Rheinisch Westfälische Technische Hochschule Aachen
Data Management and Data Exploration Group

Bachelor Thesis

Content-based Video Similarity Search Using Feature Signatures

Daniel Sabinasz

First advisor: Prof. Dr. T. Seidl
Second advisor: Prof. Dr. M. Jarke

Aachen, March 26, 2015

Erklärung

Ich versichere, die vorliegende Arbeit selbständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Aachen, den 26.03.2015

Abstract

Given an input video, how can we find visually similar videos from a database efficiently? This thesis deals with a novel approach for this problem. At the core of the approach lies a distance function that operates on so-called *feature signatures*, which represent a video as a set of cluster centers along with the cluster size, obtained by applying a clustering algorithm to a set of previously extracted feature vectors, thereby comprising a compressed yet expressive representation of the video. We consider two different distance-based similarity measures on these feature signatures, the *Earth Mover's Distance* and the *Signature Quadratic Form Distance*. In order to find the k-Nearest Neighbors of a video with respect to these distances efficiently, we consider several indexing methods, including *Metric Indexing* and *Multi-Step query processing* through lower-bounds. The new approach is evaluated and compared to existing approaches with respect to both retrieval quality and efficiency on different datasets. Experiments have shown that the novel approach yields considerably higher effectiveness than the competitive approaches at the expense of a worse efficiency.

The method is based on research done at the *Lehrstuhl für Informatik 9* at *RWTH Aachen University* by Prof. Thomas Seidl, Merih Seran Uysal and Dr. Christian Beecks with contributions from myself.

Contents

Chapter 1

Introduction

The explosion of content on the internet during the last decades has left the world of retrieval and mining of multimedia data with unprecedented challenges, especially with respect to the enormous amount of video content generated every day. For instance, about 100 hours of video are uploaded to YouTube every minute, whilst about 140.000 hours of video are watched every minute (cf. [You15]). There is a demand for this video data to be processed and indexed in order to make it available for different types of queries, whilst ensuring acceptable response times.

An arguably important task is the retrieval of video clips that are visually similar to a certain query clip. We define two videos to be *visually similar* if they depict similar contents in a similar temporal ordering. So far, this task has gained comparatively little research recognition.

Most of the existing video search engines only allow for text-based queries and the search only considers metadata such as the video title, a description text or user-specified tags. The content of the video is not taken into account. Such systems are very limited with respect to the types of queries that are possible, and with respect to the actual relevance of the retrieved results. The systems that do consider the video content are mostly systems for detecting near-duplicate versions of a video clip - i.e. video clips that were created by taking a source clip and performing certain edit operations like change of color, resolution, contrast, frame disordering, etc. They are usually not applicable for finding videos that are visually similar to a query video, unless they are a near-duplicate version of the query video.

In the context of this bachelor thesis, we introduce *FlexVis* (abbreviation for *Flexible Video Signature*), a new method for retrieving visually similar videos to a specified query video. FlexVis is based on so-called *feature signatures*, which comprise an expressive summary of the video's contents that is more compact than the video itself, allowing for an efficient

comparison between videos. The main task in the course of this bachelor thesis was to implement FlexVis, along with 4 competitive methods for video similarity search and evaluate them according to their effectiveness and efficiency on different datasets.

This thesis is outlined as follows. In Chapter 2, we give some general preliminaries for the field of content-based multimedia retrieval. Chapter 3 contains an overview of some of the existing work for video similarity search, along with a discussion of the disadvantages of these methods and suggestions for improvement. In Chapter 4, we introduce FlexVis, our new method for video similarity search. In Chapter 5, we give a brief overview of RCVS, a web-based video search engine that was developed in the course of this bachelor thesis which allows users to upload a video and delivers visually similar videos from a database, using FlexVis or one of the presented competitive methods. Chapter 6 demonstrates how FlexVis can be adapted for the task of subclip search, i.e. identifying a short query video within a longer video. In Chapter 7 we present experimental results on the effectiveness and efficiency of FlexVis in comparison to the competitive methods. Finally, in Chapter 8, we conclude this thesis and give an outlook into future research.

Chapter 2

Preliminaries

This chapter introduces some fundamental preliminaries for the problem of retrieving similar multimedia objects to a given query object. We will review how similarity between multimedia objects can be formalized and which types of queries exist with respect to this formalization.

2.1 Quantifying Similarity

In order to retrieve similar multimedia objects to a given query object, we need a way to compare the query object to the database objects and quantify the similarity or dissimilarity numerically. There are many ways in which similarity or dissimilarity between two objects can be measured, and it is highly dependent on the nature of the compared objects and on the aspects which we want to compare. For example, videos could be compared with respect to their visual content, their auditory content or meta-data such as the title or a description text.

The most common way to model similarity is by means of a *distance function*. A distance function assigns high values to objects that are dissimilar and small values to objects that are similar, reaching 0 when the two compared objects are the same. Mathematically, a distance function is defined as follows (cf. [DD09]).

Definition 2.1.1. Let X be a set. A function $\delta : X \times X \to \mathbb{R}$ is called a *distance function* if it holds for all $x, y \in X$:

- $\delta(x, x) = 0$ (reflexivity)
- $\delta(x, y) = \delta(y, x)$ (symmetry)

- $\delta(x, y) \geq 0$ (non-negativity)

When it comes to efficient query processing, as we will describe in Chapter 4, it is useful if the utilized distance function is a *metric* (cf. [DD09]).

Definition 2.1.2. Let $\delta : X \times X \to \mathbb{R}$ be a distance function. δ is called a *metric* if it holds for all $x, y, z \in X$:

- $\delta(x, y) = 0 \iff x = y$ (identity of indiscernibles)
- $\delta(x, y) \leq \delta(x, z) + \delta(z, y)$ (triangle inequality)

An alternative way to model similarity between two objects is by means of a *similarity function*, which assigns small values to objects that are dissimilar and larger values to objects that are more similar, reaching its maximum when the two compared objects are the same (cf. [BS13]).

Definition 2.1.3. Let X be a set. A function $s : X \times X \to \mathbb{R}$ is called a *similarity function* if it is symmetric and if it holds for all $x, y \in X$ that $s(x, x) \geq s(x, y)$ (maximum self-similarity).

Examples for different distance functions and similarity functions on videos are presented in Chapters 3 and 4.

2.2 Query Types

Once we have modeled the similarity for pairs of multimedia objects by means of a distance or similarity function, we can reformulate the problem of retrieving similar objects to the query object by utilizing such a function. A prominent query type is the so-called *range query*, which retrieves all database objects for which the distance to the query object lies below a certain threshold ϵ. The formal definition is given below (cf. [BS13]).

Definition 2.2.1. Let X be a set of objects, $\delta : X \times X \to \mathbb{R}$ be a distance function, $DB \subseteq X$ be a database of objects, $q \in X$ be a query object and $\epsilon \in \mathbb{R}$ be a search radius. The *range query* $range_\epsilon(q, \delta, X)$ is defined as

$$range_\epsilon(q, \delta, X) = \{x \in X \mid \delta(q, x) \leq \epsilon\}$$

This definition can be adapted for similarity functions by replacing the $\leq$-sign with a $\geq$-sign, which retrieves all objects for which the similarity to the query object lies above a certain threshold.

For range queries, it is hard to determine a suitable threshold ϵ to yield a result set of a desired size. When ϵ is too low, the result set might be very small or even empty. On the other hand, when choosing it too large, the result set might come near to including the entire database. This problem can be solved by issuing a *k-Nearest Neighbor Query* (short: kNN query) instead. In this query type, we specify the desired number of retrieved objects k instead of a distance threshold. If we assume that the distances between the query object and the database objects are pairwise distinct, the k-Nearest Neighbors are the k objects that have the smallest distance to the query object. The formal definition is given below (cf. [SK98]).

Definition 2.2.2. Let X be a set of objects, $\delta : X \times X \rightarrow \mathbb{R}$ be a distance function, $DB \subseteq X$ be a database of objects, $q \in X$ be a query object and $k \in \mathbb{N}, k \leq |DB|$. We define the *k-Nearest Neighbors* of q w.r.t. δ as the smallest set $NN_q(k) \subseteq DB$ with $|NN_q(k)| \geq k$ such that the following holds:

$$\forall o \in NN_q(k), \forall o' \in DB - NN_q(k) : \delta(o, q) < \delta(o', q)$$

Given these fundamental preliminaries for content-based multimedia retrieval, we can now continue to describe some of the existing methods for video similarity search.

Chapter 3

Related Work

In the following, we give a description of some of the competitive methods for content-based video retrieval. Although these methods were originally introduced for the task of near-duplicate detection, they show some potential for visual similarity search as well. The individual methods are characterized by some compact representation scheme of the video's contents, a similarity or distance measure on these representation schemes and, for some of them, an indexing method that allows for efficient kNN or range query processing. All methods assume that the video is given as a sequence $X = X_1, ..., X_n$ of vectors $X_i \in \mathbb{R}^d$ (e.g. framewise RGB histograms), each vector representing one video frame.

In addition to describing the methods, we also state some of their shortcomings. We will describe in Chapter 7 how these shortcomings affect the effectiveness of the methods on different video databases. For some of these shortcomings, we give suggestions for improvement.

3.1 Video Triplets (ViTri)

Shen et al. [SOZ05] suggest a representation scheme for videos called *Video Triplet* (short: ViTri), along with a similarity measure that operates on these Video Triplets and thereby tries to approximate a similarity measure called the *Ideal Video Similarity*.

3.1.1 Ideal Video Similarity

First proposed in [CZ03], the Ideal Video Similarity (short: IVS) is a measure of similarity between two videos based on the percentage of visually similar frames. Its definition is given below.

Definition 3.1.1. Given two videos represented as sets of frame vectors X and Y, a distance function $d(x, y)$ between frames and a frame similarity threshold $\epsilon \in \mathbb{R}^{\geq 0}$, the *Ideal Video Similarity* is defined as

$$IVS(X,Y) = \frac{\sum_{x \in X} 1_{\{y \in Y : d(x,y) \leq \epsilon\}} + \sum_{y \in Y} 1_{\{x \in X : d(x,y) \leq \epsilon\}}}{|X| + |Y|}$$

where $1_A = 1$ if A is not empty, 0 otherwise.

This similarity measure counts the frames in X that have a similar frame in Y and vice versa, which is then normalized by the total number of frames. The time complexity of a single computation of the IVS lies in $\mathcal{O}(|X| \cdot |Y|)$, which makes it infeasible to compute in practice. In order to allow for an efficient comparison between videos, the ViTri model first summarizes the set of frame vectors into a compact representation scheme.

3.1.2 Video Representation

The proposed idea to approximate the IVS is to cluster the set of video frames and represent each cluster as a hypersphere, called a *Video Triplet* (short: ViTri). A video is then represented as a set of these Video Triplets. This compact representation allows to approximate the IVS using the volume of overlap between the hyperspheres, which can be computed more efficiently than the IVS itself.

Given a video, represented as a set X of frame vectors, X is clustered in such a way that the inter-frame distances within a cluster are at most ϵ, which is a user-specified parameter representing the desired frame similarity threshold. Each cluster C is then represented as a so-called *Video Triplet*, which is a triplet *(O, R, D)*. O denotes the *position* of the cluster, i.e. the mean of all vectors in the cluster. R denotes the *refined radius* of the cluster. To this end, let μ and σ denote the mean and standard deviation of the distance $d(c, O)$ for all $c \in C$ and let $r = \max_{c \in C} d(c, O)$ be the cluster radius. Then the refined radius is given as $R = \min(r, \mu + \sigma)$, which is more robust to outliers than r. Finally, D denotes the *density* of the cluster, i.e. the number of frames in the cluster divided by the volume of the hypersphere with center O and radius R: $D = |C| / V_{hypersphere}(O, R)$

The proposed clustering algorithm works by applying k-means (cf. [HKP06]) with parameter k = 2 to X, which splits it into 2 clusters, and then recursively applying the same algorithm to the resulting 2 clusters until we arrive at clusters that have a refined radius of at most $\epsilon/2$.

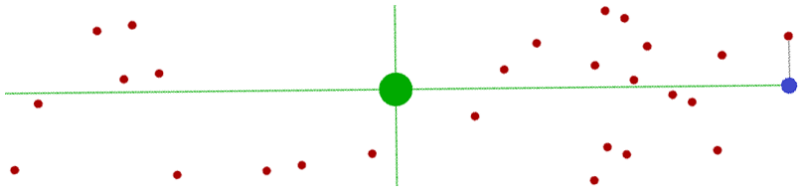


Figure 3.1: Computation of the reference point

3.1.3 Similarity Measure

Given two frame clusters C_1, C_2 and their corresponding ViTris $ViTri_1 = (O_1, R_1, D_1)$, $ViTri_2 = (O_2, R_2, D_2)$, the number of frames that are in both C_1 and C_2 is estimated by the volume of overlap between the hyperspheres, multiplied by the minimal density:

$$sim(ViTri_1, ViTri_2) = V_{intersection}((O_1, R_1), (O_2, R_2)) \cdot \min(D_1, D_2)$$

The overall similarity between two videos is then calculated by summing over the similarity between all pairs of Video Triplets from the two videos, thereby approximating the IVS.

3.1.4 Indexing

The authors propose a B^+-tree-based pruning method for efficient query processing. To this end, a global reference point O_{ref} is chosen as the furthermost projection of all ViTri positions from all database videos onto the first principal component (i.e. the direction along which the points exhibit the largest variance) by applying Principal Component Analysis (cf. [Jol02]) to the set of all ViTri positions and choosing O_{ref} to be the projection on the first principal component that has the largest distance to the data center. Figure 3.1 visualizes the computation of the reference point for 2-dimensional points. The points are depicted as red dots, the principal components as green lines and the reference point as a blue circle, corresponding to the outermost projection onto the first principal component.

All Video Triplets are then indexed in a B^+-tree, using their distances to the reference point as keys. Given two Video Triplets $ViTri_Q = (O_Q, R_Q, D_Q)$ and $ViTri_P = (O_P, R_P, D_P)$, it follows from the triangle inequality that $ViTri_Q$ and $ViTri_P$ have 0 similarity (i.e. they do not overlap) if $|d(O_Q, O_{ref}) - d(O_P, O_{ref})| \geq R_Q + \epsilon/2$. Hence, given a query video triplet set Q, we can generate a candidate set of similar videos by performing range queries on the B^+-tree for every $ViTri_Q \in Q$ with search range $[d(O_Q, O_{ref}) - R_Q - \epsilon/2, d(O_Q, O_{ref}) + R_Q + \epsilon/2]$. All other Video Triplet sets are guaranteed to have a similarity of 0, since none of their Video Triplets overlap with the query Video Triplets. This allows us to perform the kNN search on the candidate set as opposed to the entire database.

3.1.5 Issues

Since ViTri tries to approximate the IVS, it has the same conceptual issues. A significant disadvantage of IVS is that it does not take into account the temporal ordering of the video frames. When judging the similarity between two videos subjectively, the temporal ordering of the frames usually plays a role. The ViTri model, however, is invariant under frame disordering since it is based on sets of frame vectors as opposed to sequences of frame vectors. Another shortcoming is the fact that the similarity is highly dependent on the difference between the video lengths, i.e. the number of frames. For example, if we take a video X and loop it n times to generate a video Y, the similarity between X and Y approaches 0 for $n \to \infty$, even though X and Y can reasonably be considered as similar. Furthermore, IVS is fairly non-robust, i.e. slight changes in a video can cause a significant change in its similarity to other videos. For example, if we take a video X and create videos $Y_1, Y_2, ...$ by gradually increasing the brightness slightly, at some point the change in brightness will cause all frame distances to exceed the parameter ϵ, resulting in a jump of the similarity from $sim(X, Y_i) = 1$ (perfectly similar) to $sim(X, Y_{i+1}) = 0$ (completely dissimilar), even though $sim(Y_i, Y_{i+1}) = 1$.

Apart from this, it is unclear whether or not ViTri is a good approximation of IVS. The authors of [SOZ05] neglect to present experimental results as to how accurate the ViTri similarity is in comparison to IVS.

3.2 Bounded Coordinate Systems (BCS)

Huang et al. [HSS+09] propose to represent a video by means of a so-called *Bounded Coordinate System*, which is composed of the mean of all frame vectors, the directions along which the frame vector distribution exhibits the largest variance and the standard deviation along these directions. Two videos are then compared by means of a distance function on their Bounded Coordinate Systems that measures the amount of translation, rotation and scaling that is necessary to match one Bounded Coordinate System onto the other one.

3.2.1 Video Representation

Given a set of frame vectors $X \subset \mathbb{R}^d$, its Bounded Coordinate System is defined as $BCS(X) = (O, \phi_1, ..., \phi_d)$ where d is the dimensionality of the feature space, $O \in \mathbb{R}^d$ is the mean of all vectors in X and $\phi_i \in \mathbb{R}$ is the i-th *Bounded Principal Component* (BPC), which is a vector whose direction lies along the i-th principal component and whose length

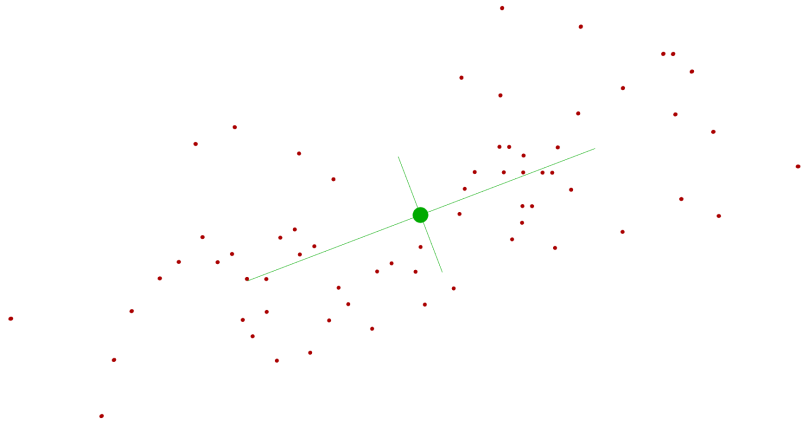


Figure 3.2: Visualization of a Bounded Coordinate System for 3-dimensional frame histograms

is $||\phi_i||_2 = \sigma_i$, which is the standard deviation of the Euclidean distance between O and the projections of all vectors in X onto the i-th principal component.

Figure 3.2 shows a visualization of a Bounded Coordinate System for 3-dimensional color histograms, extracted from a video with a duration of 6 seconds at 10 frames per second. The color histograms are depicted as red spheres, their origin is depicted as a green sphere and the Bounded Principal Components are depicted as green lines. Each Bounded Principal Component is visualized twice here, because there are two possible directions. Since histograms are L_1-normalized, they lie in a plane and, hence, have a 2-dimensional intrinsic dimensionality. Thus, the third principal component (which would be orthogonal to the other two principal components) has a variance of 0 and is therefore not visible in this visualization.

3.2.2 Distance Measure

The authors propose to compare two BCSs $BCS(X) = (O^X, \phi_1^X, ..., \phi_{d^X}^X)$ and $BCS(Y) = (O^Y, \phi_1^Y, ..., \phi_{d^Y}^Y)$ by means of the following distance function:

$$D(BCS(X), BCS(Y)) = \begin{cases} D'(BCS(X), BCS(Y)) & \text{if } d^X \geq d^Y \\ D'(BCS(Y), BCS(X)) & \text{else} \end{cases}$$

$$\text{with } D'(BCS(X), BCS(Y)) = ||O^X - O^Y|| + \frac{1}{2}(\sum_{i=1}^{d^Y} ||\phi_i^X - \phi_i^Y|| + \sum_{i=d^Y+1}^{d^X} ||\phi_i^X||)$$

The first summand measures the amount of translation that is necessary to move the origin of $BCS(X)$ to $BCS(Y)$. The other summands measure the amount of scaling and rotation that is necessary to match the respective i-th BPCs onto each other.

The authors of [HSS+09] claim $D(BCS(X), BCS(Y))$ to be a metric distance function but neglect to give a proof for this. A proof is given below.

Theorem 3.2.1. *$D(BCS(X), BCS(Y))$ is a metric distance function.*

Proof. Let $BCS(X), BCS(Y), BCS(Z)$ be bounded coordinate systems. $D(BCS(X), BCS(Y))$ fulfills the properties of a metric distance function as per Definition 2.1:

- Non-negativity: $D(BCS(X), BCS(Y)) \geq 0$ by non-negativity of the ground distance.
- Reflexivity: $D(BCS(X), BCS(X)) = 0$ by reflexivity of the ground distance.
- Symmetry:

  Assume $d^X \geq d^Y$:
  $D(BCS(X), BCS(Y)) = D'(BCS(X), BCS(Y)) = D(BCS(Y), BCS(X))$.

  Assume $d^X < d^Y$:
  $D(BCS(X), BCS(Y)) = D'(BCS(Y), BCS(X)) = D(BCS(Y), BCS(X))$.
- Identity of indiscernibles:

$$\begin{aligned}
& D(BCS(X), BCS(Y)) = 0 \\
\iff\ & ||O^X - O^Y|| + \frac{1}{2}(\sum_{i=1}^{d^Y} ||\phi_i^X - \phi_i^Y|| + \sum_{i=d^Y+1}^{d^X} ||\phi_i^X||) = 0 \\
\iff\ & O^X = O^Y \wedge d^X = d^Y \wedge \forall 1 \leq i \leq d^x : \phi_i^X = \phi_i^Y \\
\iff\ & BCS(X) = BCS(Y)
\end{aligned}$$

- Triangle inequality:

  We assume that $d^X \geq d^Z \geq d^Y$. Similar proofs can be given for all orderings of d^X, d^Y and d^Z, but they are omitted at this point.

$$\begin{aligned}
& D(BCS(X), BCS(Z)) + D(BCS(Z), BCS(Y)) \\
=\ & ||O^X - O^Z|| + \frac{1}{2}(\sum_{i=1}^{d^Z} ||\phi_i^X - \phi_i^Z|| + \sum_{i=d^Z+1}^{d^X} ||\phi_i^X||) \\
+\ & ||O^Z - O^Y|| + \frac{1}{2}(\sum_{i=1}^{d^Y} ||\phi_i^Z - \phi_i^Y|| + \sum_{i=d^Y+1}^{d^Z} ||\phi_i^Z||) \\
=\ & ||O^X - O^Z|| + ||O^Z - O^Y|| \\
+\ & \frac{1}{2}\sum_{i=1}^{d^Z} ||\phi_i^X - \phi_i^Z|| + \frac{1}{2}\sum_{i=1}^{d^Y} ||\phi_i^Z - \phi_i^Y|| + \frac{1}{2}\sum_{i=d^Z+1}^{d^X} ||\phi_i^X|| + \frac{1}{2}\sum_{i=d^Y+1}^{d^Z} ||\phi_i^Z||
\end{aligned}$$

$$
\begin{aligned}
&= ||O^X - O^Z|| + ||O^Z - O^Y|| \\
&+ \frac{1}{2}\sum_{i=1}^{d^Y}(||\phi_i^X - \phi_i^Z|| + ||\phi_i^Z - \phi_i^Y||) + \frac{1}{2}\sum_{i=d^Y+1}^{d^Z}(||\phi_i^X - \phi_i^Z|| + ||\phi_i^Z||) + \frac{1}{2}\sum_{i=d^Z+1}^{d^X}||\phi_i^X|| \\
&\geq ||O^X - O^Y|| + \frac{1}{2}(\sum_{i=1}^{d^Y}||\phi_i^X - \phi_i^Y|| + \sum_{i=d^Y+1}^{d^X}||\phi_i^X||) \\
&= D(BCS(X), BCS(Y))
\end{aligned}
$$

The last inequality follows since $||.||$ is a norm and, hence, $d(v, w) = ||v - w||$ fulfills the triangle inequality.

□

3.2.3 Indexing

For efficient range query processing, the authors propose a similar B^+-tree-based pruning technique as Video Triplet. Instead of using a single reference point, this pruning techniques uses two reference points R_1 and R_2 corresponding to the furthermost projections on either side of the first principal component. The distances of the BCSs to R_1 are indexed by a B^+-tree that contains the distances between the BCSs to R_2 at the leaf level. Given a query Q and a search radius r, a range query with range $[D(Q, R_1) - r, D(Q, R_1) + r]$ is carried out on the B^+-tree. On the resulting BCSs, a further range query with range $[D(Q, R_2) - r, D(Q, R_2) + r]$ is carried out. The original range query is then performed on the resulting candidate BCSs.

The proposed indexing method is only applicable for range queries and the authors do not propose a method for efficient kNN query processing. One way to process kNN queries using this indexing method is to perform successive range queries with increasing search radius until the result set contains at least k elements. The kNN query can then be performed on this range query result set.

3.2.4 Issues

As with ViTri, BCS suffers from the fact that it disregards the temporal ordering of the frames. Another issue lies in the distance measure: Given two BCSs X and Y, it compares the first BPC of X with the first BPC of Y, the second BPC of X with the second BPC of Y, etc. The only thing the pairs of compared BPCs have in common is the fact that they are the BPCs with the i-th largest variance of their respective BCS. If the variances do not differ significantly, then this is not a meaningful thing to do, since the ranking of BPCs

is non-robust to slight changes in the data. Adding a single frame to one of the videos could make two BPCs flip positions in the ranking, causing a high jump in the distance. A more meaningful and robust distance measure could compare each BPC from X to its most similar match in Y. This would, however, make the time complexity quadratic in the number BPCs.

3.3 Video Distance Trajectories (VDT)

Huang et al. ([HWS+09], [HSS+10]) suggest to map a video, given as a sequence of frame vectors, onto a sequence of real numbers, each number representing the distance of the corresponding frame vector to a chosen reference point. This sequence of distance values is approximated by a piecewise linear function to further summarize the representation and speed up the distance computation.

3.3.1 Video Representation

Definition 3.3.1. Given a sequence of frame vectors $X = X_1, ..., X_n$ with $X_i \in \mathbb{R}^d$ for $1 \leq i \leq n$, a reference point $o \in \mathbb{R}^d$ and a distance function $\delta : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, the *Video Distance Trajectory* $VDT(X)$ is defined as

$$VDT(X) = \delta(X_1, o), ..., \delta(X_n, o)$$

The reference point o is chosen individually for each video in such a way that the inter-frame distance information is maximally preserved. According to [HSS+10], such an optimal reference point lies along the first principal component and outside of the data range. It is hence computed in the same way as the reference point for the Video Triplet Indexing (cf. Section 3.1.4).

Once the VDT has been computed, it is segmented in such a way that each segment corresponds to a shot, i.e. a part of the video that is enclosed by camera cuts. The camera cuts are detected by checking whether the change in the distance from one frame to the next frame lies above a certain threshold ζ, i.e. if $|\delta(X_i, o) - \delta(X_{i+1}, o)| > \zeta$. Each segment of the VDT is approximated by means of a *Linear Smoothing Function*, which is a line that best fits the segment with respect to the sum of square errors.

Definition 3.3.2. Let $s = \delta(X_i, o), \delta(X_{i+1}, o), ..., \delta(X_{i'}, o)$ be a VDT segment. The *Linear Smoothing Function* LSF(s) is defined as a triplet (α, β, l) where α and β correspond to the offset and slope of the line $d_j = \alpha + \beta \cdot j$ that minimizes the sum of square error $\sum_{j=i}^{i'} (d_j - \delta(X_j, o))^2$ and $l = i' - i + 1$ corresponds to the length of the segment.

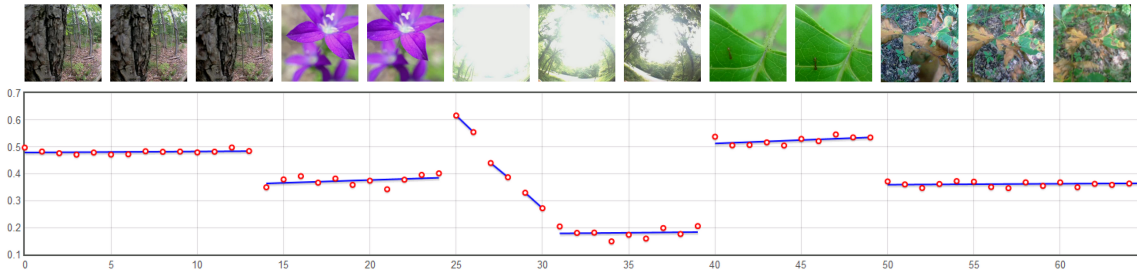
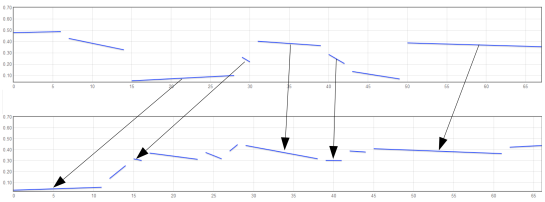
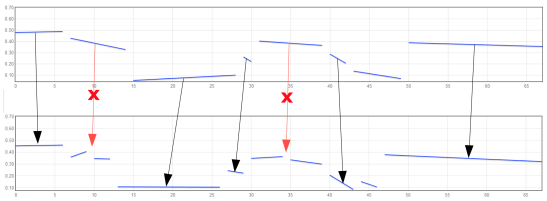


Figure 3.3: Visualization of a Video Distance Trajectory and its sequence of Linear Smoothing Functions



(a) Matching between query (top) and some unrelated database video (bottom). The length of matched LSFs sums up to 41, corresponding to a WES of $\frac{41}{68} \approx 0.6$



(b) Matching between query (top) and a near duplicate version of the query (bottom). The length of matched LSFs sums up to 38, corresponding to a WES of $\frac{38}{68} \approx 0.56$

Figure 3.4: Visualization of Weighted Edit Similarity matchings

Figure 3.3 shows the VDT of an example video of 6 seconds with 5 shots, sampled at 10 frames per second. The distance values $\delta(X_i, o)$ are depicted as red dots and the LSFs as blue lines. Note that due to lack of space only 2 frames per second are included in the figure.

3.3.2 Similarity Measure

In order to compare sequences of LSFs, the authors first define a similarity function on LSFs as follows:

Definition 3.3.3. Let $LSF_i = (\alpha_i, \beta_i, l_i)$ and $LSF_j = (\alpha_j, \beta_j, l_j)$ be two Linear Smoothing Functions. Their similarity $sim(LSF_i, LSF_j)$ is defined as

$$sim(LSF_i, LSF_j) = sim_{\updownarrow}(LSF_i, LSF_j) \cdot sim_{\angle}(LSF_i, LSF_j) \cdot sim_{\leftrightarrow}(LSF_i, LSF_j)$$

where

$$sim_{\updownarrow}(LSF_i, LSF_j) = 1 - \frac{|\alpha_i - \alpha_j|}{\alpha_{\max} - \alpha_{\min}}$$

$$sim_{\angle}(LSF_i, LSF_j) = 1 - \frac{|\beta_i - \beta_j|}{\pi}$$

$$sim_{\leftrightarrow}(LSF_i, LSF_j) = 1 - \frac{|l_i - l_j|}{\max(l_i, l_j)}$$

$\alpha_{\max}$ and $\alpha_{\min}$ correspond to the maximum and minimum α of the two LSF sequences from which the compared LSFs are taken.

According to this definition, the similarity of two LSFs is measured with respect to the amount of vertical translation, rotation and stretching that is necessary to match one LSF onto the other one. Using the similarity measure on LSFs, the authors define a similarity measure on LSF sequences called *Weighted Edit Similarity* (short: WES). This similarity measure distinguishes between a query sequence Q and a sequence V in which to search for the query. The idea is to match the LSFs of Q onto the LSFs of V. Two LSFs can be matched if their similarity lies above a certain threshold ϵ. Of all the possible matchings, an optimal matching is computed that maximizes the length of the matched LSFs. The Weighted Edit Similarity then corresponds to the percentage of query length that has been matched. The formal definition is given below.

Definition 3.3.4. Let $Q = (LSF_1^Q, LSF_2^Q, ..., LSF_m^Q)$ and $V = (LSF_1^V, LSF_2^V, ..., LSF_n^V)$ be two sequences of Linear Smoothing Functions. The *Weighted Edit Similarity* $WES(V, Q)$ is defined as

$$WES(V,Q) = \begin{cases} 0 & \text{if } n = 0 \vee m = 0 \\ WES(head(V), head(Q)) + \frac{l_i^Q}{m} & \text{if } sim(LSF_1^Q, LSF_1^V) \geq \epsilon \\ \max \begin{cases} WES(head(V), Q) \\ WES(V, head(Q)) \end{cases} & \text{else} \end{cases}$$

where $head(Q) = (LSF_2^Q, ..., LSF_m^Q)$ and $head(V) = (LSF_2^V, ..., LSF_n^V)$.

Example matchings between a query LSF sequence and two database LSF sequences are given in Figure 3.4.

3.3.3 Issues

One issue with this approach lies in the fact that the reference points are chosen individually for each video. The rationale for this, as explained in [HSS+10], is that this causes the inter-frame distance information to be maximally preserved. However, this causes two problems. When considering a video and a near-duplicate copy of that video, it is likely to happen that the reference point of the first video lies on the other side of the first principal component as the reference point of the second video. This causes the resulting VDTs to be completely dissimilar to each other. On the counter-side, the VDTs of two completely dissimilar videos might be similar if the distances to the reference points are similar, even

Figure 3.5: Example Frame Symbol Sequences for two similar videos

when the reference points have a high distance. This problem can be solved by choosing a global reference point for all videos.

The way in which the VDTs are segmented is suboptimal. It is checked whether the distance between two consecutive frames exceeds a certain threshold and if it does, the next segment starts. This is very unrobust to outliers: If only a single frame within a shot exceeds this threshold, two segments are created instead of one. Since the WES only allows for 1 to 1 matchings, this poses a problem since a segment represented by one LSF can not be matched to two segments represented by two similar LSFs. This is visualized in Figure 3.4b: The query LSFs at the starting points of the red arrows can not be matched to any LSF in the database video since no single similar LSF exists, even though there are two LSFs which, in combination, are similar to the respective query LSF. Better results might be achieved by using more sophisticated piecewise linear prediction models to approximate the VDT, e.g. Regression Trees, which yield a segmentation that is more robust to outliers than the proposed threshold-based segmentation. Adjusting the similarity measure to allow for matching one query LSF to more than one target LSF or vice versa would also make the approach less prone to differences in the segmentation.

3.4 Frame Sequence Symbolization (FRAS)

Zhou et al. [ZZS07] propose to represent a video as a digital string of cluster IDs. The idea is to compute a global clustering of the frame vectors from all database videos and assign an ID to each cluster. A video, given as a sequence of frame vectors, is then represented as a digital string such that each symbol corresponds to the ID of the cluster to which the frame vector was assigned. The distance between two video strings is measured by a modified version of the edit distance.

3.4.1 Video Representation

According to email correspondence with the authors, the clustering of all frame vectors in the database is carried out using the same algorithm as in [SOZ05], with the difference

that the clusters are extended afterwards to include all points within their radius, which allows for potential overlap among the clusters. Each cluster is represented as a tuple (id, O, r, N), where $id \in \mathbb{N}$ denotes an arbitrarily assigned cluster identifier, O denotes the mean vector of all cluster elements, $r = \max_{c \in C} d(c, O)$ denotes the cluster radius and N denotes the number of cluster elements.

Once the global clustering has been computed, a video, given as a sequence of frame vectors $X = X_1, ..., X_n$, is mapped to its so-called *Frame Symbol Sequence*, which is a string $S = s_1, ..., s_n$ where s_i corresponds to the ID of the cluster whose mean has the smallest distance to X_i, provided that this distance is within the cluster radius r. If this distance is larger than r, s_i is assigned a special symbol $-$, which expresses that this frame is dissimilar to all database frames. Figure 3.5 shows example Frame Symbol Sequences for two videos containing similar frames.

3.4.2 Distance Measure

The authors propose a distance measure on Frame Symbol Sequences called *Probability Edit Distance*, which is a variant of the *Weighted Edit Distance* (cf. [Wik15a]). Given two strings Q and S, the Weighted Edit Distance measures the cost for transforming Q into S by means of inserting, deleting or replacing symbols, where the cost of inserting a symbol s is given by $\delta(\diamond, s)$, the cost for deleting a symbol q is given by $\delta(q, \diamond)$ and the cost for replacing a symbol q by a symbol s is given by $\delta(q, s)$. Its formal definition is given below[1].

Definition 3.4.1. Let $Q = q_1, ..., q_n$ and $S = s_1, ..., s_m$ be two strings, let $start(Q) = q_1, ..., q_{n-1}$ denote the prefix of Q and $last(Q) = q_n$ denote the the last element of Q. Further, let δ be a ground distance on the symbols. The *weighted edit distance* $ED_\delta(Q, S)$ is defined as:

$$ED_\delta(Q,S) = \begin{cases} 0 & \text{if } n = m = 0 \\ \sum_{i=1}^{n} \delta(q_i, \diamond) & \text{if } m = 0 \\ \sum_{i=1}^{m} \delta(\diamond, s_i) & \text{if } n = 0 \\ ED_\delta(start(Q), start(S)) & \text{if } last(Q) = last(S) \\ \min \begin{cases} ED_\delta(start(Q), start(S)) + \delta(last(Q), last(S)) \\ ED_\delta(Q, start(S)) + \delta(\diamond, last(S)) \\ ED_\delta(start(Q), S) + \delta(last(Q), \diamond) \end{cases} & \text{else} \end{cases}$$

[1]The notation of this definition has been adopted from the slides of the lecture *Content-based Multimedia Search* held by Prof. Dr. T. Seidl and Dr. Christian Beecks in the winter term 2014/2015 at RWTH Aachen University

This recurrence equation can be computed in $\mathcal{O}(n \cdot m)$ using dynamic programming. The Probability Edit Distance between Frame Symbol Sequences is defined as the Weighted Edit Distance with a ground distance expressing the dissimilarity of the clusters to which the symbols correspond:

Definition 3.4.2. Let $Q = q_1, ..., q_n$ and $S = s_1, ..., s_m$ be two Frame Symbol Sequences, $C(id)$ be the cluster with the specified ID and $T \in \mathbb{R}$ be a user-specified threshold (denoted *probability threshold*). The *Probability Edit Distance* between Q and S is defined as $ED_\delta(Q, S)$ with

$$\delta(q, \diamond) = \delta(\diamond, s) = 1, \quad \delta(q, s) = \begin{cases} 1 & \text{if } q = - \vee s = - \\ 1 & \text{if } d(q, s) \geq T \\ 0 & \text{else} \end{cases}$$

where $d(q, s) = \frac{|C_(q) - C(s)|}{|C(q)|}$.

According to this definition, the cost for inserting or deleting a frame symbol is 1. When replacing a symbol q by s, $d(q, s)$ is computed, which corresponds to the percentage of elements in $C(q)$ that are not contained in $C(s)$. If this percentage exceeds a threshold T, then the cost for replacing q by s is 1, otherwise it is 0.

3.4.3 Issues

The Probability Edit Distance uses the percentage of non-overlapping area, $d(q, s)$, as a distance measure on frame clusters. But instead of using this distance directly as the ground distance, it is checked whether $d(q, s)$ lies above a certain threshold T and the ground distance is 1 if it does, 0 otherwise. This entails a loss of information and requires us to find a suitable parameter T. No reason is given in [ZZS07] as to why $d(q, s)$ is not used directly as the ground distance. In fact, our experiments have shown that using $d(q, s)$ as the ground distance yields better effectiveness. Apart from that, if there is no overlap between two clusters q and s, then $d(q, s) = 1$, irrespective of how near the clusters are to each other and, hence, how similar they are. A possible way to circumvent this is to simply use the Euclidean Distance between the cluster centers as the ground distance.

We have reviewed some of the state-of-the-art methods for content-based video retrieval and pointed out their shortcomings. In the following chapter, we present our new approach FlexVis.

Chapter 4

FlexVis

This chapter introduces *FlexVis*, our new approach for video similarity search. The idea of our approach is to extract a set of so-called *feature vectors* from the videos, which are vectors from the Euclidean space that describe certain local characteristic properties of the video, and then summarize these feature vectors into a structure called a *feature signature*. The distance between two videos can then be calculated by means of a distance function between feature signatures, for which we use the *Earth Mover's Distance* and the *Signature Quadratic Form Distance*. When processing a kNN query, we use an algorithm that makes use of lower bounds to the distance functions in order to speed up the retrieval process.

4.1 Features for Video Representation

The first particularity of our approach lies in the choice of features for representing the video. Our goal is to extract a set of Euclidean vectors, each of which describes certain local visual properties of the video numerically. This process is depicted visually in Figure 4.1. We first select a certain amount of sample frames from the video (e.g. 10 frames per second). For each of these frames, we select a fixed amount of equidistant sample pixels. Finally, for each sample pixel, we compute an 8-dimensional Euclidean vector $(x, y, L, a, b, \chi, \eta, t)$ describing the pixel and its context.

The first two dimensions of this vector correspond to the x and y coordinates of the pixel inside the frame. The next 3 dimensions correspond to the color of the pixel in the CIE L*a*b* color space, i.e. the lightness, the position between red and green and the position between blue and yellow (cf. [Wik15b]). Additionally, we calculate the contrast χ of a 12 x 12 neighborhood of the pixel as proposed by Tamura et al. in [TMY78], which is a measure of the dynamic range of the colors. Furthermore, we calculate the coarseness η of the pixel

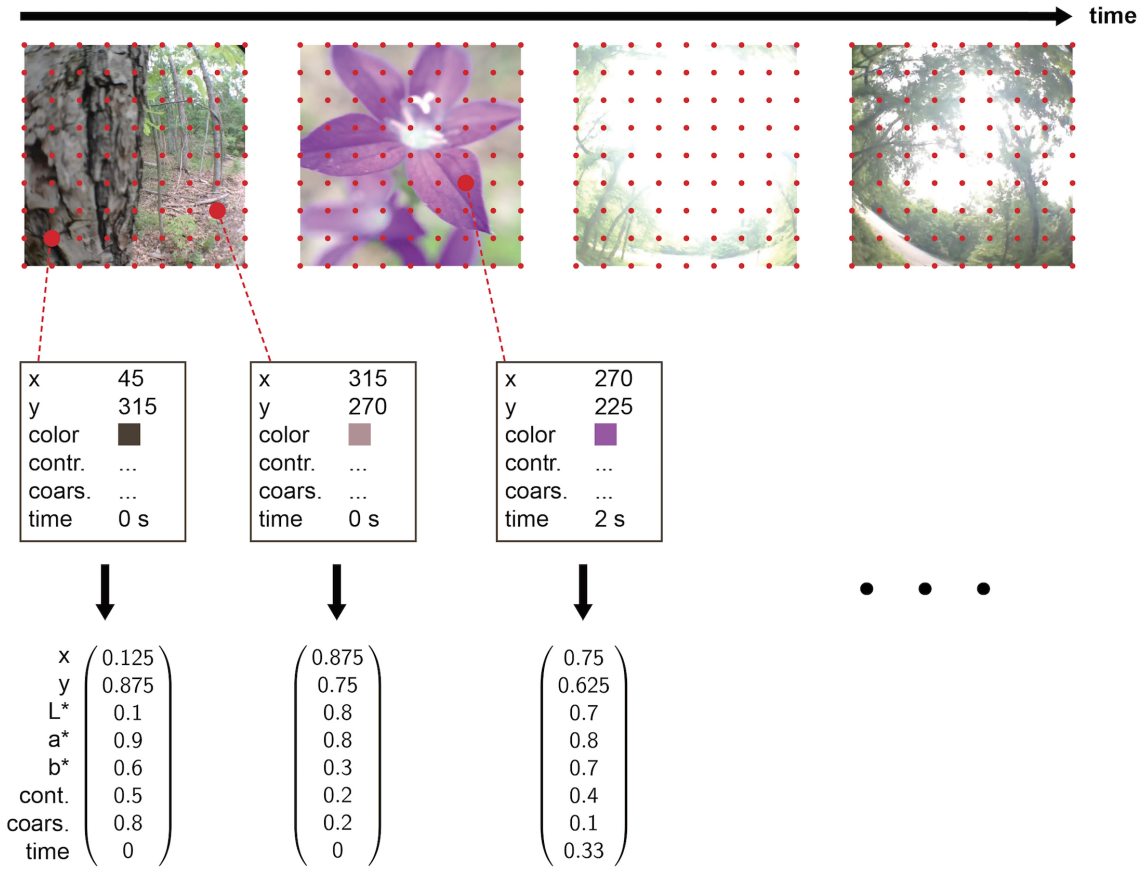


Figure 4.1: Feature extraction on videos

as proposed in [TMY78], which is a measure of how big the structures surrounding that pixel are. Finally, we add the time t of the frame from which the pixel was sampled as another dimension (in seconds from the beginning of the video).

The entries of the vectors all measure different aspects and stem from different ranges. Since we want all dimensions to have equal importance in the distance computations, irrespective of their value range, we normalize all 8 dimensions individually, yielding a vector whose entries lie between 0 and 1: The positions x and y are divided by the image width and height, respectively. The L* color coordinate ranges from 0 to 100 and is hence divided by 100. The a* and b* color coordinates range from -128 to 127. Therefore, we add 128 and divide by 255. The contrast χ ranges from 0 to 128 and is therefore divided by 128. The coarseness η ranges from 0 to 5 in our implementation and is hence divided by 5. Finally, the time is divided by the video duration.

The first 7 dimensions that describe a pixel in the context of its frame have yielded high effectiveness for the task of retrieving visually similar images (cf. [BUS10a]) and were hence adopted. Since a video can be thought of as a generalization of an image along another dimension (the time dimension), the image retrieval approach was extended simply by adding the time as another dimension to the feature vectors. The rationale for this is that there is no conceptual difference between the spatial dimensions and the time dimension. A video can be imagined to be an image changing over time. The fact that a video is usually represented as a sequence of frames is just a way to store a video digitally, and it has lead many of the video retrieval approaches to base their video representations on frame sequences, even though semantically a video can be treated reasonably as an image changing continuously over time rather than as a sequence of images.

We have seen how we can express local visual properties of a video by means of a set of feature vectors. In the next section we will see how we can summarize these vectors into a more compact representation scheme.

4.2 Feature Signatures

When computing the distance between two videos, it would be highly inefficient to take into account all of the extracted feature vectors. For most practical purposes, however, it is not necessary to do this in order to achieve a good discriminability of the videos. Most of the vectors carry redundant information or fine-grained details that do not have a significant influence on the overall similarity of two videos. Hence, we summarize all of the extracted features into a structure called a *feature signature*.

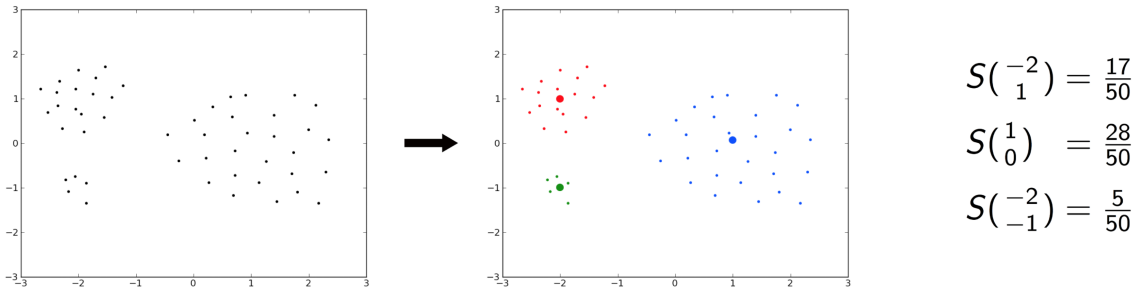


Figure 4.2: Visualization of feature signature calculation on 2-dimensional vectors

Definition 4.2.1. Let $\mathbb{F}$ be the set of all possible features. A *feature signature* X is a function $X : \mathbb{F} \to \mathbb{R}$ such that $|\{f \in \mathbb{F} \,|\, X(f) \neq 0\}| < \infty$. We refer to $R_X = \{f \in \mathbb{F} \,|\, X(f) \neq 0\}$ as the *representatives* or *centroids* of X. We use $\mathbb{S}$ to refer to the set of all feature signatures.

Intuitively, a feature signature is characterized by a set of feature vectors, called the representatives, along with a weight for each representative. A common way to calculate a feature signature is to apply a clustering algorithm (e.g. k-means, cf. [HKP06]) to the extracted set of feature vectors. From the resulting clustering, we devise a feature signature by defining the cluster means as the representatives and assigning them a weight corresponding to the relative size of the cluster, i.e. the number of cluster elements divided by the total number of extracted features:

Definition 4.2.2. Let $\mathcal{C} = C_1, ..., C_m$ be a clustering of feature vectors. We define the *clustering-induced normalized feature signature* $X_\mathcal{C}$ as $X_\mathcal{C} : \mathbb{F} \to \mathbb{R}$ with:

$$X_\mathcal{C}(f) = \begin{cases} \frac{|C_i|}{\sum_{1 \leq j \leq m} |C_j|} & \text{if } f = \frac{1}{|C_i|} \sum_{g \in C_i} g \text{ for some } i \in \{1, ..., m\} \\ 0 & \text{else} \end{cases}$$

Before applying the clustering algorithm, we multiply each dimension by a certain weight, which allows us to control the importance of that dimension for the clustering. When using k-means to calculate the clustering, we can specify the desired number of representatives k in advance. This allows us to control the expressiveness of the feature signature. The higher we choose k, the more expressive the feature signature gets, with the downside of increasing the storage size and the computational complexity of the distance computation. The experiments presented in Chapter 7 show that there is a monotonous relation between k and the effectiveness as well as the query processing time. Hence, k allows us to control the tradeoff between effectiveness and efficiency.

Figure 4.2 visualizes an exemplary calculation of a feature signature for 2-dimensional feature vectors. First, the vectors are clustered, yielding 3 clusters (red, green and blue).

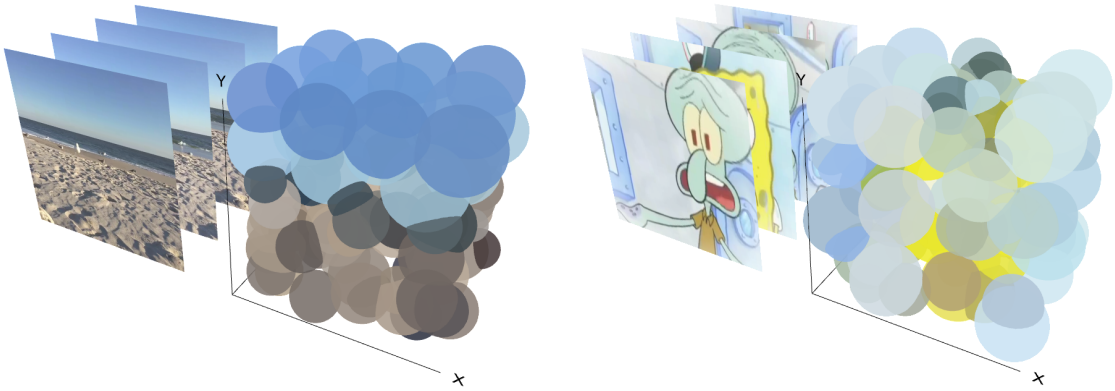


Figure 4.3: Visualizations of video feature signatures

Then we compute the cluster centers (depicted as the red, green and blue circles), which we define as the representatives of the feature signature S, and assign them a weight corresponding to the relative cluster size. Figure 4.3 shows 3D visualizations of two videos and their feature signatures with $k = 100$. Here, the clusters are represented as spheres. Their position in the 3D coordinate system corresponds to the position (x and y) and the time (t), the color of the sphere corresponds to the $L^*a^*b^*$ color dimensions of the representatives and the volume of the sphere corresponds to the weight that the feature signature assigns to the representative.

We have seen how feature signatures reduce the rather large amount of information inherent in the feature vectors into a compact representation that still reveals a lot of information about the feature distribution, since it summarizes how many feature vectors are located at which locations in the feature space. In the next section, we present two distance measures on feature signatures.

4.3 Distance Measures on Feature Signatures

Since we represent videos as feature signatures, we can measure video dissimilarity using a distance function on feature signatures. Numerous distance measures for feature signatures have been proposed (cf. [BS13] for an overview). In our experiments, we mainly focused on two distance measures, namely the *Earth Mover's Distance* and the *Signature Quadratic Form Distance*.

4.3.1 Earth Mover's Distance

Proposed in [RTG00] for the domain of content-based image retrieval, the *Earth Mover's Distance* (short: EMD) is a distance measure on feature signatures that can be thought of as the minimum required cost for transforming one feature signature into the other one. This cost is formulated by means of a transportation problem: We determine the optimal way to move the weights of the representatives from the first signature (X) to the representatives of the second signature (Y). The cost for moving a certain amount of weight is given by the amount of weight multiplied by the distance over which it is transported. A formal definition is given below.

Definition 4.3.1. Let $\mathbb{F}$ be the set of all possible features, $\delta : \mathbb{F} \times \mathbb{F} \to \mathbb{R}^{\geq 0}$ be a distance function on features and $X, Y \in \mathbb{S}$ be two feature signatures.

We call $f : \mathbb{F} \times \mathbb{F} \to \mathbb{R}$ a *feasible flow* if it holds that

- *Non-negativity constraint*: $\forall g \in R_X, h \in R_Y : f(g,h) \geq 0$
- *Source constraint*: $\forall g \in R_X : \sum_{h \in R_Y} f(g,h) \leq X(g)$
- *Target constraint*: $\forall h \in R_Y : \sum_{g \in R_X} f(g,h) \leq Y(h)$
- *Total flow constraint*: $\sum_{g \in R_X} \sum_{h \in R_Y} f(g,h) = \min\{\sum_{g \in R_X} X(g), \sum_{h \in R_Y} Y(h)\}$

Let $F = \{f \mid f \text{ is a feasible flow}\}$. Then the *Earth Mover's Distance* $EMD_\delta : \mathbb{S} \times \mathbb{S} \to \mathbb{R}^{\geq 0}$ between X and Y is defined as

$$EMD_\delta(X,Y) = \min_{f \in F} \left\{ \frac{\sum_{g \in R_X} \sum_{h \in R_Y} f(g,h) \cdot \delta(g,h)}{\min\{\sum_{g \in R_X} X(g), \sum_{h \in R_Y} Y(h)\}} \right\}$$

This definition corresponds to a linear program, i.e. an optimization problem with a linear objective function and linear constraints. It can be solved, for instance, using the *Simplex algorithm* (cf. [Van01]), which has an exponential worst-time complexity. According to [SJ08], the empirical time complexity for calculating the Earth Mover's Distance between two signatures X and Y using the simplex algorithm lies between $\mathcal{O}(n^3)$ and $\mathcal{O}(n^4)$ where $n = |R_X| + |R_Y|$.

The Earth Mover's Distance is a metric, provided that the ground distance is a metric and the compared signatures are normalized to the same value.

4.3.2 Signature Quadratic Form Distance

Proposed in [BUS10b], the *Signature Quadratic Form Distance* (short: SQFD) is a distance measure on feature signatures that compares all pairs of representatives from both

signatures. It can be thought of as an adaption of the *Quadratic Form Distance* [FBF+94] to feature signatures. The definition is given below.

Definition 4.3.2. Let X, Y be two feature signatures and let $s : \mathbb{F} \times \mathbb{F} \rightarrow \mathbb{R}^{\geq 0}$ be a similarity function on features. The *Signature Quadratic Form Distance* $SQFD_s : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}^{\geq 0}$ is defined as

$$SQFD_s(X, Y) = \sqrt{< X - Y, X - Y >_s}$$

where $< X, Y >_s : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}^{\geq 0}$ is the *Similarity Correlation*, which is defined as

$$< X, Y >_s = \sum_{f \in R_X} \sum_{g \in R_Y} X(f) \cdot Y(g) \cdot s(f, g)$$

The similarity $s(f, g)$ can be reduced to a distance function on feature vectors (e.g. the Euclidean distance or the Manhattan distance) by means of a kernel (cf. [BUS10b]). In our FlexVis implementation, we used the *Gaussian kernel* in combination with the Manhattan distance, which is defined as follows.

$$k_{\text{Gaussian}}(f, g) = e^{-\frac{||f-g||_1^2}{2\sigma^2}}$$

As shown in [BS13], the SQFD is a metric. Moreover, it fulfills Ptolemy's inequality, allowing to apply Ptolemaic indexing methods (cf. [LHSB11]). The nature of the SQFD allows for vast parallelization on GPUs that increase the query processing time by orders of magnitude (cf. [KLB+11]).

We now have a way to model the dissimilarity between two videos by means of distance functions on feature signatures. The next section describes how we can perform kNN queries with respect to these distance functions efficiently.

4.4 Efficient Query Processing

The naive way to process a kNN query entails computing the distance between the query object and all database objects, resulting in a time complexity of $\mathcal{O}(|DB|)$ (cf. [BS13]). If the distance measure is costly to compute, which is usually the case when dealing with complex multimedia objects, this is infeasible. Therefore, we rely on methods that allow us to compute the k-Nearest Neighbors without having to compute the actual distance between the query and all database objects.

One way to speed up the query processing is by means of a lower bound to the distance function δ, i.e. a function $\delta_{LB} : X \times X \rightarrow \mathbb{R}$ for which it holds that $\delta_{LB}(x, y) \leq \delta(x, y)$ for all $x, y \in X$. If we know that the k-Nearest Neighbors have a distance smaller than or

equal to $\epsilon_{\max} \in \mathbb{R}$, then we can exclude all objects o for which it holds that $\delta_{LB}(q, o) > \epsilon_{\max}$ without computing the actual distance $\delta(q, o)$, since this implies that $\delta(q, o) > \epsilon_{\max}$.

The *Multi-Step kNN Algorithm*, proposed by Seidl et al. in [SK98], utilizes a lower bound for efficient kNN search by iteratively updating the pruning distance $\epsilon_{\max}$ while scanning the database. As shown in [SK98], this algorithm is optimal with respect to the number of performed computations of the utilized distance function δ. We slightly adapted this algorithm to allow for the utilization of multiple lower bounds $\delta_{LB_1}, ..., \delta_{LB_m}$

$result \leftarrow \emptyset$
$filterRanking \leftarrow ranking(q, \delta_{LB_1}, X)$
$x \leftarrow filterRanking.getnext()$
$\epsilon_{max} \leftarrow \infty$
while $\delta_{LB_1}(q, x) \leq \epsilon_{max}$ **do**
  for $1 \leq i \leq m$ **do**
    if $\delta_{LB_i} > \epsilon_{max}$ **then** continue outer loop **end if**
  end for
  if $|result| < k$ **then**
    $result \leftarrow result \cup \{x\}$
  else
    $result \leftarrow result \cup \{x\}$
    $result \leftarrow result - \{\arg\max_{r \in result} \delta(q, r)\}$
    $\epsilon_{\max} \leftarrow \max_{r \in result} \delta(q, r)$
  end if
  $x \leftarrow filterRanking.getnext()$
end while
return result

The efficiency of this algorithm highly depends on the utilized lower bounds. A good lower bound should meet the *ICES criteria* defined by Assent et al. in [AWS06]: It should be *indexable* such that multidimensional indexing structures like X-Tree or R-Tree can be applied. Furthermore, it should be *complete*, i.e. no false drops occur, which is guaranteed by the lower-bounding property. Moreover, it should be *efficient*, i.e. its computational time complexity should be significantly lower than the complexity of the actual distance function. Finally, it should be *selective*, i.e. it should allow us to exclude as many objects as possible from the actual distance computation, which is achieved by approximating the actual distance as good as possible.

If the lower bounds have different time complexities and selectivities, then the ordering of the lower bounds plays a role for the efficiency of the algorithm. In each iteration of the outer loop, we skip the actual distance computation when one of the lower bounds exceeds

the current pruning radius ϵ_{max}. Hence, a reasonable heuristic in order to skip as early as possible is to sort the lower bounds in ascending order of their time complexities.

Lower bounds can be devised by exploiting the inner workings of the utilized distance function. There are, however, some lower bounds that are generic in nature, i.e. they are applicable to a wide variety of distance functions, as long as these distance functions fulfill certain properties. In the following, we present lower bounds for EMD and a generic lower bound for metric distance functions. In Section 7.2, we will give experimental results as to how these lower bounds and combinations thereof affect the query processing time.

4.4.1 Lower Bounds for the Earth Mover's Distance

Rubner Lower Bound

As shown by Rubner et al. in [RTG00], when using a norm-induced ground distance, the Earth Mover's Distance can be lower-bounded by the ground distance between the *weighted means* of the two signatures.

Theorem 4.4.1. *Let X be a feature signature and $\delta : \mathbb{F} \times \mathbb{F} \to \mathbb{R}$ be a norm-induced ground distance. Then it holds that*

$$Rubner(X, Y) = \delta(\overline{X}, \overline{Y}) \leq EMD_\delta(X, Y)$$

where $\overline{X}$ is the weighted mean *of X:*

$$\overline{X} = \frac{\sum_{f \in R_X} X(f) \cdot f}{\sum_{f \in R_X} X(f)}$$

Independent Minimization Lower Bound

Proposed in [UBSS14], the *Independent Minimization Lower Bound for feature signatures* (short: IM-Sig) is a lower bound for EMD that corresponds to the EMD when removing the *Target constraint* and replacing it with the *IM-Sig Target constraint* defined as follows:

$$\forall g \in R_X, h \in R_Y : f(g, h) \leq Y(h)$$

Intuitively, this modified target constraint allows to distribute the flow optimally for each representative $g \in R_X$ without considering whether the total flow coming into the target representatives exceeds their weights, as long as the flow from g to h does not exceed the weight $Y(h)$ for all target representatives $h \in R_Y$. We use $IMSig_\delta(X, Y)$ to denote the minimum cost flow with respect to the modified target constraint. Since the set of feasible flows for IM-Sig includes the set of feasible flows for EMD, it holds that $IMSig_\delta(X, Y) \leq EMD_\delta(X, Y)$.

4.4.2 Metric Lower Bound

If we are dealing with a metric distance function as per Definition 2.1, we can exploit the fact that the distance fulfills the triangle inequality to devise a lower bound (cf. [ZADB06]). Given a query q, a database object o and a set of so-called *pivot objects* P, it follows from the triangle inequality that

$$\delta_{\text{LB-Metric}}(q, o) = \max_{p \in P} |\delta(q, p) - \delta(p, o)| \leq d(q, o)$$

The distances $\delta(p, o)$ between all pivot objects $p \in P$ and all database objects $o \in DB$ can be computed in advance and stored in a so-called *pivot table* of size $|P| \cdot |DB|$. When processing a query, we only need to compute the distances $\delta(q, p)$ between the query and all pivot objects $p \in P$ and store those distances in a list. After that, $\delta_{\text{LB-Metric}}$ can be computed in $\mathcal{O}(|P|)$ trivially by looking up the stored distance values. The selectivity of this approach is highly dependent on the choice of pivot objects P and the distribution of the database objects and the query object.

We have seen how we can speed up the kNN query processing through the use of lower bounds. The next section introduces a method for reducing the number of representatives of the feature signatures in order to speed up the distance computation, resulting in an approximate kNN result.

4.4.3 Feature Signature Compression

Even though the presented lower bounds have a huge speedup potential in comparison to a sequential scan of the database, in some scenarios it might not be necessary to retrieve results in the exact order of their distances. When a user issues a query, a fast response time is usually more important than perfect accuracy of the results. In a kNN query, in many cases it is acceptable for a user that some of the retrieved results are not actually among the k-Nearest Neighbors with respect to a distance function, or that some database objects of the k-Nearest Neighbors are not among the retrieved results.

The time complexity of the distance computations on feature signatures increases with the number of representatives. Hence, reducing the number of representatives speeds up the distance computation, with the downside that the computed distance is only an approximation of the actual distance. One way to reduce the number of representatives is to simply leave out certain representatives, but this yields a significant information loss, especially if the left-out representatives are chosen arbitrarily, and hence makes the approximate distance less accurate. A better approximation can be achieved by merging representatives as follows.

Definition 4.4.1. Let X be a feature signature and $g, h \in R_X$ be two representatives. We define the *merged representative* of g and h as follows:

$$Merge(g,h) = \frac{X(g) \cdot g + X(h) \cdot h}{X(g) + X(h)}$$

Additionally, we define the *merged weight*:

$$MergeWeight(g,h) = X(g) + X(h)$$

The rationale for this definition is the following: If C_g and C_h are the clusters of features whose centroids are g and h (i.e. $g = \frac{1}{|C_g|}\sum_{f \in C_g} f$, $h = \frac{1}{|C_h|}\sum_{f \in C_h} f$), then $Merge(g,h)$ corresponds to the centroid of the union of C_g and C_h, i.e. $Merge(g,h) = \frac{1}{|C_g|+|C_h|}\sum_{f \in C_g \cup C_h} f$.

Given a set $\mathcal{C}$ of pairs of representatives, we can compress a feature signature by merging all pairs $\{g,h\} \in \mathcal{C}$:

Definition 4.4.2. Let X be a feature signature and let $\mathcal{C} \subset \{\{g,h\} \mid g,h \in R_X\}$ such that it holds for all $\{g,h\}, \{g',h'\} \in \mathcal{C}$ that g,h,g',h' are pairwise distinct (i.e. every representative gets paired with at most one other representative and this representative is different from itself). We define the *compression* of X with respect to the merged pairs of representatives $\mathcal{C}$ as

$$Comp_{\mathcal{C}}(X) : \mathbb{F} \to \mathbb{R}$$

$$Comp_{\mathcal{C}}(X)(g) = \begin{cases} 0 & \text{if } g = g' \text{ or } g = g'' \text{ for some } \{g',g''\} \in \mathcal{C} \\ MergeWeight(g',g'') & \text{if } g = Merge(g',g'') \text{ for some } \{g',g''\} \in \mathcal{C} \\ X(g) & \text{else} \end{cases}$$

Naturally, compressing a feature signature entails an information loss. This information loss is highly dependent on the set $\mathcal{C}$ of merged representatives. Merging representatives with a small distance causes less information loss than merging representatives with a high distance. Hence, we propose to construct $\mathcal{C}$ by first computing all pairwise distances between the representatives and then iterating all pairs $\{g,h\}$ in ascending order of the distances. If neither g nor h is present in a pair marked for merging, we mark $\{g,h\}$ for merging. We stop the iterative process once the distance exceeds a threshold ϵ. This

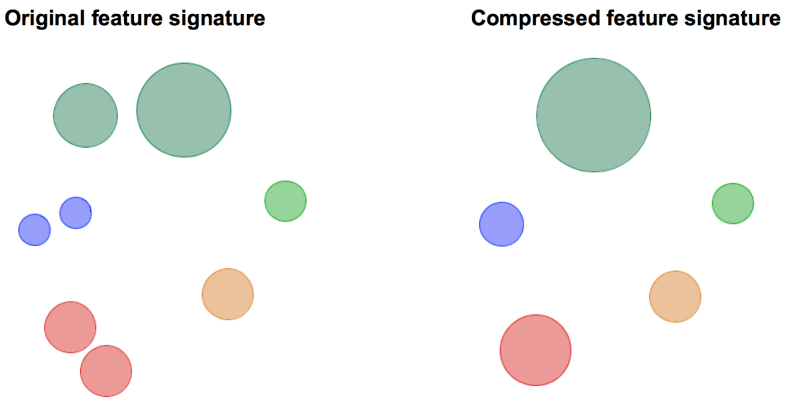


Figure 4.4: Example of a compressed feature signature

procedure is formalized in the following algorithm.

Input:

$X : \{f_1, ..., f_n\} \rightarrow \mathbb{R}$ ▷ Feature signature

$\delta : \mathbb{F} \times \mathbb{F} \rightarrow \mathbb{R}$ ▷ Ground distance between centroids

$\epsilon \in \mathbb{R}$ ▷ Distance threshold

Create empty distance matrix $D \in \mathbb{R}^{n \times n}$
for $1 \leq i \leq n$ **do**
  for $1 \leq j < i$ **do**
    $D_{i,j} \leftarrow \delta(f_i, f_j)$
  end for
end for
Sort the values of D in ascending order. Let $\{i_k, j_k\}$ be the pair of centroids with the k-th distance value.
$\mathcal{C} \leftarrow \emptyset$
for $1 \leq k \leq \frac{n \cdot (n-1)}{2}$ **do**
  if $D_{i_k,j_k} > \epsilon$ **then** break **end if**
  $\mathcal{C} \leftarrow \mathcal{C} \cup \{\{f_{i_k}, f_{j_k}\}\}$
end for
return $\mathcal{C}$

Figure 4.4 shows an example of a feature signature (left) and its compressed feature signature (right) in a 2-dimensional feature space, where each representative is depicted as a circle with an area that is proportional to the weight of that representative. The orange representative is not merged with the green representative since their distance is too high. All the other representatives are merged with their nearest neighbor, since the

distance to that neighbor is below the distance threshold.

The runtime of this algorithm is dominated by the sorting step and hence lies in $\mathcal{O}(n^2\ log\ n)$. When compressing a feature signature with this procedure, we can prove that the resulting approximated EMD value is bounded above by an additive error of at most 1.5ϵ:

Theorem 4.4.2. *Let X, Y be two 1-normalized feature signatures, $\delta : \mathbb{F} \times \mathbb{F} \to \mathbb{R}$ be a metric ground distance and $\mathcal{C}$ be a set of merged representatives such that $\delta(g,h) \leq \epsilon \in \mathbb{R}^{\geq 0}$ for all $\{g,h\} \in \mathcal{C}$. Then it holds that*

$$EMD(Comp_{\mathcal{C}}(X), Y) \leq EMD(X,Y) + 1.5\,\epsilon$$

Proof. Let f be a minimum cost flow from X to Y. We define a flow f' from $Comp_{\mathcal{C}}(X)$ to Y as follows:

$$f'(g,h) = \begin{cases} f(g',h) + f(g'',h) & \text{if } g = Merge(g',g'') \text{ for some } \{g',g''\} \in \mathcal{C} \\ f(g,h) & \text{else} \end{cases}$$

f' is a feasible flow from $Comp_{\mathcal{C}}(X)$ to Y:

- Non-negativity constraint: follows by non-negativity of f
- Source constraint: Let $g \in R_{Comp_{\mathcal{C}}(X)}$. If $g = Merge(g',g'')$ for some $\{g',g''\} \in \mathcal{C}$, it holds that

$$\sum_{h \in R_Y} f'(g,h) = \sum_{h \in R_Y} \big(f(g',h) + f(g'',h)\big) \leq X(g') + X(g'')$$
$$= MergeWeight(g',g'') = Comp_{\mathcal{C}}(X)(g)$$

  Otherwise, it holds that

$$\sum_{h \in R_Y} f'(g,h) = \sum_{h \in R_Y} f(g,h) \leq X(g) = Comp_{\mathcal{C}}(X)(g)$$

- Target constraint: Let $h \in R_Y$. Then it holds that

$$\sum_{g \in R_{Comp_{\mathcal{C}}(X)}} f'(g,h)$$
$$\overset{(1)}{=} \sum_{g \in R_X \cap R_{Comp_{\mathcal{C}}(X)}} f(g,h) + \sum_{\{g',g''\} \in \mathcal{C}} \Big[(f'(Merge(g',g''),h)\Big]$$
$$\overset{(2)}{=} \sum_{g \in R_X \cap R_{Comp_{\mathcal{C}}(X)}} f(g,h) + \sum_{\{g',g''\} \in \mathcal{C}} \Big[f(g',h) + f(g'',h)\Big]$$
$$\overset{(3)}{=} \sum_{g \in R_X} f(g,h)$$

$$\overset{(4)}{\leq} \quad X(h)$$

(1) follows by splitting the sum into two sums, where the first one sums over the flow coming from the non-merged representatives (i.e. the representatives that also exist in R_X) and the second one sums over the merged representatives. (3) follows since $R_X = (R_X \cap R_{Comp_{\mathcal{C}}(X)}) \cup \{g', g'' \mid \{g', g''\} \in \mathcal{C}\}$.

- Total flow constraint:

$$\begin{aligned}
& \sum_{g \in R_{Comp_{\mathcal{C}}(X)}} \sum_{h \in R_Y} f'(g,h) \\
\overset{(5)}{=} \quad & \sum_{g \in R_{Comp_{\mathcal{C}}(X)} \cap R_X} \sum_{h \in R_Y} f(g,h) + \sum_{\{g',g''\} \in \mathcal{C}} \sum_{h \in R_Y} \Big[f'(Merge(g',g''),h) \Big] \\
\overset{(6)}{=} \quad & \sum_{g \in R_{Comp_{\mathcal{C}}(X)} \cap R_X} \sum_{h \in R_Y} f(g,h) + \sum_{\{g',g''\} \in \mathcal{C}} \sum_{h \in R_Y} \Big[f(g',h) + f(g'',h) \Big] \\
\overset{(7)}{=} \quad & \sum_{g \in R_X} \sum_{h \in R_Y} f(g,h) \\
\overset{(8)}{=} \quad & \min\{ \sum_{g \in R_X} X(g), \sum_{h \in R_Y} Y(h) \} \\
\overset{(9)}{=} \quad & \min\{ \sum_{g \in R_{Comp_{\mathcal{C}}(X)}} Comp_{\mathcal{C}}(X)(g), \sum_{h \in R_Y} Y(h) \}
\end{aligned}$$

(5) and (7) follow in analogy to (1) and (3), respectively. (8) follows from the total flow constraint of the flow f.

We can bound the cost of the flow f' from above as follows:

$$\begin{aligned}
& \sum_{g \in R_{Comp_{\mathcal{C}}(X)}} \sum_{h \in R_Y} f'(g,h) \cdot \delta(g,h) \\
\overset{(10)}{=} \quad & \sum_{g \in R_X} \sum_{h \in R_Y} f(g,h) \cdot \delta(g,h) + \sum_{\{g',g''\} \in \mathcal{C}} \sum_{h \in R_Y} \Big[(f'(Merge(g',g''),h) \cdot \delta(Merge(g',g''),h) \\
& - f(g',h) \cdot \delta(g',h) - f(g'',h) \cdot \delta(g'',h) \Big] \\
\overset{(11)}{\leq} \quad & \sum_{g \in R_X} \sum_{h \in R_Y} f(g,h) \cdot \delta(g,h) + \sum_{\{g',g''\} \in \mathcal{C}} \sum_{h \in R_Y} \Big[(f'(Merge(g',g''),h) \cdot (\delta(g',h) + \delta(g',g'')) \\
& - f(g',h) \cdot \delta(g',h) - f(g'',h) \cdot (\delta(g',h) - \delta(g',g'')) \Big] \\
\overset{(12)}{=} \quad & \sum_{g \in R_X} \sum_{h \in R_Y} f(g,h) \cdot \delta(g,h) + \sum_{\{g',g''\} \in \mathcal{C}} \sum_{h \in R_Y} \Big[(f'(Merge(g',g''),h) + f(g'',h)) \cdot \delta(g',g'') \\
& + (f'(Merge(g',g''),h) - f(g',h) - f(g'',h)) \cdot \delta(g',h) \Big] \\
\overset{(13)}{=} \quad & \sum_{g \in R_X} \sum_{h \in R_Y} f(g,h) \cdot \delta(g,h) + \sum_{\{g',g''\} \in \mathcal{C}} \sum_{h \in R_Y} (f(g',h) + 2 \cdot f(g'',h)) \cdot \delta(g',g'')
\end{aligned}$$

$$
\begin{aligned}
&\overset{(14)}{\leq} && \sum_{g \in R_X} \sum_{h \in R_Y} f(g,h) \cdot \delta(g,h) + \sum_{\{g',g''\} \in \mathcal{C}} (X(g') + 2 \cdot X(g'')) \cdot \delta(g',g'') \\
&\overset{(15)}{\leq} && \sum_{g \in R_X} \sum_{h \in R_Y} f(g,h) \cdot \delta(g,h) + \sum_{\{g',g''\} \in \mathcal{C}} (X(g') + 2 \cdot X(g'')) \cdot \epsilon \\
&\overset{(16)}{\leq} && \sum_{g \in R_X} \sum_{h \in R_Y} f(g,h) \cdot \delta(g,h) + 1.5\,\epsilon \\
&\overset{(17)}{=} && EMD(X,Y) + 1.5\,\epsilon
\end{aligned}
$$

(10) follows in analogy to (1). (11) follows by the triangle inequality. (12) follows by reordering. (13) follows by definition of $f'(Merge(g',g''),h)$. (14) follows from the source constraint of f. (15) follows from the fact that $\delta(g',g'') \leq \epsilon$. (16) follows due to the normalization of X and the fact that we can assume without loss of generality that $X(g') \geq X(g'')$.

Since f' is a feasible flow from $Comp_{\mathcal{C}}(X)$ to Y, the minimum cost flow has a smaller cost, hence

$$EMD(Comp_{\mathcal{C}}(X), Y) \leq EMD(X,Y) + 1.5\,\epsilon$$

□

Note that we did not give a proof that this approximation error is exhaustive, i.e. that there are situations in which this approximation error actually occurs. In our experiments, the error has been observed to be much smaller (cf. Section 7.3). Since we know the set $\mathcal{C}$ when performing the compression, we can estimate an a-posteriori error bound more accurately using the error term in (14), i.e. $\sum_{\{g',g''\} \in \mathcal{C}} (X(g') + 2 \cdot X(g'')) \cdot \delta(g',g'')$.

The distance threshold ϵ acts as a tradeoff parameter between efficiency and accuracy: The higher we choose ϵ, the smaller the signature size gets, leading to an increased efficiency at the expense of a higher error.

In this chapter, we have introduced FlexVis as a new method for retrieving videos that are visually similar to a query video. We have seen how we can perform exact kNN search efficiently through lower bounding or perform approximate kNN search through signature compression. The following chapter gives an overview of RCVS, a web-based video search engine incorporating FlexVis and the competitive methods.

Chapter 5

RCVS

This chapter gives a brief overview of *RCVS* (abbreviation for *Real-time Content-based Video Similarity Search*), a web-based video search engine that was developed in the course of this bachelor thesis which allows the user to upload a video and issue a kNN query using FlexVis or one of the competitive methods presented in Chapter 3.

Figure 5.2a shows the primary user interface of RCVS. On the top, the user has the possibility to choose a retrieval method and specify certain parameters: For FlexVis, the user can specify the desired query signature size, the distance measure and indexing method (e.g. EMD with IM-Sig lower bound) as well as weights for the feature dimensions. After the user selects a video from his local computer and clicks the *Search* button, the query is processed as depicted in Figure 5.1: The video and the specified parameters are sent to the web server and then passed to the feature extractor, which generates a set of feature vectors as described in Section 4.1. The resulting set of feature vectors is in turn passed on to the feature signature calculator, which computes a feature signature using the user-specified number of clusters and dimension weights. This feature signature is passed to the kNN query processor, which retrieves the k-Nearest Neighbors from the feature signature database using the Multi-Step kNN algorithm and an optional index (e.g. a pivot table when using metric indexing). The kNN results are then passed to the web

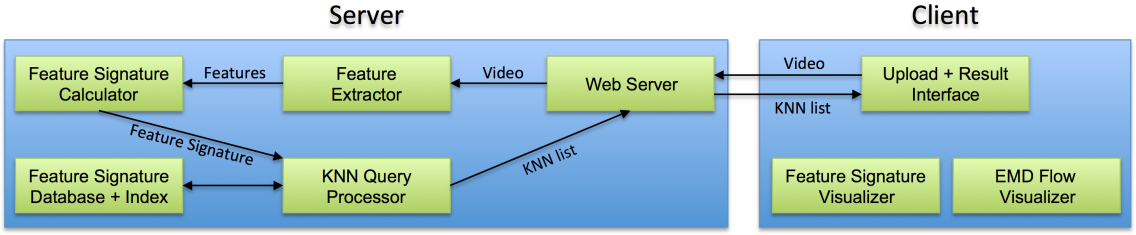
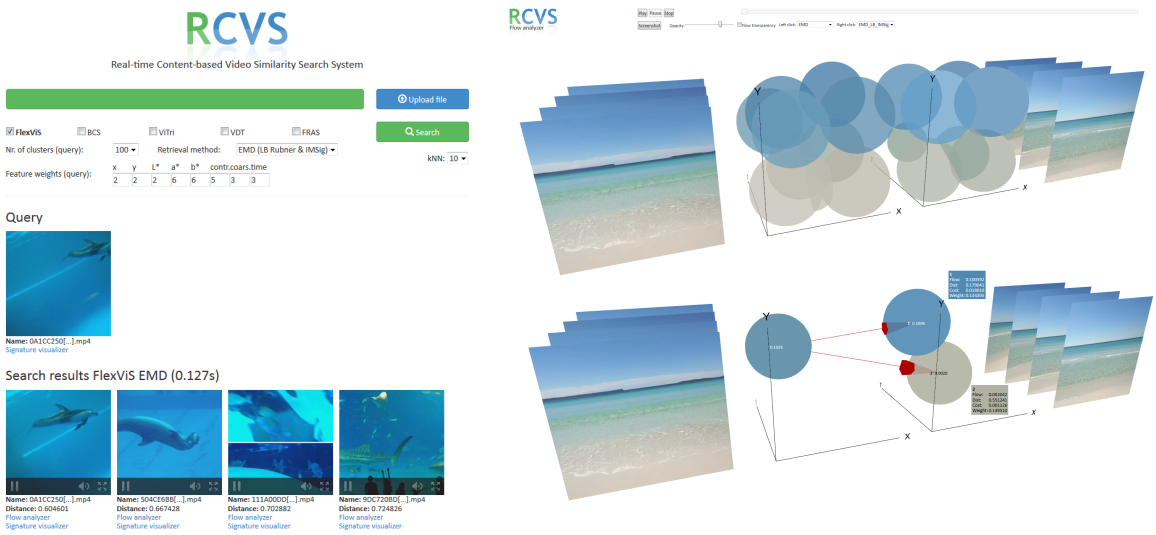


Figure 5.1: RCVS architecture



(a) Upload and Result Interface

(b) EMD Flow Visualizer

Figure 5.2: RCVS user interface

server, which sends them back to the user's browser where they are displayed as depicted at the bottom of Figure 5.2a. For the competitive methods, the procedure is analogous.

Once the search results arrive at the browser, the user has the possiblity to visualize the feature signatures of the query and the retrieved results in 3D, as explained in Section 4.2. Moreover, when using EMD for the retrieval, the user can visualize the EMD flow between the query and the retrieved database videos in order to get an understanding for how the computed distance comes about and why two videos are similar or dissimilar. To this end, the feature signatures of the two videos are displayed next to each other as depicted on the top of Figure 5.2b. By clicking on a source representative, red arrows are shown pointing at the target representatives to which the source representative sends flow, as shown at the bottom. Additionally, info boxes display the amount of flow and the distance over which the flow runs. In addition to visualizing the EMD flow, this interface also allows to visualize the IM-Sig flow for comparison.

Chapter 6

Subclip Search

So far, we have only considered the task of retrieving videos that are globally similar to the query video, i.e. videos that are similar to the query as a whole. A related but different task is to retrieve videos that contain the query clip as a subclip. Our FlexVis model in its current form is not well-suited for this task, since the centroids of the query clip have a different entry in the time dimension than their corresponding centroids in a longer database video - even when the time dimension is not normalized. Consider, for example, a query clip with a duration of 10 seconds that is to be searched in a movie of 2 hours. If the query clip starts at the 1 hour mark within the movie, then all of its centroids within the movie feature signature have their time coordinate in the interval $[3600s, 3610s]$. On the other hand, all of the centroids of the query feature signature have their time coordinate in the interval $[0s, 10s]$. Thus, the ground distance between the centroids in the query feature signature and the corresponding centroids in the movie feature signature is very large. When using the Earth Mover's Distance, it is more likely that the earth from the query signature's centroids is moved to centroids somewhere at the beginning of the movie, since they have a smaller ground distance due to a smaller value in the time dimension. This makes it impossible to differentiate between movies that contain a query clip and movies that do not. One way to circumvent this issue is to simply leave out the time dimension in the ground distance computation and only take into account the other 7 dimensions. This, however, neglects the temporal ordering of the query clip in the movie clip. The idea of our solution to this problem is to determine the number of seconds by which we need to shift the centroids of the query feature signature in order to get the smallest possible EMD value to the database video. The distance between the query and the database video is then defined as this minimal EMD value.

6.1 Translation-Invariant Earth Mover's Distance

Given a translation vector Δ, we define a function $\tau_\Delta(X)$ denoting the feature signature that we acquire when adding the vector Δ to all centroid positions of X:

Definition 6.1.1. Let $\mathbb{F} = \mathbb{R}^d$ be a Euclidean feature space and $\Delta \in \mathbb{R}^d$. The *feature signature translation* $\tau_\Delta : \mathbb{S} \to \mathbb{S}$ is defined as

$$\tau_\Delta(X)(f) = X(f - \Delta)$$

For example, given the feature signature X with $X(0) = 0.3, X(1) = 0.7$, $\tau_1(X)$ corresponds to the feature signature Y with $Y(1) = 0.3, Y(2) = 0.7$.

Given two feature signatures X and Y, we want to find a translation vector Δ_{opt} in a set of possible translation vectors Υ that minimizes the EMD between $\tau_\Delta(X)$ and Y. We term the resulting EMD value as the *Translation-Invariant Earth Mover's Distance*:

Definition 6.1.2. Let $\mathbb{F} = \mathbb{R}^d$, $\Upsilon \subseteq \mathbb{R}^d$ be a set of vectors that are closed under translation and $\delta : \mathbb{F} \times \mathbb{F} \to \mathbb{R}$ be a ground distance. We define the *Translation-Invariant Earth Mover's Distance* as

$$TIEMD(X, Y) = \min_{\Delta \in \Upsilon} EMD_\delta(\tau_\Delta(X), Y)$$

Moreover, we define the *Optimal Translation* as

$$\Delta_{opt}(X, Y) = \arg\min_{\Delta \in \Upsilon} EMD_\delta(\tau_\Delta(X), Y)$$

The set Υ allows us to specify which translations we want to allow. For example, if we want to allow a shift along the time axis for the FlexVis feature signatures, we let $\Upsilon = \{(0, 0, 0, 0, 0, 0, 0, T)^\intercal \mid T \in \mathbb{R}\}$.

Theorem 6.1.1. *$TIEMD$ is a distance function.*

Proof. Let $X, Y \in \mathbb{S}$. TIEMD fulfills the properties of a distance function.

- Non-negativity (by non-negativity of EMD):

$$TIEMD(X, Y) = \min_{\Delta \in \Upsilon} EMD(\tau_\Delta(X), Y) \geq 0$$

- Reflexivity (by reflexivity of EMD):

$$TIEMD(X, X) = \min_{\Delta \in \Upsilon} EMD(\tau_\Delta(X), X) = EMD(\tau_0(X), X)$$

- Symmetry (by symmetry of EMD):

$$TIEMD(X,Y) = \min_{\Delta\in\Upsilon} EMD(\tau_\Delta(X),Y) = \min_{\Delta\in\Upsilon} EMD(X,\tau_{-\Delta}(Y))$$

$$= \min_{-\Delta\in\Upsilon} EMD(X,\tau_{-\Delta}(Y)) = \min_{-\Delta\in\Upsilon} EMD(\tau_{-\Delta}(Y),X) = TIEMD(Y,X)$$

□

Lemma 6.1.2. *If $\Upsilon \neq \emptyset$ and $\Upsilon \neq \{0\}$, TIEMD violates the identity of indiscernibles and, hence, is not a metric.*

Proof. Since $\Upsilon \neq \emptyset$ and $\Upsilon \neq \{0\}$, there exists a $\Delta \in \Upsilon$ such that $\tau_\Delta(X) \neq X$. However, it holds that $TIEMD(\tau_\Delta(X),X) = \min_{\Delta\in\Upsilon} EMD(\tau_\Delta(X),X) = EMD(\tau_0(X),X) = 0$. □

If the set Υ of allowed translations is not finite, calculating $TIEMD$ corresponds to solving a nonlinear optimization problem with the variables $f(g,h) \forall g \in R_X, h \in R_Y$ and $\Delta_1, ..., \Delta_d \in \mathbb{R}$:

$$\min_{\Delta\in\Upsilon, f\in F} \left\{ \frac{\sum_{g\in R_X}\sum_{h\in R_Y} f(g,h)\cdot\delta(g-\Delta,h)}{\min\{\sum_{g\in R_X} X(g), \sum_{h\in R_Y} Y(h)\}} \right\}$$

subject to the same constraints as the EMD. As opposed to EMD, this is no longer a linear program, since the objective function contains summands that multiply variables $f(g,h)$ with a function of Δ. Therefore, it can not be solved using the Simplex algorithm, making it hard to compute the exact value.

6.2 Approximation of TIEMD for Subclip Search

For the task of subclip search, we only allow translations along the time axis, i.e. $\Upsilon = \{(0,0,0,0,0,0,0,T)^\intercal \mid T \in \mathbb{R}\}$. In this case, we propose to compute an approximation of TIEMD as follows.

Let Q denote the query video, X denote the database video (i.e. the video in which we want to find Q as a subclip) and T_Q and T_X denote the durations of the respective videos. If we extract subclips from X at every multiple of T_Q with a duration of $2 \cdot T_Q$ each, then we can be sure that if the query video is contained within the database video, it will be completely contained in one of the extracted subclips. Since we deal with feature signatures instead of actual videos, we can arrive at a feature signature that roughly corresponds to a feature signature of a subclip in the time interval $[t_{\min}, t_{\max}]$ by simply removing all centroids that are not within that time interval and shifting the value of the time dimension backwards by $t_{\min}$ as follows:

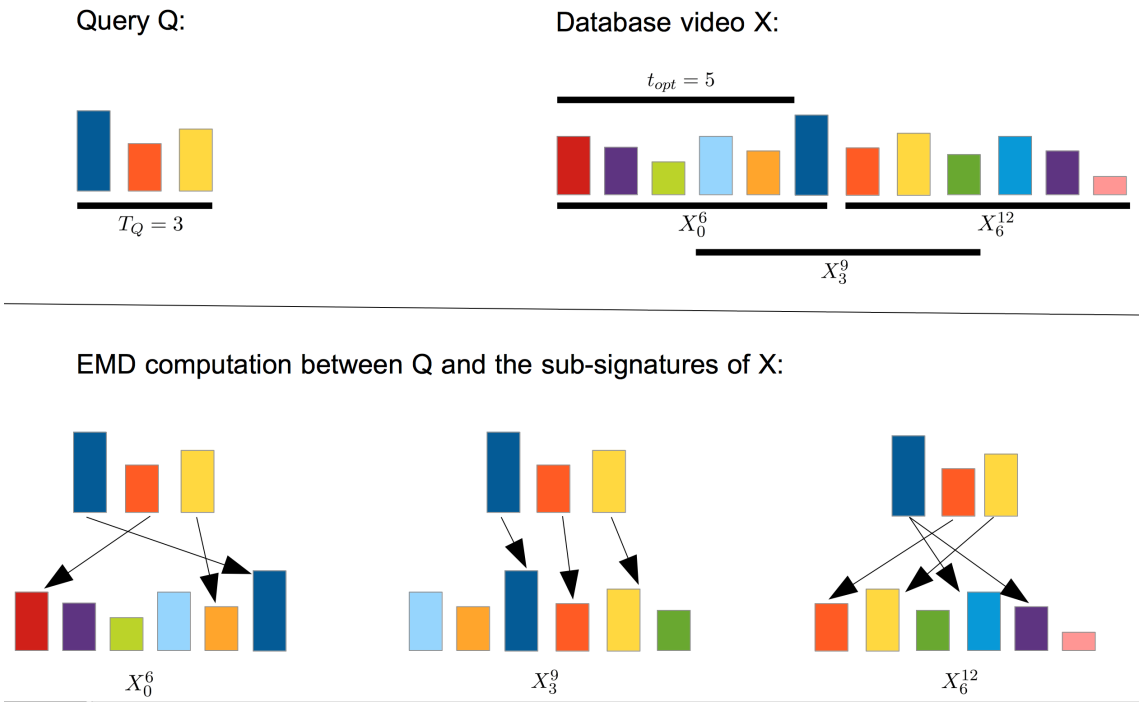


Figure 6.1: Subclip search example

Definition 6.2.1. Let X be a feature signature over $\mathbb{F} = \mathbb{R}^8$. We define the *sub-signature* of X w.r.t. a time interval $[t_{\min}, t_{\max}]$ as $X_{t_{\min}}^{t_{\max}} : \mathbb{F} \to \mathbb{R}$ with

$$X_{t_{\min}}^{t_{\max}}(f) = \begin{cases} X(f + t_{\min}) & \text{if } f_8 \in [t_{\min}, t_{\max}] \\ 0 & \text{else} \end{cases}$$

In order to approximate the TIEMD, we compute the Earth Mover's Distance between the query feature signature Q and all extracted sub-signatures $X_0^{2 \cdot T_Q}, X_{T_Q}^{3 \cdot T_Q}, X_{2 \cdot T_Q}^{4 \cdot T_Q}, \ldots$ and report the smallest distance as the approximate TIEMD.

Figure 6.1 shows an example subclip search of a query Q with duration $T_Q = 3$ inside a database video X with duration $T_X = 12$. The feature signatures contain one centroid per second, depicted as colored rectangles. We extract 3 sub-signatures from X (X_0^6, X_3^9, X_6^{12}) with a duration of $2 \cdot T_Q = 6$ seconds each, followed by a computation of the EMD between Q and the sub-signatures. The smallest EMD is observed to be $EMD(Q, X_3^9)$. Hence, we return $t_{opt} = 3$.

The number of extracted sub-signatures is $\frac{T_X}{T_Q}$, containing $|R_X| \cdot \frac{T_Q}{T_X}$ centroids on average. This leads to an overall empirical time complexity of $\mathcal{O}(\frac{T_X}{T_Q} \cdot (|R_X| \cdot (1 + \frac{T_Q}{T_X}))^3)$ to $\mathcal{O}(\frac{T_X}{T_Q} \cdot (|R_X| \cdot (1 + \frac{T_Q}{T_X}))^4)$. If we specify a maximum distance threshold ϵ in advance, we can perform pruning through lower bounds, which speeds up the procedure since it avoids costly EMD computations. The complete procedure is formalized in the following algorithm.

Input:

Q ▷ The query feature signature
X ▷ The database feature signature
T_Q ▷ Duration of the query video
T_X ▷ Duration of the database video
ϵ ▷ Maximum distance threshold

$Q \leftarrow \tau_{(0,\ldots,0,\frac{T_Q}{2})^\intercal}(Q)$ ▷ Shift Q by half its duration
$d_{\text{opt}} \leftarrow \infty$ ▷ Minimum EMD seen so far, initialized as ∞
$t_{\text{opt}} \leftarrow$ undefined ▷ Optimal time shift seen so far, initialized as undefined

for $t \leftarrow 0; t < T_X; t \leftarrow t + T_Q$ **do**
  if $RubnerLB(Q, X_t^{t+2 \cdot T_Q}) > \epsilon$ **then** continue ▷ Rubner lower bound filtering
  if $IMSig(Q, X_t^{t+2 \cdot T_Q}) > \epsilon$ **then** continue ▷ IMSig lower bound filtering
  $d \leftarrow EMD(Q, X_t^{t+2 \cdot T_Q})$ ▷ EMD between query and sub-signature in $[t, t + 2 \cdot T_Q]$
  if $d < d_{\text{opt}}$ **then** ▷ If d is smaller than d_{opt}, update d_{opt} and t_{opt}
    $d_{\text{opt}} \leftarrow d$
    $t_{\text{opt}} \leftarrow t$
  end if
end for

if $d_{\text{opt}} > \epsilon$ **then**
  return Subclip not found
else
  return $d_{\text{opt}}, t_{\text{opt}}$
end if

This procedure can be used for two different tasks. On the one hand, it determines whether or not a query clip is contained within a database video and how similar the contained version is to the query. On the other hand, it also computes at which time position the query video approximately starts within the database video, which can be used e.g. to determine the start time of a certain scene in a movie, or to extract subclips from the movie that are visually similar to the query clip.

The pruning parameter ϵ should be chosen in such a way that sub-signatures that actually contain the query are not disregarded. We can compute a suitable ϵ for a given database video by extracting a certain amount of sample query clips from the video and determining the maximum distance between the query clips and the sub-signatures of the database

video's feature signature that contain the queries.

We have seen how we can use the TIEMD for efficient subclip search. The following chapter presents experimental results for the several retrieval methods described in this thesis so far.

Chapter 7

Experiments

In the following, we present experimental results with respect to both the *effectiveness* and *efficiency* of our approach and the competitive methods presented in Chapter 3. The *effectiveness* of a retrieval method describes the quality of the retrieved results, i.e. how relevant the retrieved results are for the query. The *efficiency* describes how fast the method is at processing a query.

7.1 Effectiveness

We evaluated the effectiveness of the retrieval methods on databases of videos that are sorted into disjoint video categories. Videos in the same category are deemed to be similar to each other and dissimilar to videos from other categories. We refer to a database video as *relevant* for a query video if the two videos are in the same category. We use $Rel(q)$ to refer to the set of all videos that are relevant for a query q. Using the category information, we can evaluate how good the retrieval methods are at retrieving relevant videos.

Given a query q and a set of retrieved videos $Ret(q)$, we define the *precision* to be the fraction of retrieved videos that are relevant for the query (cf. [MRS08]):

$$Precision(q, Ret(q)) = \frac{|Ret(q) \cap Rel(q)|}{|Ret(q)|}$$

Further, we define the *recall* to be the fraction of relevant videos that were retrieved:

$$Recall(q, Ret(q)) = \frac{|Ret(q) \cap Rel(q)|}{|Rel(q)|}$$

As we can see, precision and recall are measures that operate on retrieval result sets without any ordering. However, the retrieval methods that we deal with present the k-Nearest Neighbor results as an ordered list with increasing distance (or decreasing similarity). If

we want to take this ordering into account in the evaluation, we need to extend these measures. A common way to do this is by means of a *Precision Recall Graph* (cf. [MRS08]). Given a query q, we determine the k-Nearest Neighbors for $k = 1, 2, ...$ until $Rel(q)$ is completely contained in the result set. For each of these result sets, we calculate the precision and recall and plot these values in a 2-dimensional graph.

Another common evaluation measure for ranked retrieval results is the *Mean Average Precision* (short: MAP), which is a single number that measures the average precision across all recall levels, which is in turn averaged over multiple queries. Given a set of queries $Q = q_1, ..., q_n$, it is defined as follows (cf. [MRS08]):

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{n} \frac{1}{|Rel(q_j)|} \sum_{k=1}^{|Rel(q_j)|} Precision(q_j, R_{q_j,k})$$

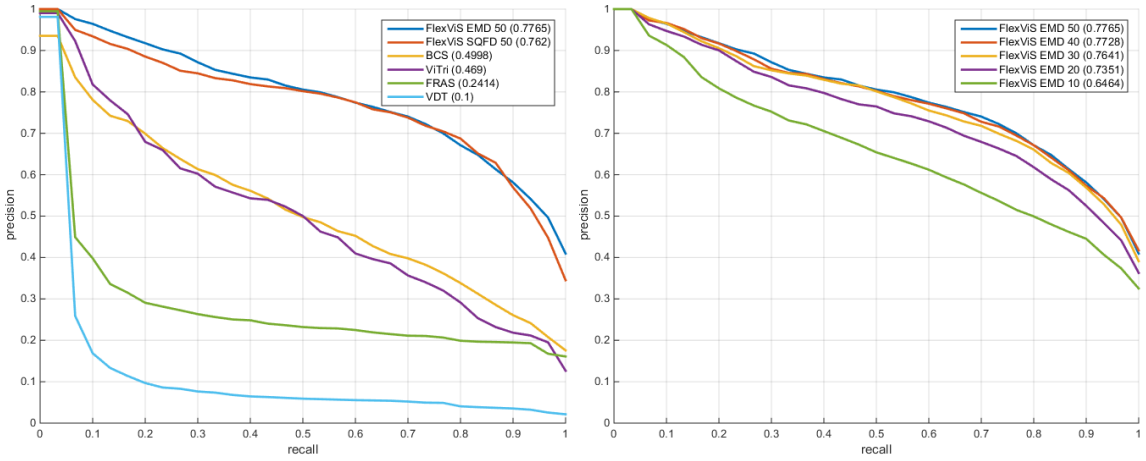
where $R_{q,k}$ denotes the set of ranked retrieval results for query q until we get to the k-th relevant object.

7.1.1 Visual Similarity Database

The first database that we used in our experiments consists of 34 categories with 30 videos per category. Two videos were sorted into the same category manually if they are visually similar by human judgement. For example, one category contains videos of soccer games, another category contains videos of beaches. All videos have a length of 6 seconds or less and were taken from `www.vine.co` or extracted as subclips from videos of `www.youtube.com`.

For the FlexVis model, we extracted feature signatures with 10, 20, 30, 40 and 50 representatives respectively using 150 sample pixels per frame at 10 frames per second, which were clustered using the feature dimension weights $(2, 2, 2, 6, 6, 5, 3, 3)$. [SOZ05], [HSS+09] and [ZZS07] suggest to use 64-dimensional RGB histograms as the framewise vectors for the competitive methods. Hence, we adopted this suggestion for our experiments using a fixed binning by splitting the R, G and B ranges into 4 bins of equal size, yielding a total number of $4^3 = 64$ bins. For VDT, we used a segmentation threshold of $\varsigma = 0.05$ and a similarity threshold $\epsilon = 0.7$, since this yielded the highest effectiveness according to experiments conducted in [HSS+10]. For ViTri, we used a clustering threshold of $\epsilon = 0.2$, since this leads to the highest effectiveness according to [SOZ05]. For FRAS, we used the same clustering threshold of $\epsilon = 0.2$ and a probability threshold of $T = 0.5$, leading to the highest effectiveness according to [ZZS07].

Figure 7.1a shows Precision Recall graphs for all of the presented retrieval methods, using 3 query videos per category. The MAP values are stated next to the method names. As we



(a) comparison of retrieval methods

(b) EMD on different signature sizes

Figure 7.1: Precision Recall graph for the Visual Similarity Database

can see, FlexVis outperforms the competitive methods. SQFD and EMD perform almost equally well, which suggests that visual similarity can be modeled both by putting all centroids into pairwise relation (as SQFD does) or by matching parts of one video to nearby similar parts of the other video (as EMD does). BCS and ViTri have similar Precision Recall graphs. This suggests that they both yield an equally expressive representation of the underlying framewise histograms. The fact that they perform worse than FlexVis suggests that the framewise RGB histograms on which they are based are not well-suited for modeling visual similarity, since they do not take into account the information considering position, time and texture like the FlexVis features presented in Section 4.1. Moreover, these histograms are based on a global partitioning of the feature space (as opposed to a local partitioning as with feature signatures), which also limits their expressiveness. FRAS and VDT perform worse, which can be explained by the fact that their video representations do not contain much information about the video content itself, but more about how the video content changes over time, which apparently is unsuitable for modeling visual similarity.

Figure 7.1b shows Precision Recall graphs for FlexVis using EMD on feature signatures with 10, 20, ..., 50 representatives, respectively. According to this graph, the effectiveness increases when using more representatives, which is due to the fact that more representatives increase the expressiveness of a signature and, hence, improve the discriminability between signatures. However, the more representatives we use, the smaller the improvement gets, suggesting that there is a limit to the effectiveness of our method on this database, regardless of how many representatives are used.

Figure 7.2: Near-Duplicate example frames

7.1.2 Near-Duplicate Databases

We also evaluated how the methods perform for near-duplicate detection. To this end, we created a software that takes a video as input and alters it with respect to several possible edit tasks. We used 3636 videos from the paper [ROS+14] as input videos. For each of the possible edit tasks, we created a ground truth database that consists of one category per original video, each category containing all of the near-duplicate versions of the respective original video:

- *Bar Insertion with Overlaying:* Consists of 10 near-duplicate videos per source video. It was created by overlaying horizontal black bars on the top and bottom of the source videos, corresponding to 2.5%, 7.5%, 12.5%, 17.5%, 20% of the video height, and analogously vertical black bars on the left and right of the video, corresponding to 2.5%, 7.5%, 12.5%, 17.5%, 20% of the video width.
- *Bar Insertion with Stretching:* Consists of 10 copies similar as above, but this time the video is stretched to fit into the remaining area instead of cropping it.
- *Brightness:* Consists of 10 copies by adding the following values to the RGB values of all pixels: -90, -72, -54, -36, -18, 0, 18, 36, 54, 72
- *Contrast:* Consists of 10 copies by multiplying the RGB values of all pixels with 0.5, 0.68, 0.86, 1.03, 1.21, 1.39, 1.57, 1.74, 1.92, 2.1 respectively and then adding 64, 41.24, 18.48, -4.27, -27.02, -49.78, -72.53, -95.29, -118.04.
- *Flip:* Consists of 6 copies by flipping the video horizontally, vertically and along the time axis (i.e. reversing it) in the following combinations: (horizontal), (vertical), (horizontal, vertical), (horizontal, time), (vertical, time), (horizontal, vertical, time)
- *Frame Deletion:* Consists of 10 copies by splitting the video into 10 parts and removing one of the 10 parts in each of the copies.
- *Frame Disordering:* Consists of 10 copies by splitting the video into 4 parts and then creating a copy for each of the following permutations of the 4 parts: 1 2 3 4, 1 2 4 3, 1 3 2 4, 1 3 4 2, 1 4 2 3, 1 4 3 2, 2 1 3 4, 2 1 4 3, 2 3 1 4, 2 3 4 1

- *Playback Speed:* Consists of 10 copies by altering the playback speed, yielding videos with the following relative durations: 0.5, 0.67, 0.83, 1, 1.17, 1.33, 1.5, 1.67, 1.83, 2
- *Resize:* Consists of 10 copies by adding the following tuples to the width and height of the video, respectively: (-115, 115), (-89, 89), (-64, 64), (-38, 38), (-13, 13), (13, -13), (38, -38), (64, -64), (89, -89), (115, -115)
- *Text Insertion:* Consists of 9 copies by splitting each video frame into a 3 x 3 grid and inserting the text "Text Insertion" into one grid per copy.

Figure 7.2 shows examples of near-duplicate versions of a single video frame according to these edit tasks.

All near-duplicate videos were created using the MPEG2 codec, whilst the source videos were encoded as MPEG4. Hence, the videos are not only altered with respect to the described edit tasks, but also with respect to a codec change. This codec change leads to a slight change in the RGB values of the pixel that is almost inperceivable to the human eye.

We evaluated the effectiveness of the retrieval methods on all of these individual tasks, using 100 queries taken from the original source videos. For the competitive methods, we used the same parameters as in the previous section. For FlexVis, we tested the effect of the time dimension weight in the clustering for $T = 1, ..., 10$ and used the parameter leading to the highest precision for the respective tasks: *Bar Insertion with Overlaying*: 6, *Bar Insertion with Stretching*: 7, *Brightness*: 7, *Contrast*: 5, *Flip*: 9, *Frame Deletion*: 9, *Frame Disordering*: 1, *Playback Speed*: 3, *Resize*: 8, *Text Insertion*: 5.

Figure 7.3 shows Precision Recall graphs for all of the edit tasks. As we can see, FlexVis with EMD outperforms the other methods in all edit tasks except for *Flip*. Overall, we can say that BCS and ViTri are serious competitive methods that seem to be limited by the choice of features (64-dimensional RGB histograms), the expressiveness of their video representation models and possibly the unsuitability of their similarity or distance measure. FRAS and VDT yield a low precision on almost all of the tasks, making them unsuitable for near-duplicate detection with respect to the presented edit tasks. This does not mean that the underlying ideas of these methods are unsuitable. We observed that the methods yield significantly higher precision when the improvements suggested in Chapter 3 are applied, but for our experiments we decided to use them in the form they were originally presented in their respective papers.

The databases *Brightness* (c), *Frame Disordering* (g) and *Playback Speed* (h) contain an unaltered copy of the original video in each category, which only differs with respect to

the utilized codec. The precision at recall level 0 therefore tells us how good the several retrieval methods are at detecting the unaltered version of the original video as the nearest neighbor and, therefore, gives us a sense of how robust the methods are to codec changes. As we can see, FlexVis with EMD and SQFD achieve a precision of 100%, which shows that the choice of features and the signature representation model are very robust to codec changes. BCS and ViTri yield a slightly worse precision, which can be explained by the fact that the codec change causes a slight change in the RGB histograms, which seems to have a comparatively large impact on the utilized distance/similarity measures, causing other videos to be more similar to some of the queries than their near-duplicate copies. VDT and FRAS yield even worse precisions, which suggests that they are non-robust to slight changes in the RGB histograms. This lack of robustness to the codec change has to be considered as a baseline when judging the performance of the retrieval methods for the several edit tasks.

Bar Insertion with Overlaying (a) and *Bar Insertion with Stretching* (b) yield similar Precision Recall graphs. After about 5 videos were retrieved, all of the methods have a precision of 20% or less. This suggests that after a sufficiently large part of the video is filled with black bars, the contribution of the black bars to the distance/similarity measures seems to be so high that videos with the same amount of black area are more similar to each other than to the source video from which they were created. Interestingly, after about 3 videos the precision of FlexVis with EMD drops dramatically, suggesting that small changes in the signatures can cause a high jump in the EMD. The other methods exhibit a smoother, line-like decay of precision with increasing bar sizes. BCS performs better on these tasks than ViTri, suggesting that the directions of largest variance used by BCS are more robust to the insertion of black bars than the number of similar frames that ViTri tries to approximate. Depending on the frame similarity threshold, two unrelated frames might be regarded as similar by ViTri if they both have a large-enough amount of black area, exacerbating the discriminability of the videos with large bar sizes.

For *Brightness* (c) and *Contrast* (d), FlexVis with EMD yields significantly larger precision than SQFD, suggesting that putting all pairs of representatives into relation with each other, as SQFD does, is not as meaningful for these tasks as matching representatives locally, as EMD does. All competitive methods yield a significant drop in precision after the first few videos, which is probably due to the fact that the utilized RGB histograms are not as robust to changes in Brightness/Contrast as the FlexVis features. For ViTri, this drop can further be explained by the fact that after a certain amount of brightness change, the distances between the query frames and the corresponding altered frames exceed the frame similarity threshold, causing a jump in the number of visually similar frames. After 8 videos (for Brightness) and 9 videos (for Contrast), all methods except FlexVis have a

precision of 0, which suggests that every query video Q is regarded to be more similar to other videos of a similar brightness than to a version of Q with very high or very low brightness.

In *Flip* (e), we observe that FlexVis performs worse than BCS and ViTri. The reason for this is that flipping the frames does not alter the resulting RGB histograms, and reversing the time does not alter the ViTri and BCS representations, since they are based on sets of frame histograms and do not take their ordering into account. Therefore, BCS and ViTri should ideally yield the same video representations for all versions of this video and, hence, result in optimal precision. The fact that they do not yield perfect precision is again due to slight differences in the encoding. Flipping the frames and reversing the time does cause a change in the FlexVis features (corresponding to a flipping of the x, y and t axes, respectively). This increases the ground distance between the representatives of the source video and the corresponding representatives of the flipped video, increasing the overall distance and, hence, making it hard to discriminate between the flipped versions of the source video and other videos. According to the precision values, this effect seems to be even stronger for EMD than for SQFD, suggesting that putting the representatives into pairwise relation seems to allow for better detection of flipped versions of the video than local matching. This can be explained by considering that the EMD flow does not always match the representatives of the original video to their corresponding flipped versions, whilst the similarity between the representatives and their flipped version at least contributes to the SQFD. VDT and FRAS yield a drop after 3 of the 6 videos, which is due to the fact that their video representations are based on frame sequences, giving importance to the temporal ordering and, hence, making it hard to detect the 3 videos that are reversed along the time axis.

We observe that EMD yields perfect accuracy for *Frame Deletion* (f), suggesting that it is fairly robust to the fact that the source signature contains centroids that have no similar match in the target signature. SQFD is slightly less robust to this, which can be explained by the fact that these dissimilar centroids are put into relation with all target centroids, leading to a high contribution to the SQFD value. BCS performs only slightly worse, prompting that the overall frame vector distribution that BCS models is largely robust to the removal of frame vectors. ViTri performs similarly well, since frame removal reduces the number of visually similar frames by the number of removed frames, causing just a slight change in the ViTri measure. VDT performs better on this task than on the other tasks, since the LSFs that do not include the removed frames stay largely unaffected by the removal of frames, causing a sufficiently large amount of query LSFs to be matched. Similarly, FRAS performs better than on the other edit tasks, since ideally the removal of frames only contributes the cost for removing one symbol per removed frame to the edit

distance.

For *Resize* (i), EMD yields almost perfect accuracy. SQFD retrieves about 70% of the near-duplicates correctly, exhibiting a sudden drop of precision afterwards. Since the position, time and color dimensions of the feature vectors are invariant under resizing, this lack of perfect accuracy can be attributed to the contrast and coarseness dimensions since they are not invariant under resizing. Moreover, it can be explained by the fact that we use slightly different sample pixels, leading to a slightly different clustering of the features. The resulting differences in the centroid positions apparently have a larger impact on the accuracy of SQFD than on EMD. Resizing does not affect the RGB histograms, so the lack of accuracy of the competitive methods is again due to the usage of different sample pixels and due to differences in the codec.

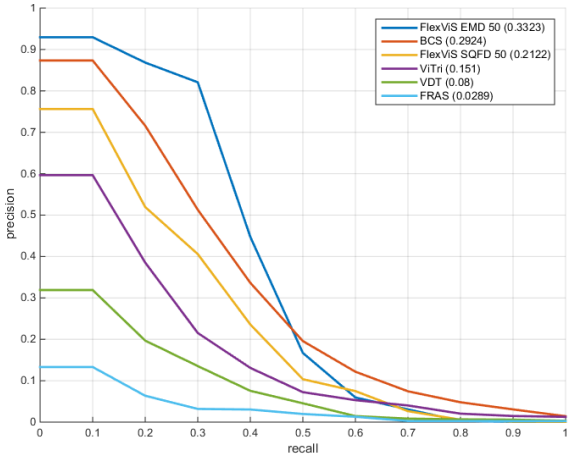
For *Text Insertion* (j), the inserted text results in centroids in the near-duplicate videos' feature signatures that have no corresponding match in the signatures of the original videos. This seems to have only a slight impact on the accuracy of EMD and SQFD. VDT performs considerably better on this task than on the other tasks, suggesting that the distances of the frames to the reference point are comparatively robust to addition of frame content since the reference point is chosen individually for each video.

7.2 Efficiency

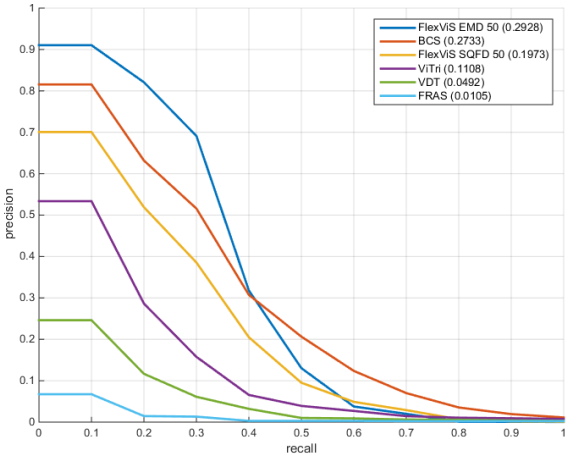
Our first set of efficiency experiments aimed at testing the effect of the database size on the query processing time. It was carried out on databases of 150000, 200000, 250000, 300000, 350000 videos, created by combining the near-duplicate databases of all edit tasks. We used 100 queries from the original source videos and retrieved the 100 nearest neighbors for each query using different lower bounds. For the metric lower bound, we used 30 randomly selected pivot elements. The experiments were carried out on an *Intel Xeon E7-4850* with 2.3 GHz without parallelization.

Figure 7.4 shows the average query processing times of the FlexVis retrieval methods with signature size 30 and the competitive methods for different database sizes. As we can see, the FlexVis methods perform worse than the competitive methods, which is due to the fact that the distance measures (EMD and SQFD) are more costly to compute. The kNN retrieval with SQFD is faster than with EMD by a factor of 201.9% on average, which can be attributed to the theoretical time complexities of the distance measures. EMD with IMSig performs faster on the compressed signatures than on the original signatures by a factor of 456.99% on average.

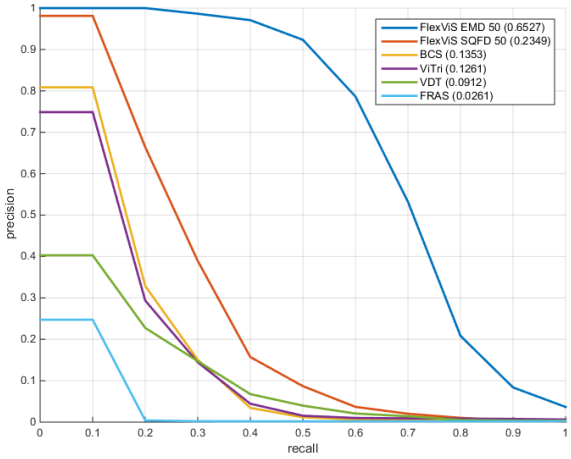
Figure 7.5a shows a comparison of the query processing times of EMD-based retrieval



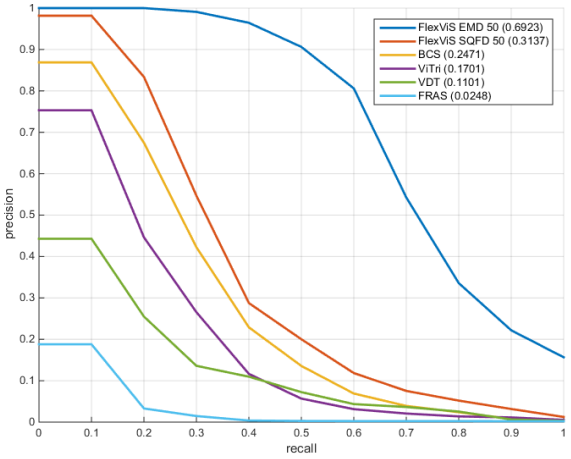
(a) Bar Insertion with Overlaying



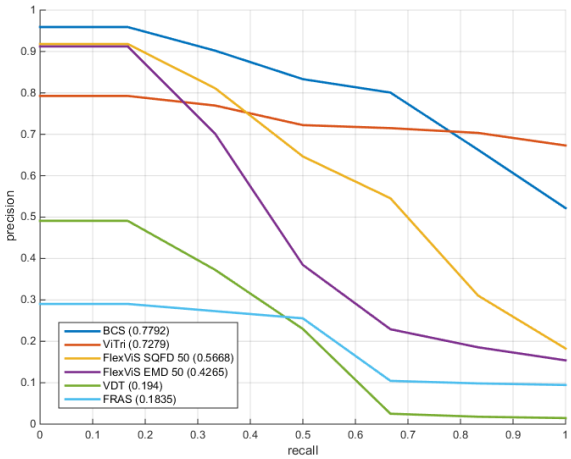
(b) Bar Insertion with Stretching



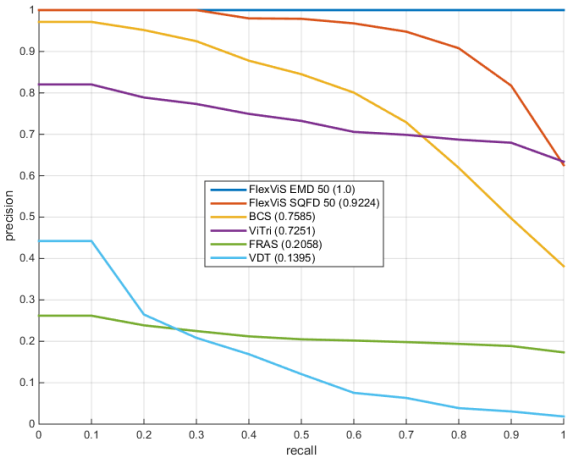
(c) Brightness



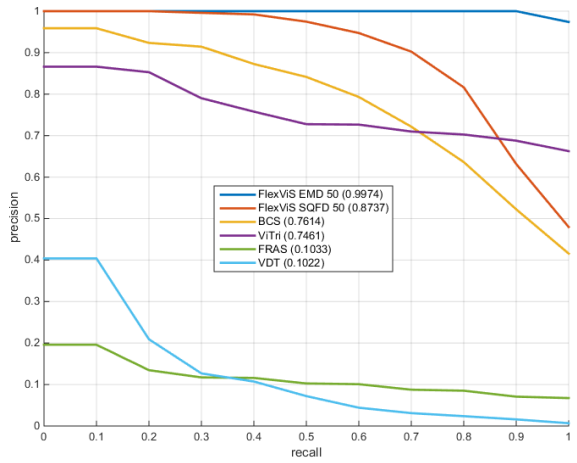
(d) Contrast



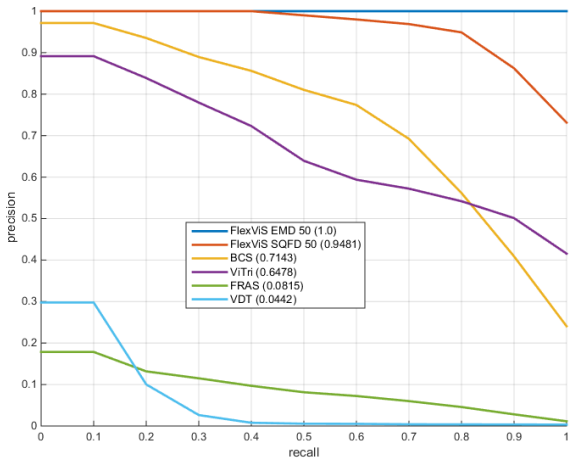
(e) Flip



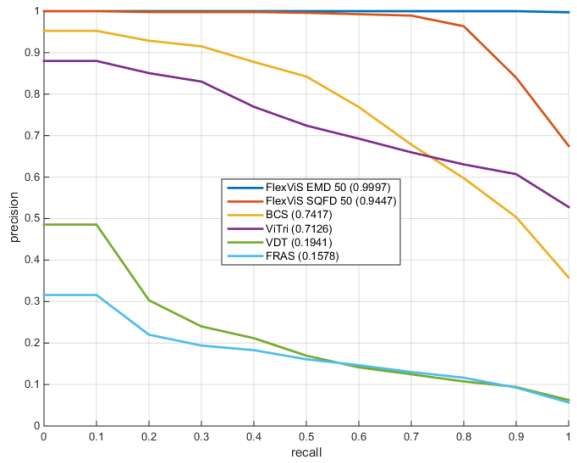
(f) Frame Deletion



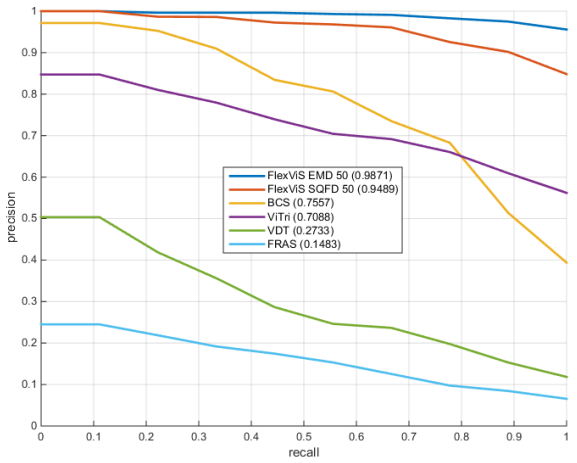
(g) Frame Disordering



(h) Playback Speed



(i) Resize



(j) Text Insertion

Figure 7.3: Precision Recall graphs for the near-duplicate databases

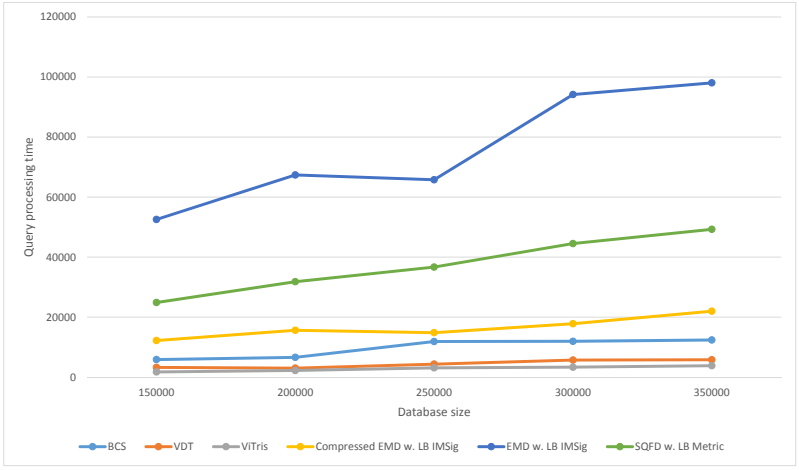


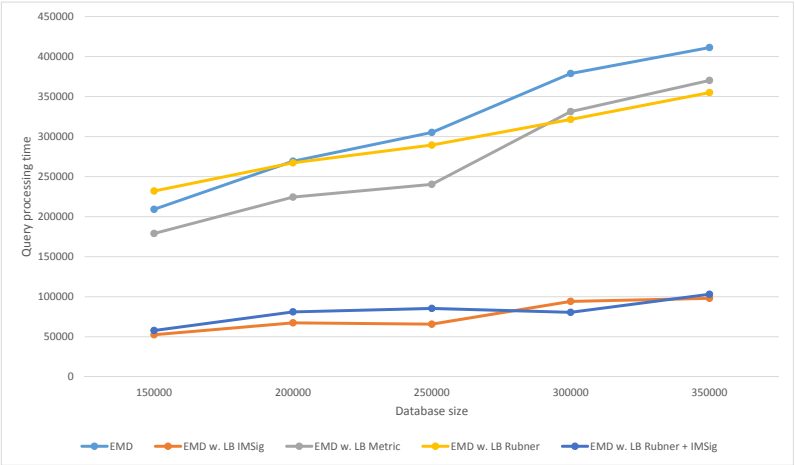
Figure 7.4: Query processing times (in ms) of FlexVis with 30 representatives on different database sizes in comparison to competitive methods

methods with different lower bounds. The query processing times are highly influenced by the number of EMD computations displayed in Figure 7.5b. The best selectivity and query processing time is achieved when using the Multi-Step kNN Algorithm in combination with the IMSig lower bound, yielding an average speedup factor of 416.43% in comparison to a sequential scan with EMD. The Rubner lower bound performs comparatively bad on this dataset, yielding an average selectivity of only 84.44%. In past experiments conducted on image databases, it yielded a selectivity of 33.22% (cf. [UBSS14]), suggesting that the bad selectivity is an attribute of the near-duplicate database. Apparently, the ranking with respect to Rubner is dissimilar to the ranking with respect to EMD on this dataset, leading to a bad selectivity of the Multi-Step kNN Algorithm.

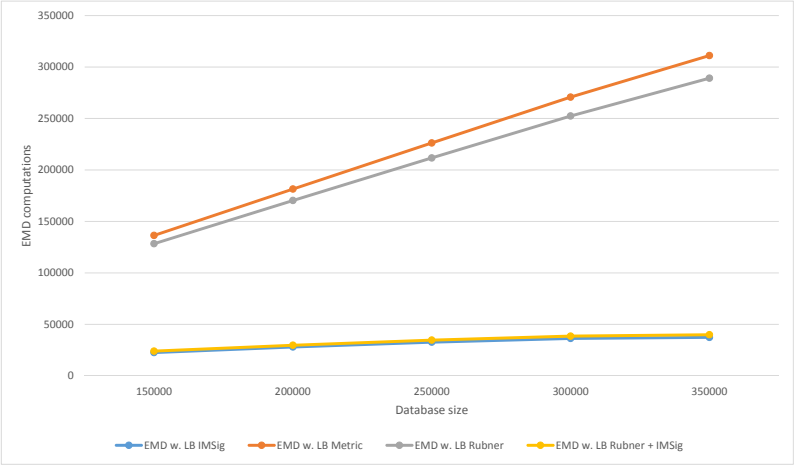
This also explains why using only the IMSig lower bound yields a better selectivity and better query processing time than using a combination of Rubner and IMSig, since in the latter case the ranking is carried out with respect to the Rubner lower bound, causing the pruning distance to decrease more slowly. This effect is best demonstrated by an example. Assume that our database consists of the videos A, B and C and that their distances to a query Q with respect to EMD, the IMSig lower bound and the Rubner lower bound are as follows:

	A	B	C
EMD	1	2	3
IMSig	1	2	3
Rubner	1	0.5	0

If we issue a 1-Nearest Neighbor query using the IMSig lower bound, our initial ranking

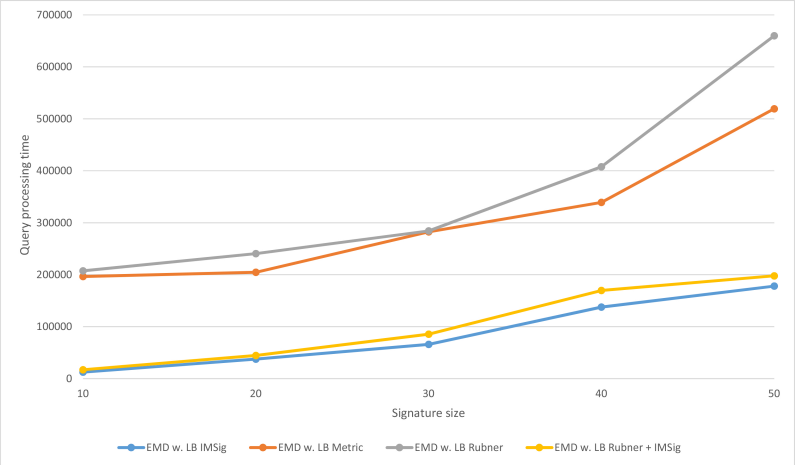


(a) Query processing times

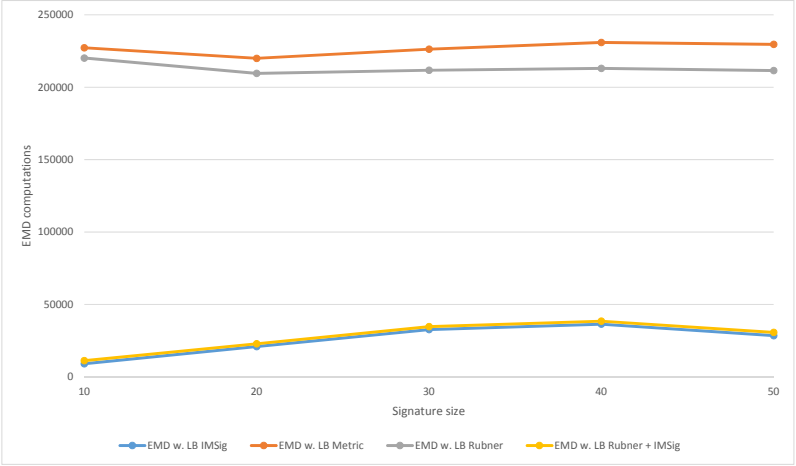


(b) Number of EMD computations

Figure 7.5: EMD efficiency with 30 representatives on different database sizes using different lower bounds



(a) Query processing times (in ms)



(b) Number of EMD computations

Figure 7.6: EMD efficiency on a database with 250000 videos using different lower bounds and different numbers of representatives

will be A, B, C. In the first iteration, we add A to the result set. In the second iteration, we compute $EMD(Q, A) = 1$ and $EMD(Q, B) = 2$. We then fix $\epsilon_{max} = EMD(Q, A) = 1$, causing the IMSig filter in the third iteration to exceed ϵ_{max}, allowing us to skip the third EMD computation. If, on the other hand, we issue the query using the Rubner and IMSig lower bound, our initial ranking will be C, B, A. In the first iteration of the algorithm, we add C to the result set. In the second iteration, we compute $EMD(Q, C) = 3$ and $EMD(Q, B) = 2$, replacing C with B in the result set and setting $\epsilon_{max} = 2$. In the third iteration, we observe that neither $IMSig(Q, A) = 1$ nor $Rubner(Q, A) = 1$ is greater than ϵ_{max}, requiring us to perform the third EMD computation $EMD(Q, A)$.

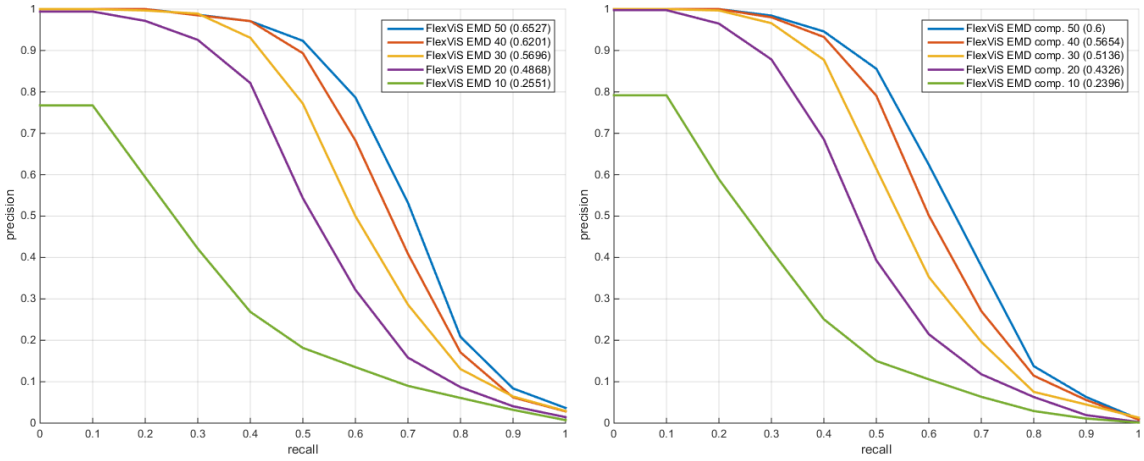
It is notable that the Metric lower bound has a slightly worse selectivity than the Rubner lower bound but still yields a better query processing time, since it can be computed more efficiently.

The next set of experiments aimed at testing the effect of the signature size on the query processing time and was carried out on a database of 250000 videos, from which we extracted feature signatures of sizes 10, 20, 30, 40, and 50. Figure 7.6 shows the query processing times and number of EMD computations for the different signature sizes. The Rubner and metric lower bound yield a largely constant number of EMD computations, suggesting that their approximation quality is independent of the signature size. The query processing times with these lower bounds do however increase with the number of representatives, which is naturally due to the cubic to quartic time complexity of the EMD computation. Interestingly, using the IMSig lower bound yields a better selectivity for 50 representatives than for 40 representatives, suggesting that the approximation quality of IMSig on this dataset gets better with increasing signature size.

7.3 EMD with Compressed Feature Signatures

We evaluated the approximation quality and computation time of EMD with compressed feature signatures on the Visual Similarity Database with a signature size of 100 using the distance threshold $\epsilon = 1$. The following table states statistics about the error, removed centroids and efficiency.

Average error $\lvert EMD(X, Y) - EMD(Comp_{\mathcal{C}}(X), Y) \rvert$	0.03607
Maximum error $\lvert EMD(X, Y) - EMD(Comp_{\mathcal{C}}(X), Y) \rvert$	0.11040
Average number of removed centroids	46.12
Average duration for original EMD computation	16.77821 ms
Average duration for EMD computation on compressed signature	6.38717 ms
Speedup factor	263%



(a) Original signatures

(b) Compressed signatures ($\epsilon = 1$)

Figure 7.7: Precision Recall graphs for EMD on the Brightness dataset

As we can see, we get a significantly better efficiency in comparison to the EMD computation on the original signature. We observe that the maximum observed error of 0.1104 is far below the theoretically proven error bound $1.5\epsilon = 1.5$.

Our next experiment aimed at measuring the precision of $EMD(Comp_{\mathcal{C}}(X), Y)$ with respect to $EMD(X, Y)$, i.e. the fraction of the k-Nearest Neighbors with respect to $EMD(Comp_{\mathcal{C}}(X), Y)$ that are contained in the k-Nearest Neighbors with respect to $EMD(X, Y)$. This experiment was carried out on the near-duplicate database with a signature size of 50, using 100 of the original videos as queries. For $k = 25, 50, 100, 200$ we observed precisions of $0.3944, 0.4118, 0.3933, 0.3594$, respectively. This suggests that even though the errors in the EMD values caused by the compression are comparatively small, the k-Nearest Neighbors with respect to the different EMD values differ significantly.

We evaluated the effectiveness of EMD with compressed feature signatures on the *Brightness* near-duplicate database. Figure 7.7 shows Precision Recall graphs of EMD on the original signatures and the compressed signatures with a distance threshold $\epsilon = 1$. We observe that the approximative nature of EMD on the compressed signatures leads to slightly worse precision values, yielding a deviation of the MAP by 0.04662 on average. Similar results are obtained for the other edit tasks, but they are omitted at this point due to space limitation.

7.4 Subclip Search

We evaluated the algorithm presented in Chapter 6 on the documentary *Home* by Yann Arthus-Betrand [AB09], which is a documentary movie with a duration of 1 hour, 33

minutes and 18 seconds mostly featuring aerial shots of various places on earth. We computed a feature signature with 55980 representatives on a set of 150 features per frame at 10 frames per second. Additionally, we extracted 933 disjoint subclips of 6 seconds each and computed individual feature signatures on all of these subclips. We then used the subclips as queries and counted how many of the subclips were identified at the correct position in the movie by our algorithm.

63.45% of the subclips were identified at the correct position, whilst 74.38%, 86.28% and 90.46% were identified within 6, 12 and 18 seconds of their original position, respectively. This suggests that most of the errors are due to nearby clips being similar to the query subclips since they show similar motifs.

The effect of different sampling sizes and different numbers of representatives on the subclip detection accuracy has yet to be investigated.

We have seen how the several retrieval methods perform on different datasets. Summarizing the results, we can conclude that the FlexVis model yields significantly better effectiveness than the competitive methods at the expense of a worse efficiency. The next chapter gives a conclusion to this thesis and suggestions for future research directions.

Chapter 8

Conclusion

In this thesis, we have given an overview of some of the state-of-the-art content-based video retrieval methods and stated their shortcomings. Moreover, we have seen how we can adapt the feature model for content-based image retrieval from [BUS10a] to videos easily by adding the time as another dimension to the feature vectors. Additionally, we have seen how our model can be adapted to allow for subclip detection.

Our experiments show that the resulting model leads to high effectiveness, outperforming the considered competitive methods with respect to detecting visually similar videos, and with respect to near duplicate detection in most of the considered tasks. However, this effectiveness comes at the price of a worse efficiency.

A future research direction consists in further investigating the efficiency of the several approaches on different datasets. The fact that we performed the efficiency experiments on a near-duplicate database containing many similar videos might cause the efficiency results to be unrepresentative of real-world databases with mostly unrelated videos. Since we observed that the IM-Sig selectivity improves when using 50 centroids instead of 40, the question arises whether the selectivity improves further when using even more representatives, which has yet to be investigated.

Since our method leads to comparatively high query processing times, there is a need for further research on how we can perform the retrieval more efficiently. Possible directions could include the investigation of other lower bounds or other indexing methods (e.g. *iDistance*, cf. [JOT+05]) for our similarity model.

Another research direction of interest could be to test the effect of different feature dimension weights on the effectiveness of the FlexVis method. Weights might also be incorporated in the distance computation, not just the clustering.

The definition of the Translation-Invariant Earth Mover's Distance shows potential for

other tasks than subclip search, e.g. searching sub-images within a larger image or improving the detection of near-duplicates by allowing translations along the color and texture dimensions. Future research could be concerned with finding better approximations of TIEMD and with extending the possible translations to other feature dimensions and testing the effectiveness of this approach for sub-image search or near-duplicate video detection. TIEMD could also be generalized to allow scaling along the feature dimensions, making it invariant under even more video transformations like contrast change, resizing or playback speed alteration.

Bibliography

[AB09] Yann Arthus-Bertrand. Home `http://www.homethemovie.org/`, 2009.

[AWS06] Ira Assent, Marc Wichterich, and Thomas Seidl. Adaptable distance functions for similarity-based multimedia retrieval. *Datenbank-Spektrum*, 19:23–31, 2006.

[BS13] Christian Beecks and Thomas Seidl. *Distance based similarity models for content based multimedia retrieval.* PhD thesis, Aachen, 2013. Zsfassung in dt. und engl. Sprache; Aachen, Techn. Hochsch., Diss., 2013.

[BUS10a] Christian Beecks, Merih Seran Uysal, and Thomas Seidl. A comparative study of similarity measures for content-based multimedia retrieval. In *Multimedia and Expo (ICME), 2010 IEEE International Conference on*, pages 1552–1557. IEEE, 2010.

[BUS10b] Christian Beecks, Merih Seran Uysal, and Thomas Seidl. Signature quadratic form distance. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, pages 438–445. ACM, 2010.

[CZ03] Sen-ching Samson Cheung and Avideh Zakhor. Efficient video similarity measurement with video signature. *Circuits and Systems for Video Technology, IEEE Transactions on*, 13(1):59–74, 2003.

[DD09] Michel Marie Deza and Elena Deza. *Encyclopedia of distances.* Springer, 2009.

[FBF+94] Christos Faloutsos, Ron Barber, Myron Flickner, Jim Hafner, Wayne Niblack, Dragutin Petkovic, and William Equitz. Efficient and effective querying by image content. *Journal of intelligent information systems*, 3(3-4):231–262, 1994.

[HKP06] Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining, southeast asia edition: Concepts and techniques.* Morgan kaufmann, 2006.

[HSS+09] Zi Huang, Heng Tao Shen, Jie Shao, Xiaofang Zhou, and Bin Cui. Bounded coordinate system indexing for real-time video clip search. *ACM Transactions on Information Systems (TOIS)*, 27(3):17, 2009.

[HSS+10] Zi Huang, Heng Tao Shen, Jie Shao, Bin Cui, and Xiaofang Zhou. Practical online near-duplicate subsequence detection for continuous video streams. *Multimedia, IEEE Transactions on*, 12(5):386–398, 2010.

[HWS+09] Zi Huang, Liping Wang, Heng Tao Shen, Jie Shao, and Xiaofang Zhou. Online near-duplicate video clip detection and retrieval: An accurate and fast system. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 1511–1514. IEEE, 2009.

[Jol02] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[JOT+05] Hosagrahar V Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. idistance: An adaptive b+-tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems (TODS)*, 30(2):364–397, 2005.

[KLB+11] Martin Kruliš, Jakub Lokoč, Christian Beecks, Tomáš Skopal, and Thomas Seidl. Processing the signature quadratic form distance on many-core gpu architectures. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 2373–2376. ACM, 2011.

[LHSB11] Jakub Lokoč, Magnus Lie Hetland, Tomáš Skopal, and Christian Beecks. Ptolemaic indexing of the signature quadratic form distance. In *Proceedings of the Fourth International Conference on SImilarity Search and APplications*, pages 9–16. ACM, 2011.

[MRS08] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[ROS+14] Miriam Redi, Neil O'Hare, Rossano Schifanella, Michele Trevisiol, and Alejandro Jaimes. 6 seconds of sound and vision: Creativity in micro-videos. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE Computer Society, 2014.

[RTG00] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

[SJ08] Sameer Shirdhonkar and David W Jacobs. Approximate earth mover's distance in linear time. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[SK98] Thomas Seidl and Hans-Peter Kriegel. Optimal multi-step k-nearest neighbor search. In *ACM SIGMOD Record*, volume 27, pages 154–165. ACM, 1998.

[SOZ05] Heng Tao Shen, Beng Chin Ooi, and Xiaofang Zhou. Towards effective indexing for very large video sequence database. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 730–741. ACM, 2005.

[TMY78] Hideyuki Tamura, Shunji Mori, and Takashi Yamawaki. Textural features corresponding to visual perception. *Systems, Man and Cybernetics, IEEE Transactions on*, 8(6):460–473, 1978.

[UBSS14] Merih Seran Uysal, Christian Beecks, Jochen Schmücking, and Thomas Seidl. Efficient filter approximation using the earth mover's distance in very large multimedia databases with feature signatures. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 979–988. ACM, 2014.

[Van01] Robert J Vanderbei. Linear programming. *Foundations and extensions, International Series in Operations Research & Management Science*, 37, 2001.

[Wik15a] Wikipedia. Edit distance `http://en.wikipedia.org/wiki/Edit_distance`, 2015.

[Wik15b] Wikipedia. Lab color space `http://en.wikipedia.org/wiki/Lab_color_space`, 2015.

[You15] YouTube. Youtube statistics `https://www.youtube.com/yt/press/statistics.html`, 2015.

[ZADB06] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity search: the metric space approach*, volume 32. Springer Science & Business Media, 2006.

[ZZS07] Xiangmin Zhou, Xiaofang Zhou, and Heng Tao Shen. Efficient similarity search by summarization in large video database. In *Proceedings of the eighteenth conference on Australasian database-Volume 63*, pages 161–167. Australian Computer Society, Inc., 2007.