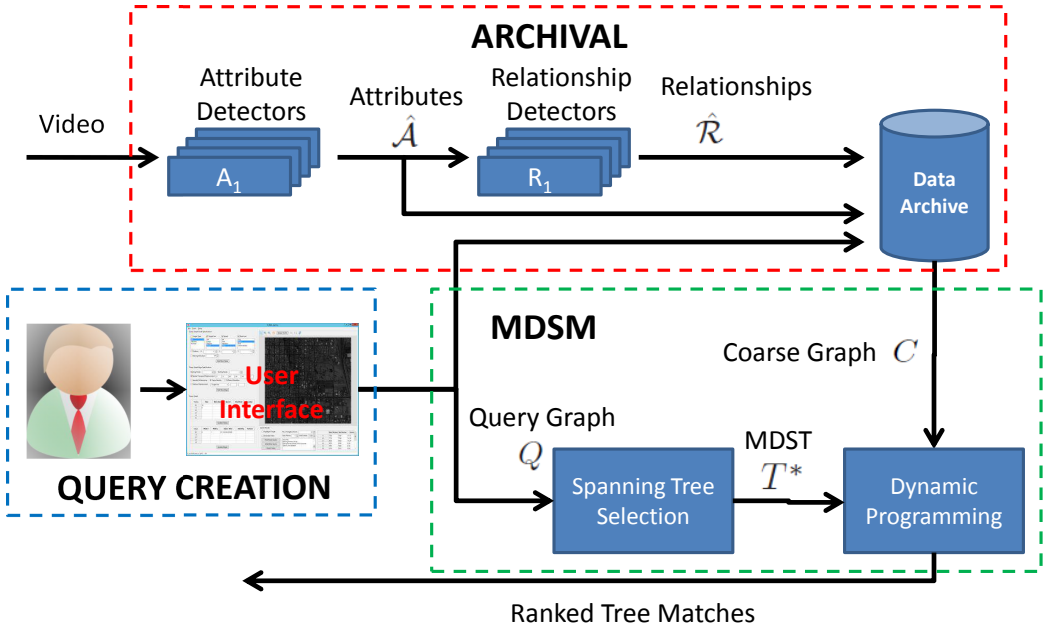

Figure 4·1: In the archival step, we take in incoming data, extract attributes and relationships and store them in hash tables. In the query creation step, a user utilizes our GUI to create a query graph that is used to extract the coarse graph C from archive data. In the Maximally Discriminative Sub-graph Matching (MDSM) step, we calculate the maximally discriminative spanning tree (MDST) T^* from the query graph, retrieve matches to it, and assemble them into ranked search results for the user.

represent attributes of an object or scene and edges represent relationships. We use this query in a sub-graph matching approach to identify activity, allowing our algorithm to effectively ignore confuser events and clutter that happen near the activity of interest. This graphical representation also makes the approach relatively agnostic to the duration of the event, allowing it to detect events that take place over several minutes.

MDST: We introduce the calculation of the Maximally Discriminative Spanning Tree, T^* based on the statistics of archive data stored in our indices. We not only exploit, as others do [Lin et al., 2014, Sun et al., 2012], the sparsity of individual elements, but calculate an optimal combination of elements to maximally reduce the archive data using a novel Maximally Discriminative Subgraph Matching (MDSM) algorithm. We show that

this approach results in a dramatically smaller corpus over which to search, and prove that the corpus must contain all high-ranking matches to the original graph.

We show that our approach, outlined in Figure 4·1, works in both airborne and building-mounted surveillance modalities, outperforms previous work and produces results comparable in quality to a brute-force search in a realistic time.

4.2 Model

Our model is composed of four parts:

Vocabulary: We define an attribute vocabulary $\mathcal{A}$ and a relationship vocabulary $\mathcal{R}$. Our attribute vocabulary consists of pyramidal-binned histograms of pixel-level features, object detectors and low-level motion features extracted from tracked data. Our relationship vocabulary consists of local, pairwise descriptors such as "near to" and "shortly after".

Query: Our query graph, $Q = G(V^q, E^q, A^q, R^q, L^q)$ is a function of five elements. For each vertex $v \in V^q$ in the graph, A^q_v is the set of attributes from $\mathcal{A}$ associated with v, and L^q_v is the set of tolerances for each of those attributes. For each edge $(v_1, v_2) \in E^q$, $R^q_{(v_1,v_2)}$ is the set of relationships associated with (v_1, v_2), and $L^q_{(v_1,v_2)}$ is the set of tolerances for each of those relationships. Value and tolerance pairs specify an expected value and the variance in that value that is acceptable to the user.

Scoring: We define similarity functions $K_{a,l}(a')$ and $K_{r,l}(r')$ for the purposes of comparing attributes a and a' with tolerance l and relationships r and r' with tolerance l.

Storage: We construct an archived representation which stores $\mathcal{A}$ and $\mathcal{R}$ given a maximal tolerance l_{max} for each attribute or relationship. This archive allows us to quickly recover the set of attributes or relationships which could match a query, independent of the specified tolerance.

4.2.1 Vocabulary

The video modality for our datasets is street surveillance, which dictates a specific set of components that a user might use to construct queries (see Sec. 4.2.2).

The attributes in our attribute vocabulary $\mathcal{A}$ come from three classes. The first is local descriptors of pixel-level attributes, like those used in [Castañón and Caron, 2012]. These attributes, like color, size, coarse motion direction, persistence and activity can be computed over tiles made of B by B pixel squares and histogrammed to provide rough descriptors of foreground image properties. As video streams in, we aggregate these tiles into space-time boxes called *atoms*, and aggregate these rectangles via pyramidal binning as shown in Figure 4·2. In our experiments, atom size is generally eight pixels in each spatial dimension, and roughly one second of frames in the temporal dimension. We calculate simple attributes and quantize them to low-dimensional histograms. This approach is doubly beneficial; histogramming and quantization naturally mitigate low-level measurement noise while simultaneously producing low-dimensional features for efficient storage.

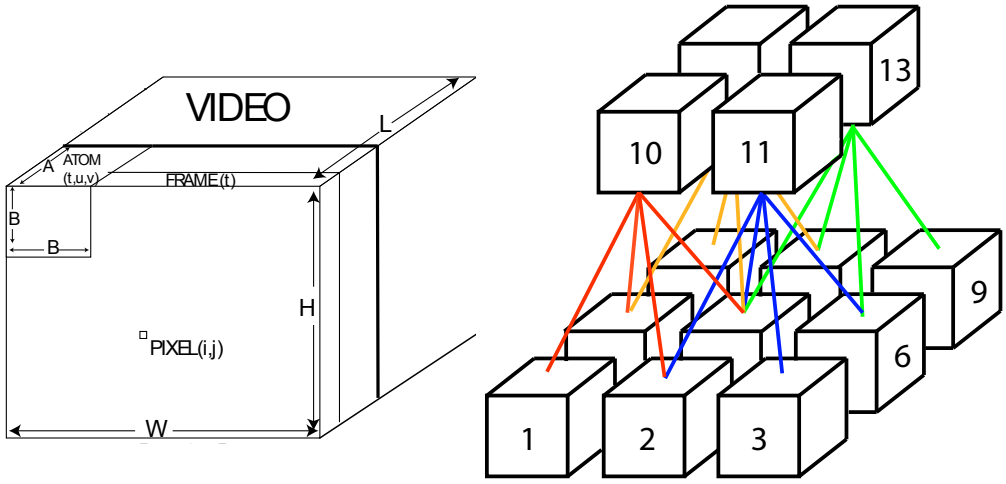

Figure 4·2: (Left) Given a video, we extract (Right) Atoms are grouped into two-level trees - every adjacent set of four atoms is aggregated into a parent, forming a set of partially overlapping trees.

The second type of attribute that we utilize comes from object detectors, such as the

Viola-Jones detector [Viola and Jones, 2001], which work particularly well in video surveillance due to stationary points of view. These detectors are used to find objects, people, and vehicles. These features are simple "type" attributes attached to the location where they were generated.

The third is tracked data [Yang et al., 2005], from which we can extract properties like identity, object size, direction of motion, and appearance/disappearance. Tracked data serves a dual purpose in our system. First, it enables us to extract motion features in extremely low-resolution/noisy video where accurate pixel-level optical flow is prohibitively difficult to compute. It also allows us access to new features indicating that an object appeared or disappeared at a given point in time. We use these descriptors to detect people getting in and out of vehicles, as well as in other scenarios.

The focus of this chapter is not attribute detection - we utilize existing methods to detect simple attributes. Our contribution lies in the utilization of these attributes in combination with indices and graph-search algorithms.

In addition to extracting attributes, we compute a relationship vocabulary $\mathcal{R}$ over pairs of extracted attributes $\mathcal{A} \times \mathcal{A}$. We do not, however, store all possible relationships, as the number of potential relationships scales as the square of the number of attributes. Moreover, storing all relationships would be counterproductive to search - a search algorithm is interested in attributes and relationships which discriminate between the activity described in the query and other activities present in the archive data. Instead, we employ a simple visual intuition: important relationships are both sparse and discriminative. They are sparse in that they are relatively rare compared to the number of potential relationships, and discriminative for that same reason. Because of this utility, we pre-compute sparse relationships like "near" and "the same as". We use a spatiotemporal index [Guttman and A, 1984] to efficiently calculate spatiotemporal relationships, and pairwise calculations to evaluate the presence of other relationships. For our datasets, our relationships describe

relative positions in space, time, and feature space, as well as proximity. We also have an identity relationship, indicating if two attributes were thought to be produced by the same object.

Some queries may include non-discriminative relationships. The relationship "far" is a good example of a non-discriminative relationship, as most objects in a long video are far from each other in space or time. Likewise, it's inefficient to calculate every possible type of spatiotemporal displacement. When we receive queries containing relationships that we have not precalculated we first use other precalculated relationships to reduce the data before looking for matches to relationships which we must calculate in real-time.

We discuss the vocabularies $\mathcal{A}$ and $\mathcal{R}$ used in our experiments in Section 4.4.

4.2.2 Query

Effective zero-shot retrieval must bridge the semantic gap between a desired activity and a representation that can be associated to the data with a particular score. We ask first that a user give us a set of attributes of things involved in the activity, and then specify relationships between those things.

The user assembles his or her attributes and relationships using a query GUI. While the structure of our approach is flexible and allows for other methods of input such as text, our datasets have a limited object vocabulary and most of our relationships are spatiotemporal. Thus, we provide the user with a canvas to arrange vertices with attributes and edges with relationships into query graph $Q = G(V^q, E^q, A^q, R^q, L^q)$. For example, in Figure 4·3, a user specifies that a vertex has the "car" attribute, the "red" attribute, and an edge with the "near" relationship with a vertex containing the "small" and "object" attributes. These vertices and edges form an Attributed Relational Graph (ARG). The vertex properties correspond to attributes in $\mathcal{A}$, while the edge properties correspond to relationships in $\mathcal{R}$.

The process of defining a graph is simple and intuitive in order for users to reliably

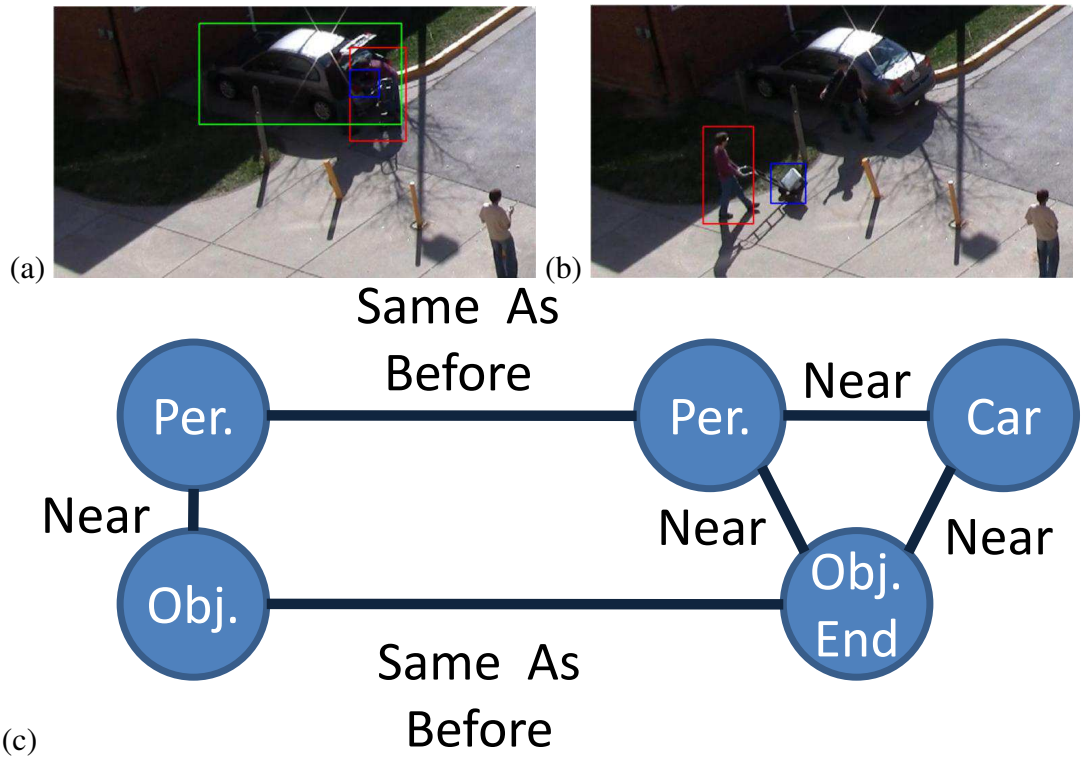

Figure 4·3: The graphical representation of an "object deposit" event.

produce meaningful queries. This begins by making sure that all of the attributes in our attribute vocabulary $\mathcal{A}$ have semantic meaning. This differentiates our approach significantly from others [Aytar et al., 2008, Jung and Park, 2009, Dalton et al., 2013, Wu et al., 2014], which use Histogram of Gradients (HOG) or spatio-temporal interest point (STIP) features to describe their actions. While it is easy for a user to create a graph out of cars, colors, or people, it requires significant expertise to create a meaningful assembly out of HOG or STIP features. Our relationships, like "near" and "same as", also represent clear semantic concepts.

While our attribute and relationship vocabularies, $\mathcal{A}$ and $\mathcal{R}$, may change based on application, we find that a relatively small vocabulary is sufficient to characterize a wide range of interesting actions with a surprising degree of nuance. Consider the activity "A person loads something into a car." Three things are involved - a person, a car to load something

into, and an object to be loaded. In order for a person to be loading an object into a car, they all have to be near each other - this creates the complete triangle graph in the right part of Fig 4·3(c). When the object is loaded, we expect it to disappear, so we add the "end" attribute to the object in that triangle graph. To add further discriminative power to the query, we can add in the requirement that we see the person near to the object at some point in the past - this increases the likelihood that the videos we find have a person bringing an object to the car, as opposed to just loitering around a car with an object. The final query graph is shown in Figure 4·3.

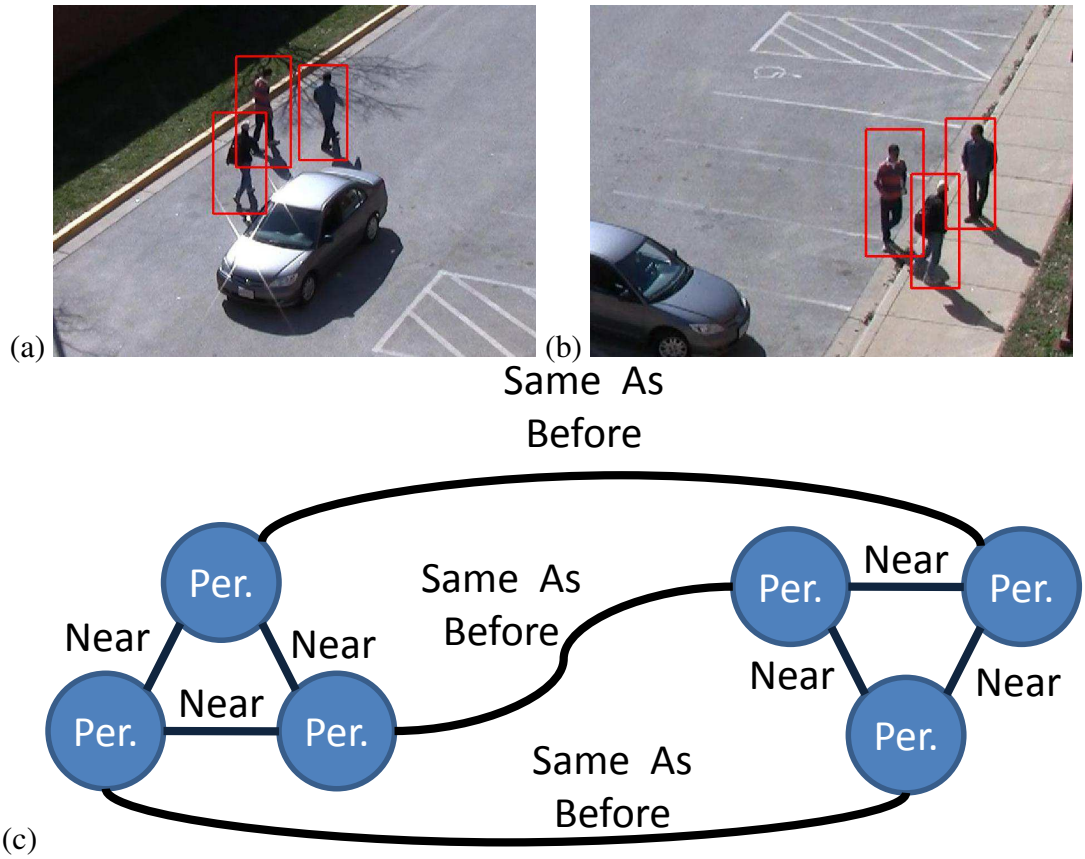

Figure 4·4: The graphical representation of a "meeting" event.

Another salient feature of query graphs is that they sparsify activity, reducing it to its essential components. Many algorithms would struggle to detect "Three people meet together for five minutes," because an activity that spans five minutes or more represents a

significant amount of data. However, the graphical representation of such a query can be made relatively simple (Figure 4·4) - all that matters is that the same three people who were all near each other in the beginning remain near each other five minutes later. Because we are going to solving a sub-graph matching problem, this allows us to treat unrelated activity that happens during our activity of interest exactly like we treat unrelated activity the rest of the time, and ignore it. This enables us to search for long-term events without worrying about unrelated activity over that time period.

The meeting example highlights both a strength and a weakness of the approach: the graph shown in Figure 4·4 is not affected by what happened in the intervening five minutes between the first meet-up and the second meet-up. The group of three could have separated and reconvened, or stayed together the whole time. However, our approach to search focuses not on the perfect modeling of the event we are searching for, but instead discriminating it from other events - and for that, this graph is sufficient.

In addition to specifying the attributes and relationships involved in the graph structure, we allow a user to specify sets of parameters L_v^q for the attributes A_v^q in vertex $v \in V^q$ to indicate confidence in an attribute or tolerance for error. These tolerances are used in the functions that calculate the similarity between pairs of attributes a and a' and pairs of relationships r and r', $K_{a,\ell}(a')$ and $K_{r,\ell}(r')$, respectively, defined below in Equation 4.6. When an attribute $a \in A_v^q$ is something like color or size, we embed it in a real vector space as μ_a. Then we measure the similarity based on RBF kernels such as normal distributions $N(\mu_a, \ell^2)$ or employ indicators on the interval $[\mu_a - \ell, \mu_a + \ell]$ with tolerance $\ell \in L_v^q$. For attributes such as object type where a distance metric is not as obvious, these functions merely indicate whether or not the attributes are identical.

4.2.3 Efficient Indexing

Discriminative attributes and relationships are powerful if you can efficiently use their sparsity to rule out things that don't match a provided query Q, with vertices V^q, each contain-

ing a set of attributes A^q, as well as relationships. In order to be able to retrieve the locations where we could potentially match each $v \in V^q$, we independently hash the locations and values of every attribute we find in the video archive to an inverted index. We use Locality Sensitive Hashing (LSH) [Datar et al., 2004] for attributes like size and color which have computable distances, and employ a standard hash table for discrete attributes like object type. Fuzzy hashing renders our search robust to minor discrepancies in query model, detection model, and storage by retrieving all archived attributes with a certain radius of the query attribute. By computing a hash code for each of a vertex's attributes, A_v^q, and taking the intersection of the bins corresponding to those codes, we can efficiently retrieve the set of matching locations to vertex v.

We apply the same logic to the set of relationships. When a relationship exists between a particular pair of locations, we hash it (and its value), to a fuzzy or normal hash table depending on the relationship type. This allows us to retrieve the set of pairs of locations which approximately satisfy a particular relationship in time proportional to the number of satisfying pairs.

The power of this approach is that it allows us to efficiently downsample the full set of archive data to the set of potentially relevant attributes and relationships, in time proportional to the number of attributes and relationships returned, as opposed to the size of original data.

4.3 Search

We find the optimal match to the query graph Q over the course of four steps. First, we independently look up each attribute and relationship in Q in our data store to retrieve a coarse graph C. Then we calculate discriminative weights for each of the relationships in the query, then calculate the maximally discriminative spanning tree T^* for that query. We create a matching graph H to encode the possible matches in the coarse graph C for

the spanning tree T^*, and finish by using a Maximally Discriminative Subgraph Matching (MDSM) approach to recover the ranked matches to T^*.

4.3.1 Coarse Graph Construction

Given a query graph Q, we construct the coarse archive graph $C = G(V^c, E^c, A^c, R^c)$ in two steps. First, for every vertex $v \in V^q$, we use our data store to retrieve the set of locations where all attributes in A^q_v are present. We add a vertex v_i to V^c for each of those locations and store the attribute values in $A^c_{v_i}$. Second, for every discriminative (hashed) edge $(v_1, v_2) \in E^q$, we retrieve the set of attribute pairs that satisfy every relationship in $R^q_{(v_1,v_2)}$ and whose attributes are in V^c. For each of these pairs, we add an edge to E^c and store the associated relationships in R^c. The computational complexity of this step is proportional to the number of vertices and edges that end up matching the query graph, $O(|V^c| + |E^c|)$.

4.3.2 Tree Selection

Downsampling the data to the set of potentially relevant vertices and edges provides incredible cost savings, but the result is still frequently a coarse graph with hundreds of thousands of vertices. Modern subgraph matching algorithms for attributed relational graphs only scale to hundreds of vertices [Cour et al., 2007], absent approximation by massive parallelization. As such, we are obliged to downsample the coarse graph further.

The optimal downsampled graph, C^*_q would contain only vertices and edges that were present in high-ranked matches to query graph Q so as to minimize the time spent performing an expensive search. We approximate C^*_q by selecting a spanning tree T of Q and downsampling the coarse graph C to C_t, the set of vertices and edges involved in high-ranking solutions to that tree. Because the tree effectively relaxes constraints (in the form of edges) of the optimal solution to the query graph, C_t is guaranteed to contain all of the solutions which match the query graph, with added false alarms. To generate such trees,

we first compute a set of weights indicating the discriminative power of each attribute and relationship in Q, then calculate the spanning tree T which maximizes that discriminative power.

Weight Computation

The choice of which spanning tree to select has significant run-time implications. Its creation involves the removal of edges from Q, and not all edges are created equal. During the archival process, we assign probabilities $p(a)$ and $p(r)$ to each attribute and relationship that we store. These functions denote the probability that a randomly-chosen attribute or relationship in the archive is a match to the attribute a or relationship r. For example, in a dataset that surveils a highway, the "car" attribute is not particularly discriminative, because almost everything on a highway is a car. The "bicycle" attribute in the same dataset would be extremely discriminative, as it is rare for people to bicycle on the highway. We show the probabilities $p(a)$ for attributes used in the VIRAT [Oh et al., 2011] data set in Figure 4·5

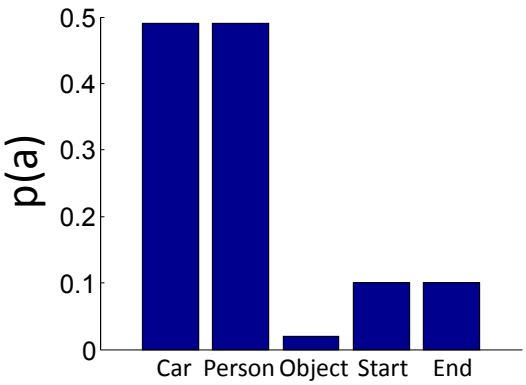

Figure 4·5: The probabilities associated with 5 of the attributes in the VIRAT dataset. Being an object, as opposed to a car or a person, is the most unlikely attribute and thus the most discriminative.

Relationships, in particular, have greater power to be discriminative because the set of potential relationships is $|\mathcal{A}|^2$. Take Figure 4·3; while the "Person near car" relationship is normally very discriminative, 80% of the dataset is shot in parking lots, where people are frequently near cars. Objects disappear near cars far less less frequently - thus, a tree rooted at the "object disappears" vertex and connecting through the "near" edge to the "car" vertex is more discriminative than one starting elsewhere.

We compute empirical values for $p(a)$ and $p(r)$ during the archival process by com-

puting the percentage of attributes (and relationships) that are returned by the hash tables described in Section 4.2.3. If relationships have not been hashed, they are assumed to be nondiscriminative and assigned values of $p(r) = 1$. If it is later determined that these relationships are discriminative, we can revise our estimate of $p(r)$.

Maximally Descriminative Spanning Tree

Absent additional information indicating the distribution of relationships and attributes in the video corpus, we assume that all attributes and relationships are generated independently. This model is clearly imperfect, as certain pairs of attributes are more likely to have a particular relationship. A speed attribute of 60 miles per hour is far more likely to be found in the same location as a car object, as opposed to a person object. Given a diverse set of video corpora, it would be possible to learn these correlations and adjust our model accordingly, but for the purposes of our work we assume independence.

Given a query tree $T = G(V^t, E^t, A^t, R^t)$, we compute weights $p(v)$ for all $v \in V^t$ and $p(e)$ for all $e \in E^t$ as the product of the matching probabilities of the attributes in node v or relationships in edge e. That is,

$$p(v) = \prod_{a \in A_v^t} p(a), \quad p(e) = \prod_{r \in R_e^t} p(r) \tag{4.1}$$

Since nodes and edges in tree T are independent, the total matching probability of the tree $p(T)$ to the archive is:

$$p(T) = \prod_{v \in V^t} p(v) \prod_{e \in E^t} p(e) \tag{4.2}$$

As noted, we are going to do a search using T instead of the original query graph Q as a method of reducing the coarse graph C. We therefore would like to select a tree T which

results in as few matches as possible, so as to generate the maximal reduction in the coarse graph. That is our novel maximially discriminative spanning tree.

Definition 4.3.1 (Maximially Discriminative Spanning Tree (MDST))**.** *We call a tree is an MDST, T^*, to a query graph Q with node weights $\forall v \in V^q, p(v)$ and edge weights $\forall e \in E^q, p(e)$, if the tree satisfies*

$$T^* = \underset{T \in \mathcal{T}}{\operatorname{argmin}}\, p(T), \tag{4.3}$$

where $\mathcal{T}$ denotes the set of all possible trees induced from query graph Q.

Lemma 4.3.1. *Assume that there is no such node in query graph Q that the matching probabilities of the node and all its connected edges are 1. Then the MDST of a query graph Q is equivalent to the minimum spanning tree of Q.*

Lemma 4.3.1 provides us with a convenient way to calculate the MDST that should produce the fewest possible matches. Notice that by matching with MDST, our method can return all the solutions which satisfy all the nodes and some of the edges in query graph Q. Given that $T^* \neq Q$, we will prove that the set of matches to MDST contains all of the matches to the original query graph.

Lemma 4.3.2. *Let M denote a match between a query graph $Q = (V^q, E^q)$ and a coarse graph C. Let $S(\cdot, C, M)$ denote the matching score between any query (e.g. graphs, trees, attributes, relationships) and C, and let $Q' = Q - T^* = (\emptyset, E^q - E^t)$ denote the residual of graph Q. We further define $S(Q, C, M) = \sum_{v \in V^q} S(v, C, M) + \sum_{e \in E^q} S(e, C, M)$. Then we have*

$$\begin{aligned} |S(Q, C, M) - S(T^*, C, M)| &= \left| \sum_{e \in Q'} S(e, C, M) \right| \\ &\leq \sum_{e \in Q'} |S(e, C, M)| \leq |Q'| \cdot \max_{e \in Q'} \max_{\tilde{M} \in \mathcal{M}} |S(e, C, \tilde{M})|, \end{aligned} \tag{4.4}$$

where $|Q'|$ denotes the number of edges in Q', and $\mathcal{M}$ denotes the set of possible matches between Q' and C.

Lemma 4.3.2 shows that the score difference between using query graph Q and using its MDST T^* is upper-bounded. Using this, we can set our parameters to make sure that all

good matches to the query graph Q are present in the set of matches to the MDST T^*. We prove this in Lemma 4.3.3.

Lemma 4.3.3. *By setting proper thresholds, we guarantee that the matches returned by MDST cover all the matches for the original query graph.*

Proof. Lemma 4.3.2 shows that the score difference between using query graph Q and using its MDST T^* is upper-bounded. Let $S_Q = \min_{M_Q \in \mathcal{M}_Q} S(Q, C, M_Q)$, where $\mathcal{M}_Q$ denotes the set of possible matches between Q and C, then in the worst case, we can set the matching score threshold δ for returning solutions as

$$\delta = S_Q - \Delta_{Q'} = S_Q - |Q'| \cdot \max_{e \in Q'} \max_{\tilde{M} \in \mathcal{M}} |S(e, C, \tilde{M})|, \tag{4.5}$$

which guarantees to return all the solutions for query graph Q. □

Although the bound for the matching score difference is not tight, in practice we find that setting these thresholds separately for nodes and edges is easy to tune and quite stable.

4.3.3 Maximally Discriminative Subgraph Matching (MDSM)

Scoring Attributes & Relationships

Because of visual ambiguity, compositional variance and spatiotemporal distortion, we do not want to search for identical matches to the query graph. Not only is it unreasonable to expect a perfect description of an activity from a user, it is improbable that all examples of that activity within the dataset fit a particular description. To this end, we look for a score function with some tolerance for error. This error manifests in two parts - a kernel function on the vertex matches, K_v, and a kernel function on the edges, K_e. For vertex attributes our similarity functions are the sum of similarities for each attribute belonging to that vertex. Similarity functions for relationships are also the sum of the similarities associated with that edge. For notational convenience we express vertex attributes, edge relationships and their corresponding tolerances in terms of binary vectors $A_v \in \{0,1\}^{|\mathcal{A}|},\ R_e \in \{0,1\}^{|\mathcal{R}|},\ L_v \in [0,1]^{|\mathcal{A}|},\ L_e \in [0,1]^{|\mathcal{R}|}$ for any arbitrary vertex v and edge e.

Let v_1 be a vertex in query graph Q, whose value and tolerance for the j-th attribute in $\mathcal{A}$ are given by $a_1(j)$ and $\ell_1(j)$, respectively. Let v_2 be a vertex from the archive graph, whose j-th attribute is given by $a_2(j)$. Let e_1,e_2 represent edges in the archive and query graphs, with values $r_1(j)$ and $r_2(j)$ with tolerance $\ell_1(j)$.

$$K_v(v_1, v_2) = \sum_{j=1}^{|\mathcal{A}|} K_{a_1(j),\ell_1(j)}(a_2(j))$$
$$K_e(e_1, e_2) = \sum_{j=1}^{|\mathcal{R}|} K_{r_1(j),\ell_1(j)}(r_2(j)) \tag{4.6}$$

Note that we are only provided with tolerance for a given vertex or edge in the query, because the other vertex or edge generally represents archived data.

Given a coarse graph C and the MDST T^*, we want to select a matching $M : V^t \to V^c$. We select the matching that maximizes our objective function $S(Q, C, M)$ comprised of two parts - vertex and edge potentials

$$S(Q, C, M) = \sum_{v_q \in V^q} K_v(v_q, M(v_q)) + \tag{4.7}$$
$$\lambda \sum_{(v_{q_1}, v_{q_2}) \in E^q} K_e(v_{q_1}, v_{q_2}, M(v_{q_1}), M(v_{q_2}))$$

where λ is a weighting parameter determining the relative value of matching vertices and edges.

Matching Graph Creation

We solve for the optimal match between the MDST T^* and archive graph C in two steps. In the first step, we build a *matching graph* $H = G(V^h, E^h)$, where each vertex $v \in V^h \subseteq V^t \times V^c$ is a tuple denoting a proposed assignment between a vertex in T^* and a vertex in C, and each edge $e \in E^h$ denotes a relationship between the two assignments. We create H by first adding matches for the root, then adding in matches to its successors which satisfy

both vertex and edge relationships described in T^* - we define score thresholds τ_v and τ_e to be the minimum score for vertices and edges. This process in described in Algorithm 3, and the expected number of vertices scales as a product of $p(T^*)$ and the size of the archive data.

Algorithm 3 Create Matching Graph

procedure CREATE MATCHING GRAPH($T^*, C, K_v(v_1, v_2), K_e(v_1, v_2, v_3, v_4)$)
  $H = G(V^h, E^h) \leftarrow \emptyset$
  Iterate from root to leaves
  for all $v^t \in V^t$ **do**
    Compute the matches to this vertex
    $N_{v_t} \leftarrow (v_t, v_c)$ where $K_v(v_t, v_c) > \tau_v$
    if $Parent(v_t) \neq \emptyset$ **then**
      $E \leftarrow \emptyset$
      for all $(v_{t_p}, v_{c_p}) \in N_{Parent(v_t)}$ **do**
        $E \leftarrow E \cup (v_t, v_c)$ where $K_e(v_{t_p}, v_{c_p}, v_t, v_c) > \tau_e$
      end for
      $N_{v_t} \leftarrow N_{v_t} \cap E$
      $V^h \leftarrow E^h \cup N_{v_t}$
      $E^h \leftarrow E^h \cup E$
    else
      $V^h \leftarrow V^h \cup N_{v_t}$
    end if
  end for
end procedure

Retrieval with MDSM

Having traversed T^* from root to leaves to create a matching graph, we traverse it from leaves to root to determine the optimal solution for each root match. For each leaf vertex in T^*, we merge vertices in H with their parents, keeping the one which has the best score. We repeat this process until only matches to root vertices are left, and then sort these root vertices by the score of the best tree which uses them. This process is described in Algorithm 4, and has complexity of $O(|E^h|)$.

This process yields a set of matches, M_{T^*}, for each potential activity - generally on

Algorithm 4 Solve for Optimal Matches

procedure OPTIMIZE MATCHES(T^*, H)
  $Score(v) \triangleq K_v(v[0], v[1])$
  $Score(v_1, v_2) \triangleq K_e(v_1[0], v_1[1], v_2[0], v_2[1])$
  Iterate from leaves to root
  for all $v_t \in T^*$ **do**
    if $Parent(v_t) \neq \emptyset$ **then**
      for all $v \in V^h$ where $v[0] == Parent(v_t)$ **do**
        $Score(v) += max_{c \in Children(v)}(Score(c) + Score(v, c))1(c[0] == v_t)$
      end for
    end if
  end for
end procedure

the order of the number of true matches in the data. We then iterate through each match $m \in M_{T^*}$ and compute $S(Q, C, m)$, filtering matches that have poor scores for the edges which were not included in the MDST. This allows us to have the speed of the MDSM approach and the effective quality of the full graph-matching result.

4.4 Experimentation

The novelty of this method is its ability to reason over events that are not necessarily extremely time-localized. Other approaches [Le et al., 2009, Castañón and Caron, 2012, Lin et al., 2014] rely on events which span relatively short amounts of time (seconds), whereas our subgraph matching formulation is agnostic to activity duration. For the purposes of comparison, we compare performance on both relatively local events such as dismounts and object loading as well as long-term events like group meetings.

4.4.1 Comparisons

The most comparable approach to our algorithm is the system described in [Castañón and Caron, 2012]. The authors formulate the optimization purely as a dynamic program and assume that the input graph Q is a line graph. Given a line graph representing a series of

temporal events, the system expects events to happen sequentially, but employs dynamic time warping to allow for flexible duration.

Specifically, given a sequence of vertices $V_1, V_2, ...V_i$, they score activities using a Markov model with three types of fixed transition probabilities. When an activity stays in the same state, the transition probability $P_{i,i}$ is a continuation or a missed detection, depending on if there is evidence in the archive to support this transition. When the activity moves to the next state, the transition probability $P_{i,i+1}$ is either a match or a deletion, depending on if there is evidence in the archive to support this transition. Given that their model is a line, they can recover the set of optimal matches in polynomial time.

While this works well for sequential events over relatively short periods of time that always have something happening, it fails at detecting events that have a graph structure, multiple salient objects at a given point in time, or minutes of duration. Specifically, the linear decay in likelihood that occurs from not seeing the next vertex is a poor model when the next piece of evidence could be minutes away. We compare to the baseline established in this method, and note that we expect comparable performance on short-term chains of events and improved performance in other areas.

We compare this baseline against two other approaches, both of which utilize the MDST reduction. In the first (MDSM, in red), we use MDSM to rank matches to the MDST, T^*. We select the top few of these matches, and re-score them using the query graph instead of the graph. In the second (brute force, in blue), we evaluate every subgraph in the matching graph H and rank it using query graph Q. By comparing directly against the scores in a brute-force fashion, we demonstrate how much error is due to imperfect modeling.

4.4.2 Datasets

We evaluate the performance of our algorithm on two data sets representing different surveillance modalities. The first is a wide-area persistent surveillance sensor flying over Yuma, Arizona. This dataset has an extremely low framerate (.66 FPS), combined with

large black-and-white frames (64 megapixels) over a large area, totalling to 1 TB of imagery, or a graph with 1 million nodes and 5 million edges. Because of the size of the area, there are relatively few pixels on a target. Vehicles have roughly 100 pixels and people have around 10. See an example of a u-turn activity in Figure 4·6.

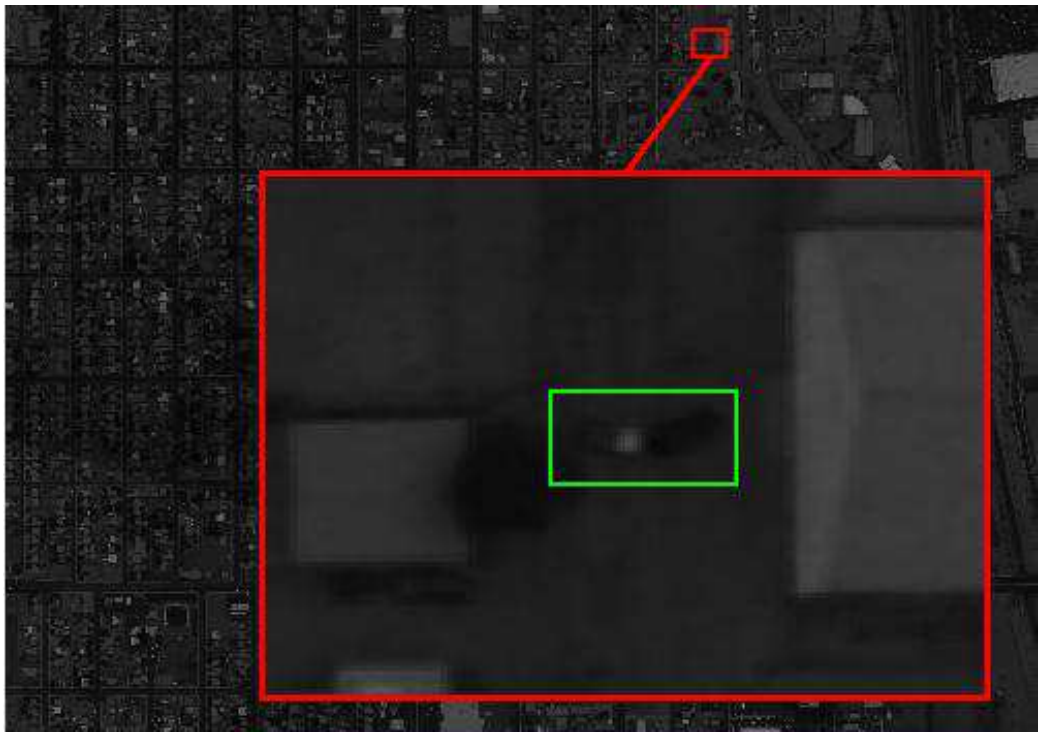

Figure 4·6: The YUMA video data set features high-resolution imagery taken at significant elevation, leaving few pixels on targets as small as people or vehicles.

Because of the low resolution on target, pixel-level features are extremely noisy in the YUMA dataset. For this dataset, we rely on a very simple tracking approach to generate track IDs, bounding boxes, time-stamps and size. We extract from these tracks a series of locations with the features of object size, motion orientation and magnitude and a start/end attribute denoting the beginning and termination of the track that serve as our attribute vocabulary $\mathcal{A}$, similar to the raw attributes extracted in [Hu et al., 2007, Castañón and Caron, 2012]. We employ object size as a way of disambiguating cars and people - carried objects are not visible at this resolution. The discriminative relationship vocabulary $\mathcal{R}$

Query	MDST Reduction	Baseline [Castañón and Caron, 2012]	Brute	Brute (MDST)	DP (MDST)
Long Meeting	85	6.5	*3e25*	4610	2.3
Short Meeting	86	6	*3e25*	10671	2.5
Deposit	<1	12.5	*4.5e16*	5	<1
Removal	<1	14.4	*4.7e16*	4.9	<1
Mount	<1	6.3	*3.7e8*	<1	<1
Dismount	<1	7.7	*3.7e8*	<1	<1
UTurn	<1	2.0269	*6.6e8*	<1	<1
Dismount	<1	<1	<1	<1	<1
Mount	<1	<1	<1	<1	<1
Suspicious Stop	<1	<1	32.2820	<1	<1

Table 4.1: The run times for baseline [Castañón and Caron, 2012], brute force and DP algorithms on the VIRAT (top) and YUMA (bottom) datasets. When a brute force algorithm is infeasible, we estimate run-time based on a sub-set of solutions.

contains a "near" operator and an identity operator which provides a confidence that two elements are from the same object. Spatiotemporal and feature displacement are the non-discriminative relationships that are used in queries, but not hashed. For the VIRAT dataset, which has significantly higher target resolution, we also include object type as a discrete attribute, as object type can reliably be detected.

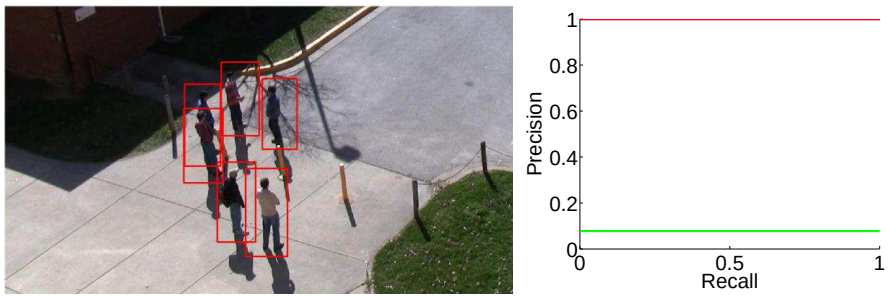

Figure 4·7: Precision/recall curves for the meeting activities in the VIRAT video dataset. Note that we optain perfect precision/recall for both brute force and dynamic programming approaches.

The queries for the YUMA dataset originate from analysts working in the surveillance community, and are similar to the queries in [Hu et al., 2007], such as mounts, dismounts,

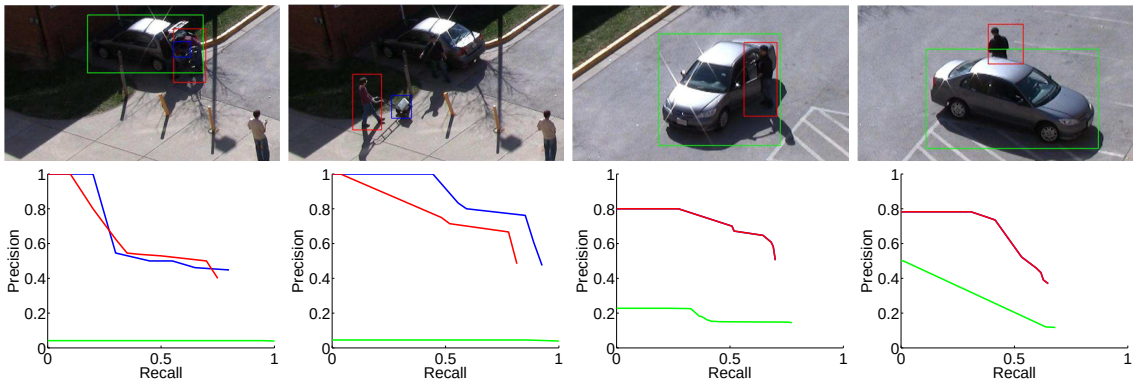

Figure 4·8: Precision/recall curves for the object deposit, object removal, mount and dismount activities in the VIRAT Ground Dataset. We show baseline [Castañón and Caron, 2012] search results in green, MDSM results in red, and brute force results in blue when they diverge from MDSM results.

suspicious stops and u-turns. We compare to a baseline approach based on feature accumulation used in [Castañón and Caron, 2012], and to a brute-force enumeration of possibilities. The precision/recall curves are shown in Figure 4·8.

The other surveillance modality we explored was street surveillance from building-mounted cameras. For this, we used the VIRAT 2.0 Ground Dataset [Oh et al., 2011], a popular surveillance dataset containing 35 GB of video and represented in a graph of 200,000 nodes and 1 million edges. These are relatively standard 2 megapixel surveillance cameras shooting at 30 frames per second. Because of the smaller field of view, there are far more pixels on target, enabling basic object recognition to be performed. As such, we define $\mathcal{A}$ to include object type (person, vehicle, bag) as well as the attributes used in the YUMA dataset. The relationship vocabulary $\mathcal{R}$ is identical. To further differentiate it from the YUMA dataset, the VIRAT ground dataset contains 315 different videos covering 11 scenes rather than a single large video covering one scene.

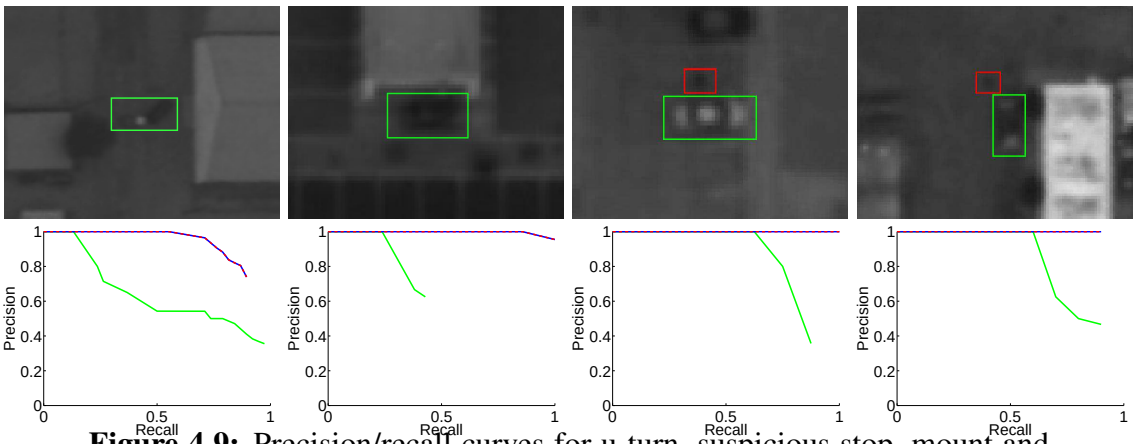

Figure 4·9: Precision/recall curves for u-turn, suspicious stop, mount and dismount activities in the YUMA dataset. We show baseline [Castañón and Caron, 2012] search results in green, unfiltered MDSM results in red, and brute force results in blue.

4.4.3 Results

We also demonstrate the run-time of our approach in Table 4.1. This demonstrates the futility of solving a large-scale graph search problem without performing intelligent reduction first. It should surprise nobody that an algorithm which must explore all $|V^q|$-sized subsets of the data will take forever to run on a large dataset. More relevant is how long it takes us to compute the MDST and downsample the data to the relevant subset: Less than a second with pre-hashed relationships (all examples except meetings), and 80 seconds when we do not have pre-hashed relationships. When we do not have hashed relationships, our algorithm must compute pair-wise relationships, which is expensive to do even when the data is significantly reduced.

The "before" relationship that is used in the meeting events is not hashed because it is not particularly local. Many elements of a dataset satisfy the relationship that one of them comes significantly before the other. Therefore, this relationship is not particularly discriminative by itself, so we would have no a-priori reason to calculate it and store it. Here we run into a classic memory vs. run-time trade off, where hashing less-discriminative relationships may turn out to later be valuable, but is prohibitively memory expensive. The

"before" attribute becomes discriminative when paired with the "same as" attribute so our algorithm performs successful reduction, albeit slower due to a lack of hashing.

The performance of our approach, compared to baseline and brute force solutions is shown in Figures 4·8 and 4·9. Our algorithm performs extremely favorably relative to the baseline, and comparably to brute force in a number of scenarios. The performance gap for simple, time-localized graphs on the YUMA data is relatively small, so the baseline approach performs relatively well. In addition, the simple/time-localized graphs have relatively few cases of complex spatial relationships that are difficult to model with time-sequences like those in Figure 4·3.

As we introduce more complex sets of relationships that are difficult to model, such as mounting and dismount in the VIRAT dataset, the performance of the baseline tails off. Finally, once we introduce large temporal gaps on the order of minutes, the ability of competing approaches to detect these events ceases, while the subgraph approach persists. In longer-term events such as bringing an object to a car and depositing it, as well as long-term group meetings, the baseline approach fails entirely while the performance of the graph-matching approach is relatively constant.

These experiments convincingly showcase our approach's ability to represent events with multiple entities over multiple time scales while still maintaining strong performance in precision and recall.

4.5 Conclusions

In this chapter, we presented a practical solution to video search in both run-time and performance quality. Our method has many advantages. First, it is flexible; we are able to model every activity present in our two video corpora, and many activities (such as long-term meetings) that were not intentionally present. Second, it is fast; we downsample the data using the MDST and solve most examples in a matter of seconds using maxi-

mally discriminative subgraph matching. Finally, use of a sparse graphical representation of complex activities makes it surprisingly flexible and robust. The subgraph matching formulation allows it to ignore foreground clutter and confusion to identify which elements of the archival data match the query optimally. These attributes combine to make it a powerful new tool in video search.

Chapter 5

Successive Search Space Reduction

This chapter focuses on a statistical approach to search space reduction in the subgraph matching problem and demonstrates its efficiency for large-scale video activity retrieval applications. Given query and archive graphs, we pose the search problem as a ranked subgraph matching problem and leverage the statistical properties of the query and archive graphs in a recursive search space reduction based on tree-matching. At each iteration, we solve for the spanning tree that leverages the statistics of the archive and query graph to optimize the volume of search space eliminated. We then perform a polynomial-time tree-matching until the search space has been minimized. We demonstrate our scaling relative to other search space reduction approaches on a synthetic dataset, and show superior performance to window-based approaches on popular large-scale video surveillance datasets.

5.1 Introduction

With advances in multimedia collection comes an increased emphasis in algorithms that reason over large data corpora. From copyright protection to video compression to activity and object recognition, there is a pressing need for high-quality algorithms that can both represent the data efficiently and draw conclusions in real-time.

Many recent high-quality representations are graph-based ([Felzenszwalb et al., 2010, Tianmin Shu et al., 2015, Choe et al., 2013]), with nodes that capture local aspects of a video or image, and edges that represent relationships between these aspects. These representations have been proven to be robust to many types of low-level distortion commonly

found in video as well as the inherent variance within object and activity classes. While these graph-based approaches provide a powerful way to represent objects and activities in images and video, they rely on graph and subgraph matching, both *NP*-hard algorithms, which can lead to challenging scaling for large datasets.

In order to render these algorithms practical, many approaches utilize search space reduction, a pre-processing step which can be done in polynomial time to reduce the problem size and enable an *NP*-hard algorithm to run in reasonable time. The most popular of these methods is a sliding window approach, which pre-defines a box of space-time and slides it through the video, processing in each box independently. With the advent of massive parallelization, this approach is particularly appealing, as each box can be evaluated independently. The primary limitation of this approach is that it assumes foreknowledge of the spatiotemporal extent of the search activity. For most long-term activities, the spatiotemporal extent can vary significantly, limiting most activity detection in large-scale video to extremely localizable activity queries.

In contrast to the approach of spatiotemporal windowing, we propose a method of search space reduction that is agnostic to the location and spatiotemporal extent of the query. We treat the archived video corpus as a graph with nodes and relationships, and make no assumptions about the embedding of those nodes in space or time. We identify discriminative subtrees of the query graph and use them to eliminate large swathes of the archival graph for the purpose of matching. We demonstrate the efficacy off this approach both on complex simulated datasets and real applications in activity search and bioinformatics.

5.1.1 Related Work

Subgraph and graph matching are problems that have been worked on extensively by several communities, with applications in a wide range of problems. Within the computer vision community, graphs are frequently used to represent images and videos [Felzenszwalb

et al., 2010]. Given a large image corpora, approximate graph matching has been used to find images that are similar to query images in both object recognition [Riesen and Bunke, 2009,Johnson et al., 2015,Zhou and De la Torre, 2013] and near-duplicate detection [Zhang and Chang, 2004] for the purposes of copyright protection. Popular approaches to solving this graph-matching problem include spectral techniques [Leordeanu and Hebert, 2005], bipartite graph matching approximations [Riesen and Bunke, 2009], and factorized graph matching [De la Torre, 2012].

Efficient subgraph matching in images has received less interest as large image datasets are generally decomposed into millions of individual images that can be represented as graphs, as opposed to one large image. However, early work on detection in aerial surveillance datasets pioneered approaches for subgraph matching in large image graphs [Christmas et al., 1995, Cordella et al., 2004].

Graphs have also proven to be robust representations of activities in video datasets [Ryoo and Aggarwal, 2010]. In small video datasets [Blank and Gorelick, 2005, Schuldt et al., 2004], where each video contains a single action, graph matching has been successfully used to identify examples of a particular activity [Çeliktutan et al., 2012]. In more complex datasets [Soomro et al., 2012, Oh et al., 2011], multiple approaches have been employed to make the graph search computationally feasible. Decomposition of activities into strings of graphs that can be detected in images [Gaur et al., 2011, Castañón and Caron, 2012] has been proposed. Similarly, other approaches search for tree-like query graphs [Ryoo and Aggarwal, 2010, Kwak et al., 2013] and approximate graphs with trees [Castañón et al., 2015]. Others [Tianmin Shu et al., 2015] rely on a sliding window approach to make the problem manageable. These approaches rely on the action or activity being searched for having relatively small spatiotemporal extent.

As subgraph matching problems grow larger, focus shifts away from finding the solution to the subgraph-matching problem and towards search space reduction. These al-

gorithms have two focuses: to rapidly identify locations where matches are unlikely to occur [Sun et al., 2012, Bhattacharjee and Jamil, 2012] and to eliminate potentially redundant searches [Solnon, 2010, Ullmann, 1976, Bonnici et al., 2013]. Most approaches employ local structure, and some recent works have utilized the aggregate statistics of the dataset to guide their search [Castañón et al., 2015].

In this chapter, we employ successive search-space refinement on both complex simulated graphs, a popular bioinformatics dataset [Huehne and Suehnel, 2009] and a large surveillance video dataset [Oh et al., 2011]. Unlike sliding window approaches [Tianmin Shu et al., 2015], we make no assumptions about the graph having small spatiotemporal extent. We also do not assume a tree-like structure in the graph, remaining both agnostic to graph structure and whether or not it is embedded in a real three-dimensional space.

5.2 Problem Setup

Subgraph matching is a common formulation for problems that seek to find one or more examples of an object/activity/structure in a larger database. In subgraph matching, we are presented with a query graph Q with node and edge sets $V(Q)$ and $E(Q)$, respectively, and an archive graph A with node set $V(A)$ and edge set $E(A)$. Given distance functions $d_v(v_1, v_2)$ measuring the distance between two nodes v_1 and v_2 and $d_e\left((v_1, v_2), (v_3, v_4)\right)$ measuring the distance between two edges (v_1, v_2) and (v_3, v_4), the best subgraph match looks to find a mapping $m : V(Q) \rightarrow V(A)$ where:

$$\underset{m \in \mathcal{M}}{\operatorname{argmax}} S\left((m, (V(Q), E(Q)), (V(A), E(A))\right) \tag{5.1}$$

$$S(m) = \sum_{v_q \in V(Q)} d_v(v_q, m(v_q)) + \sum_{e_q \in E(Q)} d_e(e_q, m(e_q)),$$

where $\mathcal{M}$ is the set of all possible matchings between the query and archive graph and $m(e_q)$ is the mapping of edges induced by the mapping of nodes $m(v_q)$.

An extension of this is subgraph ranking, which will try to return all matchings $m \in (M)$ for which $S(m)$ is within a threshold δ of an exact match. For exact subgraph ranking problems, δ is set to zero. Where subgraph ranking returns the set of all subgraphs satisfying the criterion, subgraph matching returns the single best match.

By far the most common subgraph matching problem is exact subgraph matching with discrete attributes. In these problems, each node v contains a feature vector f_v with features from the set $\mathcal{F}_v$. These features can include spatiotemporal location, speed, size, color, object or protein type. Each edge $e \in E(Q)$ contains a feature vector f_e from the space of features $\mathcal{F}_e$. In video, this may represent that two vertices are associated with the same object, or are near each other. In other datasets, it may represent a specific interaction between the two nodes. In exact subgraph matching with discrete attributes, distance functions $d_v(v_a, v_i)$ and $d_e((v_a, v_b), (v_i, v_j))$ are generally given by:

$$d_v(v_a, v_i) = \mathbb{1}_{\{f_{v_a} = f_{v_i}\}}, \quad d_e((v_a, v_b), (v_i, v_j)) = \mathbb{1}_{\{f_{(v_a, v_b)} = f_{(v_i, v_j)}\}}. \tag{5.2}$$

This problem is generally formulated as an integer quadratic program (IQP). Let $X \in \{0,1\}^{|V(Q)| \times |V(A)|}$ be an assignment matrix from query graph Q to archive graph A, and let $x \in \{0,1\}^{(|V(Q)||V(A)|)}$ be its column-wise vectorized replica. Let weight matrix W be a $|V(Q)||V(A)| \times |V(Q)||V(A)|$ matrix, where $W_{ia;jb}$ denotes the score for matching query edge (v_i^q, v_j^q) to archive edge v_a^a, v_b^a, $d_e((v_a, v_b), (v_i, v_j))$. The diagonal values of the weight matrix, $W_{ia;ia}$, represent the score for matching query node v_i^q to archive node v_a^a, $d_v(v_a, v_i)$. Then, the IQP formulation is:

$$x^* = \text{argmax}(x^T W x) \tag{5.3}$$

$$\text{s.t.} \quad x \in \{0,1\}^{(|Q||A|)}, \quad \forall a \sum_{i=1}^{|V(Q)|} x_{ia} \leq 1, \quad \forall i \sum_{a=1}^{|V(A)|} x_{ia} \leq 1 \tag{5.4}$$

While this objective suggests that our task is to find an instance with maximum similarity with the query, in practice, our goal is to identify matches that meet a user defined threshold. Specifically, we let $\mathcal{U}_\tau$ denote the collection of such instances, namely,

$$\mathcal{U}_\tau = \{m \in \mathcal{M} \colon\ S(m, (V(Q), E(Q)), (V(A), E(A))) \geq \tau\} \tag{5.5}$$

Identifying the set $\mathcal{U}_\tau$ is in general intractable involving combinatorially many choices of matching m. In the context of video retrieval, the archive graph can involves billions of nodes. In addition we typically encounter query graphs that are relatively small. Consequently, a highly effective approach is to identify a mapping function $F : V(A) \rightarrow V'$ which maps the space of archive nodes $V(A)$ to a smaller, filtered space of nodes A'.

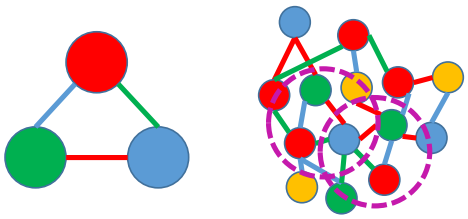

Figure 5·1: Given a query graph, our goal is to find the set of matchings in the archive graph above a particular score threshold.

5.3 Subgraph Matching by Refinement

A. Sliding Window: This approach is based on dividing up the data archive graph based on spatiotemporal bounding boxes. The set of bounding boxes represented as the set of subgraphs $\mathcal{B}$, where each element of $\mathcal{B}$ is a subgraph of the archive graph with nodes that are spatiotemporally close. For each subgraph induced by a bounding box, the subgraph matching problem can be solved independently. This approach, shown in Figure 5·2 works

best if we can infer the maximum size that a query graph could take, and select the set of bounding boxes $\mathcal{B}$ to be that size.

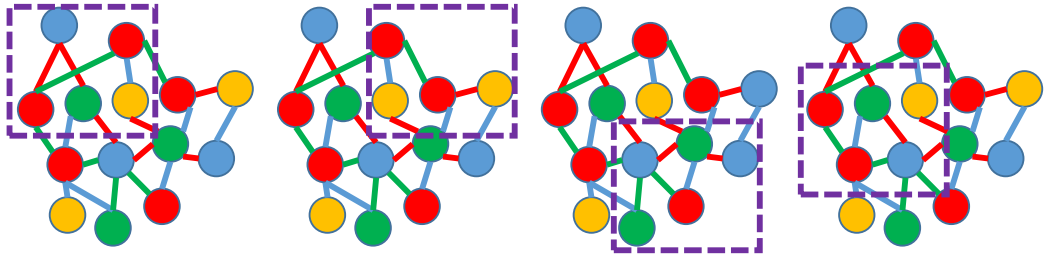

Figure 5·2: Given a query, sliding window search space reduction creates a window, shown in purple, and slides it through the space, creating a filtered archive graph containing only nodes and edges within the window.

This approach leads to a local search space around the anchor node that scales with the size of the query. We can then approximate $\mathcal{U}_\tau$ by examining only matches with a similarity greater than τ for each bounding box, i.e.,

$$\mathcal{U}_\tau = \bigcup_{b \in \mathcal{B}} \{m \in \mathcal{M} \colon\ S(m, (V(Q), E(Q)), (V(b), E(b))) \geq \tau\}, \tag{5.6}$$

where $V(b)$ and $E(b)$ are the node and edge sets, respectively, of the subgraph b.

In general, $\mathcal{B}$ is chosen as a set of overlapping windows with size roughly equal to the maximum query size that contain archive elements. This step works well when the volume of the query is relatively small with respect to the data-archive. This is generally not the case in many video retrieval problems. Furthermore, note that the this process requires scoring similarity for each bounding box, which can be cumbersome for very large graphs.

B. Node/Edge Filters: While the sliding window method is robust, it does not fully exploit the feature information of the query and archive data. This information can be used to eliminate many nodes and edges as potential candidates. It follows that if Eq. 5.5 has to be satisfied, then each node or edge must satisfy a sufficient level of similarity. This motivates the following filtered node and edge sets:

$$V'(A) = \{v_a \in V(A) \colon \exists v_i \in V(Q) \colon \ d_v(v_a, v_i) \leq \tau_v\} \tag{5.7}$$

$$E'(A) = \{(v_a, v_b) \in E(A) \colon \exists (v_i, v_j) \in E(Q) \colon \ d_e((v_a, v_b), (v_i, v_j)) \leq \tau_e\} \tag{5.8}$$

This approach involves the choice of thresholds τ_v, τ_e to ensure we meet the user-described global similarity. We denote the induced graph A' with node and edge sets $V'(A)$ and $E'(A)$ as the *filtered graph* of archive graph A.

C. Query Sub-Tree Filters: Node/Edge filtering can reduce the search space significantly if the associated query features are somewhat "rare" or unique. On the other hand many retrieval tasks are described by features that occur commonly when viewed in isolation. It is their *co-occurrence* that is uncommon.

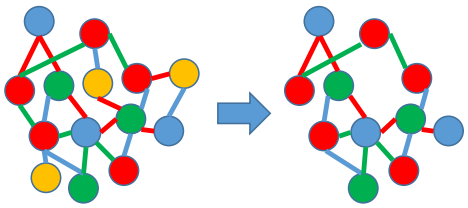

Figure 5·3: Node/Edge filters remove nodes and edges from the archive graph which could not independently match any nodes or edges in the query graph.

This suggests that we should attempt to design queries that are subgraphs of our query graph. Our goal is to seek large search space reduction (precision rate) with little degradation in recall. Note that for exact sub-graph matching, the recall rate is not impacted by deleting edges in our query graph. This is because the collection of instances that match the query with a deleted edge contains the true instances as well. We propose to use spanning trees of the query graph as queries to reduce the search space, as dynamic programming can be used to score similarity for tree queries.

The goal is to find tree queries that lead to maximal search space reduction. We propose an entropy minimization method to estimate query sub-trees. Let $P_i(\tau_v), P_{ij}(\tau_q)$ be empirical frequencies in the data-archive that match features at node $v_i \in V(Q)$ and edge

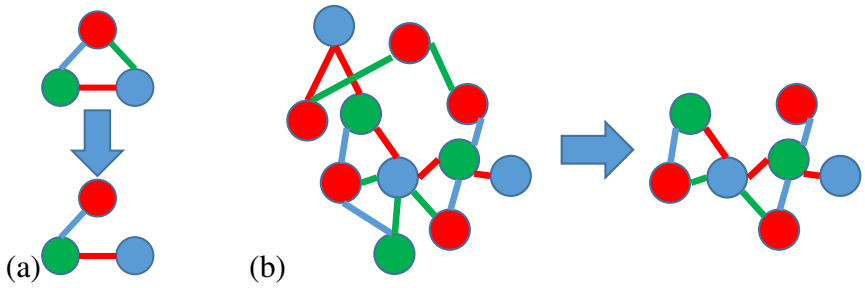

Figure 5·4: In (a), we select a minimum spanning tree of Q to be our query tree T. In (b), we solve for all matches to T in the archive graph and remove nodes not present in a matching.

$(v_i, v_j) \in E(Q)$ with distance less than τ_v and τ_e, i.e.,

$$\hat{P}_i(\tau_v) = \frac{1}{|V(A)|} \sum_{v_a \in V(A)} \mathbb{1}_{\{d(v_a, v_i) \leq \tau_v\}}, \tag{5.9}$$

$$\hat{P}_{ij}(\tau_e) = \frac{1}{|E(A)|} \sum_{(v_a, v_b) \in E(A)} \mathbb{1}_{\{d((v_a, v_b), (v_i, v_j)) \leq \tau_e\}}. \tag{5.10}$$

Let $\mathcal{T}_Q$ be the collection of trees over the query graph Q. Our goal is to select a tree T^*_Q to optimize the following function:

$$T^*_Q = \underset{T \in \mathcal{T}_Q}{\operatorname{argmin}} \sum_{v_i \in V(T)} \log(\hat{P}_i(\tau_v)) + \sum_{(v_i, v_j) \in E(T)} \log \hat{P}_{ij}(\tau_e), \tag{5.11}$$

where $V(T)$ and $E(T)$ are the node and edge sets associated with tree T.

The solution is a min-cost spanning tree as described below.

Lemma 5.3.1. *The solution to Eq. 5.11 is a minimum cost spanning tree.*

Note that if we were to remove binary constraints on the variables x_q the optimal solution has minimal entropy over all tree choices. We can expect maximal search space reduction with this query without degradation in recall rate. Indeed, if the node features and edge features of the query tree were generated independently at random in the data-archive our minimum cost query would optimally reduce the search space. We state this formally below:

Lemma 5.3.2. *Suppose the node and edge attributes take values in a finite set. Let Q be a fixed query. Let A be a data archive where the node and edge features are generated independently at random. Suppose the similarity function is binary valued (corresponding to exact matching of attributes). Define $\mathcal{U}_0(T_Q)$ as the set of matches obtained by running the sub-tree query T_Q. Then among all trees we have that,*

$$Prob\{\mathcal{U}_0(T_Q^*)\} \leq Prob\{\mathcal{U}_0(T_Q)\}$$

where the probability is taken with respect to data-archive generation.

Our approach combines commonly-employed node/edge filters with our own novel successive approach to query sub-tree filtering. Our intuition is that certain structures and sub-structures in our graph are sparse. We know from existing work [Demeyer et al., 2013,Castañón et al., 2015] that tree-matching can be performed in polynomial time, which makes it feasible even on larger problems. So we use the statistics of the archive graph to identify the structures and substructures within our query graph that are rarest. We use these to create a tree, T^* that is the most unlikely to be randomly matched in the archive, allowing us to maximally reduce the search space.

D. Successive Query Sub-Tree Filters: Due to the fact that matching trees is cheap while matching graphs is expensive, we iteratively reduce the search space via tree-matching, yielding notable improvements in run-time. In contrast to other approaches [Castañón et al., 2015], we iteratively reduce the search space by updating statistics of the graph and finding new search trees. To this end, we define the graph A'_t with node and edge sets $V'_t(A)$ and $E'_t(A)$ as the *iterative filtered graph* of archive graph A after t iterations. The initial graph A'_0 is the graph produced by applying node and edge filters, with nodes and edges $V'_0(A) = V'(A)$ and $E'_0(A) = E'(A)$ as defined in Eqns. 5.7 and 5.8.

We define $\mathcal{U}^{t-1}\left(T_Q^t\right)$ as the set of potential tree matchings, where each element of $\mathcal{U}^{t-1}\left(T_Q^t\right)$ is a matching of the archive graph A'_{t-1} to the tree query T_Q^t. Each successively filtered graph A'_t has nodes $V'_t(A) = \left\{\bigcup_{m\in\mathcal{U}^{t-1}\left(T_Q^t\right)}\bigcup_{v_q\in V(Q)} m(v_q)\right\}$ found by taking the intersection between tree query matches and an edge set $E'_t(A)$ composed of edges

remaining between nodes in $V'_t(A)$. The subtree query $T^t_{\mathcal{Q}}$ is defined as

$$T^t_{\mathcal{Q}} = \underset{T \in \mathcal{T}_{\mathcal{Q}}}{\operatorname{argmin}} \sum_{v_i \in V(T)} \log(\hat{P}^t_i(\tau_v)) + \sum_{(v_i,v_j) \in E(T)} \left(\log \hat{P}^t_{ij}(\tau_e) \right), \tag{5.12}$$

the successive frequencies P^t_i and P^t_{ij} are defined as

$$\hat{P}^t_i(\tau_v) = \frac{1}{|V'_{t-1}(A)|} \sum_{v_a \in V'_{t-1}(A)} \mathbb{1}_{\{d(v_a,v_i) \leq \tau_v\}}, \tag{5.13}$$

$$\hat{P}^t_{ij}(\tau_e) = \min \left(\frac{1}{|E'_{t-1}(A)|} \sum_{(v_a,v_b) \in E'_{t-1}(A)} \mathbb{1}_{\{d((v_a,v_b),(v_i,v_j)) \leq \tau_e\}} + \lambda U^t_{ij}, 1 \right), \tag{5.14}$$

and U^t_{ij} is defined as an indicator variable with a value of 1 if the edge ij has been previously used in a tree query and a value of 0 otherwise and λ is a user chosen exploration penalty. Note that the term λU^t_{ij} arises as our approach is iterative, and therefore we bias subsequent tree queries to explore the space of spanning trees. In general, this behavior naturally arises, however to avoid pathological cases where edges are never included in the filter tree $T^*_{\mathcal{Q}}$, we add a penalty of λ to previously explored edges.

As shown in [Castañón et al., 2015], minimizing Equation (5.12) is equivalent to computing the minimum spanning tree of a graph with edge weights $W_{i,j} = log(\hat{P}^t_{ij}(\tau_e))$. It is therefore extremely computationally efficient to compute the tree which minimizes Equation (5.12). We choose the root node of $T^t_{\mathcal{Q}}$ such that the most discriminative edges are near the root. This optimizes tree search and can be done efficiently by enumeration, as there are only at most $|V(Q)|$ options for the root node. We discuss efficient implementation of the tree search below.

Tree Matching: We solve for the optimal match between a tree $T^t_{\mathcal{Q}}$ and filtered graph A' in two steps. In the first step, we build a *matching graph* M with nodes $V(M)$ and edges $E(M)$, where each vertex $v \in V(M) \subseteq V(Q) \times V'_t(A)$ is a tuple denoting a proposed assignment between a vertex in $T^t_{\mathcal{Q}}$ and a vertex in $V'_t(A)$, and each edge $e \in E(M)$

denotes a relationship between the two assignments. We create M by first adding matches for the root node of $T^t_{\mathcal{Q}}$, then adding in matches to its successors which match both vertex and edge relationships described in $T^t_{\mathcal{Q}}$. The complexity of this approach scales as a product of $|T^t_{\mathcal{Q}}|$ and the size of the archive data.

Having traversed $T^t_{\mathcal{Q}}$ from root to leaves to create a matching graph, we traverse it from leaves to root to determine the optimal solution for each root match. For each leaf vertex in $T^t_{\mathcal{Q}}$, we merge vertices in M with their parents, keeping the one which has the best score. We repeat this process until only matches to root vertices are left, and then sort these root vertices by the score of the best tree which uses them. The complexity of this component is $O(|E(M)|)$. This process yields the set of matches $\mathcal{U}^{t-1}\left(T^t_{\mathcal{Q}}\right)$, each of which is a matching from filtered graph A'_{t-1} to the query tree $T^t_{\mathcal{Q}}$.

Given a set of matches $\mathcal{U}^{t-1}\left(T^t_{\mathcal{Q}}\right)$, we can now reduce the size of filtered graph A'_t by removing any vertices or edges that are not present in any matches $m \in M_{T^q}$. We can safely do this because we know that the $T^t_{\mathcal{Q}}$ is a sub-tree of Q, thus guaranteeing that an element which fails one of the edge or node restrictions in the $T^t_{\mathcal{Q}}$-matching process could not meet that condition in the full subgraph matching problem.

The full approach is shown in Algorithm 5.

5.4 Experimentation

We evaluated our work on both synthetic datasets and real-world datasets [Oh et al., 2011, Huehne and Suehnel, 2009]. We measured the effectiveness of the graphical formulation by precision and recall on a real-world dataset, and we measure the effectiveness of our recursive search-space reduction by run-time on the surveillance dataset and explicitly in the synthetic dataset.

Algorithm 5 Successive Search Space Refinement

procedure REFINE SEARCH SPACE($Q, A, \tau_v, \tau_e, \lambda$)
  $A'_0 = A'$ ▷ Initialize using the node/edge filtered archive graph (Eqns. 5.7 and 5.8)
  $U \leftarrow \emptyset$ ▷ Initialize the set of used edges
  $t = 0$
  while A'_t has not converged **do** ▷ Successively refine search space
    for all $v_i \in V(q)$ **do**
      Compute $\hat{P}_i^t(\tau_v)$ ▷ Compute node frequencies according to Eqn. 5.13
    end for
    for all $(v_i), (v_j) \in E(Q)$ **do**
      Compute $\hat{P}_{ij}^t(\tau_e)$ ▷ Compute edge frequencies according to Eqn. 5.14
    end for
    Find T_Q^t ▷ Compute the best query tree according to Eqn. 5.12
    $U \leftarrow U \bigcup E(T_Q^t)$ ▷ Updated the set of previously used edges
    Find $\mathcal{U}^{t-1}\left(T_Q^t\right)$ ▷ Calculate the set of potential tree matchings
    $V'_t(A) = \left\{\bigcup_{m \in \mathcal{U}^{t-1}\left(T_Q^t\right)} \bigcup_{v_q \in V(Q)} m(v_q)\right\}$ ▷ Keep nodes present in a matching
    $E'_t(A) = \left\{E'_{t-1}(A) \bigcap V'_t(A) \times V'_t(A)\right\}$ ▷ Keep edges between nodes in $V'_t(A)$
  end while
end procedure

5.4.1 Datasets

We experimented on three datasets with the intention of demonstrating both the scalability and practicality of our algorithm. As a pure subgraph matching problem, we looked at the Lena Library of Biological Macromolecules [Huehne and Suehnel, 2009], which contains a number of molecular structures. We chose to look at the densest graphs in these datasets: protein backbones and contact maps for amino acids. We also elected to use our approach on the popular VIRAT Ground 2.0 [Oh et al., 2011] dataset for video surveillance. The dataset contains 40gb of video, comprising 11 scenes and 330 videos. Resolution ranges from 100-10000 pixels on an object of pedestrian size. Our queries for this dataset were entering a vehicle and driving off, getting picked up by a vehicle, loading an object into a vehicle, taking an object out of a vehicle, and groups of people meeting for several minutes. We tested both using ground truth and using tracks (explained in 5.4.2). Unfortunately, we were unable to locate a generic object detector which could perform at all on the VIRAT

dataset, so for objects (1.6% of the dataset), we use ground truth.

Most current datasets [Geiger et al., 2012,Oh et al., 2011,Marszałek et al., 2009,Soomro et al., 2012] focus on atomic actions that are extremely local, as sliding-window techniques are designed to detect these. Note that the limited complexity of these queries does not bias results towards our algorithm; increased complexity should increase the performance gap. Our query graphs on the VIRAT dataset range from 3-6 nodes, and 2-9 edges.

While our approach is effective on these datasets, the benefits are accentuated on datasets with more complex structure. We follow the example of the bioinformatics community [De Santo et al., 2003] to simulate large graphs for sensitivity analysis. In our synthetic dataset, we use an Erdös-Rényi model to generate a query graph. In this model, the probability p_e of an edge existing between two nodes is fixed. We generate node an edge attributes from separate discrete distributions with various parameters, discussed below in 5.4.3.

5.4.2 Implementation

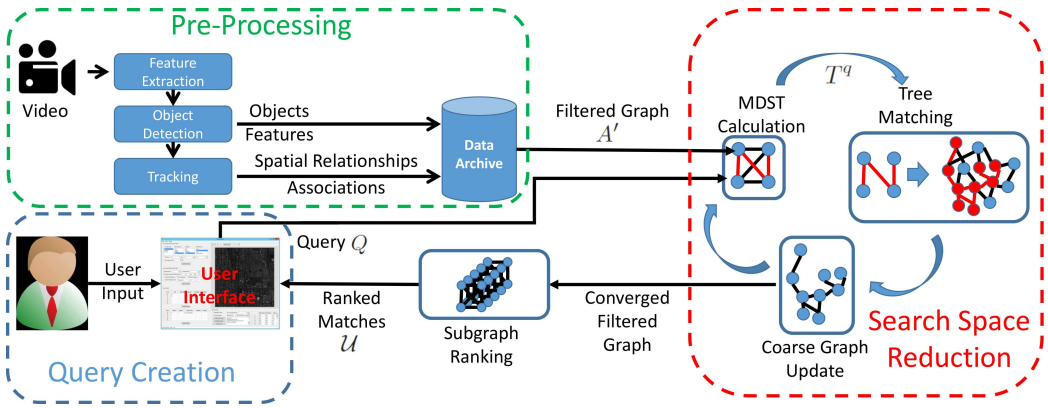

Figure 5·5: The processing chain for our system, applied to video. As video streams in, we extract features for object detection and tracking. We store the resulting detections and associations in a database. When a query Q is created by a user through our GUI, we downsample the data to create a filtered graph A'. We iteratively reduce the filtered graph through matching using our successive search space refinement algorithm, then solve a small subgraph ranking problem to produce results.

As Figure 5·5 illustrates, our system consists of 4 steps:

1. **Object Detection and Tracking.** In order to be able to detect large-scale activities, we need to be able to detect their components: objects, people and vehicles over time. We stabilize streaming video using frame registration, and used Piotr Dollár's MATLAB Toolkit [Dollár,] to extract Aggregate Channel Features [Dollar et al., 2014]. We used that toolkit's person model (trained on a different dataset) to identify pedestrians in the VIRAT dataset. Because the software did not have a pre-trained car detector, we trained one on scenes from the VIRAT dataset that we did not use.

   Due to obscuration, missed detection, shadows and other real-world issues, detections on non-vehicles are of low quality, frequently having missed/multiple detections. In order to mitigate detection error, we post-processed detections by applying non-maximal suppression, and used [Andriyenko et al., 2012] to independently generate tracks for cars and objects. Each track was broken down into a set of space-time bounding boxes that represented the trajectory, and we extracted features for each of those bounding boxes: size, velocity, direction, object type and whether that bounding box the first or the last box in the track.

2. **Storage.** Our video corpus is represented as a large archive graph in which each detection is a node. Edges incorporate both spatiotemporal information ('near') and identity information ('the same as'). We create an independent, inverted index for each feature in the dataset, and hash nodes and edges to each index based on their feature values. Each continuous feature (size, velocity, direction) was hashed to a fuzzy inverse index using locality-sensitive hashing [Datar et al., 2004], while each discrete feature (object type, track start/end) was simply hashed directly. This architecture allowed us to identify bounding boxes which matched a query graph node in time proportional to the number of matches instead of the size of the dataset - an important attribute for a system that aimed to do massive search in real-time.

3. **Query Creation.** When a user wants to locate an activity, they are asked to create a query using our query creation GUI. This allows users to take semantic concepts like 'car', 'person' and 'near' and directly assemble them into a graph. This produces the attributed query graph Q.

4. **Search.** Given query graph Q and archive graph A, the goal is to return a set of ranked video snippets which best match Q. We first use the successive search space refinement approach to eliminate elements of A that could not be present in Q. When that has converged, we use dynamic programming to rank the potential subgraphs, convert those subgraphs to video segments, and display them to the user.

Experiments were run using an intel 2.7 GHz CPU and 4 GB of memory. Note that components of our algorithm (graph filtering, tree-matching) can be parallelized, but for clarity of analysis we elected to forego this option.

5.4.3 Results

Jena Library

The Jena library comes with sets of pre-generated queries, containing graphs of up to 128 nodes. The node attributes had a relatively limited vocabulary; nodes in protein backbones had an average of five unique attributes, and nodes in amino acid contact maps aveaged 20. To determine our algorithm's efficiency at reducing the search space, we ran our successive search space refinement technique and recorded the total reduction after query sub-tree filtering.

Over the 857 pre-generated Jena Library graphs we solved, we saw an average reduction factor of 6.4 in the number of nodes remaining, and 11.4 in the number of nodes remaining. We did not see significant improvement over many iterations, in part because the structure of these queries is very tree-like - no query had twice as many edges as nodes, even as the queries became hundreds of nodes. As a result, the initial tree approximation provided the

lion's share of the reduction.

Simulated Data

In generating a synthetic dataset, we follow the example of [Bonnici et al., 2013] and [De Santo et al., 2003] by simulating large graphs. In our simulations, we randomly generate large (500-3000 node), dense graphs using the Erdös-Rényi model, with connection probabilities $p_{conn} = \{.25, .5\}$. For each graph, we randomly and uniformly select a distribution of features from the set of $|\mathcal{F}_v|$-dimensional discrete distributions, then generate features for each edge and node. We choose the induced subgraph of 10 randomly chosen nodes from the archive graph to be our query graph Q. We then apply our approach to the resulting subgraph isomorphism problem, and compare to a baseline tree matching approach that does not exploit the distribution of attributes (akin to [Bonnici et al., 2013, Kriege and Mutzel, 2012]).

Our goal is to understand under what circumstances a significant advantage can be gained by exploiting data statistics. Intuitively, successive refinement of the search space yields little advantage if the query graph resembles a tree. Likewise, little is to be gained if components of the query graph are extremely uncommon and the subgraph match is immediately apparent. Under either of these cases, all algorithms perform equivalently. As an example, consider the graph A with $|\mathcal{F}_v|$ on the order of $|A|$. In this case, matching a single pair of nodes with an edge between them guarantees that you have matched the whole query graph, so all algorithms are equivalent. In the case where $|\mathcal{F}_v|$ is 1, the graph A is effectively unlabeled, and there are no relevant statistics, so the approaches perform equivalently well. We consider the most common case in large graph datasets, where a small feature vocabulary $\mathcal{F}_v$ exists.

As expected, Figure 5·6 shows that our reduction performs at least as well as the baseline approach for all scenarios. This is expected, as our approach exploits additional information. In Figure 5·6(a) and 5·6(d), we have an isomorphism problem that is relatively

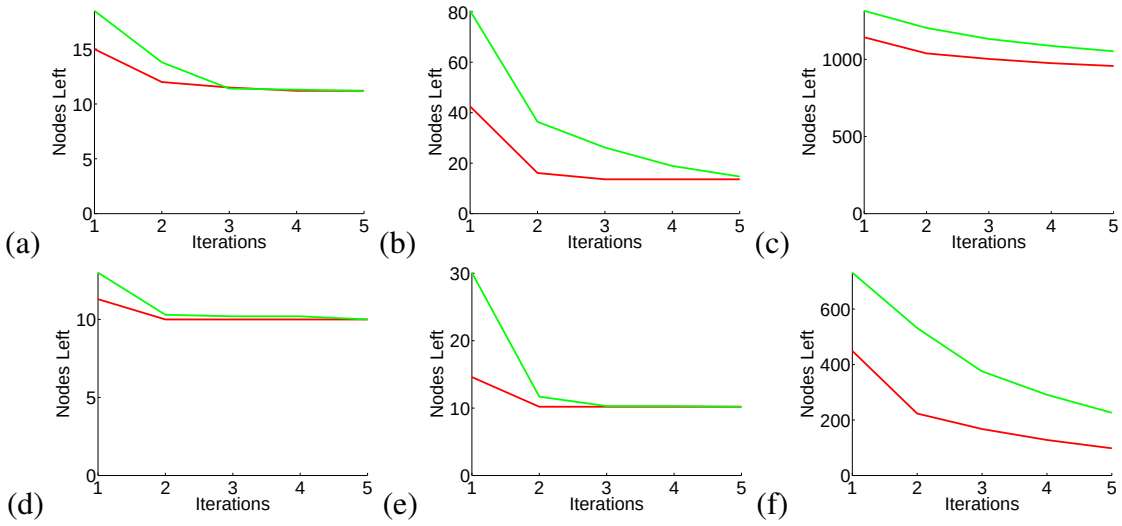

Figure 5·6: The average number of nodes remaining after each iteration for the baseline (green) and successive-search space reduction (red) approaches on archive graphs of size 500, 1000 and 3000. In the first row, $p_{conn} = .25$ and $|\mathcal{F}_v| = 20$. In the second row $p_{conn} = .5$ and $|\mathcal{F}_v| = 30$.

straightforward to solve - there are not a significant number of matches to any subtree of Q, so correct selection does not matter. As the problem size grows, however, we reach the other extreme - 5·6(c), where the graph is so densely connected that almost every node is part of a correct match. However, in the middle, where the problem is non-trivial but solvable, our approach consistently outperforms the baseline.

Using the recursive search space reduction to this technique reduces the number of nodes remaining by a factor of 3.2, and the number of edges remaining by a factor of 33 on average. Given that edges are the dominant term in most subgraph matching approaches, this shows that there is significant value to the recursive approach.

VIRAT Dataset

We tested five queries on the VIRAT dataset: mount, dismount, object deposit, and group meeting. We selected activities with relatively long temporal duration, highlighting the ability of our algorithm to selectively downsample video based on relevance, as opposed to

a small spatio-temporal window.

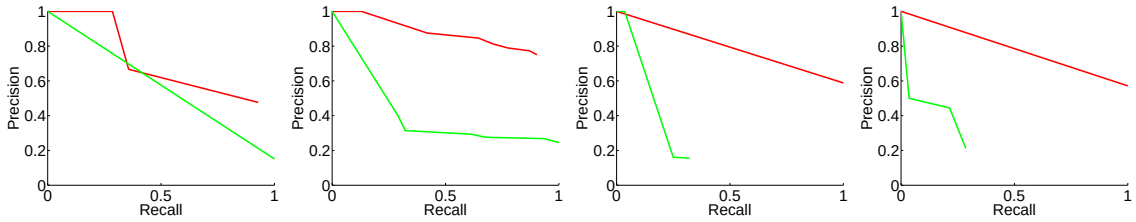

Figure 5·7: Precision/Recall curves for the Object Deposit, Object Take-out, Mount and Dismount activities in the VIRAT dataset. Recursive search space reduction is shown in red, with a baseline approach based on space-time feature accumulation shown in green. In these experiments, ground truth was used for detection and track information.

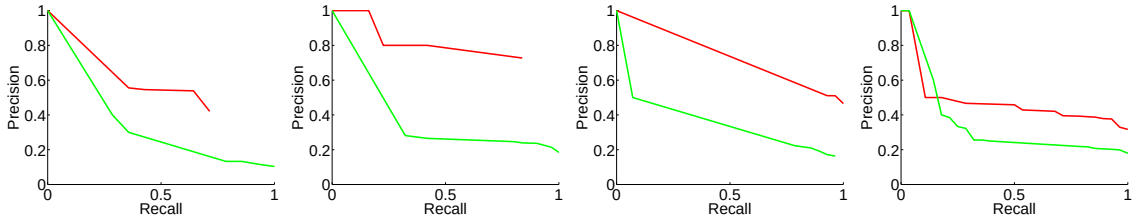

Figure 5·8: Precision/Recall curves for the Object Deposit, Object Take-out, Mount and Dismount activities in the VIRAT dataset. Recursive search space reduction is shown in red, with a baseline approach based on space-time feature accumulation shown in green. We track aggregate channel features [Dollar et al., 2014] using [Andriyenko et al., 2012] in these experiments.

In order to establish a baseline for performance, we compare algorithms on the ground truth provided in [Oh et al., 2011]. This removes the noise due to imperfect detection and tracking. The precision/recall curves for this for our algorithm, applied to ground truth detections, are shown in Figure 5·7. While timing results vary by machine, they can be indicative of feasibility. On this dataset, tree matching iterations took between .7 and 2.3 seconds, and all queries converged within three iterations.

Using the implementation described in Section 5.4.2, we generated tracks on objects and people from raw data. These detections and tracks were significantly fragmented and

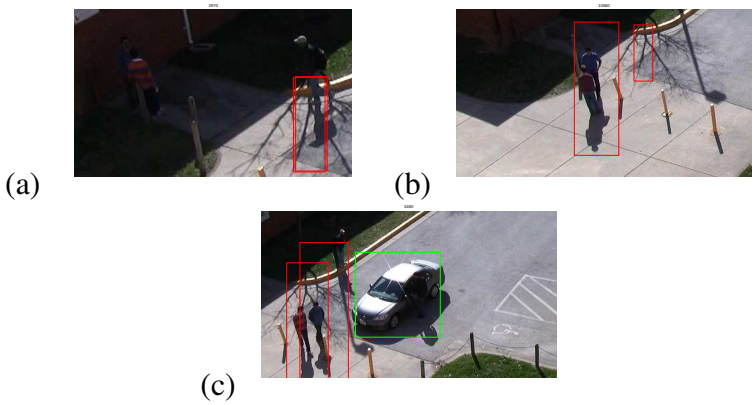

Figure 5·9: This figure highlights some of the issues involved in tracking and detection in surveillance scenarios. In (a), the algorithm misses a pair of people talking in the shadow of a building. In (b), two people talking are mis-classified as a single person. In (c), a person comes too near a car, enters its shadow, and is not detected.

noisy. However, we were able to produce strong results on the shorter term scenarios (under 1 minute) as shown in Figure 5·8. Errors in this dataset were largely due to missed detections. In Figure 5·9 we see why detections are problematic: We miss detections in the shadow of a building, near a car, and when two people are talking with eachother. However, tracks on cars are quite reliable. As a result, detecting long-term events is difficult due to identity gaps over time.

5.5 Conclusion

In this chapter, we introduced a method for iterative search space reduction in large graphs and demonstrated its efficacy on simulated large-scale video surveillance data. We demonstrated a complete system that took in raw surveillance footage, extracted features, created detections and stored these in an archived inverted index. Using a simple query GUI, we enabled a user to input activities in graphical form. In real-time, we downsample the archive data to a filtered graph A', and then use our novel recursive search space reduction algorithm to solve the *NP*-hard problem of graph matching in real-time even for large

datasets.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, we have demonstrated that a structure and relationship-based approach to search is an efficient and effective way of finding things in large datasets - both video and otherwise. Specifically, we have shown that it is reasonable and realistic to represent activities in video through combinations of simple elements structured appropriately and that these representations can be extremely discriminative in video.

In our first work, we focused on simple temporal structure with an extremely limited query vocabulary. The diverse set of activities represented and strong results compared to state-of-the-art topic modeling and feature accumulation approaches suggested that structure was an effective tool for retrieval. As further work expanded our exploitation of structure to include trees, arbitrary graphs, and non-temporal relationships like identity and proximity, it became clear that a weak set of descriptors could be significantly empowered by user-provided structure.

At the core of this advance is a simple idea that should not be lost: as structure grows more complex, it becomes more powerful, as it is less likely that it will be incidentally replicated by chance. This makes sense - a system, provided with more information, should perform better. With more complex structure comes a shared burden on user and the GUI they are using to make query creation accessible and simple. We showed that this can be done effectively in the area of video surveillance.

In 4, we also showed that, given a graphical structure provided by a user, subgraph

matching was an effective way to search over large video corpora in a manner that was agnostic to background clutter. In order to mitigate this, we again exploited structure to dramatically reduce the search space via tree matching with Maximally Discriminative Spanning Trees. We showed that tree-matching, performed in a single step or iteratively, is a strong method for reducing the search space in subgraph matching problems, both video and otherwise.

In conclusion, properly exploited structure can compensate for other deficiencies (e.g. simple attributes) in a video search system. Subgraph matching is an important way to do video search in crowded scenes, but is computationally impractical without a strong technique for search space reduction. It is our hope that future works will make use of these techniques to advance the science of search in both video and other domains.

6.2 Future Work

Like all theses grounded in an unsolved real-world problem, there are a number of potential extensions for this work. In this section, we will strive to name some of the most promising.

First is the question of query generation. Over the course of Chapters 3 through 5, we've shown that we have a compact and efficient representation for activities which differentiates them from background or unrelated activity. While having direct human input is an efficient way to solve this problem, it does assume a level of system mastery. We've found this mastery to be readily present in those who have tried our system, but in event where you'd want an untrained user to be able to search, it would be good to be able to train our model from exemplar videos, similar to [Lin et al., 2014].

Second, it would be interesting to explore how we could recover significantly disconnected/untracked queries. While the video corpora currently available are relatively straightforward and feature minimal obscuration, creating long-term tracks in arbitrary surveillance video has proven to be extremely challenging. With the aim of recovering

long-term activities like “group meeting”, it is less important for us to know exactly where somebody is for the entirety of a video than to efficiently tell if two different locations or points in time contain the same person. To this end, we could shift the burden of creating “same as” relationships from archival time to query time by using re-identification algorithms like [Zhang et al., 2014b].

Third, given time we would like to explore the possibilities of iterating with a user to refine search queries. In traditional active learning for search [Hsu et al., 2007], users inform the search engine if a result is correct or wrong. This sort of binary feedback represents the lowest-bandwidth communication channel between human and algorithm. It would be interesting to explore both if this approach is effective at refining graphs, particular in comparison to manual refinement.

Appendix A

Proofs and Additional Detail

A.1 Theorem: Low False Positives

Theorem A.1.1. *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. □

A.2 Proof of Lemma 4.3.1

Definition A.2.1 (Maximally Discriminative Spanning Tree (MDST))**.** *We call a tree is an MDST, T^*, to a query graph Q with node weights $\forall v \in V^q, p(v)$ and edge weights $\forall e \in E^q, p(e)$, if the tree satisfies*

$$T^* = \underset{T\in\mathcal{T}}{\operatorname{argmin}}\, p(T), \tag{A.1}$$

where $\mathcal{T}$ denotes the set of all possible trees induced from query graph Q.

Proof. Based on the definition of MDST, we can rewrite Eq. 4.3 as follows:

$$\begin{aligned} T^* = \underset{T\in\mathcal{T}}{\operatorname{argmin}}\, p(T) &= \underset{T(V^t,E^t)\in\mathcal{T}}{\operatorname{argmin}} \prod_{v\in V^t} p(v) \prod_{e\in E^t} p(e) \\ &\equiv \underset{T(V^t,E^t)\in\mathcal{T}}{\operatorname{argmin}} \left\{\sum_{v\in V^t}\log p(v) + \sum_{e\in E^t}\log p(e)\right\} \end{aligned} \tag{A.2}$$

Since $\forall p(v), \forall p(e)$ are real numbers between 0 and 1, $\forall \log p(v)$ and $\forall \log p(e)$ are all non-positive. Therefore, in order to minimize Eq. A.2, all the nodes in Q will be included, and all the specific edges will be included as well, which (1) form a tree together with all the nodes in Q, and (2) achieve the minimum summation. Deleting any node/edge from the tree will either increase the objective value in Eq. A.2 or destroy the tree structure. This is exactly the same as the definition of minimum spanning tree. □

A.3 Proof of Lemma 4.3.2

Lemma A.3.1. *Let M denote a match between a query graph $Q = (V^q, E^q)$ and a coarse graph C. Let $S(\cdot, C, M)$ denote the matching score between any query (e.g. graphs, trees, attributes, relationships) and C, and let $Q' = Q - T^* = (\emptyset, E^q - E^t)$ denote the residual of graph Q. We further define $S(Q, C, M) = \sum_{v \in V^q} S(v, C, M) + \sum_{e \in E^q} S(e, C, M)$. Then we have*

$$\begin{aligned} |S(Q, C, M) - S(T^*, C, M)| &= \left| \sum_{e \in Q'} S(e, C, M) \right| \\ &\leq \sum_{e \in Q'} |S(e, C, M)| \leq |Q'| \cdot \max_{e \in Q'} \max_{\tilde{M} \in \mathcal{M}} |S(e, C, \tilde{M})|, \end{aligned} \tag{A.3}$$

where $|Q'|$ denotes the number of edges in Q', and $\mathcal{M}$ denotes the set of possible matches between Q' and C.

A.4 Proof of Lemma 4.3.3

Lemma A.4.1. *By setting proper thresholds, we guarantee that the matches returned by MDST cover all the matches for the original query graph.*

Proof. Lemma 4.3.2 shows that the score difference between using query graph Q and using its MDST T^* is upper-bounded. Let $S_Q = \min_{M_Q \in \mathcal{M}_Q} S(Q, C, M_Q)$, where $\mathcal{M}_Q$ denotes the set of possible matches between Q and C, then in the worst case, we can set the matching score threshold δ for returning solutions as

$$\delta = S_Q - \Delta_{Q'} = S_Q - |Q'| \cdot \max_{e \in Q'} \max_{\tilde{M} \in \mathcal{M}} |S(e, C, \tilde{M})|, \tag{A.4}$$

which guarantees to return all the solutions for query graph Q. □

A.5 Proof of Lemma 5.3.1

$$\hat{P}_i(\tau_v) = \frac{1}{|V(A)|} \sum_{v_a \in V(A)} \mathbb{1}_{\{d(v_a, v_i) \leq \tau_v\}}, \tag{A.5}$$

$$\hat{P}_{ij}(\tau_e) = \frac{1}{|E(A)|} \sum_{(v_a, v_b) \in E(A)} \mathbb{1}_{\{d((v_a, v_b), (v_i, v_j)) \leq \tau_e\}}. \tag{A.6}$$

Let $\mathcal{T}_Q$ be the collection of trees over the query graph Q. Our goal is to select a tree T^*_Q to optimize the following function:

$$T^*_Q = \underset{T \in \mathcal{T}_Q}{\operatorname{argmin}} \sum_{v_i \in V(T)} \log(\hat{P}_i(\tau_v)) + \sum_{(v_i,v_j) \in E(T)} \log \hat{P}_{ij}(\tau_e), \tag{A.7}$$

Lemma A.5.1. *The solution to Eq. 5.11 is a minimum cost spanning tree.*

Proof. Note first that for all $T \in \mathcal{T}$, $V(T)$ is identical because they are all spanning trees of Q. Therefore, we can say that

$$T^*_Q = \underset{T \in \mathcal{T}_Q}{\operatorname{argmin}} \sum_{(v_i,v_j) \in E(T)} \log \hat{P}_{ij}(\tau_e). \tag{A.8}$$

Because $\hat{P}_{ij}(\tau_e)$ is by definition between 0 and 1, $\log \hat{P}_{ij}(\tau_e)$ is negative.

For all trees T, $|E(T)| = |V(T)| - 1$, defining a weight of $w_{ij} = \log \hat{P}_{ij}(\tau_e) + 1$ adds $|V(Q)| - 1$ to the score of each tree, but does not re-order the solutions.

Weight w_{ij} is positive, so

$$T^*_Q = \underset{T \in \mathcal{T}_Q}{\operatorname{argmin}} \sum_{(v_i,v_j) \in E(T)} w_{ij} \tag{A.9}$$

is the definition of a minimum spanning tree.

□

A.6 Proof of Lemma 5.3.2

Lemma A.6.1. *Suppose the node and edge attributes take values in a finite set. Let Q be a fixed query. Let A be a data archive where the node and edge features are generated independently at random. Suppose the similarity function is binary valued (corresponding to exact matching of attributes). Define $\mathcal{U}_0(T_Q)$ as the set of matches obtained by running the sub-tree query T_Q. Then among all trees we have that,*

$$\mathit{Prob}\{\mathcal{U}_0(T^*_Q)\} \leq \mathit{Prob}\{\mathcal{U}_0(T_Q)\}$$

where the probability is taken with respect to data-archive generation.

If the similarity function is binary valued, we can re-write equations (5.9) and (5.10) as independent of any parameters τ_v or τ_e:

$$\hat{P}_i = \frac{1}{|V(A)|} \sum_{v_a \in V(A)} \mathbb{1}_{\{d(v_a,v_i)=0}, \tag{A.10}$$

$$\hat{P}_{ij} = \frac{1}{|E(A)|} \sum_{(v_a,v_b) \in E(A)} \mathbb{1}_{\{d((v_a,v_b),(v_i,v_j))=0}. \tag{A.11}$$

Let P_i and P_{ij} denote the probability of the node or edge feature for query node i or edge ij. Then the expected number of nodes and edges in the archive graph A with this feature would be given by:

$$\mathbb{E} \sum_{v_a \in V(A)} \mathbb{1}_{\{d(v_a,v_i)=0\}}] = P_i|V(A)|, \tag{A.12}$$

$$\mathbb{E} \sum_{(v_a,v_b) \in E(A)} = \mathbb{1}_{\{d((v_a,v_b),(v_i,v_j))=0\}} = P_{i,j}|E(A)|. \tag{A.13}$$

Therefore, $\mathbb{E}\hat{P}_i = P_i$ and $\mathbb{E}\hat{P}_{ij} = P_{ij}$

Thus,

$$T^*_{\mathcal{Q}} = \underset{T \in \mathcal{T}_{\mathcal{Q}}}{\operatorname{argmin}} \sum_{v_i \in V(T)} \log P_i + \sum_{(v_i,v_j) \in E(T)} \log P_{ij}, \tag{A.14}$$

Without re-ordering trees, we can exponentiate to get:

$$T^*_{\mathcal{Q}} = \underset{T \in \mathcal{T}_{\mathcal{Q}}}{\operatorname{argmin}} \prod_{v_i \in V(T)} P_i \prod_{(v_i,v_j) \in E(T)} P_{ij} \tag{A.15}$$

Any tree T has $|E(T)|$ independently generated edge attributes and $|V(T)|$ indepedently-

generated node attributes. Thus, we can compute the probability of generating any particular tree T as:

$$\text{Prob}(T) = \prod_{v_i \in V(T)} P_i \prod_{(v_i, v_j) \in E(T)} P_{ij} \tag{A.16}$$

Thus, $T^*_{\mathcal{Q}}$ is the spanning tree of Q that has the minimum probability of being generated, and

$$\text{Prob}\{\mathcal{U}_0(T^*_{\mathcal{Q}})\} \leq \text{Prob}\{\mathcal{U}_0(T_{\mathcal{Q}})\}. \tag{A.17}$$

Appendix B

Additional Experiments

B.1 Jena Library

Query	$V(A)$	$V(A')$	$V(A'_t)$	$e(A)$	$E(A')$	$E(A'_t)$
$\|V(Q)\| < 50$	1822	814	797	4999	1133	1069
$50 \leq \|V(Q)\| < 100$	1128	387	355	4812	1377	1197
$100 \leq \|V(Q)\|$	2128	653	594	4959	1869	1642

B.2 Simulated Data

Query	$V(A)$	$V(A')$	$V(A'_t)$	$e(A)$	$E(A')$	$E(A'_t)$
$\|V(Q)\| = 10$	1000	92	10	249826	2617	26
$\|V(Q)\| = 10$	3000	106	10	2249446	4307	23
$\|V(Q)\| = 10$	5000	120	10	6248151	8973	26

B.3 VIRAT

Query	$V(A)$	$V(A')$	$V(A'_t)$	$e(A)$	$E(A')$	$E(A'_t)$
Mount	19353	2290	589	3.7e8	2913	931
Dismount	19353	3298	978	3.7e8	6145	1950
Deposit	19353	1508	100	3.7e8	10229	172
Take Out	19353	2070	116	3.7e8	13587	195
Group Meeting	19353	10687	9548	3.7e8	232047	227070

Appendix C

Software and Datasets

C.1 Software

Over the course of building a video surveillance system, we utilized a large amount of pre-existing software. In this section, we briefly describe the uses we put it to and provide the location of these packages for follow-up work.

1. **Piotr's Computer Vision Matlab Toolbox**. We used Piotr's Matlab Toolbox (PMT) for extracting aggregate channel features (ACF) from video frames and performing detection of people and cars. The toolbox can be downloaded at

   https://github.com/pdollar/toolbox.

2. **Anton Milan's Discrete-Continuous Tracker**. We used Anton Milan's implementation of his tracker [Andriyenko et al., 2012] to stitch together detections of people, objects and vehicles generated by the toolbox. The tracker can be downloaded at http://www.milanton.de/dctracking/.

3. **RTree for Python**. We used the Rtree package, a libspatialindex wrapper, to implement the spatial indexing of our bounding boxes for easy identification of spatial displacement relationships. It can be found at http://toblerity.org/rtree/.

C.2 Datasets

Here are links to the publicly available datasets that were used in this paper.

1. **VIRAT Ground 2.0 Dataset**. A 40-GB video surveillance dataset with full ground truth of people, objects, bicycles and cars. Available at http://www.viratdata.org/

2. **Jena Library of Biological Macromolecules**. Contains PDBSv1, a dataset of large, sparse graphs of RNA, DNA and Proteins. Also has datasets of protein backbones (PDBSv2) and contact maps of amino acids (PDBSv2).

   Available at http://ferrolab.dmi.unict.it/ri/datasets.html.

3. **PETS 2006** A video surveillance dataset with simple activities like abandoning objects. Available by request at http://www.cvg.reading.ac.uk/PETS2006/data.html

4. **Mit-Traffic** A video dataset for pedestrian detection and vehicle behavior. Available at http://www.ee.cuhk.edu.hk/ xgwang/MITtraffic.html

5. **Subway** Video monitoring of a subway system in black and white. Available by request from http://www.cs.technion.ac.il/ amita/.

References

I-LIDS.

Abramowitz, M., Stegun, I. A., and Others (1966). Handbook of mathematical functions. *Applied Mathematics Series*, 55:62.

Adam, A., Rivlin, E., Shimshoni, I., and Reinitz, D. (2008). Robust Real-Time Unusual Event Detection Using Multiple Fixed-Location Monitors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(3):555–560.

Andriyenko, A., Schindler, K., and Roth, S. (2012). Discrete-continuous optimization for multi-target tracking. *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (June):1926–1933.

Aytar, Y., Shah, M., and Luo, J. (2008). Utilizing semantic word similarity measures for video retrieval. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Barnich, O. and Van Droogenbroeck, M. (2011). ViBe: A universal background subtraction algorithm for video sequences. *IEEE Transactions on Image Processing*, 20(6):1709–1724.

Baugh, G. and Kokaram, A. (2010). A Viterbi tracker for local features. In *IS&T/SPIE Electronic Imaging*, pages 75430L—-75430L. International Society for Optics and Photonics.

Benezeth, Y., Jodoin, P.-M., Emile, B., Laurent, H., and Rosenberger, C. (2010). Comparative Study of Background Subtraction Algorithms. *Journal of Electronic Imaging*, 19(3):1–12.

Berretti, S., Bimbo, A. D., and Pala, P. (2007). Graph Edit Distance for Active Graph Matching in Content Based Retrieval Applications. *The Open Artificial Intelligence Journal*, 1(1):1–11.

Bhattacharjee, A. and Jamil, H. M. (2012). WSM: A novel algorithm for subgraph matching in large weighted graphs. *Journal of Intelligent Information Systems*, 38:767–784.

Blank, M. and Gorelick, L. (2005). Actions as space-time shapes. *International Conference on Computer Vision*, 29(12):2247–2253.

Blank, M., Gorelick, L., Shechtman, E., Irani, M., and Basri, R. (2005). Actions as Space-Time Shapes. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1395–1402.

Bonnici, V., Giugno, R., Pulvirenti, A., Shasha, D., and Ferro, A. (2013). A subgraph isomorphism algorithm and its application to biochemical data. *BMC bioinformatics*, 14:13.

Bouman, C. A., Shapiro, M., Cook, G. W., Atkins, C. B., and Cheng, H. (1997). Cluster: An unsupervised algorithm for modeling Gaussian mixtures.

Castañón, G. and Caron, A.-L. (2012). Exploratory search of long surveillance videos. *Proceedings of the 20th ACM International Conference on Multimedia*.

Castañón, G., Chen, Y., Zhang, Z., and Saligrama, V. (2015). Efficient Activity Retrieval through Semantic Graph Queries. *Proceedings of the 23rd ACM International Conference on Multimedia*, pages 391–400.

Çeliktutan, O., Akgul, C., Wolf, C., and Sankur, B. (2013). Graph-based analysis of physical exercise actions. *Proceedings of the 1st ACM International Workshop on Multimedia Indexing and Information Retrieval for Health Care*, pages 23–31.

Çeliktutan, O., Wolf, C., Sankur, B., and Lombardi, E. (2012). Real-time exact graph matching with application in human action recognition. In A.A. Salah (ed.) *Human Behavior Understanding* (pp. 17-28). *Lecture Notes in Computer Science*, vol. 7559. Berlin, New York: Springer.

Chang, S.-F., Chen, W., and Sundaram, H. (1998). VideoQ: a fully automated video retrieval system using motion sketches. *Proceedings of the IEEE Workshop on Applications of Computer Vision*, pages 270–271.

Chen, S., Zhang, J., Li, Y., and Zhang, J. (2012). A hierarchical model incorporating segmented regions and pixel descriptors for video background subtraction. *IEEE Transactions on Industrial Informatics*, 8(1):118–127.

Cho, M., Lee, J., and Lee, K. (2010). Reweighted random walks for graph matching. *Proceedings of the European Conference on Computer Vision*, pages 1–14.

Choe, T. E., Deng, H., Guo, F., Lee, M. W., and Haering, N. (2013). Semantic video-to-video search using sub-graph grouping and matching. In *Proceedings of the IEEE International Conference on Computer Vision*, number 1, pages 787–794.

Christmas, W., Kittler, J., and Petrou, M. (1995). Structural matching in computer vision using probabilistic relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8).

Cordella, L. P., Foggia, P., Sansone, C., and Vento, M. (2004). A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–72.

Cour, T., Srinivasan, P., and Shi, J. (2007). Balanced graph matching. *Advances in Neural Information Processing Systems*, 19:313.

Dalton, J., Allan, J., and Mirajkar, P. (2013). Zero-shot video retrieval using content and concepts. *Information and Knowledge Management*, pages 1857–1860.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. (2004). Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computation Geometry*, pages 253–262, New York, NY, USA. ACM.

De la Torre, F. (2012). Factorized graph matching. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 127–134.

De Santo, M., Foggia, P., Sansone, C., and Vento, M. (2003). A large database of graphs and its use for benchmarking graph isomorphism algorithms. *Pattern Recognition Letters*, 24(8):1067–1079.

Demeyer, S., Michoel, T., Fostier, J., Audenaert, P., Pickavet, M., and Demeester, P. (2013). The Index-Based Subgraph Matching Algorithm (ISMA): Fast Subgraph Enumeration in Large Networks Using Optimized Search Trees. *PLoS ONE*, 8(4).

Dollár, P. Piotr's Computer Vision MATLAB Toolbox.

Dollar, P., Appel, R., Belongie, S., and Perona, P. (2014). Fast feature pyramids for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(8):1532–1545.

Dollár, P. and Rabaud, V. (2005). Behavior recognition via sparse spatio-temporal features. *Performance Evaluation of Tracking and Surveillance*, pages 65–72.

Dollár, P., Tu, Z., Perona, P., and Belongie, S. (2009). Integral channel features. *Proceedings of the British Machine Vision Conference*, pages 91.1–91.11. doi:10.5244/C.23.91.

Efros, A. and Berg, A. (2003). Recognizing action at a distance. *Ninth IEEE Conference on Computer Vision*, (October).

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2010). Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–45.

Ferryman, J. (2006). PETS 2006. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Gaur, U., Zhu, Y., Song, B., and Roy-Chowdhury, A. (2011). A 'string of feature graphs' model for recognition of complex activities in natural videos. *Proceedings of the IEEE International Conference on Computer Vision*, pages 2595–2602.

Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Gionis, A., Indyk, P., and Motwani, R. (1999). Similarity Search in High Dimensions via Hashing. In *Proceedings of the Twenty-fifth International Conference on Very Large Databases*, pages 518–529.

Guttman, A. R.-t. and A (1984). dynamic index structure for spatial searching. *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages:47–57.

Horn, B. K. and Schunck, B. G. (1981). Determining optical flow. In *Artificial Intelligence*, volume 17, pages 185–203. International Society for Optics and Photonics.

Hsu, W. H., Kennedy, L. S., and Chang, S.-F. (2007). Reranking methods for visual search. *Proceedings of the IEEE Conference on Multimedia*, 14(3):14–22.

Hu, R., James, S., Wang, T., and Collomosse, J. (2013). Markov random fields for sketch based video retrieval. *Proceedings of the International Conference on Multimedia Retrieval*, page 279.

Hu, W., Xie, D., Fu, Z., Zeng, W., and Maybank, S. (2007). Semantic-based surveillance video retrieval. *IEEE Transactions on Image Processing*, 16(4):1168–1181.

Huehne, R. and Suehnel, J. (2009). The Jena Library of Biological Macromolecules–JenaLib. *Nature - Precedings*. doi:10.1038/npre.2009.3114.1

Ikizler, N. and Forsyth, D. A. (2008). Searching for complex human activities with no visual examples. *International Journal of Computer Vision*, 80(3):337–357.

Johnson, J., Krishna, R., Stark, M., Li, L.-j., Shamma, D. A., Bernstein, M. S., and Fei-fei, L. (2015). Image Retrieval using Scene Graphs. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Jung, M. Y. and Park, S. H. (2009). *Semantic Similarity Based Video Retrieval*. Springer Berlin Heidelberg.

Koutra, D., Parikh, A., Ramdas, A., and Xiang, J. (2011). Algorithms for Graph Similarity and Subgraph Matching. *Technical Report of Carnegie-Mellon-University*.

Kriege, N. and Mutzel, P. (2012). Subgraph Matching Kernels for Attributed Graphs. *Proceedings of the International Conference on Machine Learning*, pages 1015–1022.

Kuettel, D., Breitenstein, M., Gool, L., and Ferrari, V. (2010). What's going on? Discovering spatio-temporal dependencies in dynamic scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1951–1958.

Kwak, S., Han, B., and Han, J. H. (2013). Multi-agent Event Detection: Localization and Role Assignment. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2682–2689.

Laptev, I. (2005). On space-time interest points. *International Journal of Computer Vision*, 64:107–123.

Laptev, I., Marsza\lek, M., Schmid, C., and Rozenfeld, B. (2008). Learning realistic human actions from movies. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE.

Le, T.-L., Thonnat, M., Boucher, A., and Brémond, F. (2009). Surveillance Video Indexing and Retrieval Using Object Features and Semantic Events. *International Journal of Pattern Recognition and Artificial Intelligence*, 23(07):1439–1476.

Le Gall, D. (1991). MPEG: A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58.

Leordeanu, M. and Hebert, M. (2005). A spectral technique for correspondence problems using pairwise constraints. *Proceedings of the IEEE International Conference on Computer Vision*, pages 1482–1489 Vol. 2.

Li, S. Z., Zhu, L., Zhang, Z., Blake, A., Zhang, H., and Shum, H. (2002). Statistical learning of multi-view face detection. In *Proceedings of the European Conference on Computer Vision*, pages 67–81. Springer.

Li, W., Mahadevan, V., and Vasconcelos, N. (2014). Anomaly detection and localization in crowded scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(1):18–32.

Lin, D., Fidler, S., Kong, C., and Urtasun, R. (2014). Visual Semantic Search : Retrieving Videos via Complex Textual Queries. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Lowe, D. (1999). Object recognition from local scale-invariant features. *Proceedings of the IEEE International Conference on Computer Vision*, pages 1150–1157 vol.2.

Ma, S., Zhang, J., Ikizler-Cinbis, N., and Sclaroff, S. (2013). Action Recognition and Localization by Hierarchical Space-Time Segments. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2744–2751.

Marszałek, M., Laptev, I., and Schmid, C. (2009). Actions in context. In *Computer Vision and Pattern Recognition Workshops*, pages 2929–2936.

Meessen, J., Coulanges, M., Desurmont, X., and Delaigle, J.-F. (2006). Content-Based Retrieval of Video Surveillance Scenes. In *Multimedia Content Representation, Classification and Security*, pages 785–792.

Mikolajczyk, K. and Schmid, C. (2005). A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630.

Mita, T., Kaneko, T., and Hori, O. (2005). Joint haar-like features for face detection. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2, pages 1619–1626. IEEE.

Niebles, J. C., Wang, H., and Fei-Fei, L. (2008). Unsupervised Learning of Human Action Categories Using Spatial-Temporal Words. *International Journal of Computer Vision*, 79(3):299–318.

Oh, S., Hoogs, A., and Perera, A. (2011). A large-scale benchmark dataset for event recognition in surveillance video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, number 2.

Oneata, D., Verbeek, J., and Schmid, C. (2013). Action and event recognition with Fisher vectors on a compact feature set. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1817–1824.

Palatucci, M. and Pomerleau, D. (2009). Zero-shot learning with semantic output codes. *Proceedings of the Conference on Neural Information Processing Systems*, pages 1–9.

Philbin, J., Chum, O., Isard, M., Sivic, J., and Zisserman, A. (2007). Object retrieval with large vocabularies and fast spatial matching. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8.

Pitié, F., Berrani, S.-A., Kokaram, A., and Dahyot, R. (2005). Off-line multiple object tracking using candidate selection and the viterbi algorithm. In *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, volume 3, pages III—-109. IEEE.

Raptis, M., Kokkinos, I., and Soatto, S. (2012). Discovering discriminative action parts from mid-level video representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1242–1249. IEEE.

Raptis, M. and Sigal, L. (2013). Poselet key-framing: A model for human activity recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2650–2657.

Riesen, K. and Bunke, H. (2009). Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27(7):950–959.

Rodriguez, M. D., Ahmed, J., and Shah, M. (2008). Action mach a spatio-temporal maximum average correlation height filter for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE.

Rubner, Y., Tomasi, C., and Guibas, L. (2000). The Earth Mover's Distance as a Metric for Image Retrieval. *International Journal of Computer Vision*, 40(2):99–121.

Ryoo, M. S. and Aggarwal, J. K. (2010). Stochastic Representation and Recognition of High-Level Group Activities. *International Journal of Computer Vision*, 93(2):183–200.

Sadanand, S. and Corso, J. J. (2012). Action bank: A high-level representation of activity in video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1234–1241. IEEE.

Schüldt, C., Laptev, I., and Caputo, B. (2004). Recognizing human actions: a local SVM approach. In *Proceedings of the 17th International Conference on Pattern Recognition*, volume 3, pages 32–36. IEEE.

Schuldt, C., Laptev, I., and Caputo, B. (2004). Recognizing human actions: a local SVM approach. *Proceedings of the International Conference on Pattern Recognition*, pages 3–7.

Scovanner, P., Ali, S., and Shah, M. (2007). A 3-dimensional sift descriptor and its application to action recognition. In *Proceedings of the 15th ACM International Conference on Multimedia*, page 357, New York, New York, USA. ACM Press.

Shi, J. and Tomasi, C. (1994). Good features to track. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600. IEEE.

Siersdorfer, S., San Pedro, J., and Sanderson, M. (2009). Automatic video tagging using content redundancy. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 395–402. ACM.

Sivic, J. and Zisserman, A. (2003). Video Google: A Text Retrieval Approach to Object Matching in Videos. In *Ninth International IEEE Conference on Computer Vision*, volume 2, pages 1470–1477.

Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197.

Solnon, C. (2010). AllDifferent -based Filtering for Subgraph Isomorphism. *Artificial Intelligence*, 174(December 2009):850–864.

Song, X. M. and Fan, G. L. (2006). Joint Key-Frame Extraction and Object Segmentation for Content-Based Video Analysis. *Circuits and Systems for Video Technology*, 16(7):904–914.

Soomro, K., Zamir, A. R., and Shah, M. (2012). UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild. *arXiv preprint arXiv:1212.0402*, (November).

Stringa, E. and Regazzoni, C. (1998). Content-based Retrieval and Real Time Detection from Video Sequences Acquired by Surveillance Systems. In *Proceedings of the International Conference on Image Processing*, pages 138–142.

Sujatha, C. and Mudenagudi, U. (2011). A study on keyframe extraction methods for video summary. In *Proceedsings of the International Conference on Computational Intelligence and Communication Networks*, pages 73–77. IEEE.

Sun, Z., Wang, H., Shao, B., and Li, J. (2012). Efficient subgraph matching on billion node graphs. *Proceedings of the VLDB Endowment*, pages 788–799.

Taubman, D. (2000). High performance scalable image compression with EBCOT. *IEEE Transactions on Image Processing*, 9(7):1158–1170.

Taubman, D. and Marcellin, M. (2012). *JPEG2000 Image Compression Fundamentals, Standards and Practice: Image Compression Fundamentals, Standards and Practice*, volume 642. Springer Science & Business Media.

Tianmin Shu, Xie, D., Rothrock, B., Todorovic, S., and Zhu, S.-c. (2015). Joint inference of groups, events and human roles in aerial videos. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4576–4584.

Ullmann, J. R. (1976). An Algorithm for Subgraph Isomorphism. *Journal of the ACM*, 23(1):31–42.

Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Viola, P. and Jones, M. J. (2004). Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154.

Wang, H., Kläser, A., Schmid, C., and Liu, C.-L. (2013). Dense trajectories and motion boundary descriptors for action recognition. *International Journal of Computer Vision*, 103(1):60–79.

Wang, H. and Schmid, C. (2013). Action recognition with improved trajectories. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3551–3558.

Wang, M., Hong, R., Li, G., Zha, Z.-J., Yan, S., and Chua, T.-S. (2012). Event driven web video summarization by tag localization and key-shot identification. *IEEE Transactions on Multimedia*, 14(4):975–985.

Wang, M. and Wang, X. (2011). Automatic adaptation of a generic pedestrian detector to a specific traffic scene. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3401–3408. IEEE.

Wang, X., Ma, X., and E.Grimson (2009). Unsupervised Activity Perception in Crowded and Complicated Scenes Using Hierarchical Bayesian Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(3):539–555.

Wang, Y. and Mori, G. (2009). Learning a discriminative hidden part model for human action recognition. In *Advances in Neural Information Processing Systems*, pages 1721–1728.

Wang, Y. and Mori, G. (2011). Hidden part models for human action recognition: Probabilistic versus max margin. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(7):1310–1323.

Weinland, D., Boyer, E., and Ronfard, R. (2007). Action recognition from arbitrary views using 3d exemplars. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1–7. IEEE.

Wu, S., Bondugula, S., Luisier, F., Zhuang, X., and Natarajan, P. (2014). Zero-Shot Event Detection Using Multi-modal Fusion of Weakly Supervised Concepts. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2665–2672.

Xiang, T. and Gong, S. (2008). Video Behavior Profiling for Anomaly Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):893–908.

Yang, T., Li, S. Z., Pan, Q., and Li, J. (2005). Real-time Multiple Objects Tracking with Occlusion Handling in Dynamic Scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, number 60172037.

Yang, Y., Lovell, B., and Dadgostar, F. (2009). Content-Based Video Retrieval (CBVR) System for CCTV Surveillance Videos. In *Digital Image Computing: Techniques and Applications*, pages 183–187.

Yao, B., Jiang, X., Khosla, A., Lin, A. L., Guibas, L., and Fei-Fei, L. (2011). Human action recognition by learning bases of action attributes and parts. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1331–1338. IEEE.

Zhang, C. and Viola, P. A. (2008). Multiple-instance pruning for learning efficient cascade detectors. In *Advances in Neural Information Processing Systems*, pages 1681–1688.

Zhang, D.-q. and Chang, S.-f. (2004). Stochastic Attributed Relational Graph Matching for Image Near-Duplicate Detection. *Proceedings of the 12th ACM International Conference on Multimedia*.

Zhang, H., Zhou, W., Reardon, C., and Parker, L. (2014a). Simplex-based 3D spatio-temporal feature description for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2059–2066.

Zhang, S., Yang, J., and Jin, W. (2010). SAPPER: subgraph indexing and approximate matching in large graphs. *Proceedings of the VLDB Endowment*, 3:1185–1194.

Zhang, Z., Chen, Y., and Saligrama, V. (2014b). A Novel Visual Word Co-occurrence Model for Person Re-identification. *Proceedings of the European Conference on Computer Vision*.

Zhong, D. and Chang, S.-f. (1999). An integrated approach for content-based video object segmentation and retrieval. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(8):1259–1268.

Zhou, F. and De la Torre, F. (2013). Deformable Graph Matching. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2922–2929.

Zhu, Y., Nayak, N. M., and Roy-Chowdhury, A. K. (2013). Context-Aware Activity Recognition and Anomaly Detection in Video. *IEEE Journal of Selected Topics in Signal Processing*, 7(1):91–101.

CURRICULUM VITAE

Gregory Castanon

- G. Castañón, Y. Chen, Z. Zhang, V. Saligrama. **Efficient Activity Retrieval through Semantic Graph Queries**. ACM Multimedia, 2015.
- G. Castañón, M. Elgharib, V. Saligrama, P.M. Jodoin. **Retrieval in Long Surveillance Videos using User Described Motion and Object Attributes**. IEEE Transactions on Circuits and Systems for Video Technology, 2014.
- G. Castañón, A. Caron, V. Saligrama, P.M. Jodoin. **Real-Time Activity Search of Surveillance Video**. AVSS, 2012.
- G. Castañón, A. Caron, V. Saligrama, P.M. Jodoin. **Exploratory search of long surveillance videos**. ACM Multimedia, 2012.
- C. Chong, G. Castañón, N. Cooprider, S. Mori, R. Ravichandran, R. Macior. **Efficient multiple hypothesis tracking by track segment graph**. FUSION, 2009.
- E. Fortunato, W. Kreamer, S. Mori, C. Chong, G. Castañón. **Generalized Murty's algorithm with application to multiple hypothesis tracking**. FUSION, 2007.