Efficient Video Similarity Measurement and Search

by

Sen-ching Cheung

B.S. (University of Washington) 1992

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering - Electrical Engineering
and Computer Sciences

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Avideh Zakhor, Chair
Professor Lawrence A. Rowe
Professor Ray R. Larson

Fall 2002

The dissertation of Sen-ching Cheung is approved:

Chair Date

Date

Date

University of California at Berkeley

Fall 2002

Efficient Video Similarity Measurement and Search

by

Sen-ching Cheung

Abstract

Efficient Video Similarity Measurement and Search

by

Sen-ching Cheung

Doctor of Philosophy in Electrical Engineering and Computer Sciences
University of California at Berkeley

Professor Avideh Zakhor, Chair

The amount of information on the world wide web has grown enormously since its creation in 1990. Duplication of content is inevitable because there is no central management on the web. Studies have shown that many similar versions of the same text documents can be found throughout the web. This redundancy problem is more severe for multimedia content such as web video sequences, as they are often stored in multiple locations and different formats to facilitate downloading and streaming. Similar versions of the same video can also be found, unknown to content creators, when web users modify and republish original content using video editing tools. Identifying similar content can benefit many web applications and content owners. For example, it will reduce the number of similar answers to a web search and identify inappropriate use of copyright content. In this dissertation, we present a system architecture and corresponding algorithms to efficiently measure, search, and organize similar video sequences found on any large database such as the web.

We first introduce a class of randomized algorithms, called ViSig, to estimate video similarity. The basic idea is to summarize each video sequence into a small set of video frames, or a signature, that is most similar to a set of predefined random images. Theoretical and experimental results show that video similarity can be reliably estimated by the ViSig method. Even though a small signature is sufficient to estimate similarity, each frame in the signature is represented by a high-dimensional vector. Similarity search on a large database of high-dimensional vectors is a challenging problem from a computational viewpoint. To solve this problem, we propose a novel non-linear feature extraction technique that can be used in a fast similarity search system. The proposed technique combines the classical principal component analysis (PCA) with triangle inequality pruning. Experimental results show that our proposed method outperforms techniques such as the Haar Wavelet, Fastmap and PCA. To further improve retrieval performance and provide better organization of similarity search results, we also design a new graph-theoretical clustering algorithm on large databases of signatures. The algorithm treats all the signatures as an abstract threshold graph, where the threshold is determined based on local data statistics. Similar clusters are identified as highly connected regions in the graph. By measuring the retrieval performance against a ground-truth set, we show that our proposed algorithm outperforms simple thresholding and hierarchical clustering techniques.

Professor Avideh Zakhor
Dissertation Committee Chair

To Dad and Mom

Contents

List of Figures **iv**

List of Tables **vi**

1 Inroduction **1**
1.1 Video similarity measurement 4
1.2 Fast similarity Search 8
1.3 Similarity clustering 10
1.4 Organization 13
1.A Appendix: Acronyms and Notations 15

2 A similarity video search engine **18**
2.1 System description 19
2.2 Web video acquisition 21
2.3 Ground-truth construction 24

3 Measuring video similarity **30**
3.1 Ideal video similarity 31
3.2 Voronoi video similarity 37
3.3 Video signature method 41
3.4 Seed vector generation 44
3.5 Voronoi gap 49
3.6 Ranked ViSig method 53
3.7 Experimental results 58
3.7.1 Color histogram feature 58
3.7.2 Simulation results 62
3.7.3 Ground-truth results 66
3.8 Summary 74
3.A Appendix: Proofs of propositions 75

4 Fast similarity search on signatures **82**
4.1 The GEMINI approach for similarity search 85
4.2 Projection-vector mapping 89
4.3 PCA on projection vectors 97
4.4 Summary 105
4.A Appendix: Modification for speed tests 106

5 Similarity signature clustering **109**
5.1 Graphical representation of signature database 111
5.2 Signature clustering 114
5.3 Ground-truth results 118
5.4 Similar cluster statistics for web video 121
5.5 Summary 124
5.A Appendix: Proof of Proposition 5.2.1 125
5.B Appendix: Clustering algorithm 126

6 Summary and Future Work **133**

Bibliography **138**

List of Figures

2.1 *Schematic of the video search engine.* 19
2.2 *Search results of keyword "telephone".* 22
2.3 *Ground-truth clusters and their sizes (part 1 of 2).* 28
2.4 *Ground-truth clusters and their sizes (part 2 of 2).* 29

3.1 *Two video sequences with NVS equal to 0.9.* 33
3.2 *Two video sequences with IVS equal to 1/3.* 36
3.3 *The Voronoi diagram of a three-frame video.* 39
3.4 *The shaded area denotes the similar Voronoi region between the two sequences.* 40
3.5 *Examples of sequences with identical IVS's but very different VVS's.* . 45
3.6 *The unshaded region is the Voronoi gap for this pair of video sequences with IVS one.* 49
3.7 *The error probability for the hamming cube at different values of ϵ and distances k between the vectors in the video.* 53
3.8 *Values of ranking function $Q(\cdot)$ for a three-vector video sequence. Lighter colors correspond to larger values.* 56
3.9 *Quantization of the HSV color space.* 59
3.10 *Comparisons between the basic (broken-line) and ranked (solid) ViSig methods for four different signature sizes: $m = 2, 6, 10, 14$.* 69
3.11 *Precision and recall performance for ranked ViSig method at $m = 2, 6, 10, 14$.* 70
3.12 *Comparison between d_c metric (broken) and $\widehat{d}_c$ distance (solid).* . . . 71
3.13 *Comparison between the Ranked ViSig method with $m = 6$ (solid) and k-medoid with 7 representative vectors (broken).* 72
3.14 *Comparison between the symmetric and asymmetric VSS_r with $m = 6$.* 73

4.1 *Distribution of the metric $d(x, y)$ for 100,000 random pairs of signature vectors in the coordinates of $\max_{i=1,2,\ldots,100} |d(x, s_i) - d(y, s_i)|$ and $\min_{i=1,2,\ldots,100}[d(x, s_i) + d(y, s_i)]$. Different colors correspond to metric values at different ranges.* 92

4.2 *Pruning-versus-Accuracy plots for the "lower-bound", the "product" and the proposed schemes.* 96
4.3 *Pruning-versus-accuracy plot for two dimension.* 100
4.4 *Pruning-versus-accuracy plot for four dimension.* 101
4.5 *Pruning-versus-accuracy plot for eight dimension.* 102
4.6 *Pruning and Accuracy versus pruning threshold for three independent sets of queries.* 103
4.A *The left figure shows the signature similarity search that uses the ranking of the query signature. The system on the right uses the rankings of the signatures in the database. Since the right system only needs to access the top-ranked signature vectors, less memory is required to store the database.* 107

5.1 *A threshold graph with three connected components.* 113
5.2 *Precision versus recall for different clustering algorithms and simple thresholding.* 119
5.3 *Precision and Recall versus edge density threshold γ.* 121
5.4 *Distribution of cluster sizes.* 122
5.A *For both illustrations, we obtain a contradiction by replacing e of length d with a shorter edge (u, v) to obtain a better MST.* 127

List of Tables

2.1 *Statistics of collected web video sequences* 25

3.1 *Comparison between using uniform random and corel image seed vectors. The second through fifth columns are the results of using uniform random seed vectors and the rest are the corel image seed vectors. Each row contains the results of a specific test video at IVS levels 0.8, 0.6, 0.4 and 0.2. The last two rows are the averages and standard deviations over all the test sequences.* 63

3.2 *Comparison between* VSS_b *and* VSS_r *under different levels of perturbation. The table follows the same format as in Table 3.1. The perturbation levels* ϵ *tested are 0.2, 0.4, 0.8, 1.2 and 1.6.* 65

4.1 *Speed comparisons among the sequential search, the proposed scheme, Fastmap, and Haar Wavelet.* 104

5.1 *Statistics of the largest ten clusters in the database.* 123

Acknowledgements

I am deeply indebted to my advisor and mentor, Professor Avideh Zakhor, for her guidance and inspiration throughout my graduate career at Berkeley. I am truly grateful for her dedication to the quality of my research, and her insightful prospectives on numerous technical issues.

This work has also benefited from helpful discussions with several individuals. I would like to thank Prof. Lawrence A. Rowe for helping me to identify a number of key issues in my dissertation work, and always leaving his door open for discussions. I would also like to thank Professor Ray R. Larson for his helpful comments during my qualifying examination and dissertation talk. I am very grateful to both Professor Rowe and Professor Larson for reading earlier drafts of this dissertation. I would also like to thank Professor Alistair Sinclair for introducing me to many of the theoretical results in similarity search. In addition, I would like to express my gratitude to Rainer Lienhart from Intel Research Lab, and Frederic Dufaux from Compaq Cambridge Research Lab for providing the necessary video capturing routines and hyperlink datasets to begin my research.

The financial support for my research work in last three years is greatly appreciated. They include funding from AFOSR Muri Grant FDF49620-99-1-0062, NSF grant CCR-9903368, California Digital Media Innovation (DiMI) program D97-03, and Hughes Research Lab.

The companionship and support from students in the department are gratefully

acknowledged. In particular, many thanks to Thinh Nguyen, Daniel Tan, Vito Dai, Nelson Chang, Ralph Neff, John Koo, and Christian Frueh, for the insightful discussions on diverse research problems, for the tremendous help in both technical and personal matters, and for showing me that difficult problems can actually be solved in tennis court and soccer field.

I am most grateful to my parents for instilling in me the importance of science and technology at a very young age, for encouraging me to pursue my own dream, and for reminding me the importance of doing my best even in the most difficult time.

Finally, I would like to thank my wife Blanca for her love, her help and her support in my final stage of dissertation research and thesis writing. It was absolutely wonderful to have some one, late at night while I was stuck at a programming problem, bringing me a cup of hot coffee or a bowl of sweet grapes.

Chapter 1

Inroduction

The amount of information on the world wide web has grown enormously since its creation in 1990. By June 2002, commercial search engines such as Google and Fast had indexed more than two billion web-pages. There is no central management on the web, so duplication of content is inevitable. There have been a number of research studies in recent years that investigated duplicated or highly-similar web-pages and web-sites [1, 2, 3, 4, 5]. The amount of redundancy on the web, as shown by these studies, is in fact quite high – one study has shown that about 46% of all the text documents on the web have at least one "near-duplicate," that is a document which is identical except for low level details such as formatting, and 5% of them have between 10 and 100 replicas [2]. Overly-duplicated content wastes resources in storage and transmission bandwidth, and increases the effort required to mine information for both human and artificial agents.

Such a problem is more severe for web multimedia content, especially for web video sequences. Tens or even hundreds of very similar video clips are returned when sending popular video keywords such as "star wars" or "Clinton testimony" to commercial search engines. There are a number of factors contributing to such a high degree of multiplicity. Due to the stringent requirements in bandwidth and delay, web video sequences are often stored in multiple locations, formatted to various sizes and frame-rates, and compressed with different algorithms and bitrates to facilitate downloading and streaming. As multimedia authoring tools are now commonplace on personal computers, similar versions, in part or as a whole, of the same video can also be found on the web when users modify and combine original content with their own productions. Advances in automatic video analysis also enable users to easily create trailers or key-frame story-boards to summarize video sequences. All the aforementioned variations of the same video sequence share *a large percentage of visually similar frames* with each other. These are the types of similar video sequences we are interested in finding on the web. Identifying these similar content can be beneficial to content owners and web video applications such as the followings:

- As users typically do not view items beyond the first result screen from a search engine [6], it is detrimental to have all "near-duplicate" entries cluttering the top retrievals. It is better to group together similar entries before presenting the retrieval results to users. Commercial search engines such as Google and Altavista have already applied techniques to cluster similar text documents

together before presenting them to users.

- When a particular web video becomes unavailable or suffers from slow network transmission, users can opt for a better version among similar video content identified by the video search engine.

- Similarity detection algorithms can also be used for content identification when conventional techniques such as watermarking are not applicable. For example, multimedia content brokers may use similarity detection to check for copyright violation as they have no right to insert watermarks into original material.

In this dissertation, we explore the design and implemention of video similarity detection and search algorithms for large databases of video sequences such the web. There are three main challenges in building such a system: first, to design a robust and low-complexity video similarity measure; second, to support fast similarity search over potentially millions of video sequences, and third, to present the search results to users in an organized and intuitive manner. We propose, in this dissertation, a number of algorithms to tackle these problems, and demonstrate the efficiency and effectiveness of these algorithms on a large dataset of web video sequences. Before embarking on a detailed description of our system, we will first elaborate on these challenges and review existing solutions in the literature.

1.1 Video similarity measurement

As mentioned earlier, we are interested in defining a video similarity measure based on the percentage of visually similar frames or shots shared between two video sequences. This is analogous to finding the percentage of shared words or phrases between two text documents. This commonly-used document similarity measure is called a Tanimoto measure [1, 2]. While it is relatively straightforward to distinguish two different words,[1] it is much harder to identify visually similar frames or shots due to the large number of possible variations. The typical approach is to identify each frame or shot by some of its attributes such as color, texture, shape, and motion, usually represented as a high-dimensional feature vector. Visual similarity between different frames can then be gauged by a metric function between the corresponding feature vectors. To compute our target video similarity measure, we thus need to identify feature vectors from two video sequences that are "close" to each other based on the computed metric values. There exist other video similarity measures in the literature and some of them are reviewed in this section. Nonetheless, no matter which measure is used, the major challenge is how to perform the measurement efficiently. As a video sequence can potentially have thousands of feature vectors representing different shots, computing a high-dimensional metric between them becomes a daunting task. On the other hand, for every new video added to the database or a query

[1] Variants of the same word should be first converted to the root by a process known as word stemming [7, ch. 3]. Synonyms can also be identified as a unique lexical concept via the use of an electronic thesaurus [8].

video presented by a user, similarity measurements need to be performed with possibly millions of entries in the database. Thus, it is imperative to develop fast methods to compute similarity measurements for database applications.

Finding visually similar content is the central theme in the area of Content-Based Information Retrieval (CBIR). In the past decade, numerous algorithms has been proposed to identify visual content similar in color, texture, shape, motion and many other attributes [9, 10, 11, 12, 13]. Much of the video similarity research has been focused on the problem of search for a particular short segment, such as a television commercial, within a long sequence [14, 15, 16, 17]. When extending the similarity measurement to full video, the first challenge is to define a single measurement to gauge the similarity between two video sequences. To this end, several proposals can be found in the literature: in [18, 19, 20], warping distance is used to measure the temporal edit differences between video sequences. Hausdorff distance is proposed in [21] to measure the maximal dissimilarity between shots. Template matching of shot change duration is used by Indyk et al. [22] to identify the overlap between video sequences. A common step shared by all the above schemes is to match similar feature vectors between two video sequences. This usually requires searching through part of or the entire video. The full computation of these measurements thus require the storage of the entire video, and time complexity that is at least linear in the length of the video. Applying such computations to find similar content within a database of millions of video sequences is too complex in practice.

On the other hand, computing the precise value of a similarity measurement is typically unnecessary. As feature vectors are idealistic models and do not entirely capture the process of how similarity is judged in the human visual system [23], many CBIR applications only require an approximation of the underlying similarity value. As such, it is unnecessary to maintain full fidelity of the feature vector representations, and approximation schemes can be used to alleviate high computational complexity. For example, in a video similarity search system, each video in the database can be summarized into a compact fixed-size representation that can be compared to test the similarity between the two video sequences.

Two types of summarization techniques are used for similarity approximation: higher-order and first-order. Higher-order techniques summarize all feature vectors in a video as a statistical distribution. These techniques are useful in classification and semantic retrieval as they are highly adaptive and robust against small perturbation. Nonetheless, they typically assume a restricted form of density models such as Gaussian, or mixtures of Gaussian distributions, and require computationally intensive methods such as Expectation-Maximization for parameter estimation [24, 25, 26]. As a result, higher-order techniques may be impractical for matching the enormous amount of extremely diverse video content on the web. First-order techniques summarize a video by a small set of representative feature vectors. One approach is to compute the "optimal" representative vectors by minimizing the distance between the original video and its representation. If the metric is finite-dimensional Euclidean

and the distance is the sum of squared metric, the well-known k-means method can be used [27]. For general metric spaces, we can use the k-medoid method which identifies k feature vectors within the video to minimize the distance [28, 21]. Both of these algorithms are iterative with each iteration running at $O(l)$ time for k-means, and $O(l^2)$ for k-medoids, where l represents the length of the video. To summarize long video sequences, such as feature-length movies or documentaries, these methods are clearly too complex to be used in large databases.

To produce a compact summarization that is both easy to generate and capable of producing a reliable estimate of the underlying similarity, we propose a class of randomized techniques called the Video Signature (ViSig) method in Chapter 3. The ViSig method summarizes each video sequence in the database by selecting a number of its frames closest to a set of random vectors. Such a representation is called a video signature. An important result shown in Chapter 3 is that, regardless of how long the video sequences are, it is sufficient to use very small video signatures to identify those sequences that share a large fraction of similar frames. Our proposed ViSig method is also an example of first-order video summarization technique. Unlike the k-means or k-medoid methods, it is a single-pass $O(l)$ algorithm. Thus, it takes far less computation to generate a summarization for a long video sequence. On the other hand, as demonstrated by the experimental results in Chapter 3, ViSig can produce retrieval results that are comparable to other techniques that are much more computationally intensive.

1.2 Fast similarity Search

An efficient algorithm to measure video similarity is only the first step towards building a similarity video search engine. When a user presents a query in the form of a video signature to the search engine, the search engine must identify all similar signatures in the database of possibly millions of entries. The naive approach of sequential search is too slow to handle large databases, and complex comparison functions. To guarantee a fast response time, it is imperative to develop fast similarity search algorithms. Faster-than-sequential solutions have been extensively studied by the database community. Elaborate data structures, collectively known as the Spatial Access Methods (SAM), have been proposed to facilitate similarity search [29, 9, 30]. Most of these methods, however, do not scale well to high dimensional metric spaces [31]. This problem is commonly known as the "curse of dimensionality". One possible strategy to mitigate this problem is to design a transformation to map the original metric space to a low-dimensional space where a SAM structure can be efficiently applied. Such a transformation is called feature extraction mapping, and the approach of combining feature extraction with SAM is called GEneric Multimedia INdexIng (GEMINI) [9, ch. 7].

A good feature extraction mapping should be able to closely approximate distances in the high-dimensional space using the corresponding distances in the low-dimensional projected space. The most commonly used feature extraction mapping is Principal Component Analysis (PCA). PCA has been shown to be optimal in ap-

proximating Euclidean distance [32], and a myriad of techniques have been developed for generating PCA on large datasets [33]. If the underlying metric is not Euclidean, PCA is no longer optimal and more general schemes need to be used. Multidimensional Dimension Scaling (MDS) is the most general class of techniques for creating mappings that preserve a high-dimensional metric in a low-dimensional space [34]. Historically, MDS schemes have been developed for visualizing high-dimensional data on a computer screen. MDS solves a non-linear optimization problem by searching for the mapping that best approximates all the high-dimensional pairwise distances between data points. In most occasions, MDS is simply too complex to be used for similarity search.

There are other techniques less computationally intensive techniques that are developed for metric spaces. One such technique is the Fastmap algorithm proposed by Faloutsos and Lin [35]. Fastmap is a heuristics algorithm that uses Euclidean distance to approximate a general metric. The time complexity of generating a fastmap mapping is linear with respect to the size of the database. In Section 4.3, we compare the search performance of fastmap with our proposed technique. Another class of techniques constructs feature extraction mappings based on distances between the high-dimensional vectors and a set of random vectors. These kinds of "random mappings" have been shown to possess certain favorable theoretical properties. [36] and [37] have shown that a specific form of the random mappings can achieve the best possible approximation of high-dimensional distances. Unfortunately, such mappings

are quite complex, and effectively require the computations of all pairwise distances. A more practical version has been proposed in [38] for approximating a metric used in protein sequencing. An even simpler version, called Triangle-Inequality Pruning (TIP), has been proposed by Berman and Shapiro for similarity search on image databases [39]. In Chapter 4, we propose a novel feature extraction mapping for metric-space data. This technique improves upon TIP by taking into account both the upper and lower bounds offered by the triangle-inequality. It also exploits the classical PCA technique in order to achieve any target dimension. As we will demonstrate in Chapter 4, our proposed scheme outperforms many other techniques in the literature in terms of the search performance on a large database of video signatures.

1.3 Similarity clustering

For a meaningful presentation of similarity search results, we investigate the use of clustering algorithms on a large database of video signatures. The goal is to group similar video sequences into non-overlapping clusters so a user is presented an uncluttered view of the results. A clustering structure provides an efficient organization of data which allows users to rapidly focus on relevant information. For example, clustering is extensively used in the areas of browsing and navigation [40, 41] as well as story segmentation of video clips [42, 43]. There are other benefits to applying clustering techniques to large databases. It has long been observed that clustering similar data items can improve the performance of a text-based information retrieval

system [44]. A number of recent studies have demonstrated that retrieval performance on multimedia information systems can also be improved via clustering [45, 21, 25].

Clustering experiments on web text documents show that the number of clusters with similar documents is likely to be very large [1]. It is difficult to apply many popular optimization-based clustering algorithms, such as the k-means method, to our problem as many of them need the precise number of clusters as input [46, ch. 14]. Another popular class of clustering algorithms, called the hierarchical algorithms, do not have such a requirement [46, ch. 13]. Hierarchical algorithms recursively create new clusters by either subdividing or merging existing ones. Different hierarchical algorithms use different criteria to decide which clusters to merge or divide. A common approach is based on the distances between centroids of the existing clusters. Nevertheless, in general metric spaces where distances are the only measurable quantities, there may not be a sensible way to compute the centroid of a cluster. In addition, as the ViSig method is a randomized algorithm, there are uncertainties associated with each signature. Centroids computed based on erroneous signature vectors certainly do not reflect the actual locations of the clusters.

Rather than computing centroids, we can treat each data point as a vertex of a graph, and form edges between two data points if their distances are less than a certain threshold. The hierarchical clustering algorithm then considers the graphs formed at different thresholds, and identifies parts of the graphs as clusters based on their degree of connectivity. The simplest of such algorithms are the single-link and

complete-link algorithms [46]. The single-link algorithm identifies all the connected components in the graph as clusters, and the complete-link uses complete subgraphs. Both algorithms are viable candidates for clustering, but the results obtained by applying them to our data are dissatisfactory. The problem with the single-link algorithm is that its cluster definition is too lenient. As a result, a single-link algorithm produces some large clusters that contain totally irrelevant video clips. The cluster definition of the complete-link algorithm, in contrast, is too stringent – it dismisses true clusters in the presence of one single erroneous distance measurement. Ideally, a clustering algorithm should aim at identifying clusters between these two extremes of single-link and complete-link.

In Chapter 5, we propose a new hierarchical clustering algorithm that allows the user to adjust the desirable level of connectivity for cluster identification. In the proposed algorithm, a connected component forms a cluster if its edge density exceeds a user-defined threshold. Not only does the proposed algorithm produce favorable retrieval results, it admits a simple implementation based on the classical Minimum Spanning Tree (MST) algorithm by Kruskal [47]. In [48], Zahn has used MST to separate data into different clusters if the MST branch connecting them is significantly longer than the nearby edges [48]. We extend this concept to consider the connectivity of the clusters. Recently, a number of graph-theoretical clustering algorithms based on network-flow algorithms have been proposed for visual grouping and gene expression clustering [49, 50]. Compared to MST, these techniques are far

more computationally intensive, and are thus difficult to scale to very large databases.

1.4 Organization

This dissertation is organized as follows: we first provide, in Appendix 1.A, a list of all the commonly-used acronyms and notations in this dissertation. Chapter 2 provides an overview of the design of a similarity video search engine. This design overview provides a functional description of how individual components proposed in this dissertation can be applied to a realistic design. This search engine can be accessed on the web at `http://www-video.eecs.berkeley.edu/~cheungsc/cluster`. The dataset of web video sequences, which we use throughout this dissertation, is also described in this chapter.

Chapter 3 is devoted to video similarity measurement. Different video similarity models are discussed in this chapter, but the focus is on developing the ViSig method and its variations. We introduce the geometric interpretation of the ViSig method as an estimation of the intersecting volume between voronoi diagrams. This leads to the design of a number of heuristics that are essential to applying ViSig to real data. We present both experimental and simulation results to demonstrate the performance of ViSig.

Chapter 4 deals with fast similarity search on large databases of video signatures. After a brief review of a generic similarity search procedure, we focus on designing the feature extraction mapping on high-dimensional video signatures to low-dimensional

index vectors. The two main components of this mapping, namely the projection vector mapping and the PCA, are described in detail. Finally, we compare the search performance on randomly queries between the proposed technique and other state-of-the-art methods.

Chapter 5 discusses how clustering can be applied to improve retrieval performance. We first introduce the graphical representation of a database of video signatures, and define similar clusters as highly connected regions inside the graph. We then explain how these similar clusters can be identified by using a simple modification of Kruskal's minimum spanning tree algorithm. Experimental results are then presented comparing the proposed algorithm with simple thresholding, single-link and complete-link hierarchical clustering algorithms. Finally, we apply our algorithm to a large database of web video sequences, and statistically analyze the resulting clustering structure.

Chapter 6 presents a summary of the results in this dissertation along with suggestions for future work. Portions of Chapter 3 have appeared in [51, 52, 53, 54, 55], while parts of Chapters 4 and 5 have been presented in [56, 57].

1.A Appendix: Acronyms and Notations

Acronyms

NVS	Naïve Video Similarity
IVS	Ideal Video Similarity
VVS	Voronoi Video Similarity
ViSig	Video Signature
VSS_b	Basic ViSig Similarity
PDF	Probability Density Function
VSS_r	Ranked ViSig Similarity
HSV	Hue-Saturation-Value color space
GEMINI	Generic Multimedia Indexing
CC	Connected Component
MST	Minimum Spanning Tree

Notations

$(F, d(\cdot,\cdot))$	Feature vector space F with metric $d(\cdot,\cdot)$
x, y	Video frames, represented as feature vectors
X, Y	Video sequences, represented as sets of feature vectors
ϵ	Frame Similarity Threshold
1_X	Indicator function
$\lvert X\rvert$	Cardinality of set X
$\text{nvs}(X, Y; \epsilon)$	NVS between X and Y
$[X]_\epsilon$	Collection of clusters in X
$[X \cup Y]_\epsilon$	Clustered union between X and Y
$\text{ivs}(X, Y; \epsilon)$	IVS between X and Y
$V(X)$	Voronoi Diagram of video X
$V_X(x)$	Voronoi Cell of $x \in X$
$V_X(C)$	Voronoi Cell of a cluster $C \in [X]_\epsilon$

$R(X, Y; \epsilon)$	Similar Voronoi Region
$\mathrm{Vol}(A)$	Volume of a region A
$\mathrm{Prob}(A)$	Probability of event A
$\mathrm{vvs}(X, Y; \epsilon)$	VVS between X and Y
X_S	Signature of X with respect to the SV set S
$g_X(s)$, x_s	Signature vector of X with respect to s
$\mathrm{vss}_b(X_S, Y_S; \epsilon, m)$	VSS_b between X_S and Y_S
m	Number of signature vectors in a signature
$f(u; X \cup Y)$	PDF that assigns equal probability to the Voronoi Cell of each cluster in $[X \cup Y]_\epsilon$
$G(X, Y; \epsilon)$	Voronoi gap between X and Y
$Q(g_X(s))$	Ranking function for the signature vector $g_X(s)$
$\mathrm{vss}_r(X_S, Y_S; \epsilon, m)$	VSS_r between X_S and Y_S
m'	Number of top-ranked signature vectors used in computing VSS_r
$\widehat{\mathrm{vss}_r}(X_S, Y_S; \epsilon, m)$	Asymmetric VSS_r between X_S and Y_S using the ranking of X_S
$d_c^q(x_i, y_i), \widehat{d_c^q}(x_i, y_i)$	l_1 and modified l_1 color histogram distances
$d_c(x, y), \widehat{d}_c(x, y)$	Quadrant color histogram distances based on $d_c^q(\cdot, \cdot)$ and $\widehat{d_c^q}(\cdot, \cdot)$
ρ	Dominant color threshold used in $\widehat{d_c^q}(\cdot, \cdot)$
$\mathrm{rel}(X)$	The set of video sequences that are subjectively similar to video X as defined in the ground-truth set
$\mathrm{ret}(X, \epsilon)$	The set of video sequences that are declared to be similar to X by the ViSig method at ϵ level
$\mathrm{Recall}(\epsilon)$	The recall in retrieving the ground-truth by the ViSig method at ϵ level
$\mathrm{Precision}(\epsilon)$	The precision in retrieving the ground-truth by the ViSig method at ϵ level
$A(x; \epsilon)$	The result set of similarity search on feature vector x
$A_S(X_S; \epsilon)$	The result set of similarity on signature X_S
ϵ'	Pruning threshold
$C(x; \epsilon')$	The GEMINI candidate set for feature vector x
$C(X_S; \epsilon')$	The GEMINI candidate set for signatures X_S
$A'(x; \epsilon, \epsilon')$	The GEMINI result set for feature vector x
$A'(X_S; \epsilon, \epsilon')$	The GEMINI result set for signature X_S
$(F', d'(\cdot, \cdot))$	Range space and Range metric
$\mathcal{T}(x)$	Feature extraction mapping of feature vector x
$\mathcal{P}(x)$	Projection vector mapping of feature vector x
$d_{sig}(\cdot, \cdot)$	Signature distance

$V(\mathcal{G})$	Vertex set of a graph $\mathcal{G}$
$E(\mathcal{G})$	Edge set of a graph $\mathcal{G}$
$P(\mathcal{V}, \rho)$	Threshold graph on vertex set $\mathcal{V}$ and distance threshold ρ
$\Gamma(\mathcal{G})$	Edge density of graph $\mathcal{G}$
γ	Edge density threshold

Chapter 2

A similarity video search engine

This chapter describes a similarity video search engine that utilizes all our proposed algorithms, as well as the dataset behind this engine and other experiments presented throughout the dissertation. The functional description of the search engine can be found in Section 2.1. A prototype of this engine can be accessed at `http://www-video.eecs.berkeley.edu/~cheungsc/cluster`. Section 2.2 describes the web video dataset behind this search engine and the data collection process. We also derive a ground-truth set from this dataset to measure the retrieval performance of various algorithms. The construction of the ground-truth set is described in Section 2.3.

2.1 System description

The architecture of our proposed search engine is shown in Figure 2.1. The Uniform Resource Locator or URL database contains a large number of URL addresses of video hyperlinks, which are acquired by traversing the web with a web crawler. The web video capturer reads the URL addresses from the URL database, downloads the corresponding video clip, and identifies relevant meta-data terms. Meta-data terms are textual information about the video clip. They consist of terms extracted from the URL address of the video hyperlink, the description of the hyperlink, the title and address of the web-page, and auxiliary information such as the creator's name and the copyright notice embedded in the clip. The meta-data terms are stored in the cluster & meta-data database, while the video-data is sent to the signature & index generation process.

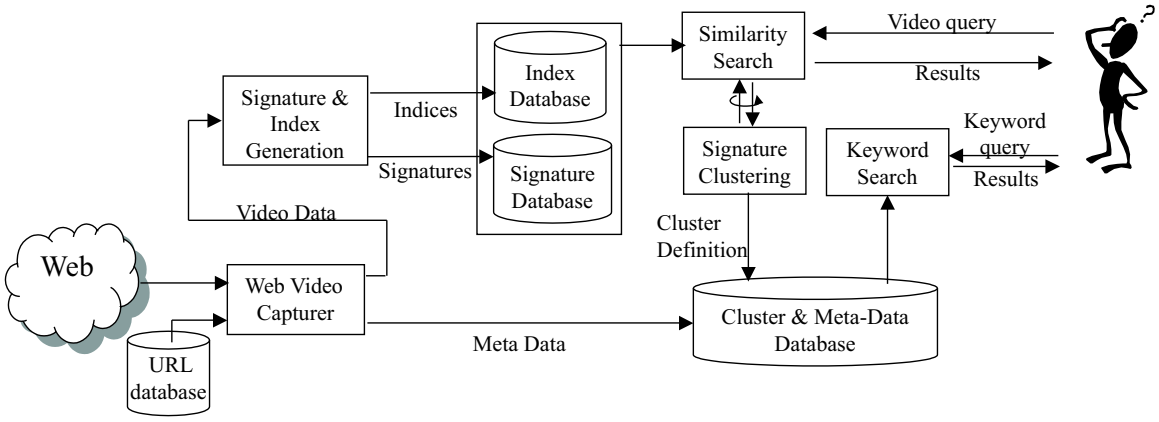


Figure 2.1: *Schematic of the video search engine.*

During the signature & index generation process, a signature and an index are

generated for each input video clip. A signature is a compact representation of a video clip, while an index is an even smaller entity that is used to facilitate similarity search on signatures. Details on how indices are used to facilitate fast similarity search on signatures are described in Chapter 4. The video search engine uses similarity search in two different modes. First, it allows a user to search for video content in the search engine database that are potentially similar to the input query video clip. To do this, a user needs to first download software to produce a signature for his/her video clip. The signature is then sent to the search engine where a similarity search is performed. Thumbnail images and hyperlink information for all signatures within a small distance threshold of the query are then presented to the user.

Similarity search is also used by the signature clustering process in our search engine. Based on the similarity search results of all the signatures, the signature clustering process identifies clusters of similar video sequences in the database. These similarity clusters can be used in many different ways: First, we show in Chapter 5 that it is possible to improve retrieval performance by returning similar clusters, rather than individual video, that are close to the input query. Second, we can use the resulting clusters to expand the results of keyword search, which is another function supported by our search engine. After the signature clustering process, membership information of all the clusters is stored in the cluster & meta-data database. For each meta-data term k in the database, we identify those clusters with at least one video clip that has k in its meta-data record. All these clusters will be returned if k is a

part of the user's keyword search. This approach expands the simple paradigm of keyword search to include those visually similar video clips that may not have any meta-data term that matches the query keyword. To illustrate this concept via an example, consider querying our search engine with the keyword "telephone." Figure 2.2 shows a screen-shot of the search results. Thumbnail images are used to represent returned clusters, which are ranked by the number of video clips in them. The detailed listing of all video clips is shown by clicking on the thumbnail image. As shown in the figure, 48 video clips relevant to the keyword "telephone" are retrieved, despite the fact that only eight clips actually have the term "telephone" in their meta-data records. These 48 video clips are organized into seven clusters of visually similar video sequences. The cluster organization provides the user a concise summary of all the visually distinctive video sequences among the search results.

2.2 Web video acquisition

In order to demonstrate the applicability of our algorithms, it is important to base our results on a representative collection of video sequences on the web. Most experimental results presented in this dissertation are based on a collection of 46,331 video sequences, crawled from the web between June and December of 1999. This section briefly describes the approach used to acquire these sequences as well as the nature of the web video sequences in our collection.

A common approach to collect data from the web is to use a web crawler. A web

crawler is a program that automatically traverses the web's hyperlink structure and retrieves desired information. As video sequences are sparsely distributed over the web, a web crawler requires substantial amount of time and resources to collect a representative data-set. Our approach to building a video collection is to send a large set of queries to the AltaVista video search engine to obtain URL addresses of web

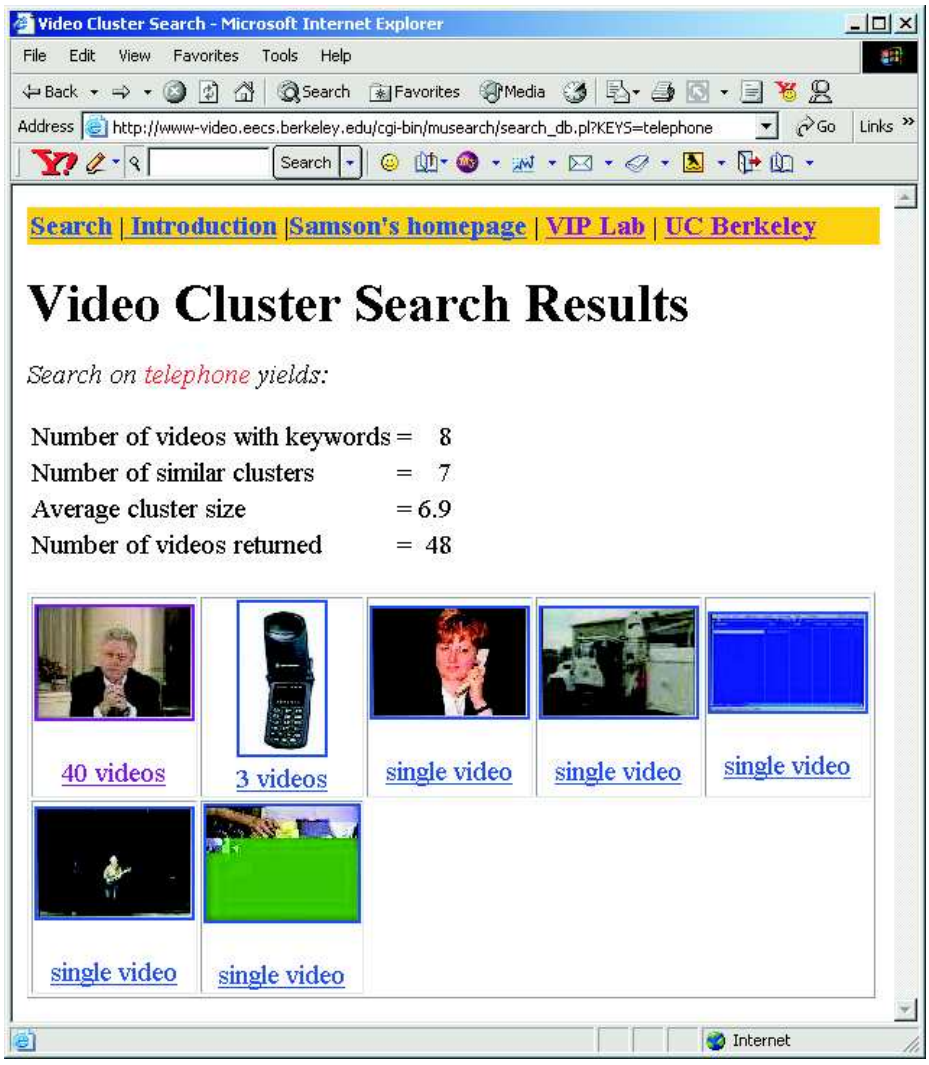


Figure 2.2: *Search results of keyword "telephone".*

video sequences. Similar methods have been used to estimate the size of the web [58]. To avoid bias towards particular types of content, our query word set consists of about 300,000 words obtained from a general search engine [59], an Internet video-on-demand site [60] and a speech recognizer vocabulary [61]. All query requests are carefully controlled so as not to overburden the search engine. Over the entire month of June 1999, about 62,000 URL addresses pointing to video content were obtained. This data-set constitutes roughly 15% of all the non-broadcast video clips on the web at that time, according to the figure estimated by Compaq Cambridge Research Laboratory in November 1998 [62].

The second step is to follow the resulting URLs and download the actual video sequences. Among all the video URLs, the most popular formats are RealVideo, Quicktime, MPEG-1, and AVI. Except for MPEG-1 which is an open standard [63], the remaining formats are proprietary. This has a significant impact on the download time since no fast bit-stream level processing can be applied, and the video sequences can only be captured after full decoding. In other words, the capture time is limited by the decoding speed or even real-time display in certain formats. RealVideo streaming format [64] presents an additional level of challenge since its display quality depends on the network conditions during the download. Depending on the settings of the content server, heavy packet losses on the network may cause delay, frame drops or corrupted frames. We developed capture software that takes the delay due to packet losses into account but fails to detect frame drops or corrupted frames. As

a result, the quality of the captured video sequences may vary significantly even for the same video downloaded from two different locations. In order to reduce storage requirements, all video sequences are re-sampled at three frames per second. For each sequence, almost identical neighboring frames with peak signal to noise ratio larger than 50 dB are removed, and the remaining frames are compressed using motion-JPEG.

After eliminating synonymous[1] and expired URL entries, we capture 46,331 video clips with total duration of approximately 1800 hours. The total disk space required for the motion-JPEG video sequences exceeds 100 Gigabytes. The total capture time was approximately 30 days using four Intel Pentium-based personal computers. In other words, on average, it takes 1.6 hours to capture 1 hour of video. The bottleneck in capturing is primarily due to the buffering delay in recording streaming RealVideo. The statistics of the four most abundant types of collected video sequences are shown in Table 2.1. Except RealVideo video sequences, most of the other sequences are less than one minute long.

2.3 Ground-truth construction

A ground-truth set is commonly used as an experimental tool to measure how well an automatic retrieval algorithm can match human judgment [44, 98]. A general

[1]Synonymous URLs are detected using the following heuristics [65] : (i) removing the port 80 designation (the default), (ii) removing the first segment of the domain name for URLs with a directory depth greater than one (to account for machine aliases), and (iii) unescaping any "escaped" characters.

Video Type	% over all clips	Duration (mean ± std-dev in minutes)
MPEG	31	0.26 ± 0.7
QuickTime	30	0.51 ± 0.6
RealVideo	22	9.57 ± 18.5
AVI	16	0.16 ± 0.3

Table 2.1: *Statistics of collected web video sequences*

ground-truth set consists of multiple clusters of data items from a large dataset. Each cluster of the ground-truth set contains all the items in the dataset that are considered to be relevant to a particular concept by a group of human subjects. By presenting each data item in the ground-truth set as a query to an automatic retrieval system, we can measure how well the system can approach human judgment.

In our particular application, the ground-truth set contains clusters of highly similar video sequences from the web video dataset. Ideally, each group in the ground-truth set should capture all of the similar versions of the same video content in the entire dataset. Rather than manually examining the entire set of more than 1800 hours of video, we adopt a best-effort approach to obtain such a ground-truth. This approach is similar to the pooling method commonly used in text retrieval systems [44]. The basic idea of pooling is to first send the same queries to different automatic retrieval systems, other than the one being tested. Then, the top-ranked results from these systems are pooled together and examined by human experts to identify the truly relevant ones. The goal of pooling is to reduce human effort by using automatic systems to eliminate a large number of irrelevant results.

For our system, the first step is to use meta-data terms to identify the initial ground-truth clusters. As described in Section 2.2, meta-data terms are extracted from the URL address and other textual information of each video. All video sequences in the dataset containing at least one of the top 1000 most frequently used meta-data terms are manually examined and grouped into clusters of similar video. Clusters which are significantly larger than others are removed to prevent bias. We obtained 107 clusters which form the initial ground-truth clusters. This method, however, may not be able to identify all the video clips in the dataset that are similar to those already in the ground-truth clusters. We further examined those video sequences in the dataset that share at least one meta-data term with the ground-truth video, and add any similar video to the corresponding clusters.

In addition to meta-data, we apply visual similarity scheme to enlarge the ground-truth as well. In particular, we identify video sequences in the dataset with color distribution similar to those already in the ground-truth. It has been shown that color is one of the most important low-level visual cue in identifying similar visual content [70]. We briefly describe here how we expand the ground-truth with color similarity, but defer all the technical details of similarity measurement until Section 3.7. We first convert every frame of a video into a color histogram feature vector. Then, each video clip in the dataset and the ground-truth set is summarized as a k-medoid of seven feature vectors. As mentioned in Chapter 1, k-medoid is a summarization technique that minimizes the distance between the original video and

its summarization [28, 21]. For each k-medoid X in the ground-truth, we identify 100 k-medoids in the dataset that are closest to X in terms of the minimum distance between all the vectors in their corresponding k-medoid representations. These 100 video clips are again manually examined to identify those visually similar to X. As a result, we obtain a ground-truth set consisting of 443 video sequences in 107 clusters. These ground-truth clusters and their sizes are shown in Figures 2.3 and 2.4. Each cluster is represented by a video frame randomly selected from one of the sequences within the cluster. The cluster size ranges from 2 to 20, with average size equal to 4.1. For all our retrieval experiments, these ground-truth clusters serve as the basis for comparison against those similar video sequences identified by the automatic retrieval algorithms.

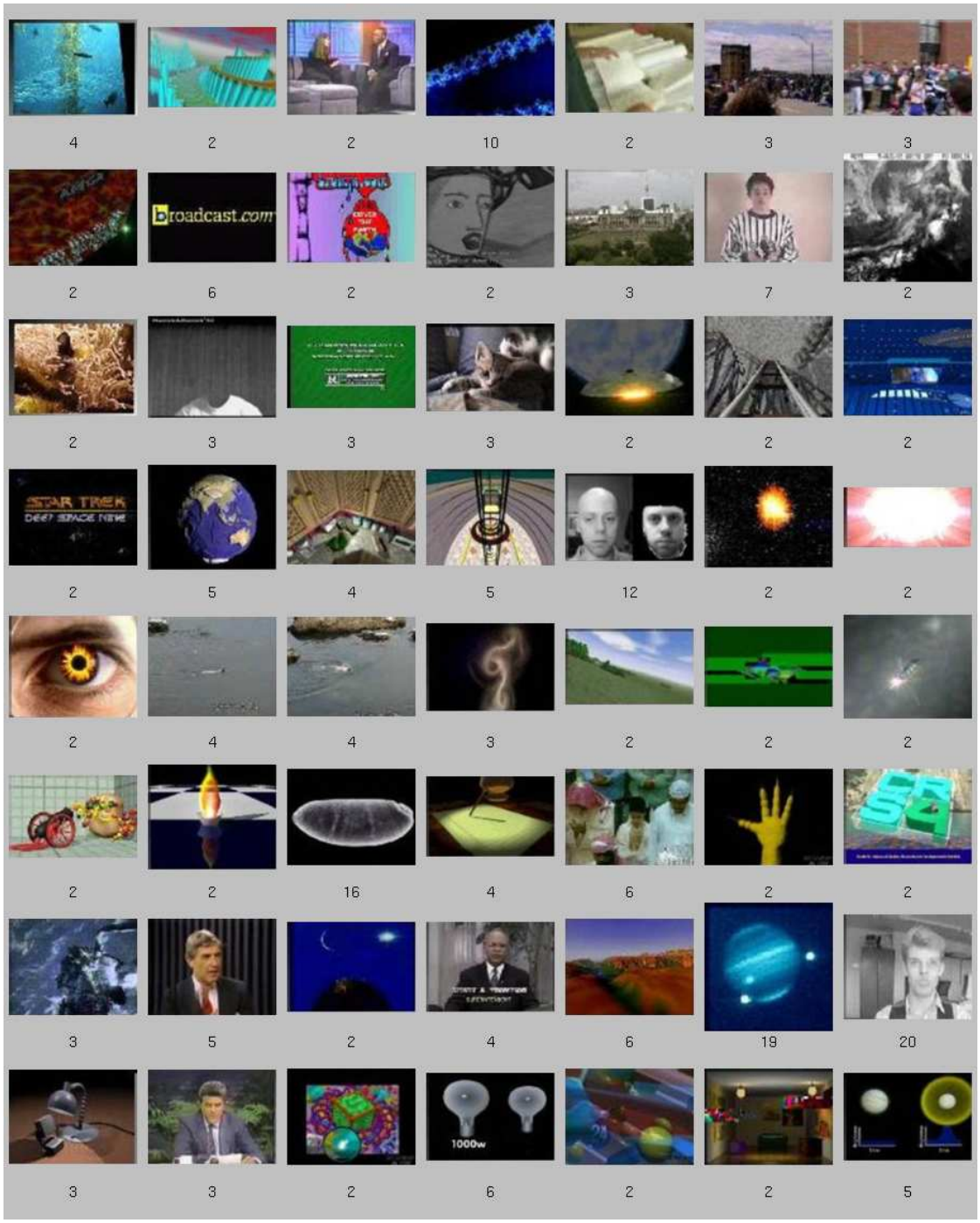


Figure 2.3: *Ground-truth clusters and their sizes (part 1 of 2).*

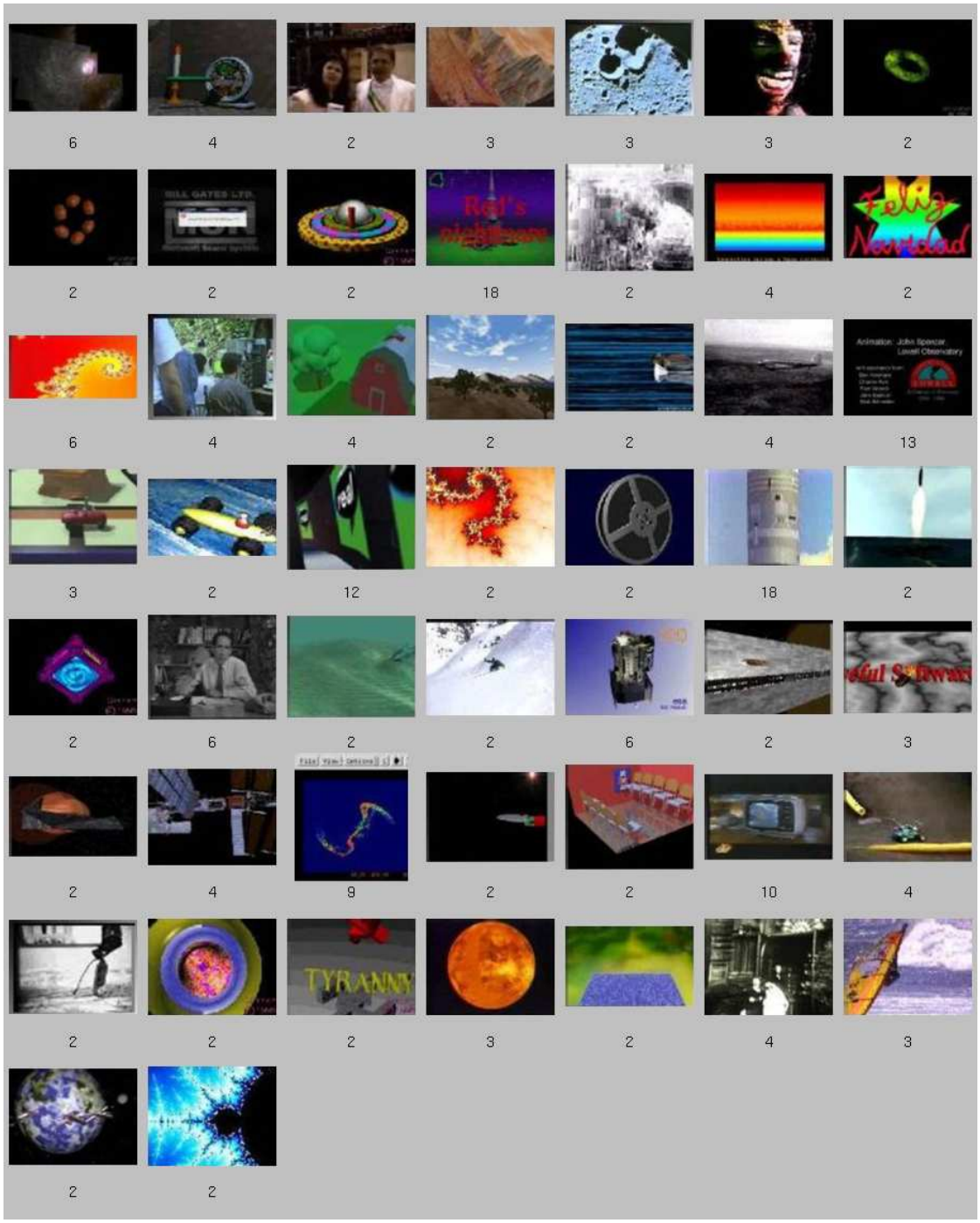


Figure 2.4: *Ground-truth clusters and their sizes (part 2 of 2).*

Chapter 3

Measuring video similarity

This chapter defines the video similarity models used in this dissertation, and describes how they can be efficiently estimated by the ViSig method. We assume that individual frames in a video sequence are represented by high-dimensional feature vectors from a metric space $(F, d(\cdot,\cdot))$[1]. In order to be robust against editing changes in the temporal domain, we define a video sequence X as a finite set of feature vectors and ignore any temporal ordering. For the remainder of this chapter, we make no distinction between a video frame and its corresponding feature vector. The metric $d(x, y)$ measures the visual dissimilarity between vectors x and y. We assume that vectors x and y are visually similar to each other if and only if $d(x, y) \leq \epsilon$ for an $\epsilon > 0$ independent of x and y. We call ϵ the Frame Similarity Threshold.

This chapter is organized as follows. Section 3.1 defines our target measure, called

[1]For all x, y in F, the function $d(x, y)$ is a metric if a) $d(x, y) \geq 0$; b) $d(x, y) = 0 \Leftrightarrow x = y$; c) $d(x, y) = d(y, x)$; d) $d(x, y) \leq d(x, z) + d(z, y)$, for all $z \in F$.

the Ideal Video Similarity (IVS), used in this chapter to gauge the visual similarity between two video sequences. As we explain in the section, this similarity measure is complex to compute exactly, and requires a significant number of vectors to represent each video. To reduce the computational complexity and the representation size, we propose an alternative form of video similarity called the Voronoi Video Similarity (VVS) in Section 3.2. This particular form of similarity leads directly to an efficient technique for representation and estimation called the ViSig method, described in detail in Section 3.3. Sections 3.4 through 3.6 analyze the scenarios where IVS cannot be reliably estimated by our proposed algorithm, and propose a number of heuristics to rectify the problems. Experimental results are presented in Section 3.7. We summarize this chapter in Section 3.8. The proofs to all propositions in this chapter can be found in Appendix 3.A.

3.1 Ideal video similarity

As mentioned in Chapter 1, we are interested in using a video similarity measure that is based on the percentage of visually similar frames between two sequences. A naive way to compute such a measure is to first find the total number of frames from each video sequence that have at least one visually similar frame in the other sequence. Then, compute the ratio of this number with the overall total number of frames as the final similarity value. We call this measure the Naïve Video Similarity (NVS):

Definition 3.1.1 Naïve Video Similarity

Let X and Y be two video sequences. The number of vectors in video X that have at least one similar vector in Y can be computed by $\sum_{x \in X} 1_{\{y \in Y : d(x,y) \leq \epsilon\}}$, where 1_A is the indicator function with $1_A = 1$ if A is not empty, and zero otherwise. The Naïve Video Similarity between X and Y, $\mathrm{nvs}(X, Y; \epsilon)$, can thus be defined as follows:

$$\mathrm{nvs}(X, Y; \epsilon) \triangleq \frac{\sum_{x \in X} 1_{\{y \in Y : d(x,y) \leq \epsilon\}} + \sum_{y \in Y} 1_{\{x \in X : d(y,x) \leq \epsilon\}}}{|X| + |Y|}, \tag{3.1}$$

where $|\cdot|$ denotes the cardinality of a set, or in our case the number of vectors in a given video.

If every vector in video X has a similar vector in Y and vice versa, $\mathrm{nvs}(X, Y; \epsilon) = 1$. If X and Y share no similar vectors at all, $\mathrm{nvs}(X, Y; \epsilon) = 0$.

Unfortunately, NVS does not always reflect our intuition of video similarity. Most real-life video sequences can be temporally separated into video shots, within which frames are visually similar. Among all possible versions of the same video, the number of frames in the same shot can be quite different. For instance, different coding schemes modify the frame rates for different playback capabilities, and video summarization algorithms use a single keyframe to represent an entire shot. As NVS is based solely on frame counts, its value is highly sensitive to these kinds of manipulations. To illustrate this problem with a pathological example, consider the two sequences shown in Figure 3.1. We represent the feature vector space as a 2-D square. Crosses and dots in the figure signify frames from two different video sequences X and Y respectively. X has two frames that are very far apart, while all the frames in Y are

clustered around one of the frames in X. This may happen, for example, when X is a key-frame sequence of a video with two distinct shots, and Y retains an entire shot of this video. Assume all eight frames in Y are within ϵ of that particular frame in X. Even though it is intuitive to say that the two sequences have 50% overlap, the measured NVS between X and Y is 90%. It is possible to rectify the problem by using shots as the fundamental unit for similarity measurement. Since we model a video as a set and ignore all temporal ordering, we instead group all visually similar vectors in a video together into non-intersecting units called clusters.

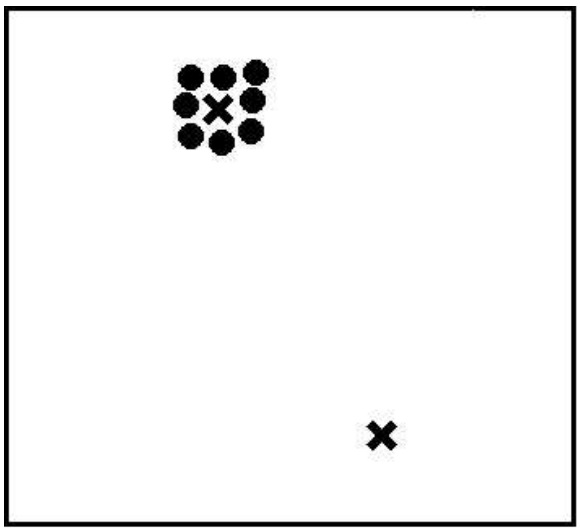


Figure 3.1: *Two video sequences with NVS equal to 0.9.*

A cluster should ideally contain only similar vectors, and no other vectors similar to the vectors in a cluster should be found in the rest of the video. Mathematically, we can express these two properties as follows: for all pairs of vectors x_i and x_j in

X, $d(x_i, x_j) \leq \epsilon$ if and only if x_i and x_j belong to the same cluster. Unfortunately, such a clustering structure may not exist for an arbitrary video X. Specifically, if $d(x_i, x_j) \leq \epsilon$ and $d(x_j, x_k) \leq \epsilon$, there is no guarantee that $d(x_i, x_k) \leq \epsilon$. If $d(x_i, x_k) > \epsilon$, there is no consistent way to group all the three vectors into clusters.

In order to have a general framework for video similarity, we adopt a relatively relaxed clustering structure by only requiring the forward condition, i.e. $d(x_i, x_j) \leq \epsilon$ implies that x_i and x_j are in the same cluster. A cluster is simply one of the connected components [66, appendix B] of a graph in which each node represents a vector in the video, and every pair of vectors within ϵ of each other is connected by an edge. We denote the collection of all clusters in video X as $[X]_\epsilon$. It is possible for such a definition to produce chain-like clusters where one end of a cluster is very far from the other end. Nonetheless, given an appropriate feature vector and a reasonably small ϵ, most clusters found in real video sequences are compact, i.e. all vectors in a cluster are similar to each other. We call a cluster ϵ-compact if all its vectors are within ϵ from each other. The clustering structure of a video can be computed by a simple hierarchical clustering algorithm called the single-link algorithm [67].

To define a similarity measure based on the visually similar portion shared between two video sequences X and Y, we consider the clustered union $[X \cup Y]_\epsilon$. If a cluster in $[X \cup Y]_\epsilon$ contains vectors from both sequences, these vectors are likely to be visually similar to each other. Thus, we call such a cluster a Similar Cluster and consider it as part of the visually similar portion. The ratio between the number of similar clusters

and the total number of clusters in $[X \cup Y]_\epsilon$ forms a reasonable similarity measure between X and Y. We call this measure the Ideal Video Similarity (IVS):

Definition 3.1.2 Ideal Video Similarity, IVS

Let X and Y be two video sequences. For each cluster C in $[X \cup Y]_\epsilon$, C contains vectors from both X and Y if and only if $1_{C\cap X} \cdot 1_{C\cap Y} = 1$. Thus, we can define the IVS between X and Y, $\mathrm{ivs}(X, Y; \epsilon)$, as follows:

$$\mathrm{ivs}(X, Y; \epsilon) \triangleq \frac{\sum_{C\in[X\cup Y]_\epsilon} 1_{C\cap X} \cdot 1_{C\cap Y}}{|[X \cup Y]_\epsilon|} \tag{3.2}$$

The main theme of this chapter is to develop efficient algorithms to estimate the IVS between a pair of video sequences. A simple pictorial example, shown in Figure 3.2, demonstrates the use of IVS. Vectors closer than ϵ are connected by dotted lines. There are altogether three clusters in the clustered union, and only one cluster has vectors from both sequences. The IVS measure is thus 1/3.

It is complex to precisely compute IVS. The clustering used in IVS depends on the distances between vectors from the two sequences. This implies that for two video sequences with l vectors each, one needs to first compute the distance between l^2 pairs of vectors before running the clustering algorithm and computing the IVS. In addition, the computation requires the entire video to be stored. The complex computation and large storage requirements are clearly undesirable for large database applications. As the exact similarity value is often not required in many applications,

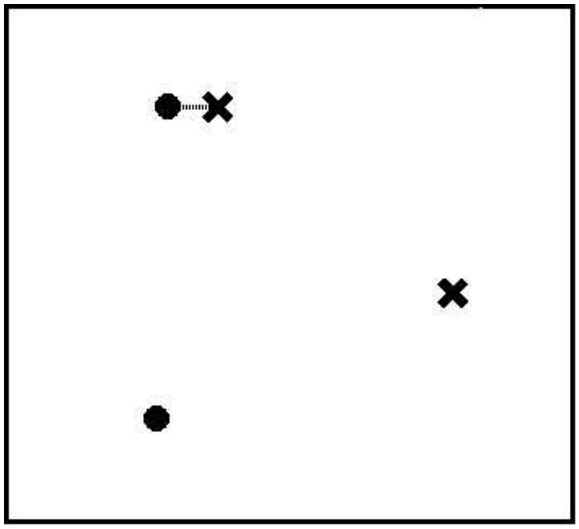


Figure 3.2: *Two video sequences with IVS equal to 1/3.*

sampling techniques can be used to estimate the true IVS. Consider the following simple sampling scheme: let each video sequence in the database be represented by m randomly selected vectors. We estimate the IVS between two sequences by counting the number of similar pairs of vectors W_m between their respective sets of sampled vectors. As long as the desired level of precision is satisfied, m should be chosen as small as possible to achieve low complexity. Nonetheless, even in the case when the IVS is as high as one, we show in the following proposition that we need a large m to find even one pair of similar vectors among the sampled vectors.

Proposition 3.1.1 *Let X and Y be two video sequences with l vectors each. Assume for every vector x in X, Y has exactly one vector y similar to it, i.e. $d(x, y) \leq \epsilon$. We also assume the same for every vector in Y. Clearly,* $\text{ivs}(X, Y; \epsilon) = 1$. *The expectation*

of the number of similar vector pairs W_m found between m randomly selected vectors from X and from Y is given below:

$$E(W_m) = \frac{m^2}{l}. \tag{3.3}$$

Despite the fact that the IVS between the video sequences is one, Equation (3.3) shows that we need, on average, $m = \sqrt{l}$ sample vectors from each video to find just one similar pair. Furthermore, comparing two sets of $\sqrt{l}$ vectors requires l high-dimensional metric computations. A better random sampling scheme should use a fixed-size record to represent each video, and require far fewer vectors to identify highly similar video sequences. Our proposed ViSig method is precisely such a scheme and is the topic of the following section.

3.2 Voronoi video similarity

As described in the previous section, the simple sampling scheme requires a large number of vectors sampled from each video to estimate IVS. The problem lies in the fact that since we sample vectors from two video sequences independently, the probability that we simultaneously sample a pair of similar vectors from them is rather small. Rather than independent sampling, the ViSig method introduces dependence by selecting vectors in each video that are similar to a set of predefined random feature vectors common to all video sequences. As a result, the ViSig method requires far

fewer sampled vectors to find a pair of similar vectors from two video sequences. The number of pairs of similar vectors found by the ViSig method depends strongly on the IVS, but does not have a one-to-one relationship with it. We call the form of similarity estimated by the ViSig method the Voronoi Video Similarity (VVS).

The term "Voronoi" in VVS is borrowed from a geometrical concept called the Voronoi Diagram. Given a video $X = \{x_t : t = 1, \ldots, l\}$, the Voronoi Diagram $V(X)$ of X is a partition of the feature space F into l Voronoi Cells $V_X(x_t)$. By definition, the Voronoi cell $V_X(x_t)$ contains all the vectors in F closer to $x_t \in X$ than to any other vectors in X, i.e. $V_X(x_t) \triangleq \{s \in F : g_X(s) = x_t \text{ and } x_t \in X\}$, where $g_X(s)$ denotes the vector in X closest[2] to s. A simple Voronoi diagram of a video is shown in Figure 3.3. We can extend the idea of the Voronoi diagram to video clusters by merging Voronoi cells of all the vectors belonging to the same cluster. In other words, for $C \in [X]_\epsilon$, $V_X(C) \triangleq \bigcup_{x \in C} V_X(x)$.

Given two video sequences X and Y and their corresponding Voronoi diagrams, we define the Similar Voronoi Region $R(X, Y; \epsilon)$ as the union of all the intersection between the Voronoi cells of those $x \in X$ and $y \in Y$ where $d(x, y) \leq \epsilon$:

$$R(X, Y; \epsilon) \triangleq \bigcup_{d(x,y) \leq \epsilon} V_X(x) \cap V_Y(y). \tag{3.4}$$

[2]If there are multiple x's in X that are equidistant to s, we choose $g_X(s)$ to be the one closest to a predefined vector in the feature space such as the origin. If there are still multiple candidates, more predefined vectors can be used until a unique $g_X(s)$ is obtained. Such an assignment strategy ensures that $g_X(s)$ depends only on X and s but not some arbitrary random choices. This is important to the ViSig method which uses $g_X(s)$ as part of a summary of X with respect to a randomly selected s. Since $g_X(s)$ depends only on X and s, sequences identical to X produce the same summary vector with respect to s.

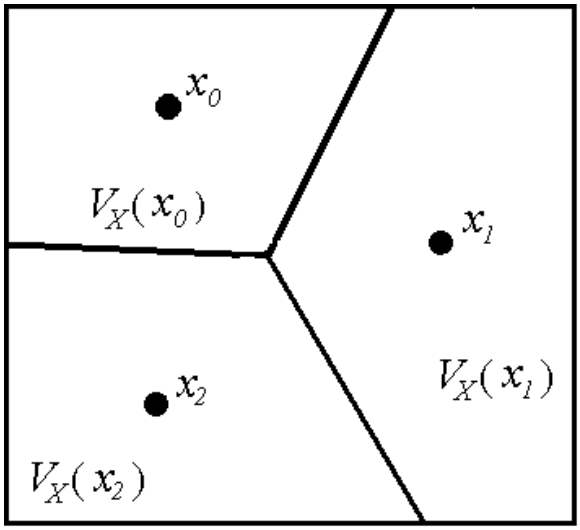


Figure 3.3: *The Voronoi diagram of a three-frame video.*

It is easy to see that if x and y are close to each other, their corresponding Voronoi cells are very likely to intersect in the neighborhood of x and y. The larger number of vectors from X and from Y that are close to each other, the larger the resulting $R(X, Y; \epsilon)$ becomes. A simple pictorial example of two video sequences with their Voronoi diagrams is shown in Figure 3.4: dots and crosses represent the vectors of the two sequences; the solid and broken lines are the boundary between the two Voronoi cells of the two sequences represented by dots and crosses respectively. The shaded region shows the similar Voronoi region between these two sequences. Similar Voronoi region is the target region whose volume defines VVS. Before providing a definition of VVS, we need to first clarify what we mean by the volume of a region in the feature space.

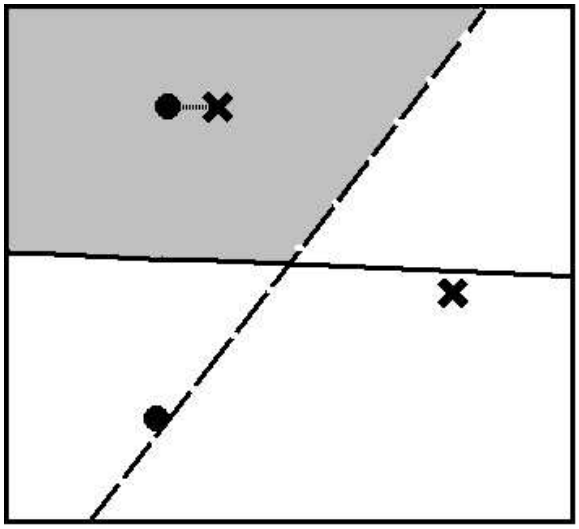


Figure 3.4: *The shaded area denotes the similar Voronoi region between the two sequences.*

We define the volume function $\mathrm{Vol} : \Omega \to \mathbb{R}$ to be the Lebesgue measure over the set, Ω, of all the measurable subsets in the feature space F [68]. For example, if F is the real line and the subset is an interval, the volume function of the subset is just the length of the interval. We assume all the Voronoi cells considered in our examples to be measurable. We further assume that F is compact in the sense that $\mathrm{Vol}(F)$ is finite. Because we are going to normalize all volume measurements by $\mathrm{Vol}(F)$, we assume that $\mathrm{Vol}(F) = 1$. To compute the volume of the similar Voronoi region $R(X, Y; \epsilon)$ between two video sequences X and Y, we first notice that individual terms inside the union in Equation (3.4) are disjoint from each other. By the basic properties of Lebesgue measure, we have

$$\mathrm{Vol}(R(X,Y;\epsilon)) = \mathrm{Vol}(\bigcup_{d(x,y)\leq\epsilon} V_X(x) \cap V_Y(y)) = \sum_{d(x,y)\leq\epsilon} \mathrm{Vol}(V_X(x) \cap V_Y(y)).$$

Thus, we define the VVS between two video sequences X and Y as follows:

$$\text{vvs}(X, Y; \epsilon) \triangleq \sum_{d(x,y) \leq \epsilon} \text{Vol}(V_X(x) \cap V_Y(y)) \tag{3.5}$$

The VVS of the two sequences shown in Figure 3.4 is the area of the shaded region, which is about 1/3 of the area of the entire feature space. Notice that for this example, the IVS is also 1/3. VVS and IVS are close to each other because the Voronoi cell for each cluster in the cluster union has roughly the same volume (area). In general, when the clusters are not uniformly distributed over the feature space, there can be a large variation among the volumes of the corresponding Voronoi cells. Consequently, VVS can be quite different from IVS. Before explaining how we can reconcile these two similarity measures, we first introduce the core algorithm in this chapter, the Basic ViSig method, as a randomized technique to estimate VVS.

3.3 Video signature method

It is straightforward to estimate $\text{vvs}(X, Y; \epsilon)$ by random sampling. First, generate a set S of m independent uniformly distributed random vectors $s_1, \ldots, s_m$, which we call Seed Vectors. By uniform distribution, we mean for every measurable subset A in F, the probability of generating a vector from A is $\text{Vol}(A)$. Second, for each seed vector $s \in S$, determine if s is inside $R(X, Y; \epsilon)$. By definition, s is inside $R(X, Y; \epsilon)$ if and only if s belongs to some Voronoi cells $V_X(x)$ and $V_Y(y)$ with $d(x, y) \leq \epsilon$. Since s must be inside the Voronoi cell of the vector closest to s in the entire video

sequence, i.e. $g_X(s)$ in X and $g_Y(s)$ in Y, an equivalent condition for $s \in R(X, Y; \epsilon)$ is $d(g_X(s), g_Y(s)) \leq \epsilon$. Since we only need $g_X(s)$ and $g_Y(s)$ to determine if each seed vector s belongs to $R(X, Y; \epsilon)$, we can summarize video X by the m-tuple $X_S \stackrel{\Delta}{=} (g_X(s_1), \ldots, g_X(s_m))$ and Y by Y_S. We call X_S and Y_S the Video Signature (ViSig), or simply signature, with respect to S of video sequences X and Y respectively. In the final step, we compute the percentage of signature vector pairs $g_X(s)$ and $g_Y(s)$ with distances less than or equal to ϵ to obtain:

$$\mathrm{vss}_b(X_S, Y_S; \epsilon, m) \stackrel{\Delta}{=} \frac{\sum_{i=1}^{m} 1_{\{d(g_X(s_i), g_Y(s_i)) \leq \epsilon\}}}{m}. \tag{3.6}$$

We call $\mathrm{vss}_b(X_S, Y_S; \epsilon, m)$ the Basic ViSig Similarity (VSS_b) between signatures X_S and Y_S. As every seed vector $s \in S$ in the above algorithm is chosen to be uniformly distributed, the probability of s being inside $R(X, Y; \epsilon)$ is simply the VVS between X and Y. Thus, $\mathrm{vss}_b(X_S, Y_S; \epsilon, m)$ forms an unbiased estimator of the VVS between X and Y. We refer to this approach of generating a signature and computing VSS_b the Basic ViSig method. To apply the Basic ViSig method to a large number of video sequences, we must use the same seed vector set S to generate all the signatures in order to compute VSS_b between an arbitrary pair of video sequences.

The number of seed vectors in S, m, is an important parameter. On one hand, m represents the number of samples used to estimate the underlying VVS and thus, a large m produces a more accurate estimation. On the other hand, the complexity of the Basic ViSig method depends on m. If a video has l vectors, it takes l metric

computations to generate a single signature vector. The number of metric computations required to compute the entire signature is thus $m \cdot l$. Also, computing the VSS_b between two signatures requires m metric computations. It is, therefore, important to determine an appropriate value of m that can satisfy both the desired estimation fidelity and the computational resources of a particular application. The following proposition provides an analytical bound on m in terms of the maximum error in estimating the VVS between any pair of video sequences in a database:

Proposition 3.3.1 *Assume we are given a database Λ with n video sequences and a set S of m random seed vectors. Define the error probability $P_{err}(m)$ to be the probability that any pair of video sequences in Λ has their m-vector VSS_b different from the true VVS value by more than a given $\gamma \in (0, 1]$, i.e.*

$$P_{err}(m) \stackrel{\Delta}{=} \mathrm{Prob}\Big(\bigcup_{X,Y \in \Lambda} \{ |\mathrm{vvs}(X, Y; \epsilon) - \mathrm{vss}_b(X_S, Y_S; \epsilon, m)| > \gamma \} \Big) \tag{3.7}$$

A sufficient condition to achieve $P_{err}(m) \leq \delta$ for a given $\delta \in (0, 1]$ is as follows:

$$m \;\; \geq \;\; \frac{2 \ln n - \ln \delta}{2\gamma^2} \tag{3.8}$$

It should be noted that the bound (3.8) in Proposition 3.3.1 only provides a sufficient condition and does not necessarily represent the tightest bound possible. Nonetheless, we can use this bound to understand the dependencies of m on various factors. First, unlike the random sampling described in Section 3.1, m does not

depend on the length of individual video sequences. This implies that it takes fewer vectors for the ViSig method to estimate the similarity between long video sequences than random vector sampling. Second, we notice that the bound on m increases with the natural logarithm of n, the size of the database. The signature size depends on n because it has to be large enough to simultaneously minimize the error of all possible pairs of comparisons, which is a function of n. Fortunately, the slow-growing logarithm makes the signature size rather insensitive to the database size, making it suitable for very large databases. The contribution of the term $\ln \delta$ is also quite insignificant. Comparatively, m is most sensitive to the choice of γ. A small γ means an accurate approximation of the similarity, but usually at the expense of a large number of sample vectors m to represent each video. The choice of γ should depend on the particular application at hand.

3.4 Seed vector generation

We have shown in the previous section that the VVS between two video sequences can be efficiently estimated by the Basic ViSig method. Unfortunately, the estimated VVS does not necessarily reflect the target measure of IVS as defined in Equation (3.2). For example, consider the two pairs of sequences in Figures 3.5(a) and (b). Dots and crosses are vectors from the two sequences, whose Voronoi diagrams are indicated by solid and broken lines respectively. The IVS's in both cases are 1/3. Nonetheless, the VVS in Figure 3.5(a) is much smaller than 1/3, while that of Figure

3.5(b) is much larger. Intuitively, as mentioned in Section 3.2, IVS and VVS are the same if clusters in the clustered union are uniformly distributed in the feature space. In the above examples, all the clusters are clumped in one small area of the feature space, making one Voronoi cell significantly larger than the other. If the similar cluster happens to reside in the smaller Voronoi cells, as in the case of Figure 3.5(a), the VVS is smaller than the IVS. On the other hand, if the similar cluster is in the larger Voronoi cell, the VVS becomes larger. This discrepancy between IVS and VVS implies that VSS_b, which is an unbiased estimator of VVS, can only be used as an estimator of IVS when IVS and VVS is close. Our goal in this section and the next is to modify the Basic ViSig method so that we can still use this method to estimate IVS even in the case when VVS and IVS are different.

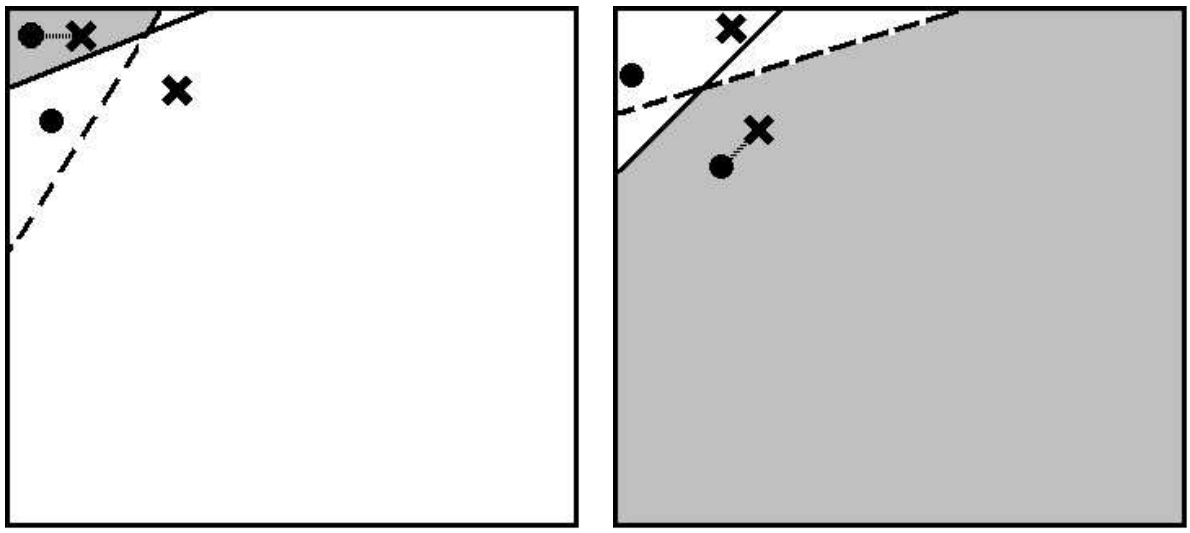


Figure 3.5: *Examples of sequences with identical IVS's but very different VVS's.*

As the Basic ViSig method estimates IVS based on uniformly-distributed seed vectors, the variation in the sizes of Voronoi cells affects the accuracy of the estimation. One possible method to amend the Basic ViSig method is to generate seed vectors based on a probability distribution such that the probability of a seed vector being in a Voronoi cell is independent of the size of the cell. Specifically, for two video sequences X and Y, we can define the Probability Density Function (PDF) based on the distribution of Voronoi cells in $[X \cup Y]_\epsilon$ at an arbitrary feature vector u as follows:

$$f(u; X \cup Y) \triangleq \frac{1}{|[X \cup Y]_\epsilon|} \cdot \frac{1}{\mathrm{Vol}(V_{X \cup Y}(C))} \tag{3.9}$$

where C is the cluster in $[X \cup Y]_\epsilon$ with $u \in V_{X \cup Y}(C)$. $f(u; X \cup Y)$ is constant within the Voronoi cell of each cluster, with the value inversely proportional to the volume of that cell. Under this PDF, the probability of a random vector u inside the Voronoi cell $V_X(C)$ for an arbitrary cluster $C \in [X \cup Y]_\epsilon$ is given by $\int_{V_X(C)} f(u; X \cup Y)\, du = 1/|[X \cup Y]_\epsilon|$. This probability does not depend on C, and thus, it is equally likely for u to be inside the Voronoi cell of any cluster in $[X \cup Y]_\epsilon$.

Recall that if we use uniform distribution to generate random seed vectors, VSS_b forms an unbiased estimate of the VVS defined in Equation (3.5). If we use $f(u; X \cup Y)$ to generate seed vectors instead, VSS_b now becomes an estimate of the following general form of VVS:

$$\sum_{d(x,y) \le \epsilon} \int_{V_X(x) \cap V_Y(y)} f(u; X \cup Y)\, du. \tag{3.10}$$

Equation (3.10) reduces to Equation (3.5) when $f(u; X \cup Y)$ is replaced by the uniform

distribution, i.e. $f(u; X \cup Y) = 1$. As shown by the following proposition, the general form of VVS in Equation (3.10) is equivalent to the IVS under certain conditions.

Proposition 3.4.1 *Assume we are given two video sequences X and Y. Assume clusters in $[X]_\epsilon$ and clusters in $[Y]_\epsilon$ either are identical, or share no vectors that are within ϵ from each other. Then, the following relation holds:*

$$\text{ivs}(X, Y; \epsilon) = \sum_{d(x,y) \leq \epsilon} \int_{V_X(x) \cap V_Y(y)} f(u; X \cup Y)\, du. \tag{3.11}$$

The significance of this proposition is that if we can generate seed vectors with $f(u; X \cup Y)$, it is possible to estimate IVS using VSS_b. The condition that all clusters in X are Y are either identical or far away from each other is to avoid the formation of a special region in the feature space called a Voronoi Gap. The concept of Voronoi gap is expounded in Section 3.5.

In practice, it is impossible to use $f(u; X \cup Y)$ to estimate the IVS between X and Y. This is because $f(u; X \cup Y)$ is specific to the two video sequences being compared, while the Basic ViSig method requires the same set of seed vectors to be used by all video sequences in the database. A heuristic approach for seed vector generation is to first select a set Ψ of training video sequences that resemble video sequences in the target database. Denote $T \stackrel{\Delta}{=} \bigcup_{Z \in \Psi} Z$. We can then generate seed vector based on the PDF $f(u; T)$, which ideally resembles the target $f(u; X \cup Y)$ for an arbitrary pair of X and Y in the database.

To generate a random seed vector s based on $f(u;T)$, we follow a four-step algorithm, called the Seed Vector Generation method, as follows:

1. Given a particular value of ϵ_{sv}, identify all the clusters in $[T]_{\epsilon_{sv}}$ using the single-link algorithm [67].

2. As $f(u;T)$ assigns equal probability to the Voronoi cell of each cluster in $[T]_{\epsilon_{sv}}$, randomly select a cluster C from $[T]_{\epsilon_{sv}}$ so that we can generate the seed vector s within $V_T(C)$.

3. As $f(u;T)$ is constant over $V_T(C)$, we should ideally generate s as a uniformly-distributed random vector over $V_T(C)$. Unless $V_T(C)$ can be easily parameterized, the only way to achieve this goal is to repeatedly generate uniform sample vectors over the entire feature space until a vector is found inside $V_T(C)$. This procedure may take an exceedingly long time if $V_T(C)$ is small. To simplify the generation, we select one of the vectors in C at random and output it as the next seed vector s.

4. Repeat the above process until the required number of seed vectors has been selected.

In Section 3.7, we compare performance of this algorithm against uniformly distributed seed vector generation in retrieving real video sequences.

3.5 Voronoi gap

We show in Proposition 3.4.1 that the general form of VVS using an appropriate PDF is identical to IVS, provided that all clusters between the two sequences are either identical or far away from each other. As feature vectors are not perfect in modeling the human visual system, visually similar clusters may have feature vectors that are close but not identical to each other. Consider the example in Figure 3.6 where vectors in similar clusters are not identical but within ϵ of each other. Clearly,

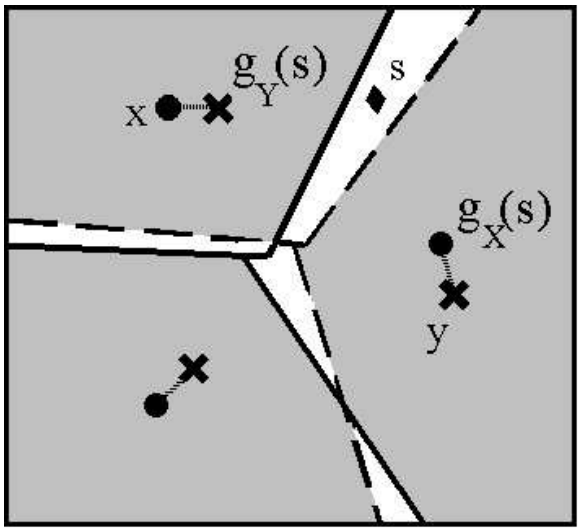


Figure 3.6: *The unshaded region is the Voronoi gap for this pair of video sequences with IVS one.*

the IVS between the two sequences shown in Figure 3.6 is one. Consider the Voronoi diagrams of the two sequences. Because the boundaries of the two Voronoi diagrams do not coincide exactly with each other, the similar Voronoi region, as indicated by the shaded area, does not occupy the entire feature space. As the general form of

VVS defined in Equation (3.10) is the weighted volume of the similar Voronoi region, it is strictly less than the IVS. The difference between the two similarities is due to the unshaded region in Figure 3.6. A large unshaded region leads to a large difference between the two similarities. If a seed vector s falls within the unshaded region in Figure 3.6, we can make two observations about the corresponding signature frames $g_X(s)$ and $g_Y(s)$ from the two sequences X and Y: (1) they are far apart from each other, i.e. $d(g_X(s), g_Y(s)) > \epsilon$; (2) they both have similar vectors in the other video, i.e. there exists $x \in X$ and $y \in Y$ such that $d(x, g_X(s)) \leq \epsilon$ and $d(y, g_Y(s)) \leq \epsilon$. These observations define a unique characteristic of a particular region, which we refer to as a Voronoi Gap. Intuitively, any seed vector in a Voronoi gap between two sequences produces a pair of dissimilar signature vectors, even though both signature vectors have a similar match in the other video. More formally, we define the Voronoi gap as follows:

Definition 3.5.1 Voronoi Gap

Let X and Y be two video sequences. The Voronoi gap $G(X, Y; \epsilon)$ of X and Y is defined by all $s \in F$ that satisfy the following criteria:

1. $d(g_X(s), g_Y(s)) > \epsilon$,
2. *there exists* $x \in X$ *such that* $d(x, g_Y(s)) \leq \epsilon$,
3. *there exists* $y \in Y$ *such that* $d(y, g_X(s)) \leq \epsilon$.

The example in Figure 3.6 seems to suggest that the Voronoi gap is small if ϵ is small. An important question is how small ϵ must be before we can ignore the contribution of the Voronoi gap. In order to have a rough idea on how ϵ affects the volume of a Voronoi gap, we present here a simple example using the h-dimensional hamming cube as our feature vector space. A hamming cube is a set containing all the h-bit binary numbers. The distance between two vectors is simply the number of bit-flips to change the binary representation of one vector to the other. Since it is a finite space, the volume function is simply the cardinality of the subset divided by 2^h. We choose the hamming cube because it is easy to analyze, and some commonly used metrics such as l_1 and l_2 can be easily embedded inside the hamming cube with little distortion [69].

To simplify the calculations, we only consider video sequences with two vectors in the h-dimensional hamming cube H. Let $X = \{x_1, x_2\}$ be a video in H. Let the distance between x_1 and x_2 be a positive integer k. We assume the two vectors in X are not similar, i.e. the distance between them is much larger than ϵ. In particular, we assume that $k > 2\epsilon$. We want to compute the "gap volume", i.e. the probability of choosing a seed vector s that is inside the Voronoi gap formed between X and some video sequence in H. Based on the definition of Voronoi gap, if a video sequence Y has a non-empty Voronoi gap with X, Y must have a vector similar to each vector in X. In other words, the IVS between X and Y must be one. Let Γ be the set of all two-vector sequences whose IVS with X is one. The gap volume is thus the volume

of the union of the Voronoi gap formed between X and each video in Γ. As shown by the following proposition, this gap probability can be calculated using the binomial distribution.

Proposition 3.5.1 *Let $X = \{x_1, x_2\}$ be a two-vector video sequence in the Hamming cube H, and Γ be the set of all two-vector sequences whose IVS with X is one. Define A to be the union of the Voronoi gap formed between X and every video in Γ, i.e.*

$$A \triangleq \bigcup_{Y \in \Gamma} G(X, Y; \epsilon).$$

Then, if $k = d(x_1, x_2)$ is an even number larger than 2ϵ, the volume of A can be computed as follows:

$$\begin{aligned} \mathrm{Vol}(A) &= \mathrm{Prob}(k/2 - \epsilon \leq R < k/2 + \epsilon) \\ &= \frac{1}{2^k} \sum_{r=k/2-\epsilon}^{k/2+\epsilon-1} \binom{k}{r} \end{aligned} \tag{3.12}$$

where R is a random variable that follows a binomial distribution with parameters k and 1/2.

We compute $\mathrm{Vol}(A)$ numerically by using the right hand side of Equation (3.12). The resulting plot of $\mathrm{Vol}(A)$ versus the distance k between the vectors in X for $\epsilon = 1, 5, 10$ is shown in Figure 3.7. $\mathrm{Vol}(A)$ decreases as k increases and as ϵ decreases, but it is hardly insignificant even when k is substantially larger than ϵ. For example, at $k = 500$ and $\epsilon = 5$, $\mathrm{Vol}(A) \approx 0.34$. It is unclear whether the same phenomenon occurs for other feature spaces. Nonetheless, rather than assuming that all Voronoi gaps are

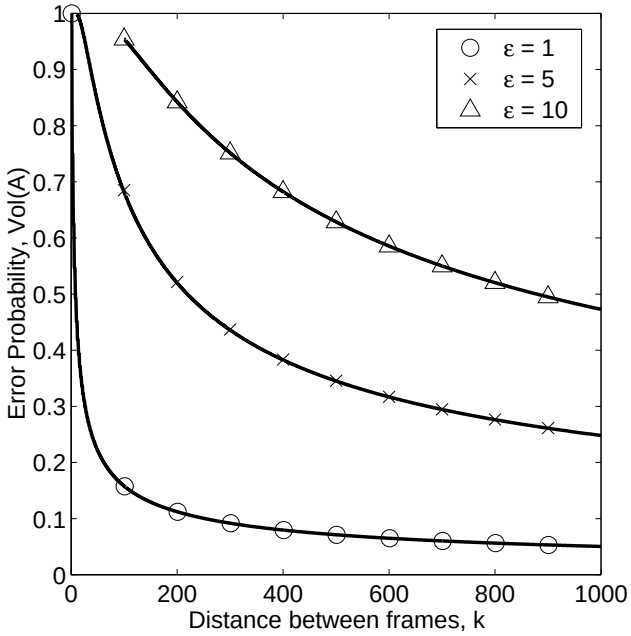


Figure 3.7: *The error probability for the hamming cube at different values of ϵ and distances k between the vectors in the video.*

insignificant, it makes sense to discard seed vectors that are inside the Voronoi gap when using the ViSig method to estimate video similarity.

3.6 Ranked ViSig method

Consider again the example in Figure 3.6. Assume that we generate m random seed vectors to compute VSS_b. If n out of m seed vectors are inside the unshaded Voronoi gap, we can reject these n seed vectors and use the remaining $(m - n)$ seed vectors for the computation. The resulting VSS_b exactly matches IVS in this example. The only caveat in this approach is that we need an efficient algorithm to determine whether a seed vector is inside the Voronoi gap. Direct application of Definition 3.5.1

is not practical, because conditions (2) and (3) in the definition require computing the distances between each signature vector of one video and all the vectors in the other video. Not only is the time complexity of comparing two signatures significantly larger than the time to compute VSS_b, it defeats the very purpose of using a compact signature to represent a video. A more efficient algorithm is needed to check if a seed vector is inside the Voronoi gap.

The remainder of this section proposes an algorithm that can be applied after generating the signature to identify those seed vectors which are more likely to be inside the Voronoi gap. In Figure 3.6, we observe that the two sequences have a pair of dissimilar vectors that are roughly equidistant from an arbitrary vector s in the Voronoi gap: x and $g_X(s)$ in the "dot" sequence, and y and $g_Y(s)$ in the "cross" sequence. They are not similar as both $d(x, g_X(s))$ and $d(y, g_Y(s))$ are clearly larger than ϵ. Intuitively, since vectors such as s inside the Voronoi gap are close to the boundaries between Voronoi cells in both sequences, it is not surprising to find dissimilar vectors such as x and $g_X(s)$ that are on either side of the boundaries to be roughly equidistant to s. This "equidistant" condition is refined in the following proposition to upper-bounding the difference between distance of s and x, and distance of s and $g_X(s)$ by 2ϵ:

Proposition 3.6.1 *Let X and Y be two video sequences. Assume all clusters in $[X \cup Y]_\epsilon$ are ϵ-compact. If a seed vector $s \in G(X, Y; \epsilon)$, there exists a vector $x \in X$ such that*

1. *x is not similar to $g_X(s)$, i.e. $d(x, g_X(s)) > \epsilon$.*

2. *x and $g_X(s)$ are roughly equidistant to s. Specifically, $d(x, s) - d(g_X(s), s) \leq 2\epsilon$.*

Similarly, we can find a $y \in Y$ that share the same properties with $g_Y(s)$.

The significance of Proposition 3.6.1 is that it provides a test for determining whether a seed vector s can ever be inside the Voronoi gap between a particular video X and any other arbitrary sequence. Specifically, if there are no vectors x in X such that x is dissimilar to $g_X(s)$ and $d(x, s)$ is within 2ϵ from $d(s, g_X(s))$, we can guarantee that s will never be inside the Voronoi gap formed between X and any other sequence. The condition that all similar clusters must be ϵ-compact is to avoid pathological chain-like clusters as discussed in Section 3.1. Such an assumption is not unrealistic for real-life video sequences.

To apply Proposition 3.6.1 in practice, we first define a Ranking Function $Q(\cdot)$ for the signature vector $g_X(s)$,

$$Q(g_X(s)) \stackrel{\Delta}{=} \min_{x \in X,\, d(x, g_X(s)) > \epsilon} d(x, s) - d(g_X(s), s). \tag{3.13}$$

An example of $Q(\cdot)$ as a function of a 2-D seed vector s is shown as a contour plot in Figure 3.8. The three crosses represent the vectors of a video. Lighter color regions correspond to the area with larger a $Q(\cdot)$ values, and thus farther away from the boundaries between Voronoi cells. By Proposition 3.6.1, if $Q(g_X(s)) > 2\epsilon$, s cannot be inside the Voronoi gap formed between X and any other sequence. In practice, however, this condition might be too restrictive in that it might not allow us to

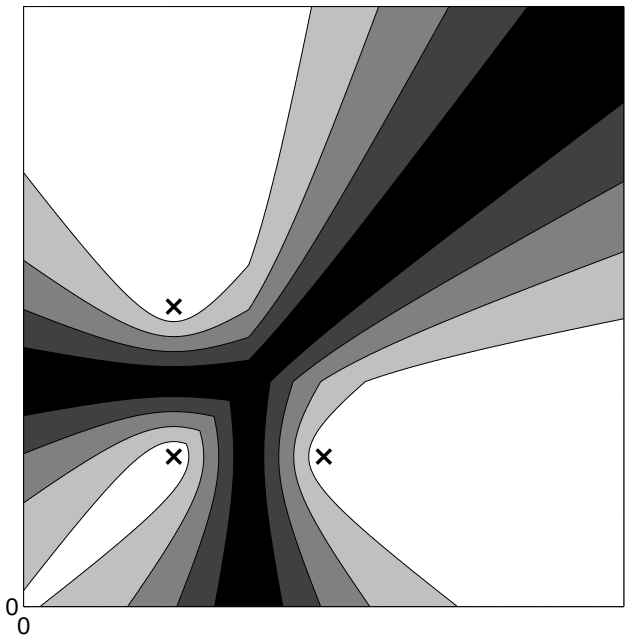


Figure 3.8: *Values of ranking function $Q(\cdot)$ for a three-vector video sequence. Lighter colors correspond to larger values.*

find any seed vector. Recall that Proposition 3.6.1 only provides a sufficient but not a necessary condition for a seed vector to be inside Voronoi gap. Thus, even if $Q(g_X(s)) \leq 2\epsilon$, it does not necessarily imply that s will be inside the Voronoi gap between X and a particular sequence.

Intuitively, in order to minimize the chances of being inside any Voronoi gap, it makes sense to use a seed vector s with as large $Q(g_X(s))$ as possible. As a result, rather than using only the signature vectors with $Q(g_X(s)) > 2\epsilon$, we generate a large number of signature vectors, and use the few with the largest $Q(g_X(s))$ for similarity measurements. Let $m' > m$ be the number of vectors in each signature. After we generate X_S by using a set S of m' seed vectors, we compute and rank $Q(g_X(s))$ for

all $g_X(s)$'s in X_S. Analogous to VSS_b defined in Equation (3.6), we define the Ranked ViSig Similarity (VSS_r) between the top-ranked signature vectors of X_S and Y_S as follows:

$$\begin{aligned}\text{vss}_r(X_S, Y_S; \epsilon, m) \triangleq \frac{1}{m}\Big(&\sum_{i=1}^{\lfloor m/2 \rfloor} 1_{\{d(g_X(s_{j[i]}), g_Y(s_{j[i]})) \le \epsilon\}} + \\ &\sum_{i=1}^{\lfloor m/2 \rfloor} 1_{\{d(g_X(s_{k[i]}), g_Y(s_{k[i]})) \le \epsilon\}}\Big)\end{aligned} \tag{3.14}$$

$j[1], \ldots, j[m']$ and $k[1], \ldots, k[m']$'s denote the rankings of the signature vectors in X_S and Y_S respectively, i.e. $Q(g_X(s_{j[1]})) \ge \ldots \ge Q(g_X(s_{j[m']}))$ and $Q(g_Y(s_{k[1]})) \ge \ldots \ge Q(g_Y(s_{k[m']}))$. We call this method of generating signature and computing VSS_r the Ranked ViSig method. Notice that in the right hand side of Equation (3.14), the first term uses the top-ranked $\lfloor m/2 \rfloor$ signature vectors from X_S to compare with the corresponding signature vectors in Y_S, and the second term uses the top-ranked $\lfloor m/2 \rfloor$ vectors from Y_S. Computing VSS_r thus requires m metric operations, the same as the basic version in Equation (3.6). Alternatively, we can use only the ranking of one signature, say X_S, and compute the asymmetric VSS_r as follows:

$$\widehat{\text{vss}_r}(X_S, Y_S; \epsilon, m) \triangleq \frac{1}{m} \sum_{i=1}^{m} 1_{\{d(g_X(s_{j[i]}), g_Y(s_{j[i]}))} \tag{3.15}$$

As we will explain in Chapter 4, we are interested in this asymmetric form of signature similarity as it leads to a more efficient implementation of fast similarity search. Theoretically, the asymmetry in (3.15) may lead to bias in the measurement. For example, if one video is a sub-sequence of the other, using the ranking of the shorter video may result in a larger asymmetric VSS_r than using that of the longer one. In

Section 3.7.3, we show that there is little difference between the two versions of VSS_r's in terms of the retrieval performance of highly similar video sequences on the web.

3.7 Experimental results

This section presents experimental results to demonstrate the performance of both the basic and ranked ViSig method. All experiments use color histograms, described in Section 3.7.1, as feature vectors. Two sets of experimental results are shown in the remainder of the section. Results of a number of controlled simulations are presented in Section 3.7.2 to demonstrate the heuristics introduced for seed vector selection in Section 3.4, and for signature vector ranking in Section 3.6. We also apply the ViSig methods to a large set of real-life web video sequences, and measure retrieval performance with respect to the ground-truth set described earlier in Section 2.3. The experimental methodology and results are presented in Section 3.7.3.

3.7.1 Color histogram feature

In our experiments, we use four 178-bin color histograms on the Hue-Saturation-Value (HSV) color space to represent each individual feature vector in a video. A color histogram is one of the most commonly used image features in content-based retrieval systems. The quantization of the color space used in the histogram is shown in Figure 3.9. This quantization is similar to the one used in [70]. The saturation

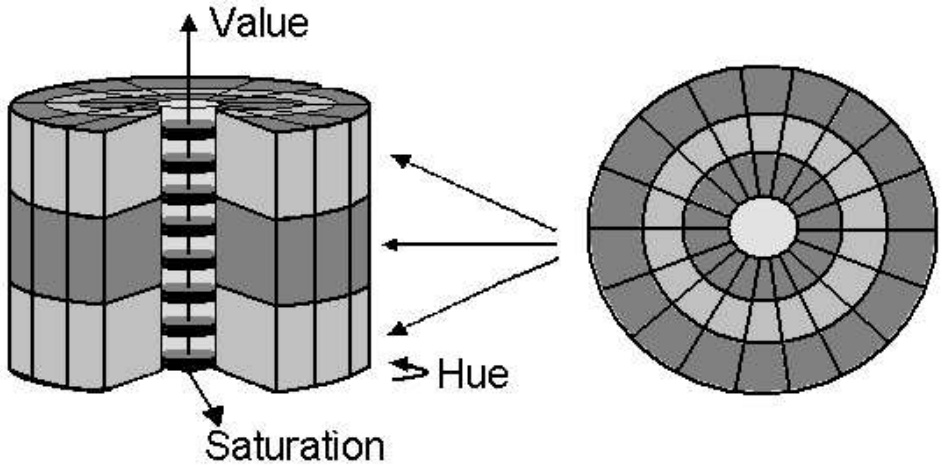


Figure 3.9: *Quantization of the HSV color space.*

(radial) dimension is uniformly quantized into 3.5 bins, with the half bin at the origin. The hue (angular) dimension is uniformly quantized at 20°-step size, resulting in 18 sectors. The quantization for the value dimension depends on the saturation value. For those colors with the saturation values near zero, a finer quantizer of 16 bins is used to better differentiate between gray-scale colors. For the rest of the color space, the value dimension is uniformly quantized into three bins. The histogram is normalized such that the sum of all the bins equals one. In order to incorporate spatial information into the image feature, the image is partitioned into four quadrants, with each quadrant having its own color histogram. As a result, the total dimension of a single feature vector becomes 712.

We use two distance measurements in comparing color histograms: the l_1 metric and a modified version of the l_1 distance with dominant color first removed. The

l_1 metric on color histogram was first used in [71] for image retrieval. It is defined by the sum of the absolute difference between each bin of the two histograms. We denote the l_1 metric between two feature vectors x and y as $d_c(x, y)$, with its precise definition stated below:

$$d_c(x,y) \triangleq \sum_{i=1}^{4} d_c^q(x_i, y_i) \text{ where } d_c^q(x_i, y_i) \triangleq \sum_{j=1}^{178} |x_i[j] - y_i[j]| \tag{3.16}$$

where x_i and y_i for $i \in \{1, 2, 3, 4\}$ represent the quadrant color histograms from the two feature vectors. A small $d_c(\cdot, \cdot)$ value usually indicates visual similarity, except when two images share the same background color. In those cases, the metric $d_c(\cdot, \cdot)$ does not correctly reflect the differences in the foreground as it is overwhelmed by the dominant background color. Such scenarios are quite common among the videos found on the web. Examples include those video sequences composed of presentation slides or graphical plots in scientific experiments. To mitigate this problem, we develop a new distance measurement which first removes the dominant color, then computes the l_1 metric for the rest of the color bins, and finally re-normalizes the result to the proper dynamic range. Specifically, this new distance measurement $\widehat{d}_c(x, y)$ between two feature vectors x and y can be defined as follows:

$$\widehat{d}_c(x,y) \triangleq \sum_{i=1}^{4} \widehat{d}_c^q(x_i, y_i)$$

$$\text{where } \widehat{d}_c^q(x_i, y_i) \triangleq \begin{cases} \frac{2}{2 - x_i[c] - y_i[c]} \sum_{j=1, j\neq c}^{178} |x_i[j] - y_i[j]| & \text{if } x_i[c] > \rho \text{ and } y_i[c] > \rho \\ \sum_{j=1}^{178} |x_i[j] - y_i[j]| & \text{otherwise.} \end{cases} \tag{3.17}$$

In Equation (3.17), the dominant color is defined to be the color c with bin value exceeding the Dominant Color Threshold ρ. ρ has to be larger than or equal to 0.5 to guarantee a single dominant color. We set $\rho = 0.5$ in our experiments. When the two feature vectors share no common dominant color, $\widehat{d_c}(\cdot,\cdot)$ reduces to $d_c(\cdot,\cdot)$.

Even though $\widehat{d_c}$ and d_c are closely related to each other, unlike d_c, $\widehat{d_c}$ is not a metric in the mathematical sense. Specifically, it does not satisfy the triangle inequality. The use of a metric space is one of the key assumptions behind the ViSig method and the fast similarity search schemes described in Chapter 4. As such, $\widehat{d_c}$ is designed in such a way that it satisfies the following proposition:

Proposition 3.7.1 *Suppose that the Dominant Color Threshold ρ is larger than 0.5 in the definition of $\widehat{d_c}(\cdot,\cdot)$ in Equation (3.17). The following inequality holds for an arbitrary pair of color histogram feature vectors x and y:*

$$\widehat{d_c}(x,y) \geq d_c(x,y) \tag{3.18}$$

In a similarity search, we are interested in finding the set of similar feature vectors whose distances with the query are within some $\epsilon > 0$. Inequality (3.18) implies that the set of similar feature vectors identified by $\widehat{d_c}$ must be a proper subset of the set identified by d_c. Thus, we can treat the use of $\widehat{d_c}$ as a post-processing step to refine the results of the similarity search obtained via d_c. For the rest of the paper, we adopt this model, and develop the theory of similarity search by assuming the use of

a true metric.

3.7.2 Simulation results

In this section, we present experimental results to verify the heuristics proposed in Sections 3.4 and 3.6. The first experiment demonstrates the effect of seed vectors on approximating the IVS by the basic ViSig method. We perform the experiment on a set of 15 video sequences selected from the MPEG-7 video data set [72][3]. This set includes a wide variety of video content including documentaries, cartoons, and television drama, etc. The average length of the test sequences is 30 minutes. We randomly drop frames from each sequence to artificially create similar versions at different levels of IVS. Signatures with respect to two different sets of seed vectors are created for all the sequences and their similar versions. The first set of seed vectors are independent random vectors, uniformly distributed on the high-dimensional histogram space. To generate such random vectors, we follow the algorithm described in [73]. The second set of seed vectors are randomly selected from a set of images in the Corel Stock Photo Collection. These images represent a diverse set of real-life images, and thus provide a reasonably good approximation to the feature vector distribution of the test sequences. We randomly choose around 4000 images from the Corel collection, and generate the required seed vectors using the seed vector generation method,

[3]The test set includes video sequences from MPEG-7 video CD's v1, v3, v4, v5, v6, v7, v8, and v9. We denote each test sequence by the CD they are in, followed by a number such as v8_1 if there are multiple sequences in the same CD.

with ϵ_{sv} set to 2.0, as described in Section 3.4. At IVS levels of 0.8, 0.6, 0.4 and 0.2, Table 3.1 contains the measured VSS_b for each test sequence and its similar version using $m = 100$ seed vectors. A good VSS_b should be close to the IVS value in the second row under the same column of the table.

Seed Vectors	Uniform Random				Corel Images			
IVS	0.8	0.6	0.4	0.2	0.8	0.6	0.4	0.2
v1_1	0.59	0.37	0.49	0.20	0.85	0.50	0.49	0.23
v1_2	0.56	0.38	0.31	0.05	0.82	0.63	0.41	0.18
v3	0.96	0.09	0.06	0.02	0.82	0.52	0.40	0.21
v4	0.82	0.75	0.55	0.24	0.92	0.44	0.48	0.25
v5_1	0.99	0.71	0.28	0.18	0.76	0.66	0.39	0.12
v5_2	0.84	0.35	0.17	0.29	0.81	0.68	0.36	0.10
v5_3	0.97	0.36	0.74	0.07	0.76	0.59	0.51	0.15
v6	1.00	0.00	0.00	0.00	0.79	0.61	0.46	0.25
v7	0.95	0.89	0.95	0.60	0.86	0.60	0.49	0.16
v8_1	0.72	0.70	0.47	0.17	0.88	0.69	0.38	0.20
v8_2	1.00	0.15	0.91	0.01	0.86	0.53	0.35	0.21
v9_1	0.95	0.85	0.54	0.15	0.93	0.56	0.44	0.18
v9_2	0.85	0.70	0.67	0.41	0.86	0.56	0.39	0.17
v9_3	0.90	0.51	0.30	0.10	0.78	0.70	0.45	0.15
v9_4	1.00	0.67	0.00	0.00	0.72	0.45	0.42	0.24
Average	0.873	0.499	0.429	0.166	0.828	0.581	0.428	0.187
Stddev	0.146	0.281	0.306	0.169	0.060	0.083	0.051	0.046

Table 3.1: *Comparison between using uniform random and corel image seed vectors. The second through fifth columns are the results of using uniform random seed vectors and the rest are the corel image seed vectors. Each row contains the results of a specific test video at IVS levels 0.8, 0.6, 0.4 and 0.2. The last two rows are the averages and standard deviations over all the test sequences.*

As shown in Table 3.1, VSS_b based on Corel images are slightly closer to the underlying IVS than those based on random vectors. More importantly, the fluctuations in the estimates, as indicated by the standard deviations, are far smaller with the Corel

images. This experiment shows that we can obtain a more consistent estimation of IVS by using seed vectors that resemble the target data than using uniformly random ones.

In the second experiment, we compare the ranked ViSig method with the basic ViSig method in approximating IVS under the presence of small feature vector displacements. As described in Section 3.5, when two vectors from two video sequences are separated by a small ϵ, the basic ViSig method may underestimate IVS due to the Voronoi gap region. To combat such a problem, we propose the ranked ViSig method in Section 3.5. In this experiment, we create similar video by adding noise to individual frames. Most of the real-life noise processes such as compression are highly video dependent, and cannot provide a wide-range of controlled noise levels for our experiment. As such, we introduce artificial noise that directly corresponds to the different noise levels as measured by our color histogram metric. As shown in [71], the $d_c^q(\cdot,\cdot)$ metric defined in Equation (3.16), is equal to twice the percentage of the pixels between two images that are of different colors. For example, if the l_1 metric between two histograms is 0.4, it implies that 20% of the pixels in the corresponding images have different colors. Thus, to inject a particular ϵ noise level to a feature vector, we determine the fraction of the pixels that need to have different colors and randomly assign colors to them. The color assignment is performed in such a way that ϵ noise level is achieved exactly.

Five ϵ levels are tested in our experiments: 0.2, 0.4, 0.8, 1.2 and 1.6. As every

feature vector contains four color histograms, an ϵ of 1.6, results in an average noise level of 0.4 for each histogram. No frames are dropped, so the IVS between the original sequence and the similar version is always one. A basic signature with $m = 100$ and a ranked signature with $m' = 500$ are generated for each pair of video sequences. All seed vectors are randomly sampled from the Corel dataset. To ensure the same computational complexity between the two methods, the top $m/2 = 50$ ranked signature vectors are used in computing VSS_r. The results are shown in Table 3.2. The averages and standard deviations over the entire set are shown in the last

Algorithm	VSS_b					VSS_r				
ϵ	0.2	0.4	0.8	1.2	1.6	0.2	0.4	0.8	1.2	1.6
v1_1	0.89	0.76	0.62	0.54	0.36	1.00	1.00	0.90	0.87	0.74
v1_2	0.81	0.73	0.55	0.47	0.34	1.00	0.98	0.83	0.73	0.62
v3	0.90	0.76	0.70	0.42	0.36	1.00	1.00	0.96	0.87	0.72
v4	0.86	0.74	0.64	0.48	0.38	1.00	1.00	0.96	0.83	0.74
v5_1	0.90	0.77	0.64	0.45	0.41	1.00	1.00	0.98	0.79	0.86
v5_2	0.96	0.81	0.52	0.66	0.56	1.00	1.00	1.00	0.86	0.78
v5_3	0.88	0.83	0.59	0.42	0.39	1.00	1.00	0.90	0.83	0.74
v6	0.88	0.72	0.64	0.49	0.49	1.00	1.00	0.98	0.92	0.78
v7	0.89	0.84	0.68	0.46	0.43	1.00	1.00	1.00	0.91	0.78
v8_1	0.85	0.67	0.58	0.52	0.30	1.00	1.00	0.87	0.79	0.73
v8_2	0.90	0.80	0.72	0.59	0.56	1.00	1.00	0.99	0.95	0.86
v9_1	0.87	0.77	0.62	0.67	0.48	1.00	0.99	0.89	0.84	0.82
v9_2	0.82	0.70	0.55	0.50	0.37	1.00	1.00	0.90	0.78	0.59
v9_3	0.86	0.65	0.66	0.49	0.40	1.00	1.00	0.91	0.70	0.58
v9_4	0.92	0.86	0.71	0.61	0.53	1.00	1.00	0.93	0.89	0.82
Average	0.879	0.761	0.628	0.518	0.424	1.000	0.998	0.933	0.837	0.744
Stddev	0.038	0.061	0.061	0.080	0.082	0.000	0.006	0.052	0.070	0.088

Table 3.2: *Comparison between* VSS_b *and* VSS_r *under different levels of perturbation. The table follows the same format as in Table 3.1. The perturbation levels* ϵ *tested are 0.2, 0.4, 0.8, 1.2 and 1.6.*

two rows. Because the IVS is fixed at one, the approximation is better if the measured similarity is closer to one. The amount of error increases for both methods as the noise level increases. Nevertheless, the VSS_r measurements are significantly closer to one than VSS_b. For high levels of IVS and small levels of perturbation, this experiment demonstrates that the ranked ViSig method provides much better estimation of IVS than the basic version.

3.7.3 Ground-truth results

Besides the simulation results presented in the previous section, we also measure the performance of the ViSig method based on how well it can identify the ground-truth set of similar video clips described earlier in Section 2.3. When using the ViSig method to identify similar video sequences, we declare two sequences to be similar if their VSS_b or VSS_r is larger than a certain threshold $\lambda \in [0, 1]$. In the experiments, we fix λ at 0.5 and report the retrieval results for different numbers of signature vectors, m, and the frame similarity threshold, ϵ. Our choice of fixing λ at 0.5 is based on the following reasoning: as the dataset is composed of extremely heterogeneous contents, it is rare to find partially similar video sequences. We notice that most video sequences in our dataset are either very similar to each other, or not similar at all. If ϵ is appropriately chosen to match subjective similarity, and m is large enough to keep sampling error small, we would expect the VSS for an arbitrary pair of signatures to be close to either one or zero, corresponding to either similar or

dissimilar video sequences in the dataset. We thus fix λ at 0.5 to balance the possible false-positive and false-negative errors, and vary ϵ to trace the whole spectrum of retrieval performance.

To accommodate such a testing strategy, we make a minor modification in the ranked ViSig method: recall that we use the ranking function $Q(\cdot)$ as defined in Equation (3.13) to rank all vectors in a signature. Since $Q(\cdot)$ depends on ϵ and its computation requires the entire video sequence, it is cumbersome to recompute it whenever a different ϵ is used. ϵ is used in the $Q(\cdot)$ function to identify the clustering structure within a single video. Since most video sequences are compactly clustered, we notice that their $Q(\cdot)$ values remain roughly constant for a large range of ϵ. As a result, we a priori fix ϵ to be 2.0 to compute $Q(\cdot)$, and do not recompute them even when we modify ϵ to obtain different retrieval results.

The performance measurements used in our experiments are recall and precision as defined below. Let Λ be the web video dataset and Φ be the ground-truth set. For a video $X \in \Phi$, we define the Relevant Set to X, $\mathrm{rel}(X)$, to be the ground-truth cluster that contains X, minus X itself. We also define the Return Set to X, $\mathrm{ret}(X, \epsilon)$, as the set of video sequences in the database which are declared to be similar to X by the ViSig method, i.e. $\mathrm{ret}(X, \epsilon) \stackrel{\Delta}{=} \{Y \in \Lambda : \mathrm{vss}(X_S, Y_S; \epsilon) \geq 0.5\} \setminus \{X\}$. $\mathrm{vss}(\cdot, \cdot)$ can be either VSS_b or VSS_r. By comparing the return and relevant sets of all the video

sequences in the ground-truth set, we define Recall and Precision as follows:

$$\begin{aligned} \text{Recall}(\epsilon) &\triangleq \frac{\sum_{X\in\Phi} |\text{rel}(X) \cap \text{ret}(X,\epsilon)|}{\sum_{X\in\Phi} |\text{rel}(X)|} \\ \text{Precision}(\epsilon) &\triangleq \frac{\sum_{X\in\Phi} |\text{rel}(X) \cap \text{ret}(X,\epsilon)|}{\sum_{X\in\Phi} |\text{ret}(X,\epsilon)|}. \end{aligned} \tag{3.19}$$

Thus, recall computes the fraction of all ground-truth video sequences that can be retrieved by the algorithm. Precision measures the fraction retrieved by the algorithm that are relevant. By varying ϵ, we can measure the retrieval performance of the ViSig methods for a wide range of recall values.

The goal of the first experiment is to compare the retrieval performance between the basic and the ranked ViSig methods at different signature sizes. $\widehat{d_c}$ distance, defined in (3.17), is used in this experiment. Seed vectors are randomly selected by the seed vector generation method, with ϵ_{sv} set to 2.0, from a set of keyframes representing the video sequences in the dataset. These keyframes are extracted by the AltaVista search engine and captured during data collection process. Each video is represented by a single keyframe. For the ranked ViSig method, $m' = 100$ keyframes are randomly selected from the keyframe set to produce the seed vector set which is used for all signature sizes, m. For each signature size in the basic ViSig method, we average the results of four independent sets of randomly selected keyframes in order to smooth out the statistical variation due to the limited signature sizes. The plots in Figure 3.10 show the precision versus recall curves for four different signature sizes: m = 2, 6, 10, and 14. The ranked ViSig method outperforms the basic ViSig method in all four cases. Figure 3.11 shows the ranked method's results across different signature

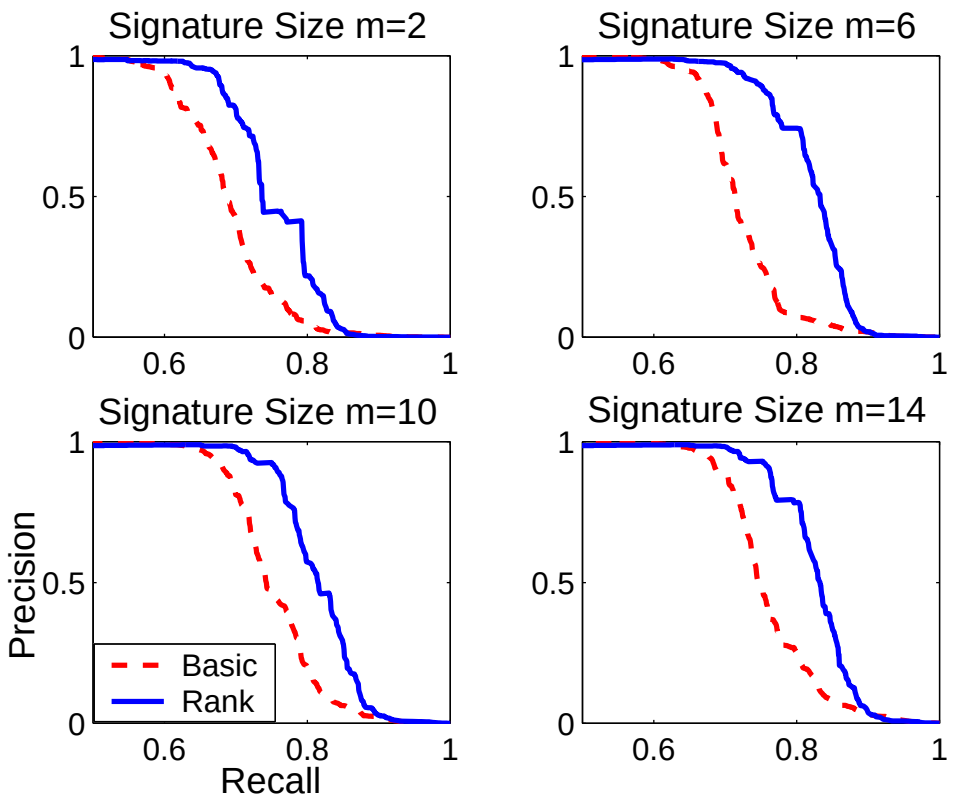


Figure 3.10: *Comparisons between the basic (broken-line) and ranked (solid) ViSig methods for four different signature sizes:* $m = 2, 6, 10, 14$.

sizes. As shown in the figure, there is a substantial gain in performance when m is increased from two to six. Further increase in m does not produce much gain. The precision-recall curves all decline sharply once they reach beyond 75% for recall and 90% for precision. Thus, we conclude that $m = 6$ is adequate for the ranked ViSig method to retrieve the ground-truth from the dataset.

In the second experiment, we test the difference between using the $\widehat{d}_c$ distance and the d_c metric on the color histogram. As described in Section 3.7.1, d_c metric represents a l_1 metric between the two color histograms, while $\widehat{d}_c$ distance removes the

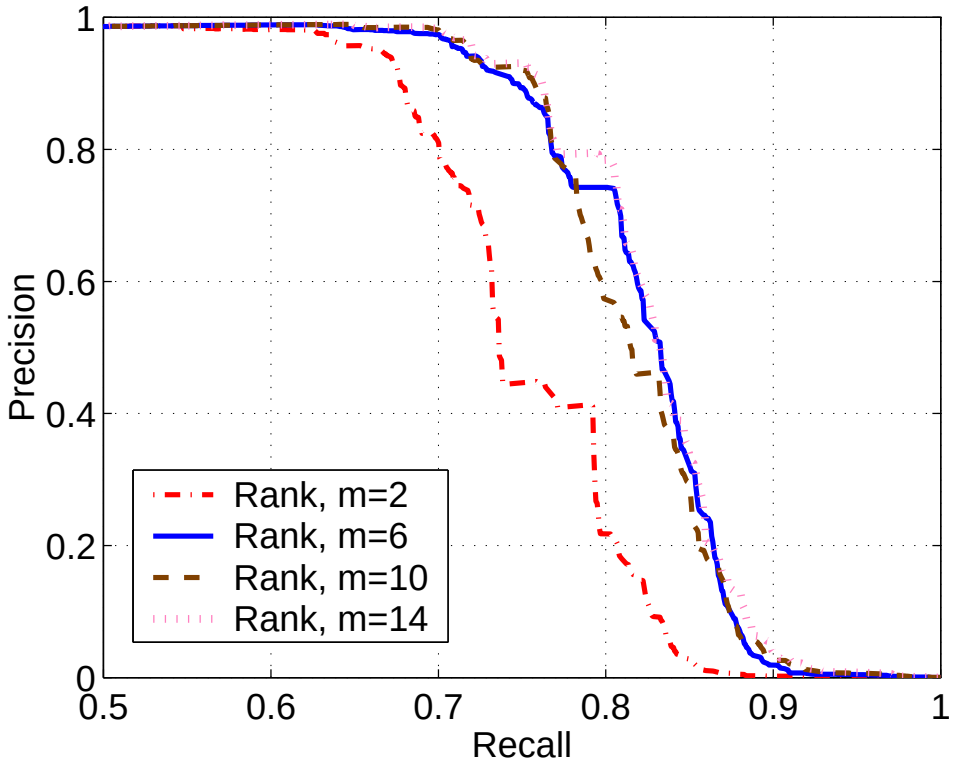


Figure 3.11: *Precision and recall performance for ranked ViSig method at* $m = 2, 6, 10, 14$.

shared dominant color before computing a l_1 metric. The same ranked ViSig method with $m = 6$ is used. Figure 3.12 shows that $\widehat{d}_c$ distance significantly outperforms the straightforward d_c metric.

In the third experiment, we compare the retrieval performance between k-medoid and the ranked ViSig method. As described in Chapter 2, k-medoid is a summarization technique that minimizes the distance between the original video and its summarization. Specifically, given a l-vector video $X = \{x_t : t = 1, \ldots, l\}$, the k-medoid of X is defined to be a set of k vectors $x_{t_1}, \ldots, x_{t_k}$ in X that minimize the

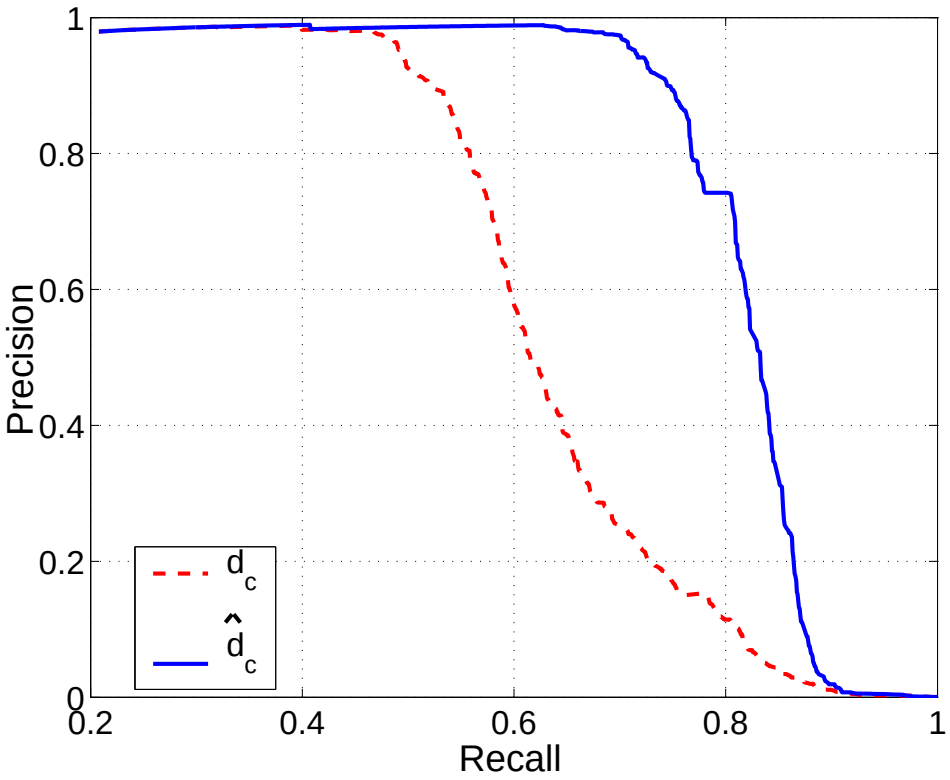


Figure 3.12: *Comparison between d_c metric (broken) and $\widehat{d_c}$ distance (solid).*

following cost function:

$$\sum_{t=1}^{l} \min_{j=1,\ldots,k} d(x_t, x_{t_j}) \tag{3.20}$$

Due to the large number of possible choices in selecting k vectors from a set, it is computationally impractical to precisely solve this minimization problem. In our experiment, we use the PAM algorithm proposed in [28] to compute a k-medoid with $k = 7$ for each video clip in our dataset. The PAM algorithm is iterative, and the time complexity for each iteration is on the order of l^2. Given $\epsilon > 0$, we declare two k-medoids to be similar if the shortest distance between vectors of the two k-medoids

is within ϵ. We plot the precision-recall curves for k-medoid and the ranked ViSig method with $m = 6$ in Figure 3.13. The k-medoid technique provides a slightly better retrieval performance. The advantage seems to be small considering the complexity advantage of the ViSig method over the PAM algorithm – First, computing VSS_r needs six metric computations but comparing two 7-medoid representations requires 49. Second, the ViSig method generates signatures in $O(l)$ time with l being the number of vectors in a video, while the PAM algorithm is an iterative $O(l^2)$ algorithm.

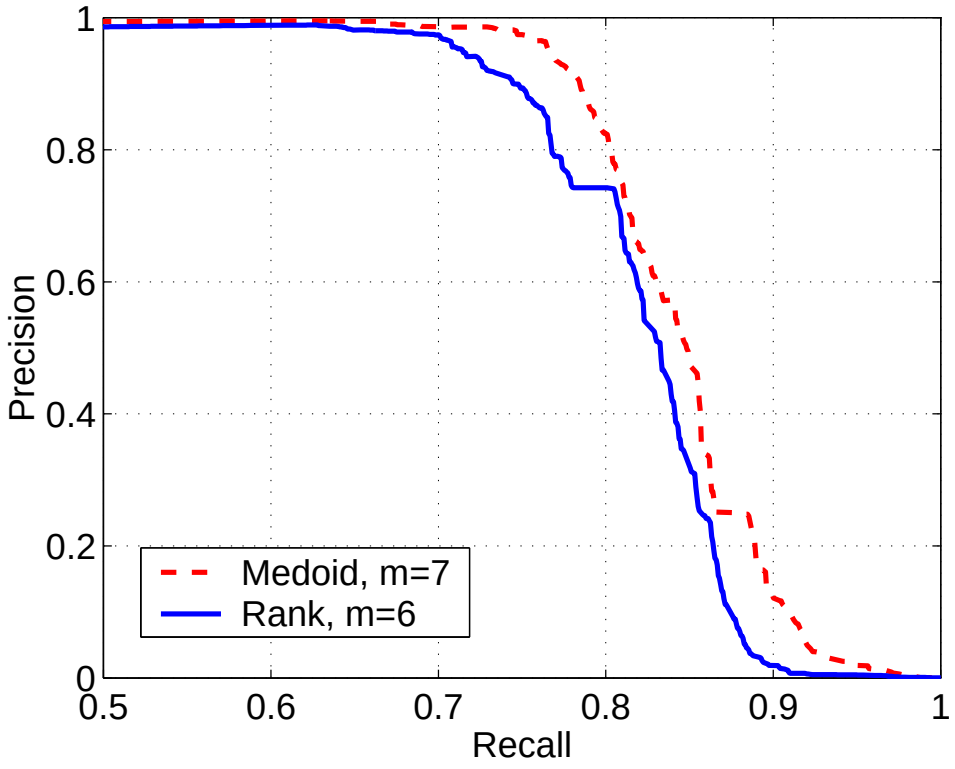


Figure 3.13: *Comparison between the Ranked ViSig method with $m = 6$ (solid) and k-medoid with 7 representative vectors (broken).*

Finally, we compare the retrieval performance between the symmetric VSS_r described in Equation (3.14), and the asymmetric VSS_r in Equation (3.15). We set $m' = 100$ and $m = 6$ for both similarity measures so that their computational complexities are identical. Figure 3.14 shows the precision-recall curves for the two schemes. Though not identical, the two measures give very similar retrieval performance in identifying the ground-truth set. On the other hand, the asymmetric version, as we will explain in Chapter 4, leads to a more efficient implementation of fast similarity search. Consequently, we focus primarily on this asymmetric similarity measurement between signatures for the remainder of this dissertation.

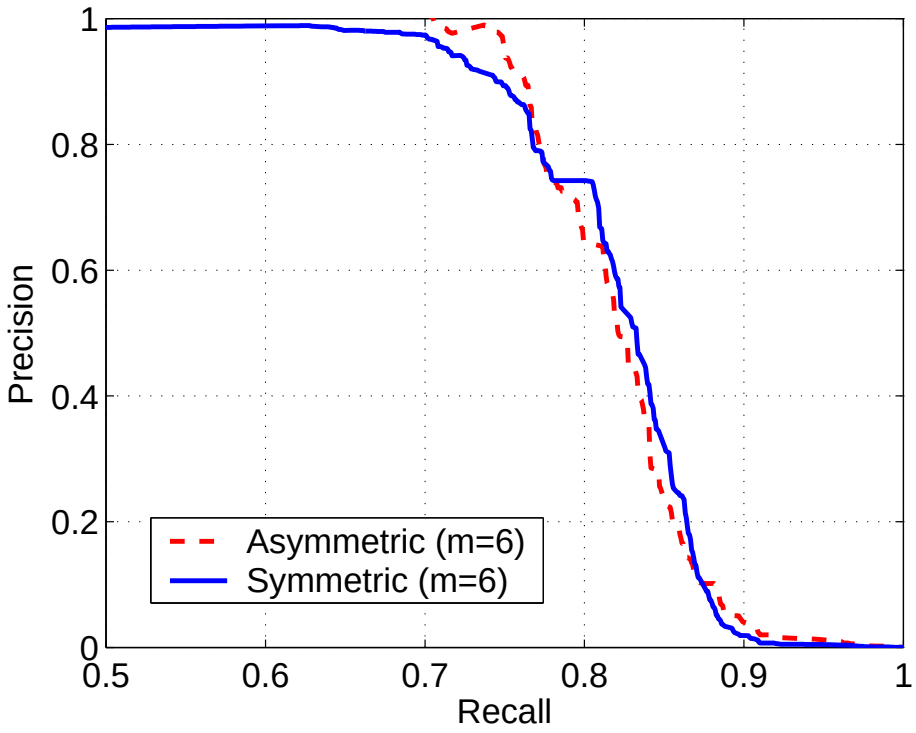


Figure 3.14: *Comparison between the symmetric and asymmetric* VSS_r *with* $m = 6$.

3.8 Summary

In this chapter, we have defined IVS as a video similarity measure for identifying similar web video sequences. Since IVS is complex to compute in practice, we have introduced an alternative measure called VVS, which can be efficiently estimated by the basic ViSig method. The basic ViSig method summarizes each video sequence into a small set of its frames, called a signature, that are closest to a set of random seed vectors. In applying the basic ViSig method to a large database, we have shown that the size of a signature depends on the desired fidelity of the measurements and the logarithm of the database size.

In practice, IVS and VVS can be quite different depending on individual video sequences. In order to reconcile the differences, we have extended the basic ViSig method in two directions. First, we have shown that the seed vectors used to generate signatures must resemble the statistics of the video sequences in the database. Second, we have proposed a ranking scheme to identify signature vectors that are most robust for similarity measurement. This new method of comparing signatures is called the ranked ViSig method. We have presented simulation results on a set of MPEG-7 test sequences to show the performances of these extensions. Lastly, we have also compared the retrieval performance of the two ViSig methods with the k-medoid scheme based on a groundtruth set from a large set of web video sequences.

3.A Appendix: Proofs of propositions

Proof of Proposition 3.1.1

Without loss of generality, let $X = \{x_1, x_2, \ldots, x_l\}$ and $Y = \{y_1, y_2, \ldots, y_l\}$ with $d(x_i, y_i) \leq \epsilon$ for $i = 1, \ldots, l$. Let Z_i be a binary random variable such that $Z_i = 1$ if both x_i and y_i are chosen as sampled frames, and 0 otherwise. Since W_m is the total number of similar pairs between the two set of sampled frames, it can be computed by summing all the Z_i's:

$$\begin{aligned} W_m &= \sum_{i=1}^{l} Z_i \\ E(W_m) &= \sum_{i=1}^{l} E(Z_i) = \sum_{i=1}^{l} \text{Prob}(Z_i = 1) \end{aligned}$$

Since we independently sample m frames from each sequence, the probability that $Z_i = 1$ for any i is $(m/l)^2$. This implies that $E(W_m) = m^2/l$. □

Proof of Proposition 3.3.1

To simplify the notation, let $\rho(X, Y) = \text{vvs}(X, Y; \epsilon)$ and $\hat{\rho}(X, Y) = \text{vss}_b(X_S, Y_S; \epsilon, m)$. For an arbitrary pair of X and Y, we can bound the probability of the event $|\rho(X, Y) - \hat{\rho}(X, Y)| > \gamma$ by the Hoeffding Inequality [74]:

$$\text{Prob}(|\rho(X, Y) - \hat{\rho}(X, Y)| > \gamma) \leq 2\exp(-2\gamma^2 m) \tag{3.A}$$

To find an upper bound for $P_{err}(m)$, we can combine (3.A) and the union bound as

follows:

$$\begin{aligned} P_{err}(m) &= \text{Prob}\Big(\bigcup_{X,Y \in \Lambda} |\rho(X,Y) - \hat{\rho}(X,Y)| > \gamma \Big) \\ &\leq \sum_{X,Y \in \Lambda} \text{Prob}(|\rho(X,Y) - \hat{\rho}(X,Y)| > \gamma) \\ &\leq \frac{n^2}{2} \cdot 2\exp(-2\gamma^2 m) \end{aligned}$$

A sufficient condition for $P_{err}(m) \leq \delta$ is thus

$$\begin{aligned} \frac{n^2}{2} \cdot 2\exp(-2\gamma^2 m) &\leq \delta \\ m &\geq \frac{2\ln n - \ln \delta}{2\gamma^2} \qquad \square \end{aligned}$$

Proof of Proposition 3.4.1

For each term inside the summation on the right hand side of Equation (3.11), $d(x,y)$ must be smaller than or equal to ϵ. If $d(x,y) \leq \epsilon$, our assumption implies that both x and y must be in the same cluster C belonging to both $[X]_\epsilon$ and $[Y]_\epsilon$. As a result, we can rewrite the right hand side of Equation (3.11) based only on clusters in $[X]_\epsilon \cap [Y]_\epsilon$:

$$\sum_{d(x,y) \leq \epsilon} \int_{V_X(x) \cap V_Y(y)} f(u; X \cup Y)\, du. = \sum_{C \in [X]_\epsilon \cap [Y]_\epsilon} \sum_{z \in C} \int_{V_X(z) \cap V_Y(z)} f(u; X \cup Y)\, du \quad \text{(3.B)}$$

Based on the definition of a voronoi cell, it is easy to see that $V_X(z) \cap V_Y(z) = V_{X \cup Y}(z)$ for all $z \in C$ with $C \in [X]_\epsilon \cap [Y]_\epsilon$. Substituting this relationship into Equation (3.B),

we obtain:

$$\begin{aligned}
\sum_{d(x,y)\le\epsilon} \int_{V_X(x)\cap V_Y(y)} f(u; X\cup Y)\, du &= \sum_{C\in[X]_\epsilon\cap[Y]_\epsilon} \int_{V_{X\cup Y}(C)} f(u; X\cup Y)\, du \\
&= \sum_{C\in[X]_\epsilon\cap[Y]_\epsilon} \int_{V_{X\cup Y}(C)} \frac{1}{|[X\cup Y]_\epsilon|\cdot \mathrm{Vol}(V_{X\cup Y}(C))}\, du \\
&= \frac{1}{|[X\cup Y]_\epsilon|} \sum_{C\in[X]_\epsilon\cap[Y]_\epsilon} \frac{\int_{V_{X\cup Y}(C)} du}{\mathrm{Vol}(V_{X\cup Y}(C))} \\
&= |[X]_\epsilon \cap [Y]_\epsilon| / |[X\cup Y]_\epsilon|
\end{aligned}$$

Finally, we note that $[X]_\epsilon \cap [Y]_\epsilon$ is in fact the set of all Similar Clusters in $[X\cup Y]_\epsilon$, and thus the last expression equals to the IVS. The reason is that for any Similar Cluster C in $[X\cup Y]_\epsilon$, C must have at least one $x\in X$ and one $y\in Y$ such that $d(x,y)\le\epsilon$. By our assumption, C must be in both $[X]_\epsilon$ and $[Y]_\epsilon$. □

Proof of Proposition 3.5.1

Without loss of generality, we assume that x_1 is at the origin with all zeros, and x_2 has k 1's in the rightmost positions. Clearly, $d(x_1,x_2)=k$. Throughout this proof, when we mention a particular sequence $Y\in\Gamma$, we adopt the convention that $Y=\{y_1,y_2\}$ with $d(x_1,y_1)\le\epsilon$ and $d(x_2,y_2)\le\epsilon$.

We first divide the region A into two partitions based on the proximity to the frames in X:

$$A_1 \stackrel{\Delta}{=} \{s\in A : g_X(s)=x_1\} \text{ and } A_2 \stackrel{\Delta}{=} \{s\in A : g_X(s)=x_2\}$$

We adopt the convention that if there are multiple frames in a video Z that are

equidistant to a random vector s, $g_Z(s)$ is defined to be the frame furthest away from the origin. This implies that all vectors equidistant to both frames in X are elements of A_2. Let s be an arbitrary vector in H, and R be the random variable that denotes the number of 1's in the rightmost k bit positions of s. The probability that R equals to a particular r with $r \leq k$ is as follows:

$$\text{Prob}(R = r) = \frac{1}{2^k}\binom{k}{r}$$

Thus, R follows a binomial distribution of parameters k and $1/2$. In this proof, we show the following relationship between A_2 and R:

$$\text{Vol}(A_2) = \text{Prob}(k/2 \leq R < k/2 + \epsilon) \tag{3.C}$$

With an almost identical argument, we can show the following:

$$\text{Vol}(A_1) = \text{Prob}(k/2 - \epsilon \leq R < k/2) \tag{3.D}$$

Since $\text{Vol}(A) = \text{Vol}(A_1) + \text{Vol}(A_2)$, the desired result follows.

To prove Equation (3.C), we first show if $k/2 \leq R < k/2 + \epsilon$, then $s \in A_2$. Be the definitions of A and A_2, we need to show two things: (1) $g_X(s) = x_2$; (2) there exists a $Y \in \Gamma$ such that $s \in G(X, Y; \epsilon)$, or equivalently, $g_Y(s) = y_1$. To show (1), we rewrite $R = k/2 + N$ where $0 \leq N < \epsilon$ and let the number of 1's in s be L. Consider the distances between s and x_1, and between s and x_2. Since x_1 is all zeros, $d(s, x_1) = L$. As x_2 has all its 1's in the rightmost k position, $d(s, x_2) = (L - R) + (k - R) =$

$L + k - 2R$. Thus,

$$\begin{aligned} d(s, x_1) - d(s, x_2) &= L - (L + k - 2R) \\ &= 2R - k \\ &= 2N \geq 0, \end{aligned}$$

which implies that $g_X(s) = x_2$. To show (2), we define y_1 to be a h-bit binary number with all zeros, except for ϵ 1's in the positions which are randomly chosen from the R 1's in the rightmost k bits of s. We can do that because $R \geq k/2 \geq \epsilon$. Clearly, $d(x_1, y_1) = \epsilon$ and $d(s, y_1) = L - \epsilon$. Next, we define y_2 by toggling ϵ out of k 1's in x_2. The positions we toggle are randomly chosen from the same R 1's bits in s. As a result, $d(x_2, y_2) = \epsilon$ and $d(s, y_2) = (L - R) + (k - R + \epsilon) = L + \epsilon - 2N$. Clearly, $Y \stackrel{\Delta}{=} \{y_1, y_2\}$ belongs to Γ. Since

$$\begin{aligned} d(s, y_2) - d(s, y_1) &= (L + \epsilon - 2N) - (L - \epsilon) \\ &= 2(\epsilon - N) > 0, \end{aligned}$$

$g_Y(s) = y_1$ and consequently, $s \in G(X, Y; \epsilon)$.

Now we show the other direction: if $s \in A_2$, then $k/2 \leq R < k/2 + \epsilon$. Since $s \in A_2$, we have $g_X(s) = x_2$ which implies that $L = d(s, x_1) \geq d(s, x_2) = L + k - 2R$ or $k/2 \leq R$. Also, there exists a $Y \in \Gamma$ with $s \in G(X, Y; \epsilon)$. This implies $g_Y(s) = y_1$, or equivalently, $d(s, y_1) < d(s, y_2)$. This inequality is strict as equality will force $g_Y(s) = y_2$ by the convention we adopt for $g_Y(\cdot)$. The terms on both sides of the inequality can be bounded using the triangle inequality: $d(s, y_1) \geq d(s, x_1) - d(x_1, y_1) = L - \epsilon$

and $d(s, y_2) \leq d(s, x_2) + d(x_2, y_2) = L + k - 2R + \epsilon$. Combining both bounds, we have

$$L - \epsilon < L + k - 2R + \epsilon \Rightarrow R < k/2 + \epsilon$$

This completes the proof for Equation (3.C). The proof of Equation (3.D) follows the same argument with the roles of x_1 and x_2 reversed. Combining the two equations, we obtain the desired result. $\square$

Proof of Proposition 3.6.1

We prove the case for video X and the proof is identical for Y. Since $s \in G(X, Y; \epsilon)$, we have $d(g_X(s), g_Y(s)) > \epsilon$ and there exists $x \in X$ with $d(x, g_Y(s)) \leq \epsilon$. Since all Similar Clusters in $[X \cup Y]_\epsilon$ are ϵ-compact, $g_X(s)$ cannot be in the same cluster with x and $g_Y(s)$. Thus, we have $d(g_X(s), x) > \epsilon$. It remains to show that $d(x, s) - d(g_X(s), s) \leq 2\epsilon$. Using the triangle inequality, we have

$$\begin{aligned} d(x, s) - d(g_X(s), s) &\leq d(x, g_Y(s)) + d(g_Y(s), s) - d(g_X(s), s) \\ &\leq \epsilon + d(g_Y(s), s) - d(g_X(s), s) \end{aligned} \tag{3.E}$$

$s \in G(X, Y; \epsilon)$ also implies that there exists $y \in Y$ such that $d(y, g_X(s)) \leq \epsilon$. By the definition of $g_Y(s)$, $d(g_Y(s), s) \leq d(y, s)$. Thus, we can replace $g_Y(s)$ with y in (3.E) and combine with the triangle inequality to obtain:

$$\begin{aligned} d(x, s) - d(g_X(s), s) &\leq \epsilon + d(y, s) - d(g_X(s), s) \\ &\leq \epsilon + d(y, g_X(s)) \\ &\leq 2\epsilon. \qquad \square \end{aligned}$$

Proof of Proposition 3.7.1

We want to show that the following relationship holds for any two color histogram feature vectors x and y,

$$d_c(x, y) \leq \widehat{d}_c(x, y)$$

It suffices to show that $d_c^q(x_i, y_i) \leq \widehat{d}_c^q(x_i, y_i)$ for $i = 1, 2, 3, 4$. These two quantities are identical when there is no common dominant color. When there is a dominant color at bin c, we have $x_i[c], y_i[c] \geq \rho \geq 0.5$. Without loss of generality, we assume that $y_i[c] \geq x_i[c]$.

$$\begin{aligned} d_c^q(x_i, y_i) &= \sum_j |x_i[j] - y_i[j]| \\ &= (1 - x_i[c]) - (1 - y_i[c]) + \sum_{j \neq c} |x_i[j] - y_i[j]| \\ &= \sum_{j \neq c} x_i[j] - \sum_{j \neq c} y_i[j] + \sum_{j \neq c} |x_i[j] - y_i[j]| \\ &\leq 2 \cdot \sum_{j \neq c} |x_i[j] - y_i[j]| \end{aligned}$$

As $\rho > 0.5$, the normalization factor $2/(2 - x_i[c] - y_i[c]) \geq 2/(2 - 0.5 - 0.5) = 2$. This implies

$$d_c^q(x_i, y_i) \leq \frac{2}{2 - x_i[c] - y_i[c]} \sum_{j \neq c} |x_i[j] - y_i[j]| = \widehat{d}_c^q(x_i, y_i),$$

and the result follows. □

Chapter 4

Fast similarity search on signatures

In this chapter, we consider the problem of developing fast similarity search algorithms for feature vectors in a metric space. We focus exclusively on metric-space data for two reasons: First, it is a very general similarity model, and many algorithms have been developed in the literature to address the metric-space similarity problem. Second, based on a simple procedure, it is straightforward to apply any metric-space algorithms to solve the similarity search problem for signature data. Given a query signature X_S, the goal of the similarity search is to identify all signatures in a large database whose signature similarities with X_S exceed a certain positive threshold λ. In the context of similarity search, we use the asymmetric VSS_r, as defined in Equation (3.15), to measure similarity between signatures. As shown in Section 3.7, asymmetric VSS_r produces similar retrieval performance as the symmetric version, and significantly outperforms the basic ViSig similarity. We opt for this asymmet-

ric definition because, by using the following procedure, it can reduce the signature similarity search problem into a metric-space similarity search problem:

Procedure 4.0.1 Signature Similarity Search

1. *Let* $X_S = (g_X(s_1), \ldots, g_X(s_{m'}))$ *be the query signature, and* $j[1], \ldots, j[m']$ *be the corresponding ranking.*

2. *For each of the* m *top-ranked signature vectors* $g_X(s_{j[i]})$*, identify those signature vectors* $g_Y(s_{j[i]})$ *in the database with* $d(g_X(s_{j[i]}), g_Y(s_{j[i]})) \leq \epsilon$*. This is a similarity search problem in the metric space* $(F, d(\cdot))$*.*

3. *Return all the signatures* Y_S *in the database which have at least* $\lfloor \lambda \cdot m \rfloor$ *of their signature vectors identified in the second step. This is equivalent to finding all the* Y_S *in the database with* $\widehat{\mathrm{vss}_r}(X_S, Y_S; \epsilon, m) \geq \lambda$*.*

In step two of the above procedure, by using only the ranking of X_S, we turn the signature similarity search problem into m similarity search problems in the metric space $(F, d(\cdot))$. This step is computationally intensive due to the large size of the database and the complexity of high-dimensional metric computation. The result of this step is a set of signatures from the database which share at least one similar signature vector with the query. Step three of Procedure 4.0.1 then searches this set for all those with more than $\lambda \cdot m$ similar signature vectors. This step is straightforward as there are no metric computations involved, and the set of signatures that survives

the second step is typically small. As a result, this chapter will focus on efficient algorithms to implement the second step of the procedure, which is similarity search on metric data. Unless specified otherwise, we follow the convention in Section 3.7.3 by assuming λ to be 0.5 and parameterizing our algorithms based only on ϵ.

This chapter is organized as follows. We first introduce a general framework called GEMINI in Section 4.1, on which we base our design of similarity search algorithms. We also describe how to measure the performance of specific fast search algorithms. The main step in GEMINI, called the feature extraction step, is to design a mapping from the metric space of feature vectors to a very low-dimensional space where similarity search can be carried out efficiently. We propose a novel feature extraction mapping for signature data in Sections 4.2 and 4.3. This mapping consists of two steps. First, each high-dimensional signature vector is mapped into a particular form of low-dimensional vector, which we refer to as a projection vector. The projection-vector mapping is described in detail in Section 4.2. Second, classical principal component analysis (PCA) is applied to the projection vector to further transform it into an index vector of even lower dimension, as specified by the user. Section 4.3 presents experimental results to compare our scheme with other techniques proposed in the literature.

4.1 The GEMINI approach for similarity search

Generic Multimedia Indexing, or GEMINI, is an approach for fast similarity search on data endowed with a metric function. Our description of GEMINI here is based on [9, ch.7]. Given a query vector x and a database D of feature vectors, we define the resulting set $A(x;\epsilon)$ of the similarity search on x as follows:

$$A(x;\epsilon) \stackrel{\Delta}{=} \{y \in D : d(x,y) \leq \epsilon\}. \tag{4.1}$$

Obviously, we can compute $A(x;\epsilon)$ by a sequential search on the database to identify those vectors that are within ϵ of x. The GEMINI approach, as outlined below, attempts to avoid the complexity of a sequential search by projecting the vectors into a low-dimensional metric space where similarity search can be performed efficiently:

Procedure 4.1.1 GEMINI

1. *Design a feature extraction mapping $\mathcal{T}$ which maps feature vectors from the metric space $(F, d(\cdot))$ to a very low dimensional metric space $(F', d'(\cdot))$. We call the vectors in F' the Range Vectors and $d'(\cdot)$ the Range Metric.*

2. *For every feature vector y in a database D, compute the corresponding range vector $\mathcal{T}(y)$ and store it in a SAM structure.*

3. *Given an input query feature vector x, first compute $\mathcal{T}(x)$, and then utilize the SAM structure computed in step 2 to perform a similarity search on $\mathcal{T}(x)$. The distance threshold used in this similarity search, which we refer to as Pruning*

Threshold and denote by ϵ', depends on both ϵ and $\mathcal{T}$. We will show how ϵ' is determined experimentally in Section 4.3. The set of feature vectors that correspond to the result of this similarity search is called the Candidate Set:

$$C(x;\epsilon') \triangleq \{y \in D : d'(\mathcal{T}(x), \mathcal{T}(y)) \leq \epsilon'\} \tag{4.2}$$

4. *It is possible that some feature vectors in $C(x;\epsilon')$ are far away from x. To complete the search, we sequentially compute the full metric function $d(\cdot)$ between x and each of the vectors in $C(x;\epsilon')$, and identify those that are truly within ϵ of x. We denote this resulting set as $A'(x;\epsilon,\epsilon')$:*

$$A'(x;\epsilon,\epsilon') \triangleq \{y \in C(x;\epsilon') : d(x,y) \leq \epsilon\}. \tag{4.3}$$

We can easily extend the GEMINI approach to handle similarity search on signatures by using Procedure 4.0.1 discussed in the beginning of this chapter. In the signature case, we denote the candidate set, the resulting set from sequential search, and the GEMINI resulting set as $C_S(X_S;\epsilon')$, $A_S(X_S;\epsilon)$, and $A'_S(X_S;\epsilon,\epsilon')$ respectively.

GEMINI solves the similarity search problem exactly if $A'_S(X_S;\epsilon,\epsilon')$ is identical to $A_S(X_S;\epsilon)$. GEMINI is more efficient than sequential search if the following two conditions hold:

1. The dimension of the range space is small. The dimension directly affects the speed of similarity search on range vectors in the following two aspects: first, a low-dimensional metric is typically faster to compute than the high-dimensional

one; and second, as shown in [31], we can expect a typical SAM structure to deliver faster-than-sequential search time if the dimension is below ten.

2. A typical candidate set is small enough so that few full metric computations are required in the last step of GEMINI.

In this dissertation, we assume the first condition holds, and do not delve into details of the design and implementation of any particular SAM structure, as it has been extensively studied elsewhere [29, 9, 30]. Instead, we focus on the design of a feature extraction mapping $\mathcal{T}$ to achieve the second condition, specifically, making the candidate set as small as possible. Based on the definition in Equation (4.2), a candidate set can be made arbitrarily small by decreasing the pruning threshold ϵ'. Nonetheless, ϵ' cannot be too small, otherwise most or all of the correct retrievals in $A_S(X_S; \epsilon)$ may be excluded. As a result, there is a trade-off between the complexity, as measured by the size of the candidate set, and the accuracy, as measured by the fraction of the correct retrievals retained. Specifically, we define two quantities, *Accuracy* and *Pruning*, to quantify this trade-off, and use them to evaluate the performances of different feature extraction mappings.

Let R be a typical set of query signatures. *Accuracy* is defined as the ratio between the sizes of the resulting sets obtained by GEMINI and by sequential search:

$$Accuracy(\epsilon, \epsilon') \triangleq \frac{\sum_{X_S \in R} |A'_S(X_S; \epsilon, \epsilon')|}{\sum_{X_S \in R} |A_S(X_S; \epsilon)|}, \tag{4.4}$$

where $|A|$ denotes the cardinality of set A. The dynamic range of *Accuracy* is from zero to one, with one corresponding to the perfect retrieval. The complexity is measured by *Pruning*, which is defined as the relative difference in the numbers of metric computations between GEMINI and sequential search:

$$Pruning(\epsilon') \triangleq \frac{|R| \cdot |D| - \sum_{X_S \in R} |C_S(X_S; \epsilon')|}{|R| \cdot |D|} \tag{4.5}$$

The dynamic range of *Pruning* is also between zero and one, with one corresponding to the maximum reduction in complexity. We can explain the numerator and denominator in Equation (4.5) as follows: the number of metric computations required by the sequential search is the product of the number of queries and the size of the database, i.e. $|R| \cdot |D|$. For GEMINI, we only count the number of metric computations performed on the candidate sets described in step 4 of Procedure 4.1.1, i.e. $\sum_{X_S \in R} |C_S(X_S; \epsilon')|$. We ignore the computational time of step 3, i.e. the similarity search on the range vectors, as we assume that it is independent of the choice of ϵ'. This assumption is certainly valid for sequential search on range vectors – no matter what value for ϵ' is chosen, the sequential search must compute the range metric between the query range vector and all range vectors in the database. On the other hand, the assumption might not necessarily hold for a particular SAM design. The interaction between SAM and our proposed feature extraction mapping is a topic that requires further study.

All experiments reported in this paper are based on the dataset of signatures generated from the 46,331 web video clips described in Section 2.2. We refer to this

test dataset as SIGDB. Based on our prior experimental results in Section 3.7.3, we use $m' = 100$ seed vectors for each signature and the top $m = 6$ ranked signature vectors for computing the signature similarity. The seed vectors are based on keyframes sampled from the video sequences in the database. ϵ_C used in computing the ranking function $Q(\cdot)$ in Equation (3.13) is set to be 2.0.

4.2 Projection-vector mapping

Let S be the set of seed vectors given by $\{s_1, s_2, \ldots, s_m\}$. Let x_s and y_s be the signature vectors in signatures X_S and Y_S that correspond to the same seed vector $s \in S$. Consider the following m-dimensional vector,

$$\mathcal{T}(x_s) \triangleq (d(x_s, s_1), d(x_s, s_2), \ldots, d(x_s, s_m)), \tag{4.6}$$

as a feature extraction mapping of x_s. We are interested in this particular formulation because of the following two reasons. First, this mapping makes use of quantities that have already been computed, and thus require no additional complex metric computations – the quantities $d(x_s, s_i)$ for $i = 1, \ldots, m$ are used to identify which feature vectors in X become signature vectors. Second, the distance $d(x_s, y_s)$ between any two signature vectors x_s and y_s can be related to the coordinates of the corresponding range vectors $\mathcal{T}(x_s)$ and $\mathcal{T}(y_s)$ by the triangle inequalities:

$$|d(x_s, s_i) - d(y_s, s_i)| \leq d(x_s, y_s) \leq d(x_s, s_i) + d(y_s, s_i), \quad i = 1, 2, \ldots, m \tag{4.7}$$

The above inequalities are instrumental in designing the feature extraction mapping. Our goal is to make use of these inequalities to design a range metric function $d'(\cdot)$ such that small $d(x_s, y_s)$ values correspond to "small" $d'(\mathcal{T}(x_s), \mathcal{T}(y_s))$ values and vice versa.

The mapping $\mathcal{T}(\cdot)$ and its variations have been previously proposed in the literature for feature extraction [36, 37, 38, 39]. These techniques typically use a l_p-metric as the range metric between $\mathcal{T}(x_s)$ and $\mathcal{T}(y_s)$. The most commonly-used l_p-metric functions include l_1, l_2 and l_∞, and are defined as follows:

$$\begin{aligned} l_1(\mathcal{T}(x_s), \mathcal{T}(y_s)) &\triangleq \frac{1}{m} \cdot \sum_{i=1}^{m} |d(x_s, s_i) - d(y_s, s_i)| \\ l_2(\mathcal{T}(x_s), \mathcal{T}(y_s)) &\triangleq \left(\frac{1}{m} \cdot \sum_{i=1}^{m} [d(x_s, s_i) - d(y_s, s_i)]^2 \right)^{1/2} \\ l_\infty(\mathcal{T}(x_s), \mathcal{T}(y_s)) &\triangleq \max_{i=1,2,\ldots,m} |d(x_s, s_i) - d(y_s, s_i)| \end{aligned} \tag{4.8}$$

We use a normalization factor of $1/m$ in the definitions of l_1 and l_2 so they are of the same order of magnitude as the l_∞-metric. All three metric functions defined in (4.8) are composed of some variation of the absolute differences between the coordinates of $\mathcal{T}(x_s)$ and $\mathcal{T}(y_s)$, i.e. $|d(x_s, s_i) - d(y_s, s_i)|$, for $i = 1, 2, \ldots, m$. These absolute differences, however, appear only in the lower-bound half of the triangle inequalities in (4.7).

The lower bound is preferred in the literature because perfect *Accuracy* can be guaranteed by simply setting the pruning threshold ϵ' to be the same as the similarity threshold ϵ. This is the so-called the Contractive Property of lower bounds, which

can be explained by the following inequalities:

$$l_1(\mathcal{T}(x_s), \mathcal{T}(y_s)) \leq l_2(\mathcal{T}(x_s), \mathcal{T}(y_s)) \leq l_\infty(\mathcal{T}(x_s), \mathcal{T}(y_s)) \leq d(x_s, y_s) \tag{4.9}$$

The inequality $l_\infty(\mathcal{T}(x_s), \mathcal{T}(y_s)) \leq d(x_s, y_s)$ can be shown by taking the maximum over all the lower bounds of the triangle inequalities in (4.7). The proofs of the other inequalities in (4.9) can be found in [33]. If we choose any of the l_1, l_2, or l_∞ as the range metric $d'(\cdot)$ and set $\epsilon' = \epsilon$, $d(x_s, y_s) \leq \epsilon$ will imply $d'(\mathcal{T}(x_s), \mathcal{T}(y_s)) \leq \epsilon'$ for any pair of signature vectors x_s and y_s. This further implies the candidate set $C(x_s; \epsilon')$, defined in Equation (4.2), must contain the true resulting set $A(x_s; \epsilon)$, described in Equation (4.1). As a result, GEMINI can produce perfect *Accuracy* by an exhaustive search on $C(x_s; \epsilon')$.

Nevertheless, for similarity search on the web, fast search time is typically more desirable than perfect *Accuracy*. By using a simple experiment, we can demonstrate that a better em Pruning-Accuracy tradeoff is achievable by combining both the upper and lower bounds of the triangle inequalities. Our experiment is based on a small random set of signature vectors sampled from the SIGDB that was introduced in Section 4.1. 100,000 pairs of signature vectors, all corresponding to the same, arbitrarily-chosen seed vector, are randomly sampled from the SIGDB. For each pair of signature vectors x_s and y_s, we compute $d(x_s, y_s)$ using the color histogram distance $d_c(\cdot)$ in Equation (3.16), illustrated in Figure 4.1 as a function of one lower and one upper bound, defined as follows. For the lower bound, we take the maximum over all the individual lower bounds in (4.7), which is identical to $l_\infty(\mathcal{T}(x_s), \mathcal{T}(y_s))$. For

the upper bound, we use a similar approach and take the minimum of the individual upper bounds in (4.7) to form an $\alpha(\cdot)$ function:

$$\alpha(\mathcal{T}(x_s), \mathcal{T}(y_s)) \triangleq \min_{i=1,2,\ldots,m} [d(x_s, s_i) + d(y_s, s_i)] \tag{4.10}$$

Different colored points in Figure 4.1 correspond to different ranges of values for $d(x_s, y_s)$. All the points in the plot are confined within a triangular area. The left edge

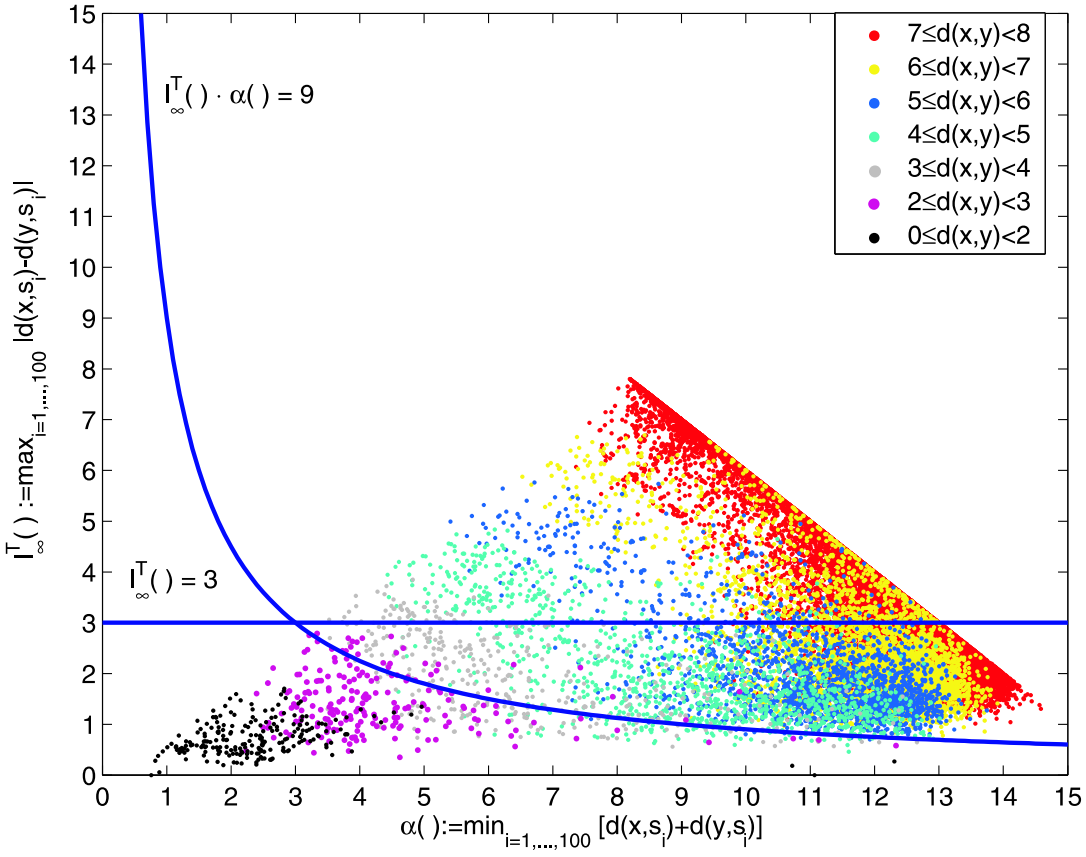


Figure 4.1: *Distribution of the metric $d(x, y)$ for 100,000 random pairs of signature vectors in the coordinates of $\max_{i=1,2,\ldots,100} |d(x, s_i) - d(y, s_i)|$ and $\min_{i=1,2,\ldots,100} [d(x, s_i) + d(y, s_i)]$. Different colors correspond to metric values at different ranges.*

of the triangle is due to the fact that $l_\infty(\mathcal{T}(x_s), \mathcal{T}(y_s))$ is always smaller than or equal

to $\alpha(\mathcal{T}(x_s), \mathcal{T}(y_s))$. The right boundary with tightly packed red points implies the inequality $\alpha(\mathcal{T}(x_s), \mathcal{T}(y_s)) + l_\infty(\mathcal{T}(x_s), \mathcal{T}(y_s)) \leq 16$. This inequality can be explained as follows: let s_j be the seed vector that achieves the maximum in $l_\infty(\mathcal{T}(x_s), \mathcal{T}(y_s))$ for a particular pair x_s and y_s. This implies that $\alpha(\mathcal{T}(x_s), \mathcal{T}(x_s)) + l_\infty(\mathcal{T}(x_s), \mathcal{T}(y_s)) = \alpha(\mathcal{T}(x_s), \mathcal{T}(y_s)) + |d(x_s, s_j) - d(y_s, s_j)| \leq d(x_s, s_j) + d(y_s, s_j) + |d(x_s, s_j) - d(y_s, s_j)|$. The last inequality holds because $\alpha(\mathcal{T}(x_s), \mathcal{T}(y_s))$ is the minimum of $d(x_s, s_i) + d(y_s, s_i)$ for all s_i's. The last expression is also equal to twice the larger value between $d(x_s, s_j)$ and $d(y_s, s_j)$. For the color-histogram feature vector used in the experiment, $d_c(\cdot)$ cannot be larger than eight, and thus the whole expression cannot be larger than $2 \cdot 8 = 16$.

If we set ϵ to be 3.0, which is a reasonable value to identify the majority of visually similar web video in our dataset, then metric values found in a typical resulting set $A(x_s; \epsilon)$ will include those black and magenta data points in Figure 4.1. Our goal then is to separate these "small-metric" points, from the rest of the "large-metric" points. If we use $l_\infty(\cdot)$ as the range metric, a typical candidate set based on the inequality $l_\infty(\mathcal{T}(x_s), \mathcal{T}(y_s)) \leq \epsilon'$ will include all the points below a horizontal line at level ϵ'. An example of such a set with $\epsilon' = 3$ is shown in Figure 4.1. Even though all the small-metric points are within the candidate set, many of the large-metric points are also erroneously included as they have small $l_\infty(\cdot)$ values. It is clear, based on the shape of the distribution of the small-metric points, that a better separating function should combine both $l_\infty(\cdot)$ and $\alpha(\cdot)$. One possible choice is based on their product,

$\beta(\cdot)$, defined as:

$$\beta(\mathcal{T}(x_s), \mathcal{T}(y_s)) \triangleq \alpha(\mathcal{T}(x_s), \mathcal{T}(y_s)) \cdot l_\infty(\mathcal{T}(x_s), \mathcal{T}(y_s)) \tag{4.11}$$

As shown in the figure, even though the candidate set defined by $\beta(\mathcal{T}(x_s), \mathcal{T}(y_s)) \leq 9$ misses a few small-metric points, it excludes a much larger set of large-metric points than $l_\infty(\cdot)$. The problem with $\beta(\cdot)$ is that it is not a metric[1]. As a result, we cannot directly employ $\beta(\cdot)$ as our range metric.

The $\beta(\cdot)$ function in (4.11) is defined as the product of $\alpha(\cdot)$ and $l_\infty(\cdot)$. The α and l_∞ functions represent the aggregate bounds of all the inequalities in (4.7). Rather than using the two aggregate bounds, it is easier to form a metric by using the product of the bounds from the individual inequalities as follows:

$$[d(x_s, s_i) + d(y_s, s_i)] \cdot |d(x_s, s_i) - d(y_s, s_i)| = |d(x_s, s_i)^2 - d(y_s, s_i)^2|, \;\; i = 1, 2, \ldots, m \tag{4.12}$$

Note that Equation (4.12) is in the form of an absolute difference. While absolute differences also appear in the definitions of l_p metrics in Equation (4.8), the one in Equation (4.12) is the absolute difference between the *squares* of the coordinates in $\mathcal{T}(\cdot)$. Thus, it is conceivable to propose a new metric $\zeta(\cdot)$ that combines l_2 with this

[1] $\beta(\cdot)$ is not a metric because, for an arbitrary pair of m-dimensional vectors u and v, $\beta(u, v)$ becomes zero when u and v share a zero in any one of the coordinates, rather than $u = v$ as required by a true metric.

absolute difference of *squares* of coordinates as follows[2]:

$$\zeta(\mathcal{T}(x_s), \mathcal{T}(y_s)) \triangleq \left(\frac{1}{m} \cdot \sum_{i=1}^{m} [d(x_s, s_i)^2 - d(y_s, s_i)^2]^2 \right)^{1/2} \tag{4.13}$$

This metric applied to $\mathcal{T}(\cdot)$ can also be interpreted as the $l_2(\mathcal{P}(x_s), \mathcal{P}(y_s))$, where $\mathcal{P}(x_s)$ is a new feature extraction mapping defined by:

$$\mathcal{P}(x_s) \triangleq (d(x_s, s_1)^2, d(x_s, s_2)^2, \ldots, d(x_s, s_m)^2) \tag{4.14}$$

We call $\mathcal{P}(x_s)$ the Projection Vector of x_s. Therefore, l_2 metric with the projection vector mapping is equivalent to applying the $\zeta(\cdot)$ metric onto $\mathcal{T}(\cdot)$ mapping. Our particular choice of l_2-metric will be explained in Section 4.3. We now justify the use of the projection-vector mapping based on experimental results on signature data.

Unlike $l_\infty(\cdot)$ and $\beta(\cdot)$, it is difficult to show the candidate set of $l_2(\mathcal{P}(x_s), \mathcal{P}(y_s)) \leq \epsilon'$ in Figure 4.1 because $l_2(\cdot)$ cannot be written in terms of the ordinate and abscissa of the graph. In addition, the simple experiment on which we have based our intuition so far is quite limited – the sample size is small and the comparisons are between individual signature vectors, rather than full signatures. To produce measurements on a more realistic setting, we expand our experiments to the full SIGDB signature database using Procedure 4.0.1, and test different range metrics within the GEMINI framework for the similarity search. The augmented color histogram distance $\widehat{d}_c$, as described in Equation (3.17), is used between color histograms, and ϵ is again

[2]Technically, $\zeta(\cdot)$ forms a metric only for real vectors with non-negative (or non-positive) coordinates, which is the case for our $\mathcal{T}(\cdot)$ vectors. If both positive and negative coordinates are allowed, $\zeta(\cdot)$ fails to become a metric as $\zeta(u, v) = 0$ when the coordinates u and v have the same magnitudes but different signs.

set at 3.0. The three schemes tested are the "lower-bound" scheme based on the mapping $\mathcal{T}(\cdot)$ in Equation (4.6) and the l_∞-metric, the "product" scheme based on $\mathcal{T}(\cdot)$ and the $\beta(\cdot)$ function defined in (4.11), and the proposed scheme based on $\mathcal{P}(\cdot)$ in Equation (4.14) and the l_2-metric. A random query set of 1000 signatures are drawn from the SIGDB, and the *Pruning* and *Accuracy* values, as defined in (4.5) and (4.4) respectively, are measured at different ϵ'. The resulting plots of *Pruning* versus *Accuracy* are shown in Figure 4.2. A good feature extraction mapping should

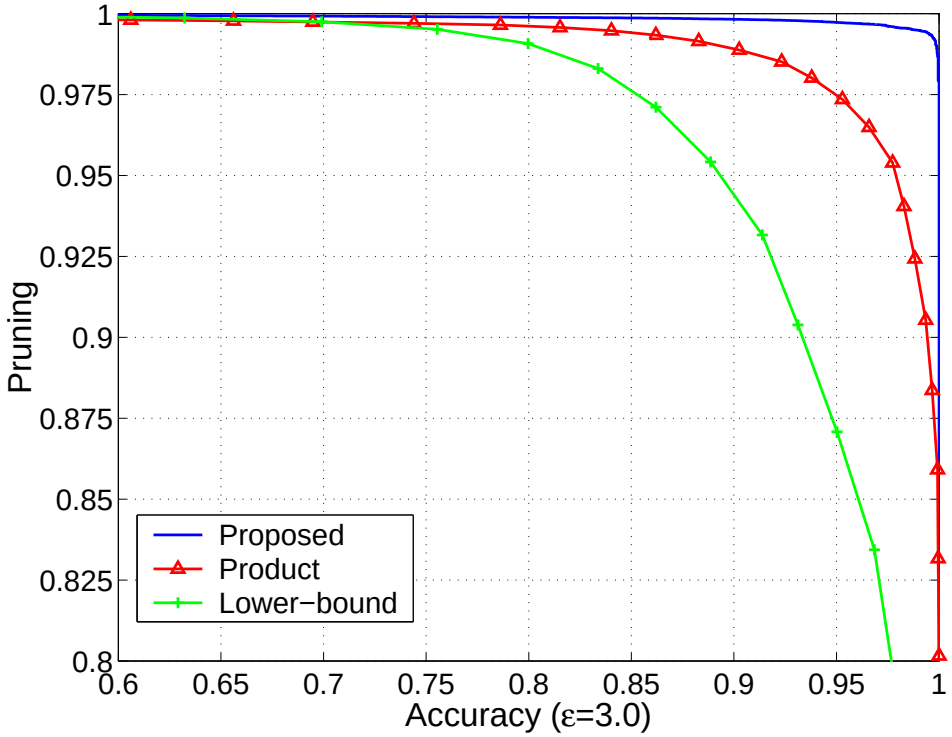


Figure 4.2: *Pruning-versus-Accuracy plots for the "lower-bound", the "product" and the proposed schemes.*

achieve *pruning* and *Accuracy* that are as close to one as possible. As shown in the figure, our proposed scheme clearly out-performs both the "lower-bound" and

"product" schemes by achieving much higher *Pruning* at the same *Accuracy* level. Also, as expected, the "product" scheme out-performs the "lower-bound" scheme as the "product" scheme exploits both the upper and lower bounds of the triangle inequality.

4.3 PCA on projection vectors

The last experiment in the previous section clearly demonstrates the superiority of the projection vector mapping $\mathcal{P}(x_s)$ with l_2-metric. Nevertheless, the dimension of $\mathcal{P}(x_s)$ is $m = 100$ for the SIGDB. Despite the fact that m is smaller than the dimension of our original feature vectors, i.e. 712, it is still much larger that what most SAM structures can handle. We address this problem by applying the classical PCA to transform the projection vectors into vectors of any target dimension, which we call Index Vectors. The collection of index vectors for each of the signature vectors in a signature is called an Index. It has been shown that, among all possible linear dimension reduction mappings, index vectors computed by the PCA method produce the least distortion in l_2 distances [32]. This further justifies our choice of using l_2 between projection vectors in Equation (4.13). PCA is straightforward to compute as well – it can be computed by first scanning all the data to estimate the covariance matrix, and then projecting the data to the subspace spanned by the most significant eigenvectors of the covariance matrix. Many numerical methods, such as SVD or variants of QR methods, have been developed to find eigenvectors

of covariance matrices [33]. If the dimension of the original space is too large, online update algorithms such as Lanczos Recursions or Subspace Iteration may be more appropriate [75].

The transformation from a signature, first to a projection, and finally to an index can in principal be considered as one feature extraction mapping to be used in GEMINI. In the remainder of this section, we present experimental results comparing our proposed scheme with other existing approaches of feature extraction mapping in the literature. Here are the descriptions of the schemes we have tested:

PCA While PCA is applied on the projection vectors in our proposed scheme, it can also be applied directly onto the 712-dimensional color histogram feature vectors. In this scheme, we apply PCA to reduce the dimension of the color histograms, and use l_2-metric on the resulting range vectors. The use of PCA on the color histogram can be justified as follows: even though PCA is only optimal for l_2 metric, the d_c metric used between the color-histogram feature vectors can be bounded above and below by l_2 as follows[3]:

$$\frac{1}{\sqrt{712}} \cdot l_2(x_s, y_s) \leq d_c(x_s, y_s) \leq l_2(x_s, y_s) \tag{4.15}$$

It is thus conceivable to use the l_2 metric to approximate the d_c metric.

Fastmap Faloutsos and Lin have proposed a feature extraction mapping called the Fastmap to approximate a general metric space with a low-dimensional l_2 space [35].

[3]The proof of the inequality can be found in [33]. Following the same convention as in Equation (4.8), the l_2 metric is normalized by the dimension of the vector.

Fastmap is a randomized algorithm that iteratively computes each dimension of a range vector by projecting the data onto the "axis" spanned by the two points with maximal separation. l_2 distance is used to compare range vectors.

Haar Another method for feature extraction in color histograms is to apply a Haar wavelet transform on the histogram bin values according to the adjacency of colors in the color space. The index vector is composed of a few low-pass coefficients from the transform. The Haar wavelet approach used in this dissertation is based on the scheme described in the MPEG-7 standard [76]. The l_1 metric is used to compare range vectors.

To compute an appropriate feature extraction mapping for a database, our proposed method and the PCA scheme must scan the entire database once. Fastmap requires multiple scans to find maximally separated data points in the database. The simplest technique is Haar as it is a fixed transform and does not depend on the data at all. We exclude all quadratic-time methods such as Multidimensional scaling [34] or SparseMap [38] in our comparisons as they do not scale well to large databases.

We follow the same procedure described in Section 4.2 to measure *Accuracy* and *Pruning* for all the schemes being tested. Since most of the schemes require training data to generate the mappings, we arbitrarily split SIGDB into two halves – we call one half the "training" SIGDB, which is used for building the mapping, and the other half the "testing" SIGDB, which is used for the actual testing. In order to ensure the suitability of incorporating these schemes into GEMINI, we focus on using very low

dimensions for range vectors. We test all the schemes for dimensions two, four and eight. The corresponding *Pruning-Accuracy* plots are shown in Figures 4.3 through 4.5. Our proposed scheme produces the best performance in all dimensions tested, followed by Haar, Fastmap and PCA. The gain of the proposed scheme over the second best scheme, however, diminishes as the dimension increases.

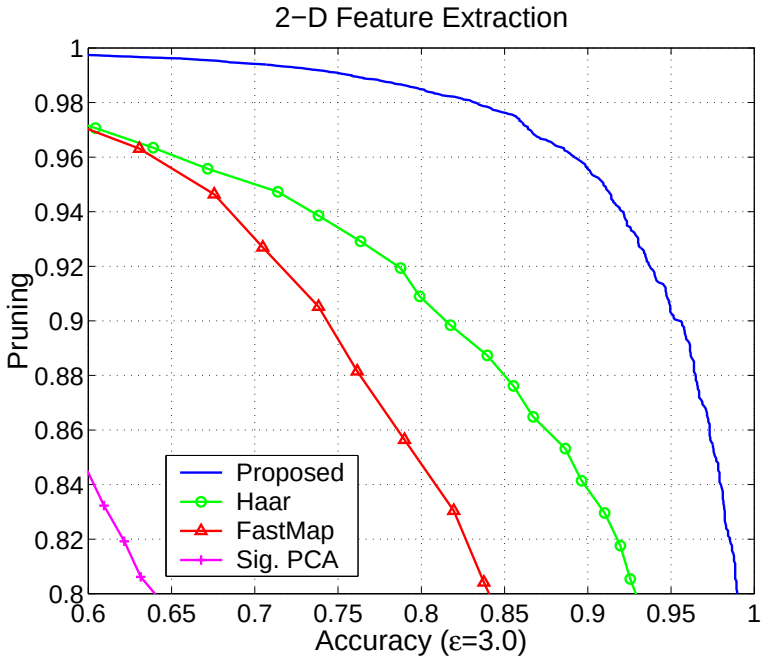


Figure 4.3: *Pruning-versus-accuracy plot for two dimension.*

In applying the feature extraction scheme in a fast similarity search, we need to choose a particular value of the pruning threshold ϵ' to compute the candidate set. Given the target dimension, *Accuracy*, and *Pruning*, one possible approach is to set ϵ' to a value that attains the particular level of performance in a previously completed experiment. Thus, an important question to answer is whether the relationship be-

tween ϵ' and the corresponding *Pruning* and *Accuracy* holds for queries other than the ones being tested. To answer this question, we repeat the experiments on our proposed scheme at dimension eight for three independent sets of random queries. Each set has 1000 signatures randomly drawn from the testing SIGDB. For each set of queries, different values of *Pruning* and *Accuracy* are measured by varying ϵ'. The experiment is also repeated for three different values of ϵ, namely 2, 3, and 4. The resulting *Pruning* and *Accuracy* versus ϵ' plots for the three query sets and different values of ϵ are shown in Figure 4.6. As shown in the figure, there is little variation in the amount of *Pruning* among the three sets. There is some variation in the *Accuracy* for small ϵ, but the variation diminishes as ϵ becomes larger. The maximum differences in *Accuracy* among the three sets over all possible values of ϵ' are 0.12,

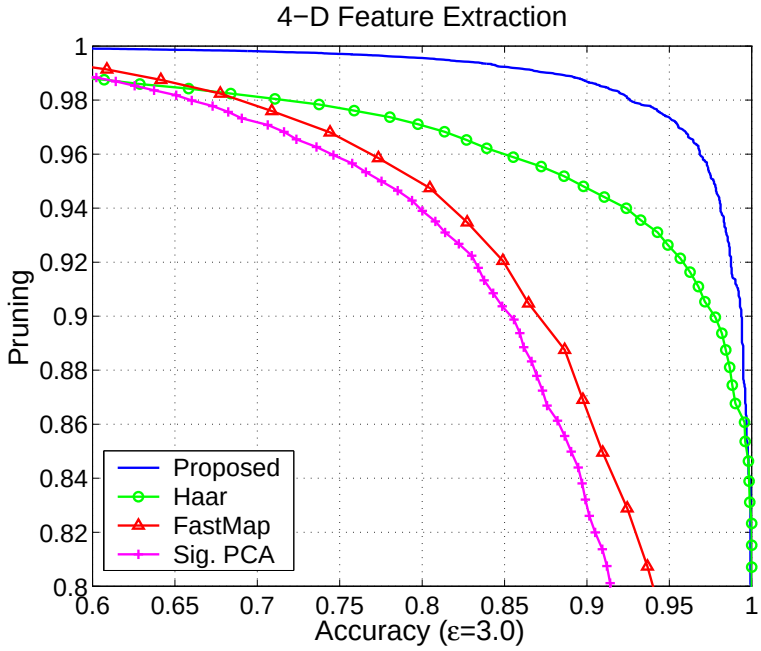


Figure 4.4: *Pruning-versus-accuracy plot for four dimension.*

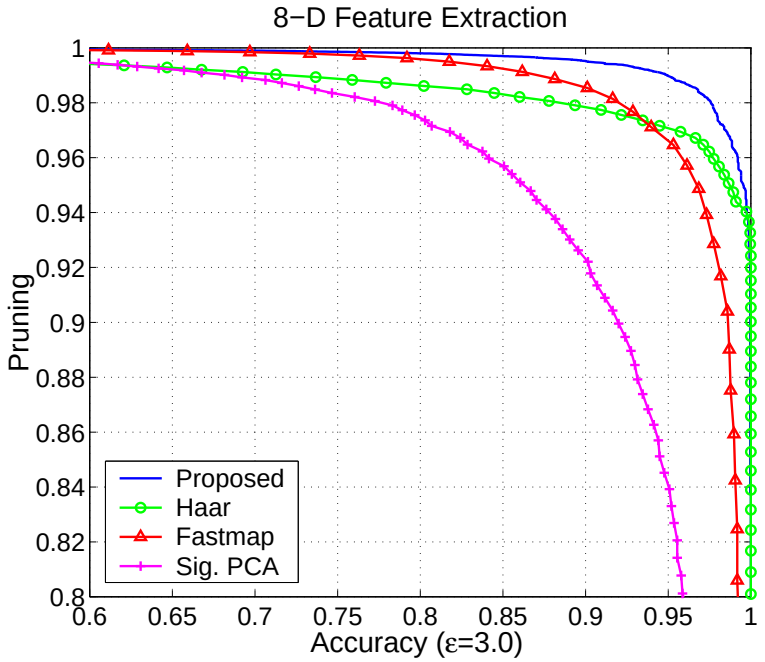


Figure 4.5: *Pruning-versus-accuracy plot for eight dimension.*

0.06, and 0.04 for $\epsilon = 2$, 3, and 4 respectively. These fluctuations are small compared to the high *Accuracy* required by typical applications.

We conclude this section with a number of speed measurements on a particular platform. Rather than measuring the performance of the entire GEMINI system, we make some simplifications so as to focus on measuring the performance of various feature extraction techniques. The most significant simplification is the absence of a SAM structure in our implementation. The primary function of a SAM structure is to provide fast similarity search on the low-dimensional range vectors. In our implementation, we have chosen to replace it with a simple sequential search. However, we measure time for sequential search separately so we can compare different

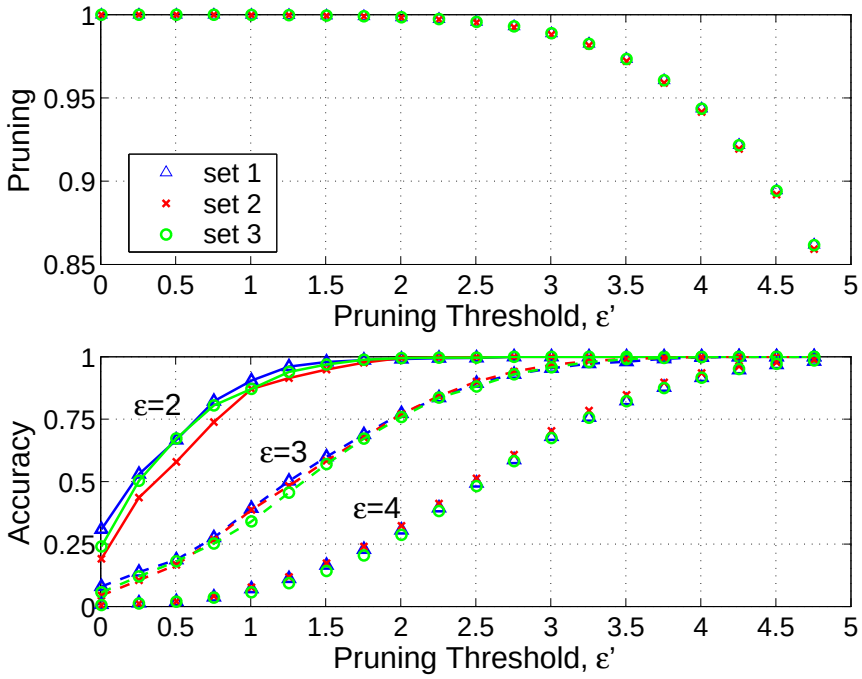


Figure 4.6: *Pruning and Accuracy versus pruning threshold for three independent sets of queries.*

schemes. Another function provided by a SAM structure is memory management for large databases. When a database is too large to fit within main memory, a SAM structure stores similar data items in contiguous regions on disk. This representation can minimize the number of slow random disk accesses during a similarity search. As the size of our test dataset is moderate, we fit the entire database of signatures and indices within main memory which eliminates the need for memory management[4]

We perform our experiments on a Dell PowerEdge 6300 Server with four 550MHz Intel Xeon processors and 1 Gigabyte of memory. As all the tests are run under a single

[4]To put the entire database inside the memory, we have made some modifications on how queries are compared against the signature database. Details of the modifications can be found in Appendix 4.A.

thread, only a single processor is used. The testing SIGDB, which contains 23,206 video signatures each consisting of 100 signature vectors, and their corresponding 8-dimensional indices are loaded into memory. 100 queries are randomly sampled from the testing SIGDB, and the time to perform the similarity search for a single query is measured. Pruning thresholds are chosen, based on the previous experiments, to hit the 90% *Accuracy* level for similarity searches at $\epsilon = 3.0$. As a reference, we also measure the performance of sequential search on signatures with no feature extraction. The results are shown in Table 4.1.

Feature Extraction	Sequential	Proposed	Fastmap	Haar	PCA
Accuracy	1.00	0.89	0.91	0.92	0.89
Index time (ms)	-	131 ± 0.8	131 ± 1.5	152 ± 1.3	130 ± 1.4
Refinement time (ms)	6730 ± 35	33 ± 8	75 ± 11	123 ± 28	401 ± 75
Candidate Size/query	-	109 ± 27	262 ± 39	428 ± 97	1386 ± 257

Table 4.1: *Speed comparisons among the sequential search, the proposed scheme, Fastmap, and Haar Wavelet.*

The Index time is the time required for the sequential search on range vectors to identify the candidate sets. The averages and their standard error at 95% confidence interval are shown. As the Sequential scheme does not use range vectors, no number is reported. The proposed scheme, fastmap, and PCA all use the l_2 distance on range vectors and thus, result in roughly the same index time. Haar requires slightly larger index time for its l_1 distance computation. The refinement time is the time required to perform the full signature distance computations on the candidate sets. Our proposed scheme outperforms all other feature extraction schemes in refinement

time. The large standard error in the refinement time is due to the variation in the size of candidate sets, as depicted in the last row of the table. Combining the index time and refinement time, the proposed scheme is roughly 41 times faster than the sequential search on signatures.

4.4 Summary

This chapter discussed the GEMINI framework for fast similarity search on metric data. In particular, we have focused on the feature extraction mapping which is the key design step in GEMINI. We have proposed a novel feature extraction mapping, and applied it to a large database of video signatures. The feature extraction mapping consists of two steps. In the first step, each signature vector is mapped into a projection vector. The projection vector is composed of the squared distances between the signature vectors and the seed vectors. Unlike other designs proposed in the literature, the projection-vector mapping provides better search performance by taking advantage of both the upper and lower bounds of the triangle inequalities. In the second step, the dimension of the projection vectors is further reduced by using PCA. PCA is appropriate as we have designed the projection vectors to be used with the l_2 distance. We have shown experimentally that this technique provides a better trade-off between *Accuracy* and *Pruning* as compared with PCA on feature vectors, Fastmap, and Haar Wavelet.

4.A Appendix: Modification for speed tests

The modification is illustrated in Figure 4.A. The figure on the left shows the original design while the one on the right shows the modification. Both figures show a query signature being compared with a database of signatures. To simplify the explanation, we assume that each signature has only four signature vectors, generated with respect to the seed vectors s_1, s_2, s_3, s_4, and the two signature vectors with the highest ranks are used in the comparison. In other words, we use $m = 4$ and $m' = 2$ in computing the asymmetric VSS_r between the query and each signature in the database.

The left diagram in Figure 4.A depicts the second step in Procedure 4.0.1 where each of the top-ranked vectors from the query is compared with the database. This step is nothing more than a series of metric-based similarity searches, which is supported by most SAM structures. There is, however, one drawback of this design: similarity search for different query signatures may access different portions of the database, as these query signatures have different top-ranked vectors. If the main memory is not large enough to hold the entire signature database, part of the database must be stored in the file system, which have much longer access time than the memory. To support fast response time for similarity search, many SAM structures thus implement sophisticated memory management techniques to minimize access to the slow file system.

Since we have not incorporated any SAM structure in our experiments, we resolve

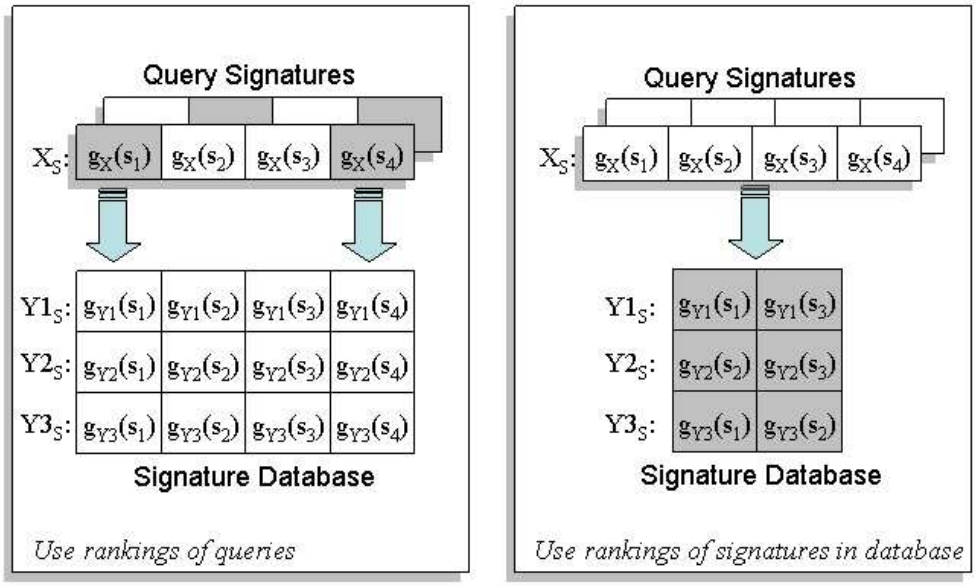


Figure 4.A: *The left figure shows the signature similarity search that uses the ranking of the query signature. The system on the right uses the rankings of the signatures in the database. Since the right system only needs to access the top-ranked signature vectors, less memory is required to store the database.*

this problem by reversing the roles of the query signature and the signature in the database. Specifically, rather than using the ranking of the query signature in calculating the asymmetric VSS_r, we use the ranking of the signature in the database instead. This simple modification enables us to keep only the top-ranked vectors of each signature from the database in the memory, regardless of the input queries. We can illustrate this concept using the right diagram in Figure 4.A. We first load into the memory the top-ranked vectors of all the signatures in the database. For

each query signature X_S presented to the system, it is sequentially compared with each signature Y_S in the database using Y_S's ranking. For example, to compute the similarity between X_S and $Y1_S$, we use the vectors with respect to s_1 and s_3; for X_S and $Y2_S$, we use the vectors with respect to s_2 and s_3, and so on. There is no need to access the lower-ranked vectors of the signatures in the database as they are never used. To understand the significance of this change, consider the SIGDB in our experiments which stores $m = 100$ vectors for each signature, and uses only $m' = 6$ top-ranked vectors for computing the similarity. This simple modification allows us to reduce the memory requirement by more than a factor of $100/6 \approx 16$, and enable us to store the entire signature database in the memory. Furthermore, all the feature extraction mappings discussed before can be directly applied to this design as the same ranking applies to both signature and index vectors. On the other hand, it should be noted that this design is not appropriate in the systems where SAM is used. The reason is that the comparison is now done in a signature-by-signature basis, which implies that the signature similarity search can no longer be decomposed into a series of metric-based similarity search. It should also be noted that this new design does not necessarily produce identical retrieval results as the old one where the ranking of the query is used. Nevertheless, we have found little discrepancy between the two schemes in the experimental results with our ground-truth set.

Chapter 5

Similarity signature clustering

By applying the similarity search techniques described in Chapter 4 to every signature in a database, it is possible to identify rapidly video sequences that are similar in the database to a given query. Beyond the limited capability of the feature vector in capturing visual similarity, there are a number of reasons for less than perfect retrieval performance from the above process. First, as explained in Section 3.5, if an overwhelming number of seed vectors fall inside the voronoi gap, the ViSig method may underestimate the true similarity between two video sequences. Second, the fast similarity search techniques, introduced in Chapter 4, trade off accuracy with speed performance. As a result, some similar signature pairs in the database may be erroneously left out. Third, we have assumed thus far that ϵ is known and constant for all signatures, which is certainly not the case for the diverse content found on the web. To address the above problems, we need a post-processing step to further

process the information computed by the similarity search step.

This chapter turns to one example of a post-processing step – identifying clusters of similar signatures within the database. Based on the measured signature similarities and some assumptions about visually similar video clips, we run an offline clustering algorithm to partition the entire database into clusters of signatures. Video clips within the same cluster are assumed to be similar to each other, and an individual cluster, rather than a video clip, is made the smallest unit for retrieval. We attempt to use this clustering structure to mitigate the aforementioned problems by simultaneously considering the relationship among all the signatures in a database. In particular, we propose a new hierarchical clustering algorithm that is robust against erroneous or missing measurement, and capable of adaptively choosing the distance threshold based on local statistics among similar signatures. We demonstrate that our proposed algorithm produces better retrieval performance than simple thresholding used in Chapter 3, and other clustering algorithms proposed in the literature.

This chapter is organized as follows. Section 5.1 describes the graphical representation of a signature database for developing the clustering algorithm. The details of the algorithm are presented in Section 5.2. In Section 5.3, we compare our algorithm to other techniques based on retrieval performance of the ground-truth set. Finally, Section 5.4 presents a number of interesting statistics about the distribution of similar clusters that we identify in our web video database.

5.1 Graphical representation of signature database

To describe our clustering algorithm in a general framework, we treat the set of signatures and their similarity relationship as a graph. A graph $\mathcal{G}$ has a set of vertices $V(\mathcal{G})$ and a set of edges $E(\mathcal{G}) \subset V(\mathcal{G}) \times V(\mathcal{G})$. All edges are undirected. In many occasions, we only consider a portion of a graph. $\mathcal{G}'$ is a subgraph of $\mathcal{G}$ if $V(\mathcal{G}') \subset V(\mathcal{G})$ and $E(\mathcal{G}') \subset E(\mathcal{G})$. If the subgraph $\mathcal{G}'$ inherits all the edges between its vertices from the original graph, $\mathcal{G}'$ is called an induced subgraph. In our application, each signature in the database is represented as a vertex in a graph. Before applying the clustering algorithm on the signature data, we assume that there is an edge between every pair of signatures, with the edge length indicating the measured signature similarity between the two signatures. Because it is more convenient to use a distance function to measure the edge length, we define the following Signature Distance:

$$d_{sig}(X_S, Y_S) \stackrel{\Delta}{=} \text{median}\{d(g_X(s_{j[i]}), g_Y(s_{j[i]})) : i = 1, 2, \ldots, m\}, \tag{5.1}$$

where $j[i]$ denotes the ranking of the signature vector $g_X(s_i)$, for $i = 1, 2, ..., m$. We ignore the rankings of the signature vectors in Y_S in order to make it compatible with the fast similarity search algorithm presented in Chapter 4. As we have experimentally shown before, there is little difference between using the ranking of one signature and using the rankings of both. Thus, we ignore the asymmetry in Equation (5.1) and treat every edge as undirected. The median operator is used in Equation (5.1) because of the following reason. Recall the ground-truth experiment in Section 3.7.3,

we defined two signatures X_S and Y_S to be similar if more than half of their signature vectors are within ϵ of each other. By using the median operator in $d_{sig}(\cdot)$, we can simply state the same condition by using $d_{sig}(X_S, Y_S) \leq \epsilon$).

Given a graph of signatures, the goal of a clustering algorithm is to identify truly similar video clips based on the overall structure of the graph. We can interpret the clustering process as a process of removing edges between pairs of vertices that represent dissimilar video clips. Typically, an edge should be removed if the two corresponding vertices have large signature distances. Nevertheless, even if the signature distance between two vertices is large, there might still be a need for an edge if, for example, there has been an error in the distance measurement. Such an error may be revealed if there are many other signatures that are simultaneously close to both of them. Thus, the clustering algorithm needs to make use of all measured distance relationships to infer the most reasonable placement of edges. Since it is computationally infeasible to search all possible placements of edges, we only consider a special subset of subgraphs called threshold graphs. A threshold graph $P(\mathcal{V}, \rho)$ has a vertex set $\mathcal{V}$, and has an edge between every two of its vertices if the distance between them is strictly less than $\rho > 0$.

In the absence of any error in distance measurement and similarity search, we assume that the largest possible signature distance between two truly similar video clips is μ. The choice of μ depends on the feature vector as well as the data. For the feature vector to be useful in similarity detection, μ is typically much smaller

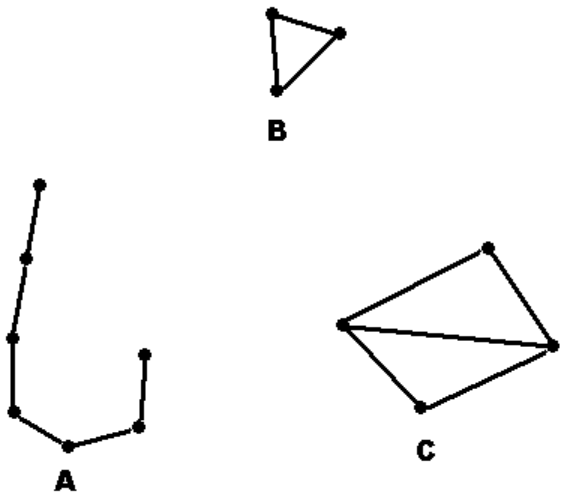


Figure 5.1: *A threshold graph with three connected components.*

than the maximum distance value, and the threshold graph $P(\mathcal{V}, \mu)$ is very sparse. Our algorithm considers all the threshold graphs $P(\mathcal{V}, \rho)$ with $\rho \leq \mu$. Since each of the subgraph $P(\mathcal{V}, \rho)$ is sparse, they are likely to contain many isolated connected components. A connected component, or CC, of a graph is an induced subgraph in which all vertices are reachable from each other, but completely disconnected from the rest of the graph. A pictorial example of a threshold graph with three CC's is shown in Figure 5.1.

CC's in the threshold graph $P(\mathcal{V}, \rho)$ are prime candidates for similar clusters: all signatures in a CC $\mathcal{C}$ are at least ρ away from the rest of the database. If $\mathcal{C}$ is also a complete graph, which means that there is an edge between every pair of signatures, intuitively it corresponds to what a similar cluster should be. All video sequences

in the cluster are similar to each other but far away from the rest of the database. In practice, full completeness is too stringent of a requirement to impose because the randomness in signatures may erroneously amplify the distance between similar video sequences. Thus, as long as $\mathcal{C}$ is close to a complete subgraph, it is reasonable to assume that it represents a similar cluster. To this end, we need a measurement of the density of edges inside a CC. Note that for any CC $\mathcal{C}$, there are at least $|V(\mathcal{C})|-1$ edges as $\mathcal{C}$ is connected, and at most $|V(\mathcal{C})| \cdot (|V(\mathcal{C})|-1)/2$ edges if there is one edge between every pair of vertices. We can thus define an edge density function $\Gamma(\mathcal{C})$ between these two extremes as:

$$\Gamma(\mathcal{C}) \triangleq \begin{cases} \frac{|E(\mathcal{C})|-(|V(\mathcal{C})|-1)}{|V(\mathcal{C})|\cdot\frac{(|V(\mathcal{C})|-1)}{2}-(|V(\mathcal{C})|-1)} & \text{if } |V(\mathcal{C})|>2 \\ 1 & \text{otherwise,} \end{cases} \tag{5.2}$$

$\Gamma(\mathcal{C})$ evaluates to 0 when $\mathcal{C}$ is barely connected, and to 1 when $\mathcal{C}$ is complete. For the three clusters shown in Figure 5.1, the edge densities for A, B, and C are 0, 1, and 2/3 respectively. We define a similar cluster to be a CC whose edge density exceeds a fixed threshold $\gamma \in (0, 1]$.

5.2 Signature clustering

We are now ready to describe our clustering algorithm: given a database of signatures $\mathcal{V}$, we compute $P(\mathcal{V}, \mu)$ by performing a fast similarity search on each signature to identify all those that are within distance μ away. The resulting $P(\mathcal{V}, \mu)$ is composed of CC's with varying edge densities. Those CC's with edge densities larger than

γ are identified as similar clusters and removed from the graph. For the remaining CC's, we start removing edges in decreasing order of length until some similar clusters emerge. To avoid bias, edges of the same length are removed simultaneously. This process of edge removal is equivalent to lowering the distance threshold ρ until the graph is partitioned into multiple CC's. CC's with high enough edge densities are identified as similar clusters. This process of lowering distance threshold and checking edge density is repeated until we exhaust all the CC's.

The key step of the above algorithm is to find the appropriate distance threshold to partition a CC $\mathcal{C}$ once it is found not be a similar cluster. A naive approach is to check whether $\mathcal{C}$ remains connected after recursively removing the longest edge, or edges, from $\mathcal{C}$. This approach is computationally expensive as we need to check connectedness after each edge removal. A more efficient approach is to use the Minimum Spanning Tree (MST) of $\mathcal{C}$. A MST $\mathcal{T}$ of a connected graph $\mathcal{C}$ is a subgraph that connects all vertices with the least sum of edge lengths. The following proposition explains why it is possible to use MST in our clustering algorithm:

Proposition 5.2.1 *Let $\mathcal{C}$ be a connected component in $P(\mathcal{V}, \rho)$ and $\mathcal{T}$ be a MST of $\mathcal{C}$. Let d be the length of the longest branch in $\mathcal{T}$. Consider the following partition of $\mathcal{T}$:*

$$\mathcal{T} = \mathcal{E} \cup \mathcal{T}_1 \cup \mathcal{T}_2 \cup \ldots \cup \mathcal{T}_N, \tag{5.3}$$

where $\mathcal{E}$ consists of all the branches of length d, and $\mathcal{T}_i$'s are individual connected components in $\mathcal{T}$ after the removal of $\mathcal{E}$. We can characterize the connectedness of

the threshold graphs $P(V(\mathcal{C}), \rho')$ *for* $\rho' \geq d$ *as follows:*

1. *for* $\rho' > d$*, the threshold graph* $P(V(\mathcal{C}), \rho')$ *is connected;*

2. *for* $\rho' = d$*, the threshold graph* $P(V(\mathcal{C}), \rho')$ *is composed of* N *connected components* $\{\mathcal{C}_\rangle, i = 1, 2, \ldots N\}$ *with* $V(\mathcal{C}_\rangle) = V(\mathcal{T}_\rangle)$*. As a consequence,* $\mathcal{T}_\rangle$ *is a MST of* $\mathcal{C}_\rangle$*.*

The proof of Proposition 5.2.1 can be found in Appendix 5.A. The implications of Proposition 5.2.1 are as follows. First, it shows that $\mathcal{C}$ stays connected until the distance threshold is lowered beyond d, the length of the longest branch in the MST. In other words, d is the correct threshold to decompose $\mathcal{C}$ into CC's. Second, we also show that, after partitioning $\mathcal{C}$ into a new set of CC's, the subtree of $\mathcal{T}$ in each newly-formed CC is also a MST. Hence, we can further partition the new CC's without computing a new MST for them. In a nutshell, all the distance thresholds required for clustering can be obtained by computing the MST for the original threshold graph $P(\mathcal{V}, \mu)$.

Based on Proposition 5.2.1, we can efficiently implement our clustering algorithm in two steps. In the first step, we use the Kruskal algorithm described in [66, ch. 23] to construct the MST: edges are examined in increasing order of length and the tree is progressively built by including edges that join CC's together. The only difference is that whenever a new branch is added to the MST, we compute the edge density of the CC on either side of this branch. As explained before, the length of each MST

branch is the right distance threshold to partition a graph into CC's, and the edge density of each CC is computed based on all their edges shorter than the threshold. Recall that the Kruskal algorithm builds the MST by sequentially examining all edges in the increasing order of their lengths. When a new MST branch is identified, all the edges that contribute to the edge densities of the CC's on either side of this new branch must already have been examined by the algorithm. Thus, we can compute the edge density by simply keeping track of the number of edges in each CC thus far examined by the Kruskal algorithm, before it is linked by a new MST branch. The time complexity of the first step of this algorithm is the same as the Kruskal algorithm, which is $O(e \log e)$ where e is the number of edges in $P(\mathcal{V}, \mu)$.

In the second step, we identify similar clusters by repeatedly setting the distance threshold to the length of the branches in the MST, and checking the pre-computed edge density for each newly-formed CC. A CC becomes a similar cluster if its edge density exceeds γ. For each threshold tested, all the information required to identify similar clusters is pre-computed, and the time complexity is simply $O(1)$. As a result, the time complexity of the second step is $O(|\mathcal{V}|)$, the same order as the maximum number of edges in the MST. The time complexity of the entire algorithm is thus $O(e \log e) + O(|\mathcal{V}|) \approx O(e \log e)$. A pseudo-code implementation of this algorithm can be found in Appendix 5.B.

In the actual implementation, we compute the edge lengths by performing a similarity search on each video in the database. The maximum distance threshold μ is set

at 4.0 and the pruning threshold ϵ' for similarity search is set at 3.0. The resulting threshold graph has 1,094,691 edges and 36,311 nodes. The edges are then stored in a sorted database of edges based on a public-domain embedded database system called the Berkeley DB version 4.0. The construction of the sorted edge database takes around 201 seconds on a 500 MHz Intel Xeon with 1GB of memory. Running on the same machine, the MST construction and the clustering requires 3.8 seconds and 0.3 seconds, respectively.

5.3 Ground-truth results

This section compares the experimental results on applying the proposed algorithm with simple-thresholding, single-link and complete-link techniques to the SIGDB described in Section 4.1. The performance is measured based on how well an algorithm can identify the ground-truth set introduced in Section 2.3.

Recall from Section 3.7.3, retrieval performance is measured based on recall and precision defined in (3.19). Recall and precision are defined with respect to the concepts of relevant and return sets. Given a query X_S in the ground-truth, the relevant set is always the similar cluster in the ground-truth that contains X_S, excluding X_S itself. In Section 3.7.3, simple thresholding is used and the return set is all signatures in SIGDB that are within a certain distance threshold of X_S. Different recall and precision values can be obtained by changing the distance threshold. When a cluster-

ing algorithm is used, the return set is simply the cluster that contains q[1]. To obtain a full spectrum of different recall and precision values, the parameter used for single-link and complete-link is the distance threshold that defines the minimum distance between clusters. For our proposed algorithm, it is the edge density threshold γ.

Figure 5.2 shows precision versus recall for the four algorithms. As shown in Fig-

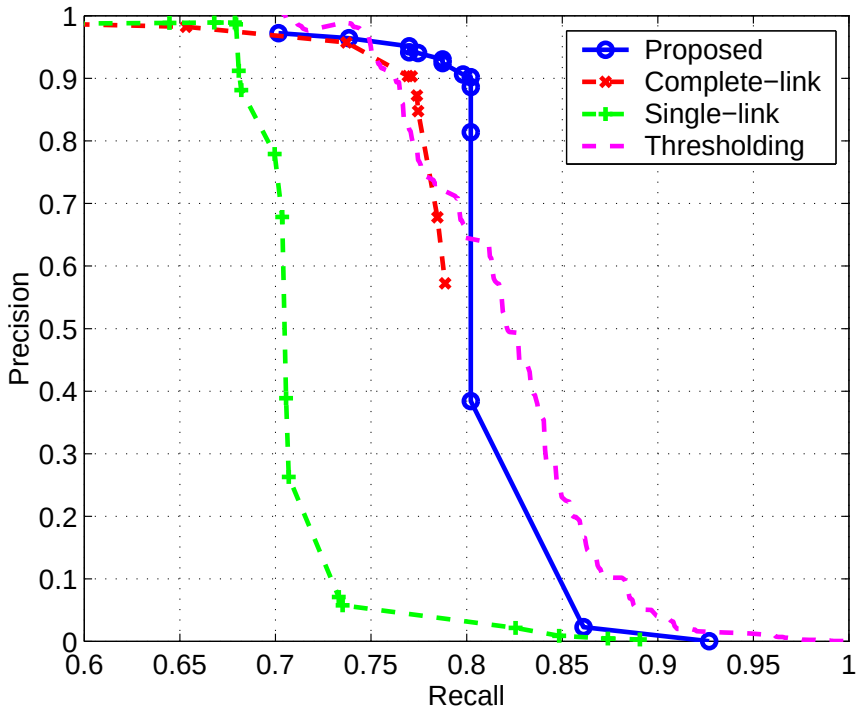


Figure 5.2: *Precision versus recall for different clustering algorithms and simple thresholding.*

ure 5.2, the single-link algorithm exhibits the worst performance. As the distance threshold increases, the single-link algorithm erroneously chains together non-similar

[1]For the general case when q is not part of the database, we can define an ad-hoc distance between q and each cluster based on, for example, the minimum of the distances between q and each of the elements in the cluster. The return set will simply be the cluster closest to q.

video clips together in a cluster. This significantly reduces precision if one of the query signatures falls within these large clusters. Simple thresholding does not look beyond the immediate neighbors and thus produces much better performance as compared with the single-link algorithm. The complete-link algorithm adds an additional constraint that neighbors can be grouped in the same cluster if they are all within the same distance threshold. Initially, this translates into a small improvement in precision over simple thresholding. Due to the strict cluster-forming criterion, the complete-link algorithm ignores many similar video clips in the ground-truth set until the distance threshold is raised to almost the maximum value. As the distance threshold is global to the entire dataset in complete-link, some of the smaller clusters begin to join together when the threshold is large. This joining results in a drop in the precision as recall improves. Our proposed algorithm delivers a good trade-off between precision and recall – before the steep descend in precision, our algorithm achieves retrieval performance of around 80% recall and 90% precision.

Figure 5.3 shows the corresponding plots of precision and recall versus the edge density threshold γ. Larger γ values correspond to more strict criterion in forming clusters, leading to larger precision and smaller recall. The retrieval performance stays around 80% recall and 90% precision for the γ value in between 0.1 and 0.35. Even though recall starts to degrade when γ becomes larger, it stays relatively close to 80% for γ as large as 0.9. This implies that our clustering algorithm should work for a relatively large range of γ values.

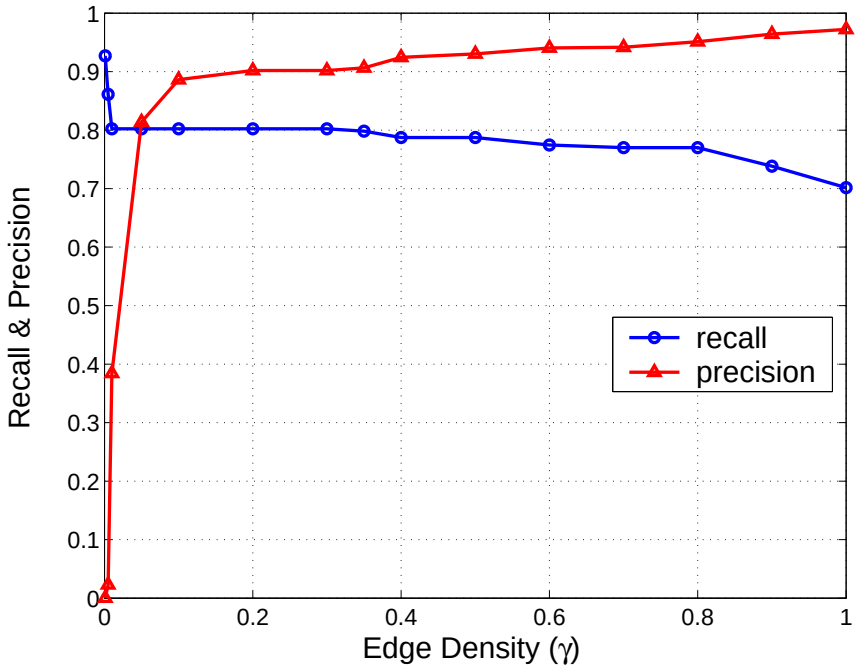


Figure 5.3: *Precision and Recall versus edge density threshold* γ.

5.4 Similar cluster statistics for web video

To further study how similar video clips are distributed on the web, we set γ to 0.2, and produce a clustering structure on our experimental database. The resultant clustering structure has a total of 7,056 clusters, with average cluster size of 2.95. Since there are 46,331 video clips in the database, $7056 \cdot 2.95/46331 \approx 45\%$ of the video clips in our database have at least one possibly similar version. Figure 5.4 shows the distribution of cluster sizes. The majority of the clusters are very small – 25% of the clusters have only two video clips in them. Nonetheless, there are a few very large clusters. The abundance of similar versions in these clusters may indicate that

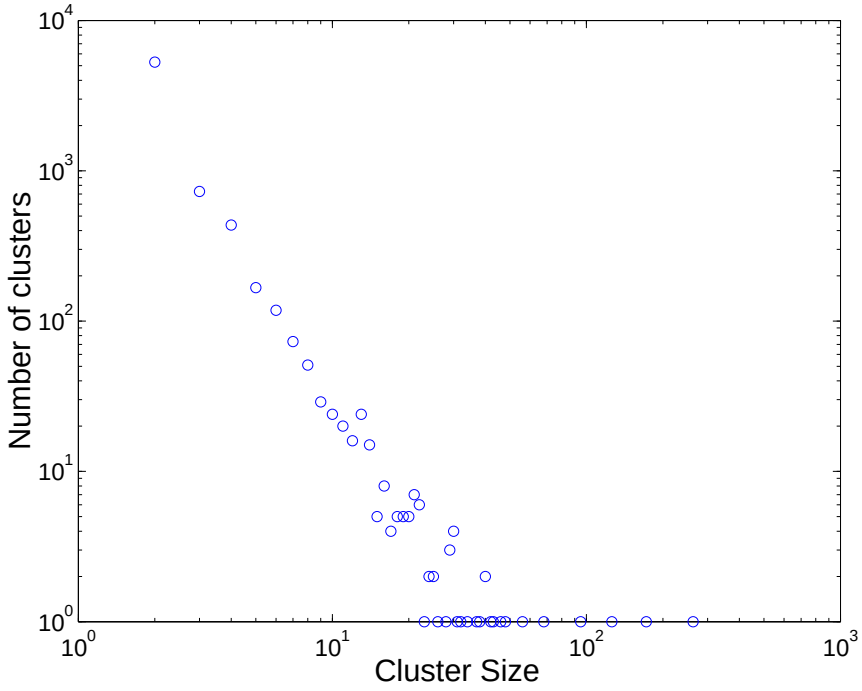


Figure 5.4: *Distribution of cluster sizes.*

these video clips are very popular in the web community. Table 5.1 lists the top ten clusters identified in the clustering structure.

We provide each cluster with a label in the first column for ease of reference. The second column indicates the size of each cluster. In the third column, we consider the diversity of web locations where similar video clips are found. We notice that it is quite common for a content creator to put similar video clips such as those compressed in different formats on the same web-page. Diversity is the ratio between the number of distinct web-pages in each cluster and the cluster size. A large ratio implies that video clips in that cluster originated from diverse locations. The fourth column indicates

Label	Size	Diversity	Misclassified?	Descriptions or Explanations
A	263	0.12	Y	Share a segment of red letters on black background
B	172	0.70	N	Dancing Baby from tv show "Ally McBeal"
C	126	0.43	Y	Share a segment of "MTV News" sign
D	95	0.01	Y	Share a segment of "chv.net" sign
E	68	0.01	N	An error message from chv.net server
F	56	0.98	N	Angry man hitting a computer
G	48	0.19	N	Mathematical plots of wave equation
H	46	0.09	N	Segments of President Clinton taped testimony
I	43	0.42	N	SOHO Astronomical Video
J	42	0.08	N	Synthetic Aperture Radar Video

Table 5.1: *Statistics of the largest ten clusters in the database.*

the correctness of the cluster – an "N" indicates that more than 95% of the video clips in the cluster are declared to be similar by manual inspection. As shown in the table, clusters A, C and D are wrongly classified. Upon careful examination, we found that all the video clips in each of these clusters share a visually-similar segment, from which multiple signature vectors are selected. As a result, they are classified to be "similar" even though the remainder of the content in these clips is very different. A possible remedy to this problem is to raise the required number of similar signature vectors shared between two signatures before declaring them as similar. Among those which are correctly classified, clusters E, G, H and J consist of clips that are mostly from the same web-page. Some of them are identical, such as the error message found in cluster E. Others have very similar visual contents such as those in G, H and J. These types of sequences are generated intentionally by the content creators, and provide

little information on how video sequences are copied and modified by different web users. On the other hand, clusters B, F, and I have relatively high diversity values. Video sequences in cluster B are from a popular television show, cluster F contains a humorous sequence of a man's frustration towards his computer, and cluster I contains astronomical video clips from a large-scale, multi-nation research project. Large clusters with high diversity values seem to indicate that the corresponding video content is of interest to a large number of web users. Such information can be used to provide better ranking for retrieval results – popular content should be ranked higher as they are more likely to be requested by users.

5.5 Summary

This chapter discussed the use of clustering algorithms on large databases of video signatures. We have applied clustering algorithms to mitigate possible error incurred by signature distance measurement and fast similarity search, and to adaptively choose the distance threshold based on local statistics. To this end, we have proposed a novel graph-theoretical algorithm that identifies clusters based on graph connectivity. In this algorithm, the database of signatures is modeled as a threshold graph. Starting from a reasonably large distance threshold, the algorithm considers all connected components formed at each distance threshold, and identifies those with large edge densities as clusters. Rather than checking for every possible distance threshold, we have shown that it is sufficient to consider only those belonged to the

minimum spanning tree (MST) of the original threshold graph. As a result, we have implemented our clustering algorithm based on Kruskal's classical MST algorithm. Our algorithm outperforms simple-thresholding, single-link, and complete-link clustering algorithms in terms of the retrieval performance of the ground-truth set from SIGDB. We have also applied our algorithm to identify similar clusters in the SIGDB. The majority of the clusters we have found are small, but there are a number of large clusters. We have found that large clusters with video clips originating from diverse locations are good indicators of popular video content.

5.A Appendix: Proof of Proposition 5.2.1

The first statement is obvious – since the lengths of all the edges in $\mathcal{T}$ are at most d, $\mathcal{T}$ is a subgraph of $P(V(\mathcal{C}), \rho')$ for all $\rho' > d$. As $\mathcal{T}$ and $P(V(\mathcal{C}), \rho')$ share the same set of vertices and $\mathcal{T}$ is connected, $P(V(\mathcal{C}), \rho')$ is connected.

To prove the second statement, we first show that $P(V(\mathcal{C}), d)$ is not connected. Pick any edge e of length d in $\mathcal{T}$. By removing this edge, we partition $\mathcal{T}$ into two disjoint trees $\mathcal{T}_a$ and $\mathcal{T}_b$. If $P(V(\mathcal{C}), d)$ is connected, vertices in $\mathcal{T}_a$ and in $\mathcal{T}_b$ must be connected by some edges in $P(V(\mathcal{C}), d)$. Let (u, v) be such an edge with $u \in V(\mathcal{T}_a)$ and $v \in V(\mathcal{T}_b)$ as illustrated in Figure 5.A(a). By adding (u, v) to $\mathcal{T}_a$ and $\mathcal{T}_b$, we form a new spanning tree of $\mathcal{C}$, $\mathcal{T}'$. Since (u, v) is an edge in $P(V(\mathcal{C}), d)$, the length of (u, v) must be strictly less than d. As a result, the total length of all the branches in $\mathcal{T}'$ is shorter than that of $\mathcal{T}$. This contradicts the assumption that $\mathcal{T}$ is a MST.

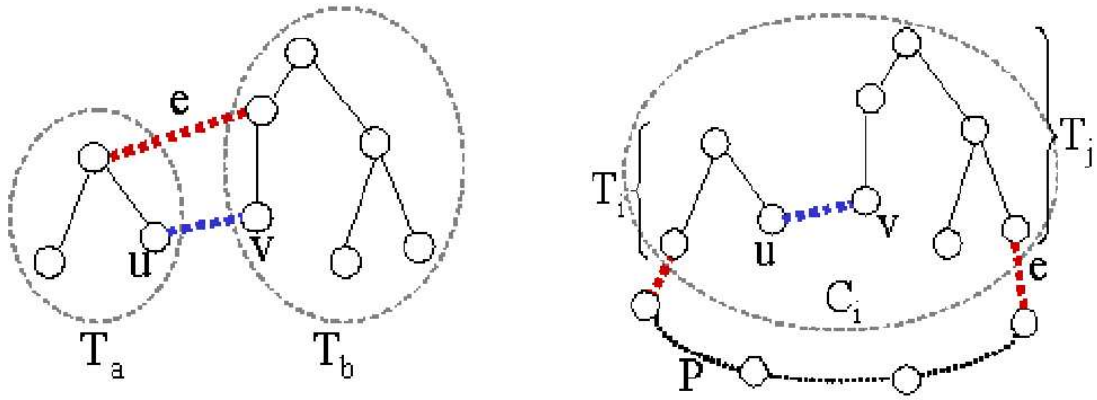
Hence, $P(V(\mathcal{C}), d)$ must be disconnected.

Since all the subtrees $\mathcal{T}_j$'s have edges shorter than d, they are all subgraphs of $P(V(\mathcal{C}), d)$. Thus, each of the connected component $\mathcal{C}_i$ must contain an integral number of $\mathcal{T}_j$'s. If $\mathcal{C}_i$ has only one subtree $\mathcal{T}_i$, $\mathcal{T}_i$ must be the MST of $\mathcal{C}_i$ – otherwise we can reduce the total edge length of $\mathcal{T}$ by using the MST of $\mathcal{C}_i$ to replace $\mathcal{T}_i$.

The goal thus is to show that each $\mathcal{C}_i$ corresponds to exactly one $\mathcal{T}_j$. Assume it is not the case and there exists a $\mathcal{C}_i$ which contains two or more subtrees. Choose two subtrees $\mathcal{T}_i$ and T_j in C_i such that there exists an $(u, v) \in E(\mathcal{C}_i)$ with $u \in \mathcal{T}_i, v \in \mathcal{T}_j$. Such a triplet of $\mathcal{T}_i$, $\mathcal{T}_j$ and (u, v) must exist as $\mathcal{C}_i$ is connected. In addition, (u, v) is not a branch in $\mathcal{T}$ otherwise it would be part of $\mathcal{T}_i$ or $\mathcal{T}_j$. On the original tree $\mathcal{T}$, there exists a path P between u and v. Since $\mathcal{T}_i$ and $\mathcal{T}_j$ are disconnected with respect to $\mathcal{T}$, and all the edges connecting them to the rest of $\mathcal{T}$ are of length d, P must contain an edge e of length d. This is demonstrated in Figure 5.A(b). Replacing e with (u, v) in the original $\mathcal{T}$ will form a new spanning tree of $\mathcal{C}$ with shorter total length as $d(u, v) < d$. Again we obtain a contradiction and hence, each $\mathcal{C}_i$ corresponds to one and only one $\mathcal{T}_j$.

5.B Appendix: Clustering algorithm

In this appendix, we describe a pseudo-code implementation of the clustering algorithm introduced in Section 5.2. The implementation consists of two main routines: BUILD-MST for the construction of the MST and calculations of the edge densities,



(a) (b)

Figure 5.A: *For both illustrations, we obtain a contradiction by replacing e of length d with a shorter edge (u, v) to obtain a better MST.*

and CLUSTER for identifying the clusters based on the computed MST and edge densities.

The BUILD-MST routine is a simple extension of the Kruskal algorithm. Our implementation is based on the version of Kruskal found in [66, p. 569]. The input to BUILD-MST is a $\mathcal{G}$ with N vertices. There are two output objects to this routine: $\mathcal{T}$ is the MST of $\mathcal{G}$, and *edgeDensity* is a two-dimensional array that stores the edge densities of the two CC's attached to each branch (X_S, Y_S) in $\mathcal{T}$, with the distance threshold setting at $d_{sig}(X_S, Y_S)$. The pseudo-code implementation of BUILD-MST is as follows:

BUILD-MST($\mathcal{G}$)
1 $\mathcal{T} \leftarrow \emptyset$
2 **for** each vertex $v \in V(\mathcal{G})$
3 **do** MAKE-SET(v)
4 sort $E(\mathcal{G})$ by nondecreasing $d_{sig}(\cdot, \cdot)$
5 $i \leftarrow 0$
6 **for** each edge $(X_S, Y_S) \in E(\mathcal{G})$, in order by nondecreasing $d_{sig}(X_S, Y_S)$

```
7       do
8           a ← FIND-SET(X_S)
9           b ← FIND-SET(Y_S)
10          w ← d_sig(X_S, Y_S)
11          if  a = b
12          then
13              a.numEdge ← a.numEdge + 1
14              if  w ≠ a.longestEdge
15              then
16                  a.longestEdge ← w
17                  a.numLongEdge ← 1
18              else
19                  a.numLongEdge ← a.numLongEdge + 1
20          else
21              T ← T ∪ (X_S, Y_S)
22              edgeDensity[i] ← (GAMMA(a, w), GAMMA(b, w))
23              COMBINE(a, b)
24          i ← i + 1
25 return (T, edgeDensity)
```

To understand BUILD-MST, we need to introduce the disjoint-set data structure used in the routine. The disjoint-set is used to represent a partition of a data set [66, ch. 21]. In our application, the data set is the set of all vertices and the partition is formed by the sets of vertices inside each connected component. Associated with each set are a number of status variables: *numNode* and *numEdge* are the number of vertices and edges inside the CC; *longestEdge* is the length of the longest edge inside the CC; *numLongEdge* is the number of edges in the CC with their lengths equal to *longestEdge*, and finally, *lastBranch* represents the length of the last MST branch attached to the CC. We use three routines from the disjoint-set data structure described in [66, ch. 21]: MAKE-SET(X_S) creates a singleton set with X_S; FIND-SET(X_S) returns the set containing X_S as an element, and COMBINE(X_S,Y_S) merges

the two sets containing X_S, Y_S.

Line 1-3 in BUILD-MST initializes the data structures by creating a singleton set for each vertex in the graph. In line 4, all the edges in the input graph are sorted in increasing order of their lengths. These edges are then sequentially added to the disjoint sets, which join to form CC's based on how they are connected by the new edges. For each edge (X_S, Y_S), lines 8 and 9 identify the CC's that contain X_S and Y_S. If X_S and Y_S belong to the same CC, (X_S, Y_S) is not part of the MST. In this case, we only need to update the status variables to account for the new edge added to the CC. Line 13 updates *numEdge* in the CC, while lines 15 through 19 update *longestEdge* and *numLongEdge*. If X_S and Y_S belong to different CC's, edge (X_S, Y_S) is the shortest edge joining the two CC's, and thus become part of the MST as indicated in line 13. The edge densities of the two CC's are computed by the routine GAMMA in line 21. GAMMA is shown as follows:

```
GAMMA(CC a, Distance w)
1   if   w = a.lastBranch
2   then
3        return -1
4   if   w = a.longestEdge
5   then
6        return Γ(a.numNode, a.numEdge − a.numLongEdge)
7   else
8        return Γ(a.numNode, a.numEdge)
```

Lines 1 to 3 of GAMMA handles the case when the new MST edge is the same length as the last MST edge attached to the CC. Recall that we are interested in computing the edge density of the CC when the distance threshold is set to be the

length of the new MST edge. In this scenario, the set of vertices in this CC are actually *disconnected* as the MST edge(s) added earlier disappear as well. As a result, it is meaningless to compute the edge density and thus we set it t -1. If the new MST edge is longer than $lastBranch$ but is the same length as the longest internal edges of the CC, we need to discount those internal edges as they will disappear. Hence, in line 6, we compute the edge density Γ, as defined in Equation (5.2), based on a discounted number of edges $numEdge - numLongEdge$. Line 8 represents the default case when the MST edge is longer than all edges in the CC.

Back to line 23 in BUILD-MST: after computing the edge densities, we combine the two CC's into one by using the COMBINE routine. COMBINE is listed below:

```
COMBINE(CC a, CC b, Distance w)
1   CC c ← UNION(a, b)
2   c.lastBranch ← w
3   c.longestEdge ← w
4   c.numNode ← a.numNode + b.numNode
5   c.numEdge ← a.numEdge + b.numEdge
6   c.numLongEdge ← 0
7   if  w = a.longestEdge
8   then
9       c.numLongEdge ← c.numLongEdge + a.numLongEdge
10  if  w = b.longestEdge
11  then
12      c.numLongEdge ← c.numLongEdge + b.numLongEdge
```

COMBINE is a simple routine that first joins the two sets by invoking the UNION routine, and then updates all the status variables to reflect the new CC. This concludes the BUILD-MST routine. The time complexity of BUILD-MST is in the same order of Kruskal which is $O(e \log e)$, where e is the number of edges in the graph.

The second part of the clustering algorithm, CLUSTER, identifies clusters based on the MST and the edge densities computed in the first part. It is listed below:

```
CLUSTER(Graph T, Array edgeDensity, Edge Density γ)
1   i ← 0
2   for each edge (X_S, Y_S) ∈ E(T), in reverse order of insertion
3   do
4       Delete (X_S, Y_S) from T
5       if   edgeDensity[i][0] ≥ γ
6       then
7           Remove connected components in T that contains X_S
8       if   edgeDensity[i][1] ≥ γ
9       then
10          Remove connected components in T that contains Y_S
11      i ← i + 1
```

In CLUSTER, all MST branches are scanned and deleted in the reverse order of how they are created in BUILD-MST. For each branch, the edge densities on either side of the branch are examined. If the edge density exceeds the density threshold, the whole CC is identified as a cluster and subsequently deleted. The deletion of branches and CC's can be easily implemented by using an adjacency-list data structure – for each vertex X_S of the MST, we associate a double linked list that stores all the vertices adjacent to X_S. Rather than searching for the next MST branch to delete, we can implement line 3 by simply deleting the last elements in the linked lists corresponding to the end vertices of the branch. This is because the deletion follows the reverse order of insertion. The time complexity for this step is $O(1)$. To delete the whole CC, we need to carry out a depth-first or breadth-first search to identify all the connected vertices. The time complexity is the same as the number of MST branches in that

CC, which is one less than its number of vertices. The above analysis shows that the complexity of the entire routine is simply $O(N)$, where N is the number of vertices.

Chapter 6

Summary and Future Work

This dissertation considered the problem of building a similarity search engine for a large and diverse database of video sequences such as the web. We tackled this problem from three different aspects: efficient and effective representation of video sequences, fast similarity search, and search result organization. Our main contribution towards the representation of video sequences is the development of a class of randomized techniques called Video Signature (ViSig). ViSig summarizes an entire video sequence, in linear time of the length of the video, into a small set of representative feature vectors called a signature. We performed analytical analysis, simulations, as well as ground-truth experiments to demonstrate the validity of ViSig – we demonstrated that it is possible to use small, fixed-size signatures to reliably estimate the underlying complex video similarity measurement. Signature thus constitutes the fundamental unit of similarity search and retrieval for our video

database system.

In developing a fast similarity search technique for large databases of signatures, we considered the more general problem of similarity search for metric data. In particular, we focused on the Generic Multimedia Indexing (GEMINI) approach of similarity search, and developed a novel feature extraction mapping that combines random projections and classical Principal Component Analysis (PCA). We first utilized the squared distances between signature vectors and seed vectors to form a projection vector. Then, the dimension of the projection vectors was further reduced by using PCA. Experimental results show that this new mapping can provide better trade-off between accuracy and pruning than other techniques, specifically, PCA, Fastmap, Haar Wavelet, and Triangle-Inequality Pruning (TIP).

To provide a compact organization of similarity search results, we investigated the use of clustering algorithms to group video sequences into similar clusters. Due to the random nature of ViSig and the less-than-perfect accuracy of fast search techniques, we developed a robust clustering algorithm that identifies densely connected components formed at different distance thresholds as clusters. This algorithm admits an efficient implementation based on the classical Kruskal algorithm. We showed experimentally that this clustering algorithm provides better retrieval performance than schemes such as simple thresholding, single-link, and complete-link hierarchical clustering algorithms.

As a proof of concepts, we combined all the proposed algorithms to construct

a video similarity search engine that contains more than 46,000 video clips crawled from the world-wide-web. Our analysis on this large dataset indicated that more than 45% of the web video clips had at least one visually similar version. Even though the majority of similar clusters we found had no more than two video clips, there were a few very large clusters with their sizes exceeded 100. These large clusters are good indications of popular and important video content.

We have described, in this dissertation, our initial effort in tackling the main challenges in providing similarity search for large video databases. There are, nonetheless, many exciting and challenging issues remained to be solved. In the sequel, we highlight some of the key problems pertaining to the algorithms proposed in this dissertation.

In developing the ViSig methods, we focused on two design heuristics: 1) the use of seed vectors that closely resemble the feature vector distribution of the real data, and 2) the use of ranking in comparing two signatures. The performance of these two heuristics are experimentally demonstrated. Nevertheless, some remaining issues still deserve further investigation. First, how the difference between the distributions of the real data and the seed vectors can affect the performance of ViSig? Second, the introduction of ranking creates a bias in the estimation of Ideal Video Similarity (IVS), as ranking favors seed vectors that are further away from the Voronoi cell boundary. Can such a bias be estimated based on some easily computed quantities from the video sequence? Beyond these specific design issues, a perhaps more important area

to explore is the extension of ViSig to other applications. The basic premise of our development of ViSig is the recognition of the importance of IVS as a similarity measurement. IVS defines a general similarity measurement between two sets of objects endowed with a metric function. By using ViSig, we have demonstrated one particular application of IVS, which is to identify highly similar video sequences found on the web. It should be an interesting and fruitful research direction to apply the entire ViSig framework to other types of pattern matching and retrieval problems.

The motivation of using squared distances in our proposed feature extraction mapping is to capture both the upper and lower bounds from the triangle inequalities. Even though we only demonstrated the merit of the proposed mapping on the color histogram data, the triangle inequality concept is a general property of any metric space. As such, we expect that our proposed technique can also be applied to other metric spaces, and we are currently exploring the feasibility of this mapping in genomic data.

Another research direction is to study the effect of seed vectors on the performance of similarity search. The proposed feature extraction mapping is based on distances between signature vectors and seed vectors randomly sampled from a training dataset. It is conceivable that more sophisticated methods can be used to select better seed vectors so the resulting mapping produces a better trade-off between accuracy and pruning. A study on the similar TIP approach has shown that search performance

can indeed be improved by carefully choosing the seed vectors[1] to match the data [39].

Another issue that we did not addressed in this dissertation was how to maintain the clustering structure when video signatures are inserted or deleted from the database. Based on the current design, we can first update the sorted edge database, then re-build the minimum spanning tree, and finally re-cluster based on the new tree. The update of the edge database can be carried out efficiently using B-tree index. On the other hand, re-building the MST runs in linear time with respect to the number of edges and re-clustering in linear time with respect to the number of nodes. One possible approach to speed up the last two steps is to perform a local repair on the MST in the case when the new signatures affect only a small portion of the graph.

[1]The term "key" was used in the original paper.

Bibliography

[1] A.Z. Broder, S.C. Glassman, M.S. Manasse, and G. Zweig, "Syntactic clustering of the web," in *Sixth International World Wide Web Conference*, Sept. 1997, vol. 29, no.8-13 of *Computer Networks and ISDN Systems*, pp. 1157–66.

[2] N. Shivakumar and H. Garcia-Molina, "Finding near-replicas of documents on the web," in *World Wide Web and Databases. International Workshop WebDB'98*, Valencia, Spain, Mar. 1998, pp. 204–12.

[3] K. Bharat and A. Broder, "Mirror, mirror on the web: a study of host pairs with replicated content," in *Proceedings of the Eighth International World Wide Web Conference*, May 1999, vol. 31, no.11-16 of *Computer Networks and ISDN Systems*, pp. 1579–90.

[4] K. Bharat, A. Broder, J. Dean, and M. Henzinger, "A comparison of techniques to find mirrored hosts on the WWW," *Journal of the American Society of Information Science*, vol. 51, no. 12, pp. 1114–1122, 2000.

[5] T. Kelly and J. Mogul, "Aliasing on the world wide web: Prevalence and performance implications," in *Proceedings of the 11th International World Wide Web Conference*, Honolulu, Hawaii, May 2002.

[6] C. Silverstein, M. Henzinger, J. Marais, and Michael Moricz, "Analysis of a very large altavista query log," Tech. Rep. SRC-Technical Note 1998-014, Compaq Systems Research Center, 1998.

[7] I. Witten, A. Moffat, and T. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*, Kaufmann Publishers, second edition, 1999.

[8] C. Fellbaum, Ed., *WordNet: An Electronic Lexical Database*, MIT Press, 1998.

[9] C. Faloutsos, *Searching Multimedia Databases by Content*, Kluwer Academic Publishers, 1996.

[10] M. Maybury and K. Jones, Eds., *Intellignet Multimedia Information Retrieval*, MIT Press, 1997.

[11] B. Perry et al., *Content-based access to multimedia information – from technology trends to state of the art*, chapter 4.3, Kluwer Academic Publishers, Massachusetts, U.S.A., 1999.

[12] V. Castelli and L. D. Bergman, Eds., *Image Databases: Search and Retrieval of Digital Imagery*, John Wiley & Sons, 2001.

[13] S. Santini and J. D. Gibson, *Exploratory Image Databases*, Academic Press, 2001.

[14] R. Lienhart, C. Kuhmunch, and W. Effelsberg, "On the detection and recognition of television commercials," in *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, Ottawa, Ontario, June 1997, pp. 509–16.

[15] R. Mohan, "Video sequence matching," in *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, Seattle, WA, May 1998, vol. 6, pp. 3697–3700.

[16] A. Hampapur and R. Bolle, "Feature based indexing for media tracking," in *Proceedings of IEEE International Conference on Multimedia and Expo*, New York, NY, July 2000, vol. 3, pp. 1709–12.

[17] A. Hampapur, K.-H. Hyun, and R. Bolle, "Comparison of sequence matching techniques for video copy detection," in *Proceedings of SPIE – Storage and Retrieval for Media Databases 2002*, San Jose, CA, January 2002, vol. 4676, pp. 194–201.

[18] M.R. Naphade, R. Wang, and T.S. Huang, "Multimodal pattern matching for audio-visual query and retrieval," in *Proceedings of the Storage and Retrieval for Media Datbases 2001*, San Jose, USA, jan 2001, vol. 4315, pp. 188–195.

[19] D. Adjeroh, I. King, and M.C. Lee, "A distance measure for video sequence similarity matching," in *Proceedings International Workshop on Multi-Media Database Management Systems*, Dayton, OH, USA, Aug. 1998, pp. 72–9.

[20] R. Lienhart, W. Effelsberg, and R. Jain, "VisualGREP: A systematic method to compare and retrieve video sequences," in *Proceedings of storage and retrieval for image and video databases VI*. SPIE, Jan. 1998, vol. 3312, pp. 271–82.

[21] H.S. Chang, S. Sull, and S.U. Lee, "Efficient video indexing scheme for content-based retrieval," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 8, pp. 1269–79, Dec 1999.

[22] P. Indyk, G. Iyengar, and N. Shivakumar, "Finding pirated video sequences on the internet," Tech. Rep., Stanford Infolab, Feb. 1999.

[23] S. Santini and R. Jain, "Similarity measures," *IEEE Tran. on Pattern Analysis and Machine Intelligence*, vol. 21, no. 9, pp. 871–83, Sept 1999.

[24] H. Greenspan, J. Goldberger, and A. Mayer, "A probabilistic framework for spatio-temporal video representation," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2001.

[25] G. Iyengar and A.B. Lippman, "Distributional clustering for efficient content-based retrieval of images and video," in *Proceedings 1998 International Conference on Image Processing*, Vancouver, B.C., Canada, 2000, vol. III, pp. 81–4.

[26] N. Vasconcelos, "On the complexity of probabilistic image retrieval," in *Proceedings Eighth IEEE International Conference on Computer Vision*, Vancouver, B.C., Canada, 2001, vol. 2, pp. 400–407.

[27] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *5th Berkeley Symposium on Mathematical Statistics*, 1967, vol. 1, pp. 281–97.

[28] L. Kaufman and P.J. Rousseeuw, *Finding Groups in Data*, John Wiley & Sons, New York, 1990.

[29] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, 1989.

[30] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*. 1997, pp. 426–435, Morgan Kaufmann.

[31] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *Proceedings of the 24th International Conference on Very-Large Databases (VLDB'98)*, New York, NY, USA, Aug. 1998, pp. 194–205.

[32] J. Hotelling, "Analysis of a complex of statistical variables into principal components," *J. of Educational Psychology*, vol. 24, pp. 417–441, 1933.

[33] G. Golub and C. van Loan, *Matrix Computation*, The Johns Hopkins University Press, 3rd edition, 1996.

[34] Trevor F. Cox and Michael A.A. Cox, *Multidimensional scaling*, Boca Raton : Chapman & Hall, second edition, 2001.

[35] C. Faloutsos and King-Ip Lin, "Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets," in *Proceedings of ACM-SIGMOD*, May 1995, pp. 163–174.

[36] J. Bourgain, "On lipschitz embedding of finite metric spaces in hilbert space," *Israel Journal of Mathematics*, vol. 52, pp. 46–52, 1985.

[37] N. Linial, E. London, and Y. Rabinovich, "The geometry of graphs and some of its algorithmic applictions," *Combinatorica*, vol. 15, no. 2, pp. 215–45, 1995.

[38] G. Hristescu and M. Farach-Colton, "Cluster-preserving embedding of proteins," Tech. Rep. DIMACS 99-50, Rutgers University, Piscataway, USA, 1999.

[39] A. P. Berman and L. G. Shapiro, "A flexible image database system for content-based retrieval," *Computer Vision and Image Understanding*, vol. 75, no. 1/2, pp. 175–195, July/August 1999.

[40] S. Krishnamachari and M. Abdel-Mottaleb, "Image browsing using hierarchical clustering," in *IEEE International Symposium on computer and communications*, July 1999.

[41] A. Vellaikal and C.-C.J. Kuo, "Hierarchical clustering techniques for image database organization and summarization," in *Proceedings of the SPIE – Multimedia Storage and Archiving Systems III*, Boston, USA, Nov 1998, vol. 3527, pp. 68–79.

[42] A. Hanjalic, R.L. Lagendijk, and J. Biemond, "Automated high-level movie segmentation for advanced video-retrieval systems," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, no. 4, pp. 580–8, June 1999.

[43] M. Yeung, B.-L. Yeo, and B. Liu, "Segmentation of video by clustering and graph analysis," *Computer Vision and Image Understanding*, vol. 71, no. 1, pp. 94–109, July 1998.

[44] K. Sparck Jones and C. van Rijsbergen, "Report on the need for and provision of an "ideal" information retrieval test collection," Tech. Rep. British Library Research and Development Report 5266, Computer Laboratory, University of Cambridge, 1975.

[45] S. Aksoy and R.M. Haralick, "Graph-theoretic clustering for image grouping and retrieval," in *Proceedings IEEE CVPR.*, June 1999, vol. 1, pp. 63–8.

[46] S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, Academic Press, 1999.

[47] J.B. Kruskal, "On the shortest spanning subtree of a graph and the travelling salesman problem.," *Proceedings of the American Mathematical Society*, vol. 7, pp. 48–50, 1956.

[48] C.T. Zahn, "Graph-theoretcial methods for detecting and describing gestalt clusters," *IEEE Transactions on Computers*, vol. 20, no. 1, pp. 65–86, January 1971.

[49] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, Aug 2000.

[50] R. Shamir and R. Sharan, "Algorithmic approaches to clustering gene expression data," in *Current Topics in Computational Molecular Biology*, T. Jiang, M. Zhang, and Y. Xu, Eds., pp. 269–300. MIT Press, Feb 2002.

[51] S.-C. Cheung and A. Zakhor, "Estimation of web video multiplicity," in *Proceedings of the SPIE – Internet Imaging*, San Jose, California, Jan. 2000, vol. 3964, pp. 34–6.

[52] S.-C. Cheung and A. Zakhor, "Efficient video similarity measurement and search," in *Proceedings of 7th IEEE International Conference on Image Processing*, Vancouver, British Columbia, Sept. 2000, vol. 1, pp. 85–88.

[53] S.-C. Cheung and A. Zakhor, "Efficient video similarity measurement with video signature," *Accepted to IEEE Transactions on Circuits and Systems for Video Technology*, 2003.

[54] S.-C. Cheung and A. Zakhor, "Efficient video similarity measurement with video signature," *Proceedings of the 9th IEEE International Conference on Image Processing*, vol. 1, pp. 621–624, Sept. 2002.

[55] S.-C. Cheung and A. Zakhor, "Efficient video similarity measurement using video signatures," in *Handbook of Video Databases*, B. Furht and O. Marques, Eds., chapter 31. CRC Press, 2003.

[56] S.-C. Cheung and A. Zakhor, "Video similarity detection with video signature clustering," in *Proceedings of 8th IEEE International Conference on Image Processing*, Thessaloniki, Greece, Sept. 2001, vol. 1, pp. 649–652.

[57] S.-C. Cheung and A. Zakhor, "Towards building a similar video search engine for the world-wide-web," *Submitted to IEEE Transactions on Multimedia*, 2002.

[58] K. Bharat and A. Broder, "A technique for measuring the relative size and overlap of public web search engines," in *Proceedings of the Seventh International*

World Wide Web Conference, 1998, vol. 30, no.1-7 of *Computer Networks and ISDN Systems*, pp. 379–88.

[59] Yahoo! Inc., http://www.yahoo.com, *Yahoo! Categories*.

[60] VideoSeeker, http://www.videoseeker.com, *VideoSeeker*.

[61] International Computer Science Institute, http://www.icsi.berkeley.edu/ dpwe/isrintro, *ICSI Speech recognition software*.

[62] M. Swain, "Searching for multimedia on the world wide web," Tech. Rep. Tech. Rep. CRL99/1, Cambridge Research Laboratory, 1999.

[63] ISO/IEC, *ISO/IEC 11172-2:1993 : Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 2:Video*, Nov. 1992.

[64] RealNetworks, http://www.real.com/devzone/library/whitepapers/overview.html, *RealVideo Technical White Paper*, Feb. 1997.

[65] S. Lawrence and C. Lee Giles, "Searching the world wide web," *Science*, vol. 280, pp. 98–100, Apr. 1998.

[66] T. Cormen, C. Leiserson, and R. Riverst, *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusetts, 2nd edition, 2001.

[67] R. Sibson, "Slink: An optimally efficient algorithm for the single-link cluster method," *The Computer Journal*, vol. 16, no. 1, pp. 30–4, 1973.

[68] H.L. Royden, *Real Analysis*, Macmillan Publishing Company, 1988.

[69] P. Indyk, *High-dimensional computational geometry*, Ph.D. thesis, Stanford University, 2000.

[70] J.R. Smith, *Integrated Spatial and Feature Image Systems: Retrieval, Analysis and Compression*, Ph.D. thesis, Columbia University, 1997.

[71] M.J. Swain and D.H. Ballard, "Color indexing," *International Journal of Computer Vision*, vol. 7, no. 1, pp. 11–32, November 1991.

[72] MPEG-7 Requirements Group, "Description of mpeg-7 content set," Tech. Rep. N2467, ISO/IEC JTC1/SC29/WG11, 1998.

[73] A. Woronow, "Generating random numbers on a simplex," *Computers and Geosciences*, vol. 19, no. 1, pp. 81–88, 1993.

[74] G.R. Grimmett and D.R. Stirzaker, *Probability and Random Processes*, Oxford Science Publications, 1992.

[75] P. Comon and G. Golub, "Tracking a few extreme singular values and vectors in signal processing," *Proceedings of the IEEE*, vol. 78, pp. 1327–1343, 1990.

[76] L. Cieplinski, S. Jeannin, M. Kim, and J.-R. Ohm, "Visual working draft 4.0," Tech. Rep. W3522, ISO/IEC JTC1/SC29/WG11, July 200.

[77] T. Gevers and A.W.M. Smeulders, "Image retrieval by multi-scale illumination invariant indexing," in *Multimedia Information Analysis and Retrieval. IAPR International Workshop, MINAR'98*, Hong Kong, China, Aug. 1998, pp. 96–108.

[78] M. Flicker et al., "Automatic and semiautomatic methods for image annotation and retrieval in qbic," in *Proceedings of storage and retrieval for image and video databases III*. SPIE, Jan. 1995, vol. 2420, pp. 24–35.

[79] A. Girgensohn and J. Boreczky, "Time-constrained keyframe selection technique," *Multimedia Tools and Applications*, vol. 11, pp. 347–358, 2000.

[80] B. Günsel, Y. Fu, and A.M. Tekalp, "Hierarchical temporal video segmentation and content characterization," in *Proceedings of the SPIE – Multimedia Storage and Archiving Systems II*, Dallas, USA, 1997, vol. 3229, pp. 46–56.

[81] X. Sun, M.S. Kankanhalli, Y. Zhu, and J. Wu, "Content-based representative frame extraction for digital video," in *IEEE Conference of Multimedia Computing and Systems*, Austin, USA, 1998, pp. 190–3.

[82] E. Sahouria and A. Zakhor, "Content analysis of video using principal components," in *Proceedings 1998 International Conference on Image Processing, volume 3*, Chicago, IL, USA, Oct. 1998, pp. 541–5.

[83] M. R. Naphade, M.M. Yeung, and B.-L. Yeo, "A novel scheme for fast and efficient video sequence matching using compact signatures," in *Proceedings of SPIE – Storage and Retrieval for Media Databases 2000*, San Jose, CA, January 2000, vol. 3972, pp. 564–572.

[84] M.A. Smith and T. Kanade, "Video skimming and characterization through the combination of image and language understanding," in *Proceedings 1998 IEEE International Workshop on Content-Based Access of Image and Video Database*, Bombay, India, Jan. 1998, pp. 61–70.

[85] J.M. Gauch et al., "Real time video scene detection and classification," *Information Processing & Management*, vol. 35, no. 3, pp. 381–400, 1999.

[86] S.-F. Chang, W. Chen, and H. Sundaram, "VideoQ: a fully automated video retrieval system using motion sketches," in *Proceedings Fourth IEEE Workshop on Applications of Computer Vision*, Princeton, New Jersey, Oct. 1998, pp. 270–1.

[87] S. Park, J.-S. Cho, and K.-H. Hyun, "Indexing technique for similarity matching in large video databases," in *Proceedings of SPIE – Storage and Retrieval for Media Databases 2002*, San Jose, CA, January 2002, vol. 4676, pp. 214–22.

[88] E. Kushilevitz, R. Ostrovsky, and Y. Rabani, "Efficient search for approximate nearest neighbor in high dimensional spaces," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, May 1998, pp. 614–23.

[89] Jon M. Kleinberg, "Two algorithms for nearest-neighbor search in high dimensions," in *Proceedings of the Twelveth Annual ACM Symposium on Theory of Computing*, May 1997, pp. 599–608.

[90] J. Barros et al., "Using the triangle inequality to reduce the number of comparisons required for similarity-based retrieval," in *Proceedings of SPIE – Storage and Retrieval for Still Image and Video Databases*, San Jose, CA, Feb 1996, vol. 2670, pp. 392–403.

[91] G.R. Hjaltason and H. Samet, "Contractive embedding methods for similarity searching in metric spaces," Tech. Rep. CS-TR-4102, Computer Science Department, University of Maryland, College Park, USA, Jan 2000.

[92] G. Hristescu and M. Farach-Colton, "Cofe: a scalable method for feature extraction from complex objects," in *Proceedings of Second International Conference on Data Warehousing and Knowledge Discovery, DaWaK 2000*, London, UK, Sept 2000, pp. 358–71.

[93] P.M. Aoki, "Generalizing "search" in generalized search trees," in *Proceedings of the 14th International Conference on Data Engineering*, Orlando, FL, Feb 1998, pp. 380–9.

[94] R. L. Scheaffer, W. Mendenhall III, and R. L. Ott, *Elementary Survey Sampling*, Duxbury Press, fifth edition, 1996.

[95] P. S. Levy and S. Lemeshow, *Sampling of Populations Methods and Applications*, John Wiley & Sons, Inc., third edition, 1999.

[96] F. Aurenhammer, "Voronoi diagrams – a survey of a fundamental geometric data structure," *Computing Surveys*, vol. 23, no. 3, pp. 345–405, Sept. 1991.

[97] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill computer science series, 1983.

[98] NIST, http://trec.nist.gov, *Text REtrieval Conference (TREC) Home page*.

[99] E. M. Voorhees and D. Harman, "Overview of the seventh text retrieval conference (trec-7)," in *Proceedings of the 7th Text Retrieval Conference (TREC-7)*, November 1998.

[100] C. J. van Rijsbergen, *Information retrieval*, Butterworth & Co (Publishers) Ltd, second edition, 1979.

[101] S. Deerwester, S.T. Dumas, G.W. Furnas, T.K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, Sept. 1990.

[102] AltaVista, http://www.altavista.com, *AltaVista Image, Audio and Video search*.

[103] *Ditto.com*, http://www.ditto.com.

[104] Scour Inc., http://www.scour.net, *Scour*.

[105] AltaVista Company, "Press release: Altavista extends multimedia search capabilities with rich new content and web's largest multimedia index," http://doc.altavista.com/company_info/press/pr020700.html, February 2000.

[106] Inktomi Corp., "Inktomi webmap," http://www2.inktomi.com/webmap, January 2000.

[107] S. Lawrence and L. Giles, "Accessibility and distribution of information on the web," *Nature*, vol. 400, pp. 107–9, July 1999.

[108] W. Cohen, H. Kautz, and D. McAllester, "Hardening soft information sources," in *Knowledge Discovery and Data Mining*, 2000, pp. 255–259.

[109] Alvaro E. Monge and Charles Elkan, "An efficient domain-independent algorithm for detecting approximately duplicate database records," in *Proceedings of SIGMOD WOrkshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'97)*, 1997.

[110] N. Shivakumar, *Detecting Digital Copyright Violations on the Internet*, Ph.D. thesis, Stanford University, August 1999.

[111] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, "Trawling the web for emerging cyber-communities," in *Proceedings of the Eight International World Wide Web Conference*, May 1999, pp. 1481–93.

[112] A.Z. Broder et al., "Graph structure in the web," in *Proceedings of the Ninth International World Wide Web Conference*, May 2000.

[113] G.C. Langelaar, I. Setyawan, and R.L. Lagendijk, "Watermarking digital image and video data. a state-of-the-art overview," *IEEE Signal Processing Magazine*, vol. 17, no. 5, pp. 20–46, Sept 2000.

[114] R.A. Fisher, "The user of multiple measurements in taxonomic problems," *Ann. Eugenics*, vol. 7, no. 2, pp. 111–132, 1936.

[115] F. Murtagh, "Comments on "parallel algorithms for hierarchical clustering and cluster validity"," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 10, pp. 1056–8, Oct 1992.

[116] B.S. Everitt, *Cluster Analysis*, Halsted Press, third edition, 1993.

[117] S. Pettie and V. Ramachandran, "An optimal minimum spanning tree algorithm," in *Proceedings 27th International Colloquium on Automata, Languages and Programming*, July 2000, vol. LNCS 1853, pp. 49–60.

[118] Bela Bollobas, *Random Graphs*, Academic Press, 1985.

[119] S.N. Bernstein, *The Theory of Probabilities*, Gastehizdat Publishing House, 1946.