Efficient adaptive retrieval and mining in large multimedia databases

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften
der Rheinisch-Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades einer Doktorin der
Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatikerin
Ira Assent

aus Aachen

Berichter: Universitätsprofessor Dr. rer. nat. Thomas Seidl
Professor Dr. Christian Jensen

Tag der mündlichen Prüfung: 26. Februar 2008

To my late parents

Contents

List of Figures

Acknowledgements

Many people have played an important role in this thesis. I would like to express my thanks to each and everyone of you, even if not explicitly mentioned here.

Foremost, I would like to thank my supervisor, Prof. Dr. Thomas Seidl for initiating this research, for many interesting and fruitful discussions, and much support. You have always been accessible and willing to help.

I am very grateful to Prof. Dr. Christian Jensen, for agreeing to act as a co-referee for this thesis. Thank you for your interest in my research and for very useful discussions.

Thanks to all my colleagues and students in our group for both interesting discussions and a very friendly and enjoyable atmosphere.

And last but not least, I would like to thank all my friends and family. Most importantly, thank you, Eric, for all your love and support.

I dedicate this thesis to my late parents. Thank you for your love and encouragement. I wish I could celebrate completion of this thesis with you.

Abstract

Multimedia data ranging from images to videos and time series is created in numerous scientific, commercial and home applications. Access to increasingly large data volumes stored in multimedia databases is a core task to retrieve similar objects or to generate an overview of the entire content. Examples include retrieval of similar magnetic resonance images for diagnostic purposes, or automatic detection of customer segments for sales promotion.

Meaningful retrieval and pattern detection require content-based methods that describe the relevant characteristics of multimedia objects. As opposed to manual keyword annotation techniques that are typically infeasible for large data volumes, content-based approaches use similarity models to process multimedia data. Similarity models specify appropriate features and their relationship for effective content based access.

As most multimedia features require many different attributes, high dimensionality of multimedia features and huge database sizes are major challenges for efficient and effective retrieval and mining.

In this work, very common feature types for multimedia data are studied: histogram and time series data. Histograms are used for a variety of features such as color, shape or texture. Time series data is prevalent for sensor measurements, stock data, and may even be applied to shapes and other features as well. For these data types, effective adaptable similarity

models are usually computationally far too complex for usage in large high dimensional multimedia databases. Therefore efficient algorithms for these effective models are proposed.

In this work, indexing techniques are used that allow for efficient query processing and mining by restricting the search space to task relevant data. Multistep filter-and-refine approaches using novel filter functions with quality guarantees ensure that fast response times are achieved without any loss of result accuracy.

This thesis is structured as follows: first, in the Preliminaries, an overview over the thesis and the major challenges in multimedia retrieval and mining is given. Part I discusses histogram retrieval, Part II studies time series retrieval. In Part III, efficient and effective histogram data mining is proposed, and Part IV presents novel time series mining techniques. Finally, this work is concluded and future research directions are suggested.

Zusammenfassung

Multimediadaten, von Bildern über Videos bis hin zu Zeitreihen, entstehen in vielen wissenschaftlichen, kommerziellen oder privaten Anwendungen. Der Zugriff auf die zunehmend großen Datenmengen, die in Multimediadatenbanken gespeichert werden, ist eine Hauptaufgabe für die Suche ähnlicher Objekte oder für Überblicksdarstellungen des gesamten Datenbankinhalts. Beispiele sind die Kernspinresonanztomographie für diagnostische Zwecke oder automatische Ermittlung von Kundengruppen für die Verkaufsförderung.

Sinnvolle Suche und Mustererkennung setzen inhaltsbasierte Methoden voraus, die die relevanten Charakteristika der Multimediadaten beschreiben. Im Gegensatz zur manuellen Verschlagwortung, die typischerweise für große Datenmengen nicht praktikabel ist, verwenden inhaltsbasierte Methoden Ähnlichkeitsmodelle um Multimediadaten zu verarbeiten. Ähnlichkeitsmodelle spezifizieren entsprechende Merkmale und deren Verhältnis für effektiven inhaltsbasierten Zugriff.

Da die meisten Multimediamerkmale viele verschiedene Attribute voraussetzen, stellen die hohe Dimensionalität der Multimediamerkmale sowie die riesigen Datenbankgrößen hauptsächliche Herausforderungen für effizienten und effektiven Suchzugriff und Data Mining dar.

In dieser Arbeit werden weitverbreitete Merkmalstypen für Multimedi-

adaten untersucht: Histogramm- und Zeitreihendaten. Histogramme werden für viele verschiedene Merkmale wie Farbe, Form oder Textur genutzt. Zeitreihen sind vorherrschend im Bereich Sensormesswerte, Börsendaten und können sogar für Formen und weitere Merkmale verwendet werden. Für diese Datentypen sind effektive Ähnlichkeitsmodelle normalerweise von zu hoher Berechnungskomplexität, als daß sie in großen hochdimensionalen Multimediadatenbanken eingesetzt werden könnten. Daher werden effiziente Algorithmen für diese effektiven Modelle vorgeschlagen.

Genutzt werden in dieser Arbeit Indexierungstechniken, die effiziente Anfragebearbeitung und Mining ermöglichen, indem sie den Suchraum auf die für die Aufgabe relevanten Daten beschränken. Mehrstufige Filter-und-Verfeinerungsansätze mit neuen Filterfunktionen, die Qualitätsgarantien erfüllen, stellen sicher, daß schnelle Antwortzeiten ohne Qualitätsverlust erreicht werden.

Diese Arbeit ist wie folgt gegliedert: zunächst wird in der Einleitung ein Überblick über die Arbeit und die Hauptherausforderungen im Zugriff und Mining auf Multimediadaten gegeben. Teil I erörtert Zugriff auf Histogrammdaten, Teil II auf Zeitreihen. In Teil III wird effizientes und effektives Data Mining für Histogramme vorgeschlagen, und Teil IV stellt neue Miningtechniken für Zeitreihen vor. Zum Schluß wird diese Arbeit zusammengefasst und zukünftige Forschungsrichtungen werden vorgeschlagen.

Preliminaries

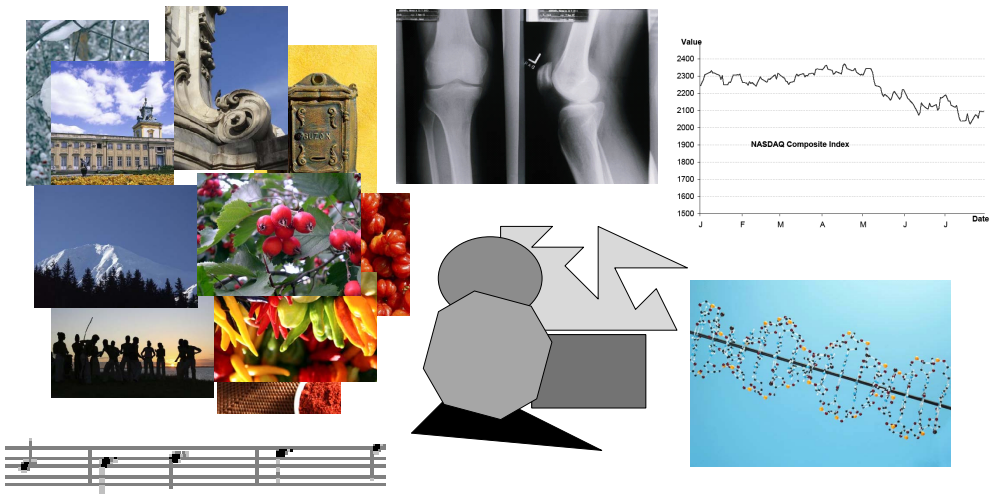


Figure 1: Examples of multimedia data

Multimedia Databases

There is tremendous growth in multimedia data in application domains such as medicine, engineering, biology and numerous others. New technologies such as computer tomography or digital photography produce increasingly large volumes of data. At the same time, decreasing storage prices allow archiving large amounts of multimedia data at relatively low cost. For database technology, large multimedia databases require efficient and effective strategies for accessing and processing of multimedia data. In this part, we give an overview over the novel challenges when dealing with large volumes of multimedia data, outline the main characteristics of typical data types and the query and mining tasks typically associated with multimedia applications.

Multimedia (lat. multum, medium) data is the combination of multiple types of information representations. Typical examples include images, audio, time series, or video. Multimedia is typically characterized by differ-

ent channels of communication (senses) in humans. Humans listen to audio, look at images, or use both senses in video. Their combination to multimedia allows humans to access information in various ways. Examples of typical multimedia content are given in Figure 1: stock data time series, music, medical magnetic resonance images, shape and color images, as well as gene expression data.

Storing and accessing multimedia content may be supported by database technology [Pra97]. Traditional database management systems have established themselves as reliable and powerful techniques for archiving and retrieving textual or numeric data. Providing guarantees in terms of recovery, concurrency control, multi-user isolation, and data durability, database management systems provide indexes on stored content for efficient access. Data is accessed in a key-based manner, which is appropriate for traditional data types.

Multimedia content access leads to novel challenges in database technology. Besides direct retrieval based on key values such as the date an image was created, novel access methods are required. Meta data information is typically not adequate for most applications, e.g. users typically search for images not with respect to their size on hard disk, but with respect to content related features like color or shape. Consequently, multimedia database systems should provide content-based access. This allows users to access the data in a meaningful way, e.g. by supporting the diagnostic process of medical experts by automatically retrieving magnetic resonance images of similar medical examination results.

Likewise, aggregating the content of multimedia databases is not straightforward. Unlike traditional reporting tasks where database queries for the (aggregated) number of sales in a given month for a certain store do not ex-

tend to multimedia content in a straightforward manner. Aggregating multimedia objects, or, more precisely, their respective feature values, in such a way, is not meaningful. For example, the average feature distribution is not an information that allows users to understand the content of their multimedia databases. Knowledge extraction techniques aggregate multimedia data with respect to type of media and application focus such that meaningful summaries are formed. This allows users to understand the type of data that is gathered by their applications and to detect trends and constant patterns that might remain unknown to them otherwise.

Multimedia databases thus should provide

- content-based access
- generation of knowledge
- handling of usually large data volumes
- coping with typically high dimensional features
- good response time performance

In this work, we focus on efficient content-based retrieval and mining.

Figure 2: Similarity search

Similarity search and retrieval

Content-based retrieval searches for similar multimedia objects to a query object (Figure 2) [Fal96]. It requires efficient algorithms on effective similarity models. Depending on the feature model, the distance function and the type of query, efficient algorithms should retrieve the correct result set.

Depending on the type of query, either all objects that do not deviate from the query by more than tolerance threshold are retrieved (range query), or a fixed number of most similar objects forms the result set (nearest neighbor query). Figure 3 illustrates a range query in a two-dimensional feature space (left image). Given a tolerance threshold ε, all objects within at most ε

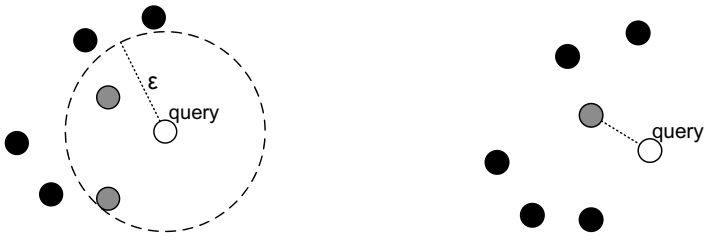


Figure 3: Range query (left) and nearest neighbor query (right)

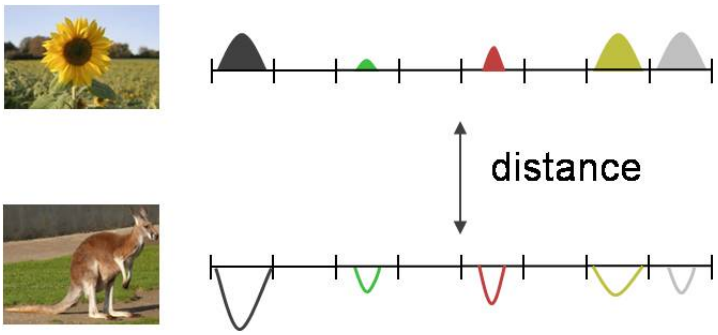


Figure 4: Distance between image histogram features

distance (depicted as a circular range around the query q) are retrieved. On the right hand side, a nearest neighbor query is illustrated. Given a fixed number k of neighbors, return all those k objects that are more similar to the query q than the rest of the database. In the illustration, the 1-nearest neighbor, the most similar object to the query, is marked in gray.

Efficient and effective retrieval requires adequate similarity models for the application domain. Similarity models define the relevant characteristics of multimedia objects, for example color distribution in images, or pixel frequencies in shapes. Such histogram features can be compared via distance functions which assign a degree of dissimilarity to any pair of histograms. An example is given in Figure 4. Two images and their respective color histograms, i.e. the relative frequencies of color pixels in each image, are depicted. To compare the images, distance functions compute a dissimilarity value on their histogram features.

The type of distance function is crucial for the effectivity of the similarity search process, as it models which multimedia objects are considered alike. For different types of features, different similarity models have been proposed. One of the simpler models that is commonly used for both histograms and time series is the Euclidean distance, a distance from the family

of L_p norms. Based on the differences of corresponding histogram bins or time series values, respectively, the overall dissimilarity between features is computed. Adaptable similarity models allow for incorporation of expert knowledge on application domain. The Earth Mover's Distance, introduced in computer vision, computes an overall match on two (histogram) features based on a ground distance in feature space. Likewise, Dynamic Time Warping, originally from speech recognition, allows for stretching and squeezing of time series to match time series features. We will formalize the respective distance functions and propose new efficient retrieval methods for these models in the course of this thesis.

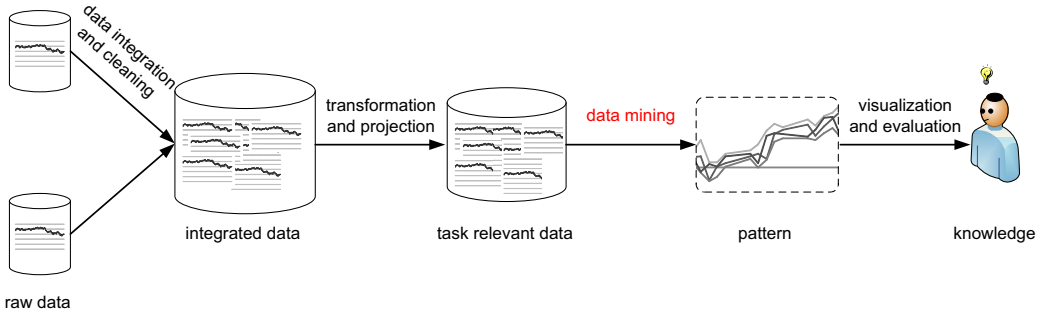


Figure 5: The knowledge discovery process

Data Mining

Knowledge discovery in databases (KDD) extracts novel, potentially useful patterns from databases for generation of knowledge about the database content as a whole.

The KDD process, depicted in Figure 5, consists of four steps from the original raw data to the actual knowledge generation [HK01]. In the first step, the data is integrated and cleaned of potential errors, then it is transformed and projected to the task relevant parts of the data. The central step, the data mining step, extracts patterns from this task relevant data, which is then visualized for human evaluation.

For data mining, a number of sub-tasks can be identified, depending on the focus of patterns that users are interested in. For example, classification assigns class labels to objects based on models trained on historical data, and association rule mining extracts general rules from transaction data. In this work, we focus on automatic aggregation of the data into meaningful groups, the so-called clustering.

Clustering groups objects such that similar ones are within the same group, whereas dissimilar ones are in different groups. Several clustering paradigms exist in the literature. The density-based paradigm defines clus-

ters as groups of density-based objects that are separated by sparsely populated regions.

We study efficient and effective density-based clustering for high dimensional and large databases in the course of this thesis.

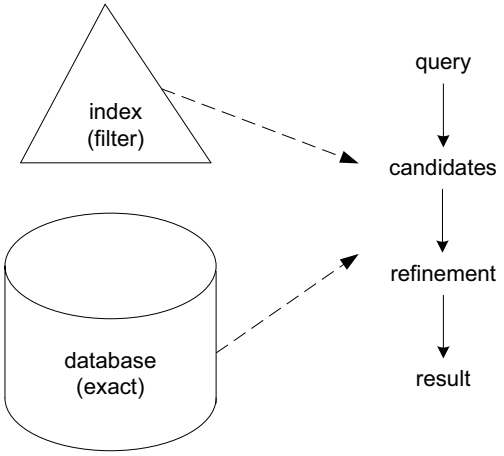


Figure 6: Multistep query processing

Structure of this thesis

In this work, we focus on efficient and effective retrieval and mining of multimedia data. We propose efficient algorithms for effective models on both histogram and time series data.

Key techniques of our work are database methods. Indexing organizes the data such that during query processing or mining, only relevant parts of the data have to be accessed to allow for efficient runtimes. Coupled with multistep filter-and-refine architectures, we are able to further enhance runtime performance by fast generation of candidate sets in a filter step that is refined to ensure neither false negatives nor false positives. This means, that no true result is dropped from the answer set, and that no unwanted object is included in the answer set. Figure 6 illustrates the basic idea: using an approximative index query, a set of candidates is generated, which is refined using the original database representation to yield the final result.

This thesis is thus organized in four parts: Parts I and II discuss retrieval of histogram data and time series data, respectively. In Parts III and IV,

histogram and time series data mining is studied, respectively. Finally, we conclude this thesis and outline future work.

Part I

Retrieving histogram data

Chapter 1

Introduction

Content-based similarity search requires effective similarity models as well as efficient query processing. Many multimedia features can be represented as histograms, i.e. vectors of attributes. For meaningful comparison of histogram features, distance functions provide dissimilarity values.

The Earth Mover's Distance was introduced in computer vision to model human perceptual similarities. Its computation, however, is too complex for usage in interactive multimedia database scenarios. In order to enable efficient query processing in large databases, we propose an index-supported multistep algorithm. We therefore develop new lower bounding approximation techniques for the Earth Mover's Distance which satisfy high quality criteria including completeness (no false dismissals), index-suitability and fast computation. We demonstrate the efficiency of our approach in extensive experiments on large image databases.

This part is structured as follows: In Chapter 2 we establish the notions and properties of the Earth Mover's Distance needed for our technique.

We then explain possible multistep algorithm concepts and their respective suitability in Chapter 3.

Chapter 4 presents several novel approximations of the Earth Mover's Distance along with an analysis of their properties and proofs of completeness. We give an extensive experimental evaluation of our multistep approach using real world color image databases.

Chapter 5 presents a case study for efficient video stream similarity search based on this multistep approach.

Exact indexing of the Earth Mover's Distance is discussed in Chapter 6.

For high-dimensional histograms, Chapter 7 introduces the vector approximation file and efficient similarity search using new Earth Mover's Distance. This includes new lower bounding approximation techniques in a multistep architecture.

Finally, we describe dimensionality reduction techniques on very high dimensional histograms for the Earth Mover's Distance in Chapter 8.

Chapter 2

Similarity search on histograms

There is tremendous growth in data produced and filed by applications in medicine, engineering, biology and numerous others. This is due to new technologies such as computer tomography as well as decreasing storage prices which allows filing of the data produced. These huge amounts of data call for efficient strategies for storing and retrieving multimedia data.

There has been substantial progress, especially in the area of content-based image retrieval (CBIR) where color images are retrieved from multimedia databases using three main concepts: first, a feature extraction model, which maps each image to a set of attributes, the so-called features. Secondly, a distance measure between these features which reflects the underlying similarity model. And finally, storage and retrieval methods for large image databases.

Several models have been proposed for the feature extraction process. The histogram approach, which maps each image to a vector of attributes encoding e.g. color distribution [HSE+95, SH94, SK97], texture [NBE+93, RPTB99], shape [AKKS99], has proven its usefulness in numerous applications and CBIR systems such as IBM's QBIC [FBF+94].

Comparing image histograms using L_p-norms such as the Euclidean distance is very sensitive to minor shifts in feature value distribution. Quadratic Forms use a ground similarity between feature bins to milden this sensitivity (cf. [FBF+94]). A recent approach from computer vision towards human similarity perception, the Earth Mover's Distance (EMD) [RGT97] models similarity as the amount of changes necessary to transform one image feature into another one.

The quality of the Earth Mover's Distance for image similarity has been demonstrated in several application scenarios: color [JLZZ04, RGT97], contour matching [GD04], texture [LSP03], melodies [TGV+03, TVW04], graphs [DSD+04], vector fields [LBH98], etc. Its high quality for image similarity search has been demonstrated in an extensive survey on distances for CBIR [RPTB99]. It has been and is being successfully used in a large number of similarity search settings including phishing detection [FWD06], medicine [DMHE+06], document retrieval [WP05], video news story tracking [UHMS06], speaker recognition [KUTR06] and many more. To benefit from the Earth Mover's Distance's high quality retrieval in these settings, efficient storage and query processing is crucial.

Formally, the Earth Mover's Distance is defined as a Linear Programming problem which can be solved using the simplex method as in [Dan51, HL90]. It is far too complex, however, for application in large multimedia databases.

To cope with this problem, we propose a database indexing and multistep approach similar to the GEMINI (generic multimedia indexing) or KNOP (k nearest neighbor optimal processing) frameworks [FRM94, SK98]. Related works following this paradigm have been successfully applied for multimedia data such as time series [CN04, FRM94] or ellipsoid queries [ABKS98, SK97]. For efficient multistep query processing, which uses an approximative filter

function to reduce the computation times of the original distance function, a tight and computationally simple approximation of the Earth Mover's Distance is essential. Our filters are motivated by geometric intuition of the Earth Mover's Distance and possible lower bounding distances. This leads to valuable insights on the nature of the Earth Mover's Distance computation, which is used to develop a more complex, but also more selective filter suitable for higher dimensionalities. The analysis of our filter properties such as index applicability and efficiency are used to conceptualise a multistep algorithm which combines the advantages of the respective filters. We proof the completeness of our approach to guarantee no false dismissals in image retrieval. Our experiments on high-dimensional color histograms in large image databases validate our findings.

The Earth Mover's Distance is flexible in adapting to various applications by employing different ground distances encoded in the cost matrix. It can also be used in a number of extensions for additional scenarios such as partial matching (losing its metric property) or generalized histograms, i.e. so-called signatures encoding image pixel clusters. In this chapter we concentrate on histograms and metric ground distances as they are a widely used representation in color image retrieval.

2.1 Distance functions

The Earth Mover's Distance was introduced in computer vision in 1997 by Rubner, Guibas and Tomasi [RGT97, RT01] to overcome inconsistencies with perceptual similarity observed in (weighted) L_p-norms and quadratic forms. To understand these difficulties, let us recall how these distance measures are defined.

Definition 2.1 L_p ***norms***

L_p norms are defined for two histograms $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$, where x_i and y_i denote the bin entries, as follows:

$$L_p(x, y) = \sqrt[p]{\sum_{i=1}^{n} (x_i - y_i)^p}$$

From this formula we immediately see that only corresponding 'bins' of the histograms are compared, ignoring relationships between 'neighboring' bins. To see what this means, consider three images and their corresponding color histograms as in Figure 2.1 (based on [RT01]). The top left one and the lower left one are different only by a slight shift in color tone. The lower right one, however, exhibits a completely different color distribution. A comparison using (weighted) L_p norms does not accurately reflect this. As difference is calculated bin-by-bin, i.e. only entries corresponding to the exact same color tones are compared, the distance between the left two images is larger than between the two ones on the right.

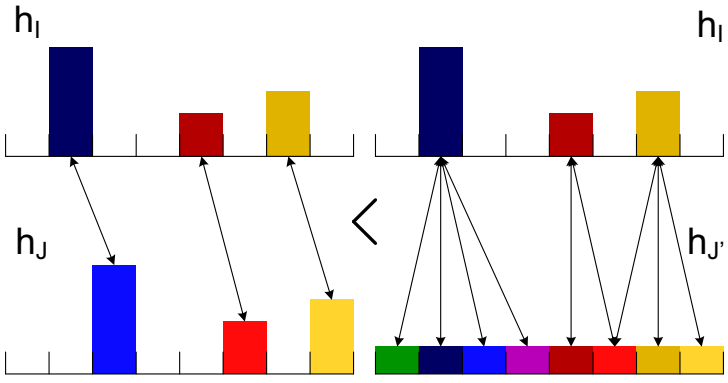


Figure 2.1: Motivation of the Earth Mover's Distance

Quadratic forms were introduced to overcome these difficulties [HSE+95, SH94]. A similarity matrix $A = [a_{ij}]$ is used to reflect the perceived color distances between bin i and bin j. Formally,

Definition 2.2 ***Quadratic forms***

Quadratic forms are defined for two histograms $x = (x_1, \ldots, x_n)$ *and* $y = (y_1, \ldots, y_n)$*, where* x_i *and* y_i *denote the bin entries, and a similarity matrix* $A = [a_{ij}]$ *where entry* a_{ij} *denotes the similarity between bin* i *and bin* j *as follows:*

$$QF_A(x, y) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i - y_i)a_{ij}(x_j - y_j)}$$

This means that all differences between arbitrary bins i and j influence the resulting distance measure weighted by a_{ij}. Reconsidering our example in Figure 2.1, depending on the similarity matrix A, the differences are merely 'smoothed' over all bins, meaning that still structural differences in images cannot be distinguished from color shifts.

2.2 The Earth Mover's Distance

The Earth Mover's Distance also uses a cost matrix $C = [c_{ij}]$ to describe distances between bins. These matrix entries c_{ij} reflect the perceived dissimilarity between the bins i and j which is called ground distance. A best match between the two images represented through their histograms x and y is then calculated by transferring all histogram entries in x to those in y at minimal cost.

Technically, this means that matrix entries do not merely weigh the differences of bin entries as in quadratic forms, but instead guide the distance

calculation. An assignment of one histogram entry x_i to another y_j is determined based on the cost of their corresponding matrix element c_{ij}.

Formally, the Earth Mover's Distance can be formalized as the search for a minimum in a linear programming scheme.

Definition 2.3 ***The Earth Mover's Distance***

The Earth Mover's Distance between histograms x and y with respect to a cost matrix $C = [c_{ij}]$ is defined as the minimum over all possible flows $F = [f_{ij}]$ between x and y as follows:

$$EMD_C(x,y) = \min_F \left\{ \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{c_{ij}}{m} f_{ij}, f_{ij} \geq 0, \sum_{j=1}^{n} f_{ij} = x_i, \sum_{i=1}^{n} f_{ij} = y_j \right\}$$

The denominator $m := \sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij}$ normalizes the Earth Mover's Distance by the total flow, i.e. the mass of the histograms.

For notationally convenience, we alternatively denote the individual constraints for the Earth Mover's Distance as positivity, source and target constraints, respectively:

$$\begin{aligned}
&\text{positivity constraint CPos:} && \forall\ 1 \leq i,j \leq d: && f_{ij} \geq 0 \\
&\text{source constraint CSource:} && \forall\ 1 \leq i\ \leq d: && \textstyle\sum_{j=1}^{d} f_{ij} = x_i \\
&\text{target constraint CTarget:} && \forall\ 1 \leq j\ \leq d: && \textstyle\sum_{i=1}^{d} f_{ij} = y_j
\end{aligned}$$

When defined this way, i.e. for histograms of equal total mass, the Earth Mover's Distance is metric as long as the ground distance is metric, i.e. it fulfills definiteness, symmetry and the triangle inequality. Formally,

Definition 2.4 ***Metric.***

A distance function dist between histograms is a metric if
$\forall$ histograms x, y, z:

- $dist(x, y) = 0 \Leftrightarrow x = y$ *(definite)*
- $dist(x, y) = dist(y, x)$ *(symmetric)*
- $dist(x, z) \leq dist(x, y) + dist(y, z)$ *(triangle inequality)*

Thus, the Earth Mover's Distance measures the effort to match one histogram against the other. All entries in one histogram need to be transferred to entries in the other at minimal cost. Similar images will only cause minor effort as there will be little differences in bin entries. Reconsidering Figure 2.1, we see that using an adequate cost matrix will indeed lead to a larger difference for the rightmost images as shifting all bin entries by one is less effort than moving some of the entries to entirely different bins.

Figure 2.2 gives an example of the Earth Mover's Distance shape in a two-dimensional feature space. An exemplary iso-contour, i.e. points which have the same Earth Mover's Distance value from the center point, is highlighted. Additionally, all image pixels are colored according to their distance, where smaller distances are visualized using darker gray tones. We can see that the Earth Mover's Distance is bounded by hyperplanes, thus a good filter

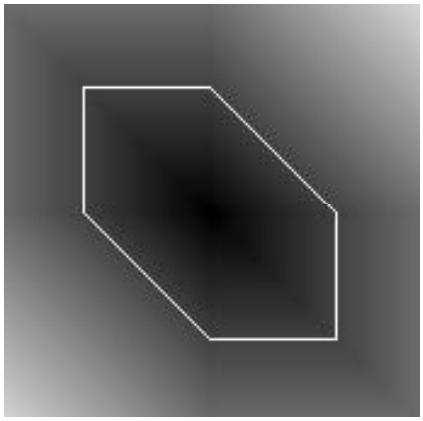


Figure 2.2: Iso-lines of the Earth Mover's Distance

approximation should describe a closely surrounding geometry. As we will see in Section 4.2, this is a good starting point for lower bounds of the Earth Mover's Distance.

The Earth Mover's Distance can be calculated using a streamlined linear programming scheme, the simplex algorithm as found in numerical mathematics literature (see e.g. [Dan51, HL90]). While this provides a solution for pairwise comparison of two histograms, it is much too expensive for large databases where numerous objects have to be compared. We therefore use a multistep algorithm to reduce expensive Earth Mover's Distance calculations while guaranteeing correctness and completeness of the result.

Chapter 3

Multistep retrieval

The need for speeding up retrieval algorithms has brought forth several approaches for avoiding unnecessary distance calculations on images which are just 'too far away' by filtering them out in advance. This is crucial for interactive similarity retrieval applications.

3.1 Multistep filter-and-refine architectures

Index structures are used to organize the data with the objective that only a small percentage needs to be accessed during query processing. They can be used in any setting where only distances between objects are given (e.g. M-tree [CPZ97]). Performance, however, is much better in a (multi-)dimensional space, where index structures such as R-trees or X-trees [BKK96, Gut84, MNPT06] can be used, since they do not only rely on distances to index the data. Additionally, information on individual dimensions in feature space is exploited.

In multidimensional index structures, the main idea is to use the information contained in the individual dimensions. Hierarchical index structures

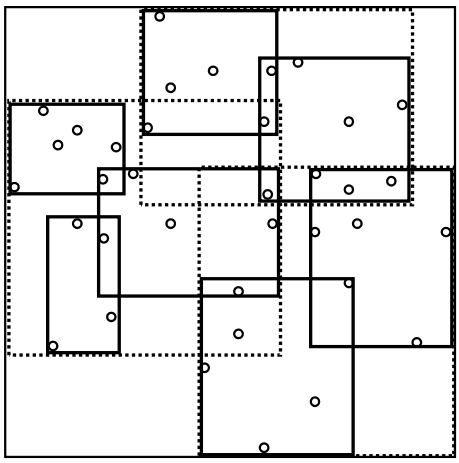


Figure 3.1: Minimum bounding rectangles geometrically

create a small-size directory of the entire database based on those dimensions. This directory is used to direct the similarity search process to determine the result set. To do so, multidimensional features are grouped together according to their respective values. They are summarized by bounding geometries kept at the next higher levels in the directory. During retrieval, the query is compared to these bounding geometries to direct the search towards the answer. In Figure 3.1, two-dimensional points are grouped in a hierarchy of minimum bounding rectangles. These rectangles are stored in a tree (Figure 3.2). Starting at the root, the query is compared to the rectangles, thus choosing only paths down to the data that are relevant for the query.

Index usage can be even more accelerated by calculating faster approximate distances on the index; thus generating candidate results which are then evaluated with the original distance to obtain the true results. Such a multistep approach has been successfully employed in several application scenarios: time series [CN04, FRM94], tumor shapes [KSF+96], GIS (geographic information systems) [BKSS94], ellipsoid queries [ABKS98, SK97],

and dimensionality reduction in general [Fal96].

The efficiency of multistep retrieval depends on the chosen filter distance which has to be tailored for the original distance function. For tumor shapes, the max morphological distance was approximated by a more efficient max granulometric filter distance [KSF+96]; for time series a dimensionality-reduced Chebyshev coefficient metric is shown to lower bound the original high-dimensional Euclidean distance [CN04].

The general idea of multistep retrieval algorithms is summarized in Figure 3.3. A query is first executed as an approximative query using a suitable index structure. This filter step, based on an approximate filter distance function, generates a set of candidates which is then evaluated using the real distance function to retrieve the desired result from the database.

3.2 Query processing

Range queries are a common query type in multimedia databases to access objects similar to an example object. A range query is defined as the set of all objects from the database DB who have a distance to a query object q

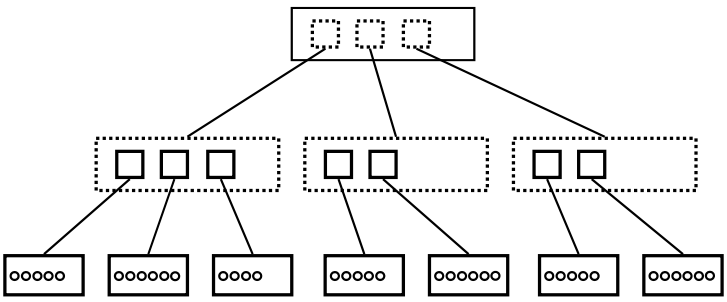


Figure 3.2: Hierarchy of minimum bounding rectangles

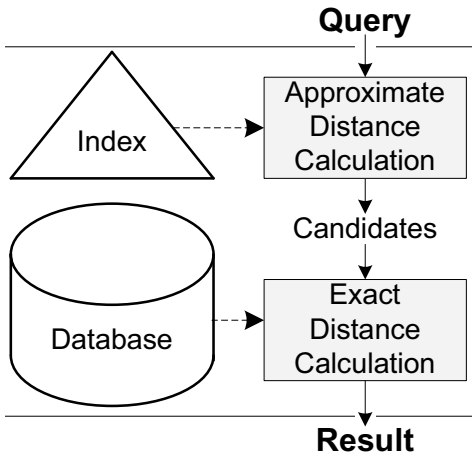


Figure 3.3: Multistep retrieval

of less than or equal to ε. Formally:

$$S_\varepsilon(q) = \{i \in DB \mid dist_{exact}(i, q) \leq \varepsilon\}$$

Range queries can be accelerated using the GEMINI multistep retrieval techniques [Fal96, FRM94]. An index is queried using the same ε threshold, but with an appropriate filter distance:

$$Cand_\varepsilon(q) = \{i \in DB \mid dist_{filter}(i, q) \leq \varepsilon\}$$

This candidate set is then evaluated to see if the actual object distance is not larger than ε. In real world applications, it is often not easy to specify a suitable ε threshold. If ε is chosen too small, the result set may be empty; if it is chosen too large, there may be too many results - in a worst case scenario the whole database may be output.

This motivates k-Nearest Neighbor (k-NN) queries, which do not require the user to specify such an ε. To a query object q the database DB is queried for those k objects which have the smallest distance from q. That is,

$S_k(q)$ is the smallest subset of the database DB with

$$\forall i \in S_k(q)\ \forall j \in DB - S_k(q) : dist_{exact}(i,q) \leq dist_{exact}(j,q)$$

In the GEMINI framework, the index is first queried using a filter distance to retrieve k candidates. The distance ε' of the k-th farthest candidate object is then used to determine the actual k-NN result by a range query. The authors show that, provided the lower bounding property holds for the distance function and the respective filter distance, the resulting output is indeed correct and complete.

An optimal competing multistep algorithm for k-NN queries which makes use of the distance information from those candidates already filtered out was devised in previous work [SK98]. First, candidates are generated using a ranking procedure on the index based on the filter distance. Here, however, as they are generated, for each candidate object the exact distance to the query q is computed and the ε' range is lowered accordingly. This guarantees optimality as to the number of candidates generated while still being correct and complete.

3.3 Filter properties

Crucial for the efficiency of multistep algorithms is the quality of the filter distance measure. This measure should meet a number of criteria (see Figure 3.4):

- **Completeness** means that the multistep algorithm does not produce false dismissals, i.e. in the filter step no actual result object should be discarded. For all of the algorithms revised in Section 3.2, completeness can be assured by a corresponding lower bounding lemma which adheres the following observation:

Lemma 3.1 ***Completeness of lower bound filters***

If a filter distance $dist_{filter}$ *lower bounds the actual object distance* $dist_{exact}$*, i.e.*

$$dist_{filter}(x, y) \leq dist_{exact}(x, y) \forall\ x, y \in DB$$

then the multistep algorithms discussed in Section 3.2 guarantee completeness.

Proofs: see [FRM94, KSF+96, SK98].

- **Selectivity** means that the filter distance should produce a candidate set as small as possible. In a worst case scenario, i.e. the filter distance returns zero for all object distances, the filter simply returns the entire database. Good selectivity is achieved by a filter which approximates the true distance as closely as possible, i.e. returns preferably large values without violating the lower bounding property.

- **Efficiency, i.e. fast single pair computation** is important for acceleration of the total response time. In larger databases, numerous

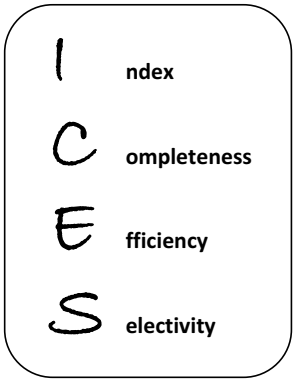


Figure 3.4: ICES filter criteria for multistep retrieval

filter distance computations are necessary, so efficiency is important for performance of the first search step.

- **Index enabled**: multistep algorithms are generally capable of working on an indexing structure which allows the search process to be guided according to the structure of the index storage. A filter distance should thus be capable of being applied in such a scenario in order to benefit from index support.

Chapter 4

Lower bounds of the Earth Mover's Distance

We present several new lower bounds for the Earth Mover's Distance in this chapter. Before doing so, we review a lower bound given in the original Earth Mover's Distance paper.

4.1 3D-averaging lower bound

The basic idea of a lower bound introduced by Rubner et al. [RTG98] is to average the features in the underlying color space. Let r_i denote the bin representatives (centroids), then the 3D-color averages, a weighted sum of the bin representatives, form a simple lower bound for the Earth Mover's Distance:

$$EMD(x, y) \geq \|\sum_{i=0}^{n} \frac{x_i r_i}{m} - \sum_{i=0}^{n} \frac{y_i r_i}{m}\|$$

As shown in [RTG98], this lower bound does help improve the computa-

tion time in smaller settings (i.e. in their case 8-dimensional features in a 20,000 color image collection). Since this lower bound (which we denote by LB_{Avg}) is fixed to three dimensions by the color space, there is no flexibility in adapting it to other dimensionalities. As shown in the experiments, the efficiency gains are thus not sufficient for high-dimensional settings. (Section 4.5).

4.2 L_p-norm-based lower bounds

Our first approach to develop lower bounding filter distances follows the geometric intuition which has been successfully applied to other complex distance functions including quadratic forms [ABKS98, BKSS94]. Common conservative approximations such as minimum bounding rectangles or minimum bounding spheres are well suited for spatial objects and, analogously, for similarity range queries that have a rigid geometry bounded by the range parameter ε.

For k-nearest neighbor queries, however, the boundary of the query region is not known in advance when query processing starts. What has to be provided as approximations are (unbounded) distance functions instead. The corresponding distance functions for rectangles and spheres are represented by weighted maximum norms (L_∞) and weighted Euclidean norms (L_2), respectively. More precisely, by our geometric approach we aim at providing the lower bounding weighted L_p norms ($p = 1, 2, \infty$) as filter distances.

The advantages of using (weighted) L_p norms are two-fold. First, a single comparison is computed in linear time $O(n)$ in an n dimensional histogram space. Secondly, these distance functions are immediately index enabled and supported by any available multi-dimensional indexing structure.

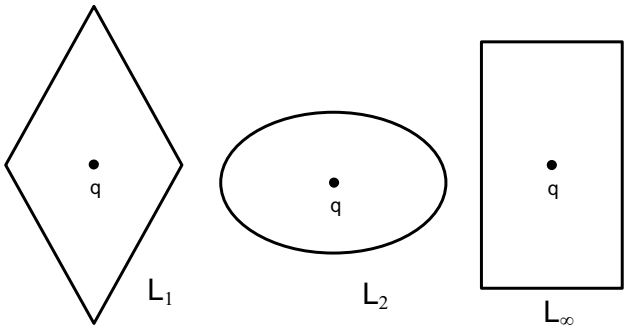


Figure 4.1: Iso-contours of weighted L_p norms

Geometrically, the iso-surfaces of the weighted L_p norms correspond to rectangles, spheres, and diamonds (Figure 4.1). Since the Earth Mover's Distance's iso-surface is bound by hyperplanes as can be observed from Figure 2.2, a hyperdiamond (L_1) or hyperrectangle (L_∞) is expected to result in a better approximation than a hypersphere (L_2).

4.2.1 Weighted L_1 lower bound

As discussed above, (weighted) L_p norms base their distance calculation on the difference between individual bin entries. The weighted L_1 (Manhattan) distance is defined by

$$WL_1(x, y) = \sum_{i=1}^{n} w_i |x_i - y_i|$$

where the w_i represent weights for the individual dimensions. These weights have to be determined as parameters of the filter. Geometrically, they stretch or compress the corresponding hyperdiamond to optimally enclose the Earth Mover's Distance iso-surface.

To determine weights which best lower bound the Earth Mover's Distance, let us take a closer look at how the Earth Mover's Distance is computed. Since we only consider metric ground distances, in our costmatrix C, the

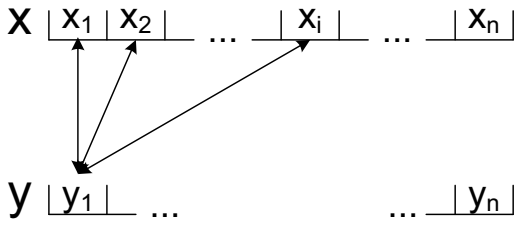


Figure 4.2: EMD flows between histograms

diagonal entries c_{ii} are zero (definiteness) and $c_{ij} = c_{ji}$ (symmetry). When searching for the minimal flow as described in Section 2.2, as much mass as possible will be moved at zero cost, i.e. between corresponding bin entries. The amount of mass which can be moved is limited by the corresponding entries in each histogram as defined in Section 2.2:

$$m = \sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij} = \sum_{i=1}^{n} x_i = \sum_{j=1}^{n} y_j$$

Figure 4.2 illustrates the computation procedure between two histograms $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$. Let us assume that $x_i \geq y_i$. As mentioned above, zero cost c_{ii} is the minimum, i.e. first, earth of x_i will be moved to y_i until y_i is filled. If there is more earth available in x_i, it cannot be absorbed by y_i. The remaining mass ($|x_i - y_i|$) will thus have to be moved at higher cost. If, to the contrary, y_i is larger than x_i all of x_i will be moved to y_i and the remaining capacity of y_i ($|x_i - y_i|$) will be filled at higher cost by other entries of x. This leads to the following theorem:

Theorem *Weighted L_1 lower bound LB_{Man}*

For any metric ground distance encoded in a costmatrix $C = [c_{ij}]$, the Earth Mover's Distance between two histograms $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ with the same total mass, i.e. $m = \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$, is always greater than or equal to the weighted Manhattan distance with weights

$w_i = min_{j,i\neq j}\{\frac{c_{ij}}{2\cdot m}\}$. Formally:

$$EMD(x,y) \geq \sum_{i=1}^{n} \min_{\substack{j \\ i\neq j}}\{\frac{c_{ij}}{2\cdot m}\}|x_i - y_i|$$

Proof:

$$EMD(x,y) = \sum_{i=1}^{n}\sum_{\substack{j=1 \\ i\neq j}}^{n} \frac{c_{ij}}{m} f_{ij} \tag{4.1}$$

$$\geq \sum_{i=1}^{n} \min_{\substack{j \\ i\neq j}}\{\frac{c_{ij}}{m}\}\sum_{\substack{j=1 \\ i\neq j}}^{n} f_{ij} \tag{4.2}$$

$$\geq \sum_{i=1}^{n} \min_{\substack{j \\ i\neq j}}\{\frac{c_{ij}}{2\cdot m}\}|x_i - y_i| \tag{4.3}$$

By definition, the Earth Mover's Distance between histograms x and y is the minimal transfer of entries in x to those in y (or vice versa). As described above, entries between corresponding bins are moved at zero cost ($c_{ii} = 0$) which does not contribute to the Earth Mover's Distance value. We therefore consider only the remainder $|x_i - y_i|$ which has to be moved to different bins $j \neq i$ (Equation 4.1).

Approximating each of the cost entries c_{ij} in one row i by their minimum allows us to combine the individual coefficients into a single one for the entire sum (Equation 4.2).

As seen before, the sum of flows from any bin x_i equals its mass: $\sum_{j=1}^{n} f_{ij} = x_i$. Since $c_{ii} = 0$ is always minimal, f_{ii} is as large as possible, i.e. $f_{ii} = y_i$ if $x_i > y_i$ (y_i cannot absorb more). Thus $\sum_{j=1,i\neq j}^{n} f_{ij} = x_i - y_i$ if $x_i > y_i$. Analogously, $\sum_{j=1,i\neq j}^{n} f_{ij} = y_i - x_i$ if $x_i < y_i$. In sum, $2\sum_{j=1,i\neq j}^{n} f_{ij} = |x_i - y_i|$.

Thus, we replace the sum of flows by half the absolute bin differences (Equation 4.3). ◊

Thus, our weighted Manhattan distance LB_{Man} lower bounds the Earth Mover's Distance which guarantees completeness in a multistep retrieval scheme as outlined in Section 3.

4.2.2 Weighted L_∞ lower bound

For the minimum bounding rectangle approximation, the corresponding filter distance function is represented by a weighted maximum norm (L_∞):

$$WL_\infty(x,y) = \max_{i=1}^{n}\{w_i|x_i - y_i|\}$$

Geometrically, this norm corresponds to a rectilinear hyperrectangle. By similar arguments as in the L_1 case, we obtain:

Theorem *Weighted L_∞ lower bound LB_{Max}*
For any metric ground distance encoded in a costmatrix $C = [c_{ij}]$, histograms $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ with $m = \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$, we have:

$$EMD(x,y) \geq \max_i\{\min_{\substack{j \\ i \neq j}}\{\frac{c_{ij}}{m}\}|x_i - y_i|\}$$

Proof:

$$EMD(x,y) \geq \sum_{\substack{i=1 \\ x_i \leq y_i}}^{n} \min_j\{\frac{c_{ij}}{m}\}|x_i - y_i| \tag{4.4}$$

$$\geq \max_{\substack{i \\ x_i \leq y_i}}\{\min_{\substack{j \\ i \neq j}}\{\frac{c_{ij}}{m}\}|x_i - y_i|\} \tag{4.5}$$

$$EMD(x,y) \geq \max_i\{\min_{\substack{j \\ i \neq j}}\{\frac{c_{ij}}{m}\}|x_i - y_i|\} \tag{4.6}$$

The proof of our weighted L_1 norm implies that considering only those cases where $x_i \leq y_i$, we obtain Equation 4.4. We approximate the sum by its maximum summand, thus yielding Equation 4.5.

Instead of approximating the Earth Mover's Distance by those cases where $x_i \leq y_i$, we could also do so by those cases where $x_i \geq y_i$. As both results lower bound the Earth Mover's Distance, we can choose the larger one of them by simply taking the maximum (Equation 4.6). $\Diamond$

4.2.3 Weighted L_2 lower bound

Another frequently used L_p norm is the L_2 norm, the Euclidean Distance:

$$WL_2(x,y) = \sqrt{\sum_{i=1}^{n} w_i(x_i - y_i)^2}$$

Geometrically, the iso-surface of the Euclidean Distance is a (hyper-)sphere, if weighted, we obtain iso-oriented ellipsoids.

Theorem *Weighted L_2 lower bound LB_{Eucl}*
For any metric ground distance encoded in a costmatrix $C = [c_{ij}]$, histograms $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ with $m = \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$, we have:

$$EMD(x,y) \geq \sqrt{\sum_{i=1}^{n} (\min_{\substack{j \\ i \neq j}} \{\frac{c_{ij}}{2 \cdot m}\})^2 (x_i - y_i)^2}$$

Proof:

$$EMD(x,y) \geq \sum_{i=1}^{n} \min_{\substack{j \\ i \neq j}} \{\frac{c_{ij}}{2 \cdot m}\} |x_i - y_i| \tag{4.7}$$

$$= \sum_{i=1}^{n} \sqrt{(\min_{\substack{j \\ i \neq j}} \{\frac{c_{ij}}{2 \cdot m}\})^2 (x_i - y_i)^2} \tag{4.8}$$

$$\geq \sqrt{\sum_{i=1}^{n} (\min_{\substack{j \\ i \neq j}} \{\frac{c_{ij}}{2 \cdot m}\})^2 (x_i - y_i)^2} \tag{4.9}$$

Equation 4.7 was proven for the weighted Manhattan lower bound. We replace the absolute value by the root of the squared value (Equation 4.8). Then the sum can be pulled inside the root, which is done in Equation 4.9.$\Diamond$

We already see from this proof that the Euclidean lower bound encloses the Manhattan lower bound and is thus of worse selectivity if used in a filter step.

While lower bounds based on L_p norms are well suited for index support and can be efficiently computed, their selectivity decreases for higher dimensions due to the so-called "curse of dimensionality". Whenever the number of bins, and therefore, the number of cost matrix entries, increases, the minimum of these is bound to decrease. Therefore, in high-dimensional settings, the selectivity of bounds based on these minima drops. This is also validated in the experiments in Section 4.5.

Moreover, index structures fall prey to the curse of dimensionality as well. In high-dimensional settings, sequential scan eventually performs better than any index structure [BBK01, WSB98]. In these high-dimensional settings, additional techniques are needed for efficient query processing. In the next section, we refine the idea of the geometrically inspired weighted L_p lower bounds. Computing for each dimension the amount of mass which has to be moved by at least the minimum cost for that dimension can be extended to a tighter, i.e. more selective, lower bound. After that, we show how these lower bounds can be combined to benefit most from their respective efficiency improvements.

4.3 Independent minimization lower bound

As stated above, we extend the concept of independently minimizing the cost for moving the mass in individual histogram bins. In contrast to the weighted L_p norms however, we do not stop at picking one minimum for each histogram dimensions. Instead, we re-approach the essential Earth Mover's

Distance method by taking into consideration that histogram bins can only absorb the mass corresponding to their respective entries ($\sum_{i=1}^{n} f_{ij} = y_j$). That is, for each dimension, we reinstall the constraint that each histogram bin should not receive more mass than specified by its value ($f_{ij} \leq y_j$). The difference to the Earth Mover's Distance remaining is then, that this constraint is weakened to the effect that the sum of flows for different dimensions might well be above this limit. To understand this, recall the Earth Mover's Distance definition:

$$EMD(x,y) = \min_{F} \left\{ \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{c_{ij}}{m} f_{ij}, f_{ij} \geq 0, \sum_{j=1}^{n} f_{ij} = x_i, \sum_{i=1}^{n} f_{ij} = y_j \right\}$$

The Independent Minimization lower bound (LB_{IM}) is then defined as follows:

$$LB_{IM}(x,y) = \min_{F} \left\{ \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{c_{ij}}{m} f_{ij}, f_{ij} \geq 0, \sum_{j=1}^{n} f_{ij} = x_i, \forall\ j\ f_{ij} \leq y_j \right\}$$

This formalizes that the difference between the new LB_{IM} and the Earth Mover's Distance is the constraint on how much mass one histogram bin may receive, the target constraint CTarget in the original Earth Mover's Distance versus the new, weaker constraint CTarget* for LB_{IM}:

$$CTarget: \ \forall\ 1 \leq j\ \leq d: \ \sum_{i=1}^{d} f_{ij} = y_j$$

$$CTarget^*: \ \forall 1 \leq j \leq d: \ f_{ij} \leq y_j$$

While the Earth Mover's Distance requires that the sum of flows to a certain bin equals its mass over all dimensions, the LB_{IM} only ensures that

for any single dimension the incoming flows do not exceed its mass. We now show that LB_{IM} indeed lower bounds the Earth Mover's Distance.

Theorem *LB_{IM} lower bound*

For any metric ground distance encoded in a costmatrix $C = [c_{ij}]$, histograms $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ with $m = \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$, we have:

$$EMD(x, y) \geq LB_{IM}(x, y).$$

Proof:

We start by proving that the constraints in LB_{IM} are indeed a weaker version of those for the Earth Mover's Distance, i.e. that for any i:

$$\sum_{i=1}^{n} f_{ij} = y_j \wedge f_{ij} \geq 0 \Rightarrow \; f_{ij} \leq y_j$$

Assume to the contrary that there exists a j such that $f_{ij} > y_j$. Then we have for the sum $\sum_{i=1}^{n} f_{ij} > y_j$ as well, which is a contradiction.

From this we infer for the two sets to be minimized:

$$\begin{aligned} S_{EMD} \;:=\; & \left\{ \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{c_{ij}}{m} f_{ij}, f_{ij} \geq 0, \sum_{j=1}^{n} f_{ij} = x_i, f_{ij} \leq y_j \right\} \subset \\ & \left\{ \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{c_{ij}}{m} f_{ij}, f_{ij} \geq 0, \sum_{j=1}^{n} f_{ij} = x_i, \sum_{i=1}^{n} f_{ij} = y_j \right\} := S_{LB_{IM}} \end{aligned}$$

Next, note that for any two sets, where one is a subset of the other, the minimum of one set is always larger than or equal to that of its superset:

$$s \subset S \;\Rightarrow\; min(S) = min(min(s), min(S \setminus s)) \leq min(s)$$

Consequently, the minimum determined as the Earth Mover's Distance is larger than or equal to that determined as the Independent Minimization lower bound:

$$LB_{IM}(x, y) = min(S_{LB_{IM}}) \leq min(S_{EMD}) = EMD(x, y) \qquad \Diamond$$

Intuitively speaking, the proof uses the fact that the Earth Mover's Distance is minimized in a more restricted space than the LB_{IM}. The minimum as specified by the Earth Mover's Distance constraints is thus always included in the search space of the LB_{IM} which can therefore only result in the same or a smaller minimum.

We have shown that the Independent Minimization indeed lower bounds the Earth Mover's Distance and is thus a complete filter. We next analyze how it can be efficiently computed. By restricting the search for a minimum by constraints over the outgoing dimensions i only, we can decompose the LB_{IM} computation into a series of smaller problems for each i. In other words, each of the three constraints, i.e.

- positivity of the flows ($f_{ij} \geq 0$),
- equality of outgoing flows to the mass of each bin ($\sum_j f_{ij} = x_i$) and
- limitation of flows to the target bins mass ($f_{ij} \leq y_j$)

can be checked for individual dimensions. The positivity of all flows f_{ij} is a constraint which does not require the consideration of more than one dimension i at a time. The same is true for the sum over all outgoing flows $\sum_j f_{ij}$: since the constraint needs to be validated for individual x_i, a separated computation yields the same result. And the last constraint on flows f_{ij} to the target bins y_j has been "individualized": the sum of flows to a target bin does no longer have to equal its mass as in the Earth Mover's Distance case ($\sum_i f_{ij} = y_j$). Instead, it merely limits the amount of mass for each i ($f_{ij} \leq y_j$). Note that this last constraint is violated by the L_p-norm-based lower bounds, where no check on the receiving bins except the corresponding one ($f_{jj} \leq y_j$) is required; the remainder ($|x_i - y_i|$) is moved at minimal cost $c_{ij}, i \neq j$ independent of the mass in $y_j, i \neq j$. Instead of minimizing

flows for all bins simultaneously via the simplex method, we are now able to minimize the cost for each histogram bin separately. This substantially decreases computation times as demonstrated in Section 4.5.

We can further improve the Independent Minimization lower bounds by adapting two properties seen for L_p-norm-based lower bounds:

(i) Restricting the flows to $f_{jj} + f_{ij} \leq y_j, i \neq j$, instead of simply to $f_{ij} \leq y_j$, can be easily fulfilled before minimization starts: we know that the flow between identical histogram bins is the mass of the smaller entry: $f_{jj} = min(x_j, y_j)$. Since $c_{jj} = 0$, we can subtract these flows f_{jj} from y_j as they do not contribute to the Earth Mover's Distance value to improve the selectivity of LB_{IM}. (Cf. the proof of LB_{Man}, Section 4.2.1).

Example.

Let $x = [4, 3, 5, 4, 5]$ and $y = [1, 2, 3, 8, 8]$.
Compute $x^1 = [x_i - min(x_i, y_i)] = [3, 2, 2, 0, 0]$
and $y^1 = [y_i - min(x_i, y_i)] = [0, 0, 0, 4, 3]$.
These histograms have the same Earth Mover's Distance value as the original ones, but further restrict for each bin the possibility of mass movement (i.e. the limit y_j is lowered), which results in a better selectivity of our LB_{IM} filter.

(ii) As for the LB_{Max} lower bound (Section 4.2.2), we can pick the maximum of $LB_{IM}(x, y)$ and $LB_{IM}(y, x)$. Switching the roles of x and y means relaxing the constraint $\sum_{j=1}^{n} f_{ij} = x_i$ to $f_{ij} \leq x_i$ instead of relaxing $\sum_{i=1}^{n} f_{ij} = y_j$ to $f_{ij} \leq y_j$. The lower bounding property is fulfilled in either case, thus we choose the larger, more selective, lower bound, i.e. the maximum of $LB_{IM}(x, y)$ and $LB_{IM}(x, y)$.

Example.

Let $C = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 1 & 0 & 1 & 2 & 3 \\ 2 & 1 & 0 & 1 & 2 \\ 3 & 2 & 1 & 0 & 1 \\ 4 & 3 & 2 & 1 & 0 \end{pmatrix}$.

Independent Minimization yields for the above x^1 to y^1:

$LB_{IM}(x, y) = x_1 * c_{14} + x_2 * c_{24} + x_3 * c_{34} = 3 * 3 + 2 * 2 + 2 * 1 = 15$,

whereas the computation from y^1 to x^1 yields:

$LB_{IM}(y, x) = 1/2 * y_4 * c_{43} * 1/2 * y_4 + c_{42} + 2/3 * y_5 * c_{53} + 1/3 * y_5 * c_{52} =$

$2 * 1 + 2 * 2 + 2 * 2 + 1 * 3 = 13$.

By taking the maximum over these two values, we are sure to obtain the LB_{IM} value closest to the Earth Mover's Distance.

4.4 Multistep filter concept

We mentioned that L_p-norm-based filters are well suited for index support and have low one-to-one computation times. As they do not provide optimal selectivity for increasing dimensionality, they are best used for low-dimensional problems. As dimensionality increases, our LB_{IM} lower bound is favorable as it yields better selectivity which eventually means that overall computation times are lower in high dimensions where many more candidates might be generated otherwise.

The combination of filters further reduces the computation time as they dismiss different sets of objects as non-candidates. Therefore, a sequential scan using a good-selectivity-filter in high dimensions on the output of an efficient low-dimensional, index-supported filter makes best use of database

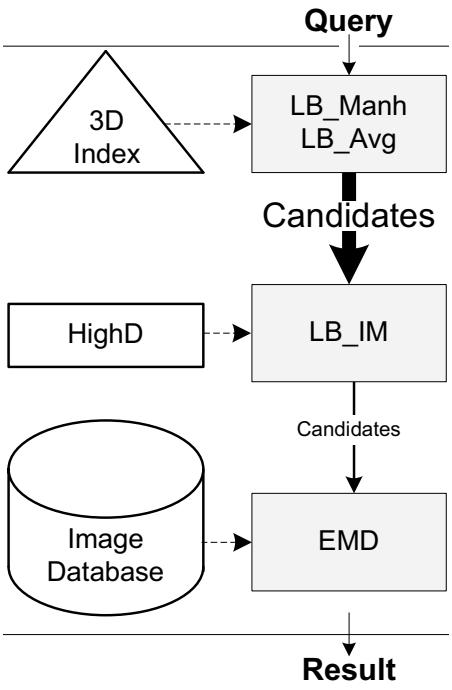


Figure 4.3: Multistep concept for lower bounds of EMD

technology available.

While index support can substantially improve response times, it has been shown many times, that with increasing dimensionality, starting as low as dimensions six or eight, indexing structures fall prey to the so-called "curse of dimensionality" [BBK01, WSB98]. This means that with higher dimensionality, large portions of the index have to be accessed, which in turn means that many more I/O activities have to be performed.

We therefore propose to integrate a reduction of dimensionality into a two-phase multistep algorithm which combines the advantages of low-dimensional index-support with the good selectivity of our multi-dimensional LB_{IM} filter (Figure 4.3). We therefore construct three-dimensional indexes based on the averaging lower bound (which can only handle three dimensions in a three-dimensional color space) or three-dimensional indexes based on the

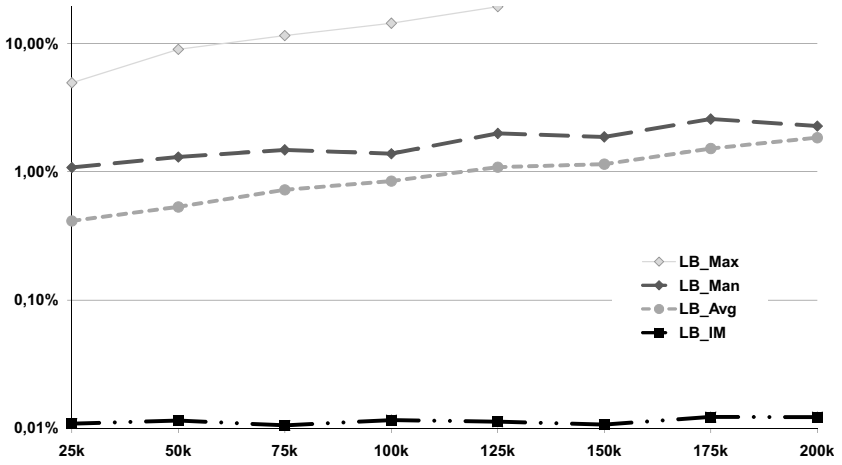


Figure 4.4: Scalability, selectivity percentage

weighted Manhattan lower bound, which, as experiments confirm, is best among our L_p-based filters. Dimensionality reduction for the weighted Manhattan lower bound is performed by simply determining those three dimensions with highest variability and discarding all other dimensions. The resulting distances are necessarily smaller, thus, lower bounding is not jeopardized. Recall the definition of the weighted Manhattan distance from Section 4.2.1: $WL_1(x, y) = \sum_{i=1}^{n} w_i|x_i - y_i|$. By reducing this to three dimensions (e.g. the first ones) we have $WL_1(x^3, y^3) = \sum_{i=1}^{3} w_i|x_i - y_i|$. Since we only drop positive summands, we are sure to lower the resulting distance value. The three-dimensional index is then built on weighted reduced histograms $w^3 \cdot x^3$.

Similarly, for the $LB_{Avg} = \| \sum_{i=1}^{n} \frac{x_i r_i}{m} - \sum_{i=1}^{n} \frac{y_i r_i}{m} \|$ lower bound, the color averages $a_i = \sum_{i=1}^{n} \frac{x_i r_i}{m}$ have to be precalculated to preserve lower bounding. The corresponding 3d-index is then built on these color averages. The resulting candidate set of the first 3d-filter may be large, but its generation involves relatively few index operations. The candidate set is then further reduced by using our LB_{IM} lower bound which has better selectivity. The final refinement computations using the Earth Mover's Distance are thus re-

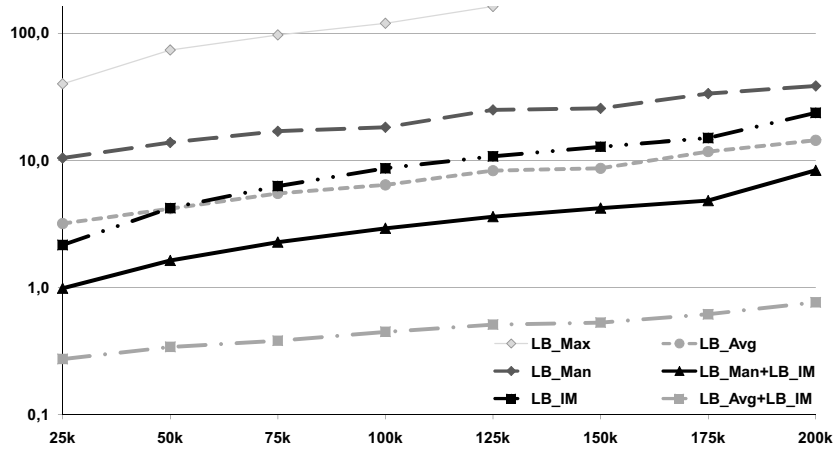


Figure 4.5: Scalability, response time in seconds

duced as much as possible. As we will see in the experimental evaluation in Section 4.5 this two-phase multistep algorithm indeed outperforms all other filter techniques.

4.5 Experiments

We ran extensive experiments on an extended version (200,000 images) of the color image database used in [ABKS98]. We implemented all lower bounds and the Earth Mover's Distance in Java (the latter is based on the original C code by Rubner [Rub98]). We used the R-tree index by Hadjieleftheriou [Had02] from the R-tree Portal. Experiments were run on Pentium 4 2.4 GHz workstations.

Using optimal query processing, we varied the number k of nearest neighbors from 1 to 20, histogram sizes from 16 to 64 dimensions, and tested the scalability of our approach by varying the database size. Using the average of 200 randomly selected query images, we recorded selectivity ratios as well as total response times for query processing.

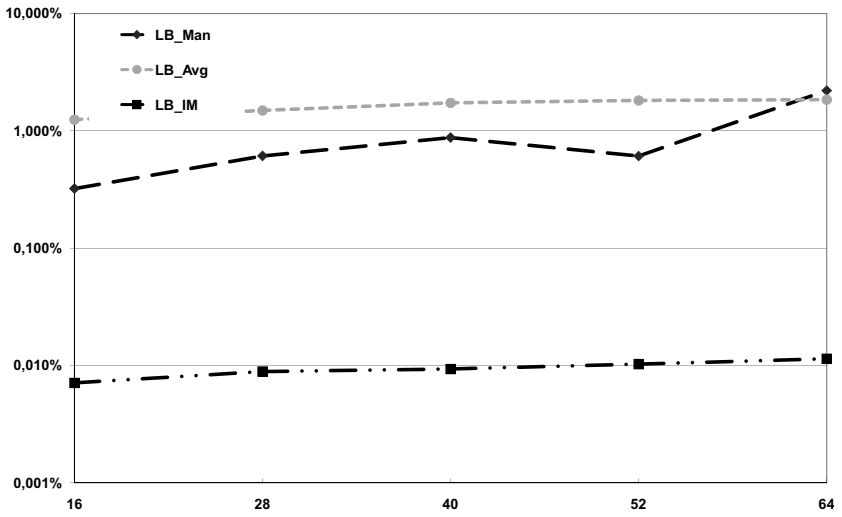


Figure 4.6: Dimensionality, selectivity percentage

Scalability. In our first experiment, we evaluated the performance of our lower bounding filters by measuring the selectivity for various database sizes. The smallest database contains 25.000 images, the largest all 200,000 images. We compared our most promising filter techniques for k nearest neighbor queries, where k was set to 10 and histograms were of highest resolution 64. The selectivity is then given as the average percentage of database images for which Earth Mover's Distance computation is necessary.

The LB_{Eucl} weighted L_2 lower bound's performance is, as was already analyzed in Section 4.2.3, far inferior to LB_{Max}, i.e. weighted L_∞ lower bound and to LB_{Man}, i.e. weighted L_1, lower bound. We therefore did not include it in our diagrams.

In the diagram in Figure 4.4 we can see that the weighted L_∞ lower bound LB_{Max} produces many more candidates than the weighted L_1 lower bound LB_{Man} or the averaging lower bound LB_{Avg}; its selectivity is inferior by an order of magnitude. This means that the Earth Mover's Distance is more closely surrounded by the hyperdiamond which corresponds to our weighted

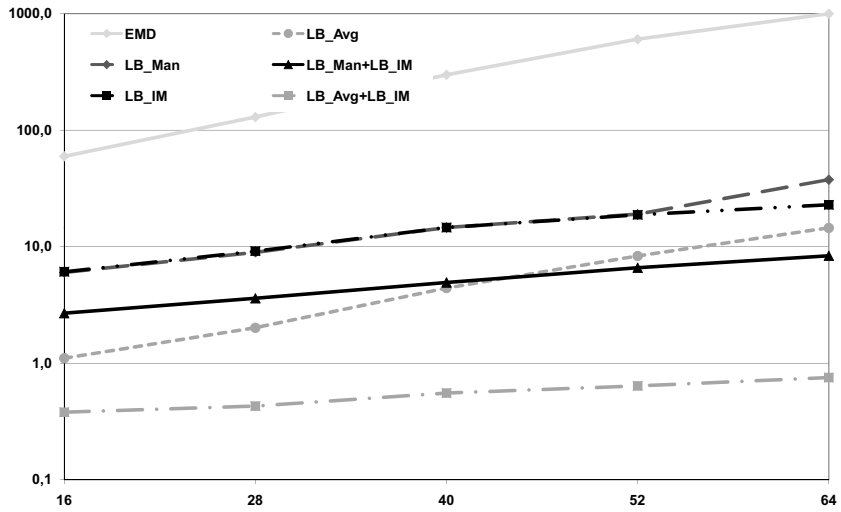


Figure 4.7: Dimensionality, response time in seconds

L_1 lower bound than by the hyperrectangle belonging to our weighted L_∞ lower bound. As LB_{Max} has far worse performance, we no longer include it in our figures.

Our new filter, the LB_{IM} lower bound is of noticeably higher selectivity. It outputs far less than 0.1% of the data as candidates for all database sizes. This is an improvement by more than two orders of magnitude to the second-best lower bound LB_{Avg}. In Figure 4.5 we see that for the same setup, the response times of LB_{Man} and LB_{Avg} are closely related to their selectivity as their computational overhead is low. LB_{IM} requires more computational effort, thus despite its far superior selectivity, it shows only similar response times. However, the combination with the former two lower bounds as presented in Section 4.4, yields the lowest response times.

Dimensionality. Another important parameter for the performance of lower bounding approximations is the size of the histograms involved. We varied histograms from dimension 16 to 64. We can see in the diagram in Figure 4.6, where the largest database with 200,000 was queried for $k = 10$ nearest

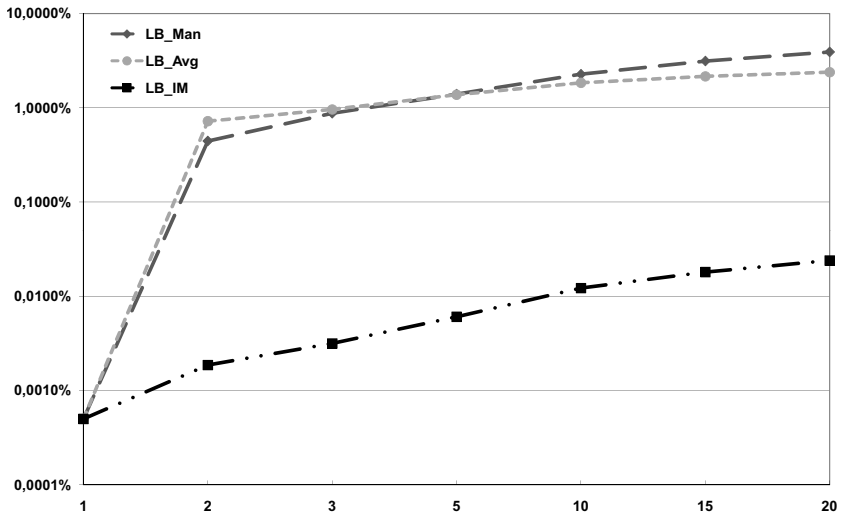


Figure 4.8: Result size, selectivity percentage

neighbors, that our LB_{IM} lower bound yields clearly best selectivity results in all cases. For finer histogram resolutions, the LB_{IM} lower bound has selectivity gains of more than two orders of magnitude. The second best lower bound is the weighted L_1 lower bound, followed by the averaging lower bound.

Figure 4.7 shows for response times that with increasing dimensionality, the computation of LB_{Avg} increases in complexity. The response times of LB_{Man} are more closely related to its selectivity ratios. As in the previous experiment, the overhead of LB_{IM} is greater than that of the other two lower bounds. Once again, its combination with a low-dimensional LB_{Avg}-index yields the best performance improvements. We include the sequential scan Earth Mover's Distance computation times as a baseline comparison. Note that the improvement for 64 dimensions comparing Earth Mover's Distance and the best multistep concept is from 1000 seconds to less than one second, i.e. more than three orders of magnitude.

Result size. To determine the influence of the number of nearest neighbors

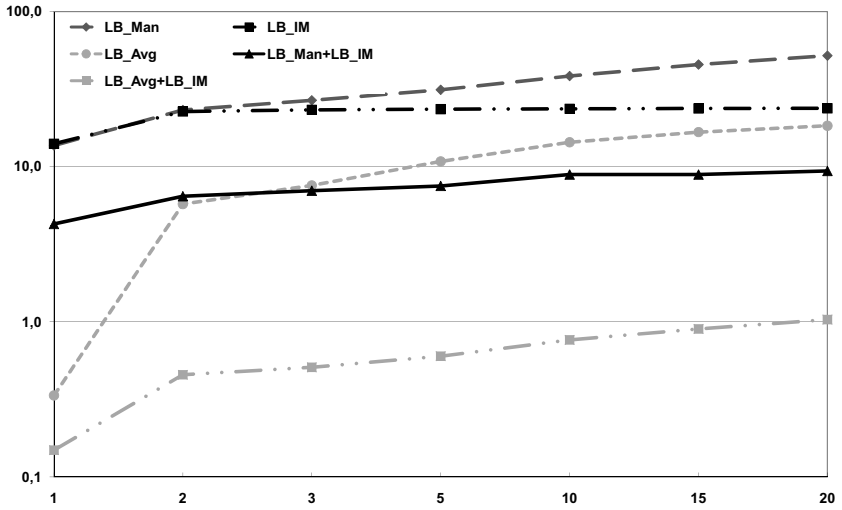


Figure 4.9: Result size, response time in seconds

queried for, we ran an experiment on the largest database of 200,000 images using 64-dimensional-histograms and searched for 1 to 20 similar images. We see in Figure 4.8 that the weighted Manhattan lower bound LB_{Man} and the color averaging approach LB_{Avg} show similar selectivity ratios, which are far inferior to those of LB_{IM} lower bound by far. In Figure 4.9 we see that the response times show corresponding behavior, where the multistep concept of Section 4.4 is once again clearly the fastest.

Query processing. Query processing performance, as discussed in Section 3.2, is illustrated for two exemplary lower bounds in Figure 4.10 and Figure 4.11. As we can see, optimal multistep query processing is noticeably more efficient than the original GEMINI approach, especially for the lower bound LB_{Man} where selectivity is improved by one order of magnitude. As the LB_{IM} lower bound already retrieves substantially fewer candidates, its selectivity gains are smaller, but still noticeable. The response times reflect these selectivity ratios. This means that choosing the optimal algorithm results in more efficient query processing.

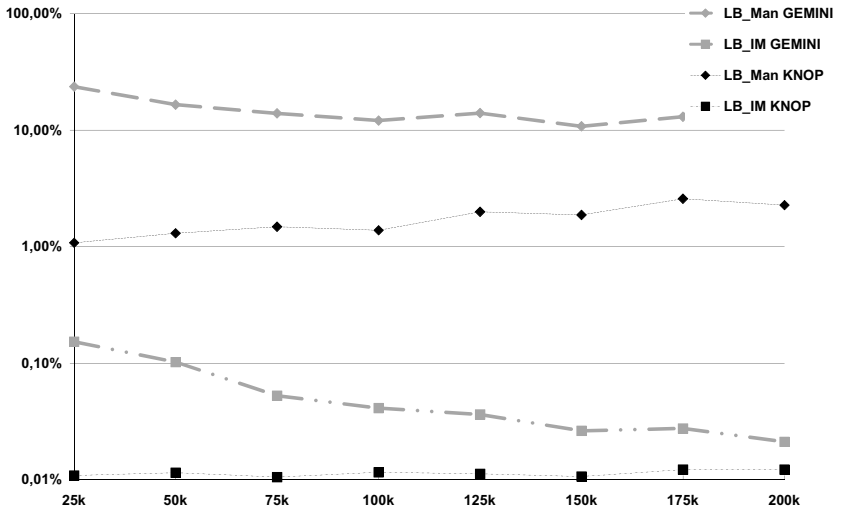


Figure 4.10: Multistep algorithm, selectivity percentage

In summary, our experiments demonstrate that the best strategy for query processing is a combination of assets. We compared computation times of two different approaches: LB_{Man}, LB_{Avg}, and LB_{IM} were tested in "simple" multistep query processing as presented in Section 3, which means that each of these filter functions is immediately followed by the exact Earth Mover's Distance calculation. Their response times were only acceptable for low dimensions and non-interactive scenarios. We compared these three with the two proposed combinations of Section 4.4, where three-dimensional LB_{Man} and LB_{Avg} were evaluated in a first step. Their candidate set is then substantially reduced by the best-selectivity-filter, the LB_{IM}. As we can see, this concept indeed results in noticeable speed-up ratios. The high selectivity of LB_{IM} thus results in the expected efficiency improvements. By building a small three-dimensional index based on simpler filter functions and combining it with a highly selective second LB_{IM} filter, index support can be profited from while expensive Earth Mover's Distance computations are minimized.

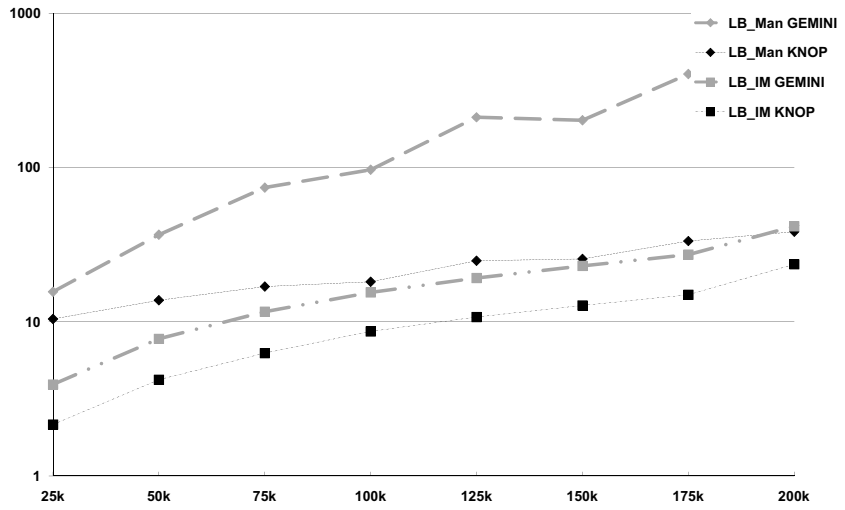


Figure 4.11: Multistep algorithm, response time in seconds

4.6 Conclusion

Search in large image or other multimedia databases highly dependends on the underlying similarity model. The Earth Mover's Distance, proposed in computer vision literature, is an interesting new approach towards achieving high-quality content-based retrieval. Despite its advantages, this distance measure is computed via a linear programming algorithm which is too slow for today's huge and interactive multimedia applications.

We therefore developed several new filters designed for usage in a multistep retrieval concept. We analyzed and demonstrated in our experiments that L_p-norm-based filters are easy to compute, they can be used with indexing structures, and achieve satisfactory selectivity for smaller dimensions and non-interactive settings. As dimensionality increases, more complex filters with higher selectivity ratios are needed. We present such an approach which is indeed closer to the Earth Mover's Distance, the LB_{IM} lower bound. We combined our filters in a multistep algorithm making use of the advan-

tages of the filters involved. High dimensions are problematic for most index structures; we therefore use a dimensionality reduction implicit in the averaging lower bound or explicit in our Manhattan reduction. As this still leaves us with far too many candidates, two-phase filtering that combines the power of the individual filters, ensures an efficient retrieval procedure.

Chapter 5

Case study: video stream retrieval

In this chapter, we study an application of the multistep concept for the Earth Mover's Distance as discussed in the previous chapters. Using this concept, we are able to efficiently retrieve similar image tiles in real time applications. This case study thus demonstrates both efficiency and effectivity in a practical context.

The idea of the prototype system is to attract customers by a video streaming application. A video camera records people passing by, and a monitor shows an alienated version of the setting accordingly. The idea is to replace the image on the video screen by a mosaic of similar images to draw peoples' attention to the location. Our aim is to alienate the video stream of a camera and project the result onto a screen for passers-by.

This "mosaic" of video stream data is a challenge for state of the art technology. For successful implementation, several aspects are of key importance: real-time computation of the alienated images, and, at the same time, a similarity computation similar to human perception for resulting recognizable

mosaic tiles.

Brief, our system has to meet the following requirements:

- good correspondence with human perception of similarity ⇒ Earth Mover's Distance
- real time responses ⇒ our multistep concept
- scalability ⇒ our multistep concept
- robustness to small changes in incoming video frames ⇒ priority ordering according to the degree of change

Our prototype, called AttentionAttractor, takes frames coming in from the video camera, breaks them down into tiles for the alienation module. This module sorts incoming tiles by a priority score depending on the changes perceived in the video image and holds them in a priority queue. Similarity search is subsequently performed on an image database making use of indexing and multistep query processing, thus ensuring real time response times. The resulting alienated tiles are handed over to the graphical user interface (GUI) for display on the screen.

This project makes use of three important building blocks: on the one hand, we adopt a similarity model capable of capturing important aspects of

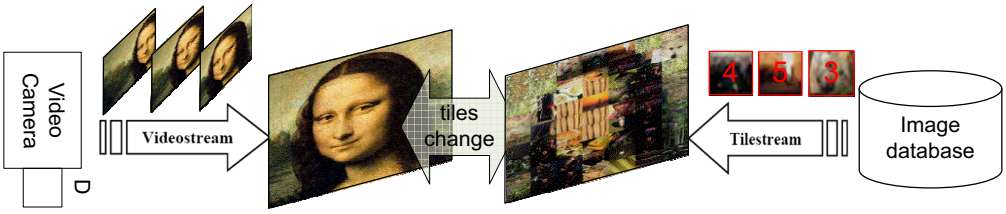


Figure 5.1: Concept of the AttentionAttractor

human perception, and, on the other hand, we use multistep query processing to speed up computation times. Moreover, we explore inherent characteristics of video stream data, such as repetition or slow change to deal with the challenges the demanding application poses. These aspects are discussed with more detail in the following.

5.1 Similarity search and prioritizing

In real-time video stream processing, it is important to ensure that the computational power is well spent. As video images change, their corresponding tiles should be updated accordingly. It is important to prioritize tile similarity search according to the following aspects as lengthy computations may be outdated by the time they are completed.

One aspect to consider is the similarity model. It should be of high quality, as noted before, to ensure convincing results for human eyes. However, the degree of similarity refinement should be adjustable according to the degree of change encountered in the video stream. Whenever changes occur only locally or only small changes are detected, more time can be spent on computing similarity for other tiles.

Another aspect is the immediate reflection of change in the video. If all tiles are queried for most similar alienations before the entire image is updated, the delay in response times is infeasible. Therefore, progressive computation of results, displaying processed tiles immediately, enhances the overall impression of the quality of the mosaic video.

Finally, we have the aspect of ordering of tiles: if tiles are simply processed in the order arriving from left to right, top-down, the result is less likely to immediately reflect the areas of greatest change. These areas, however, are

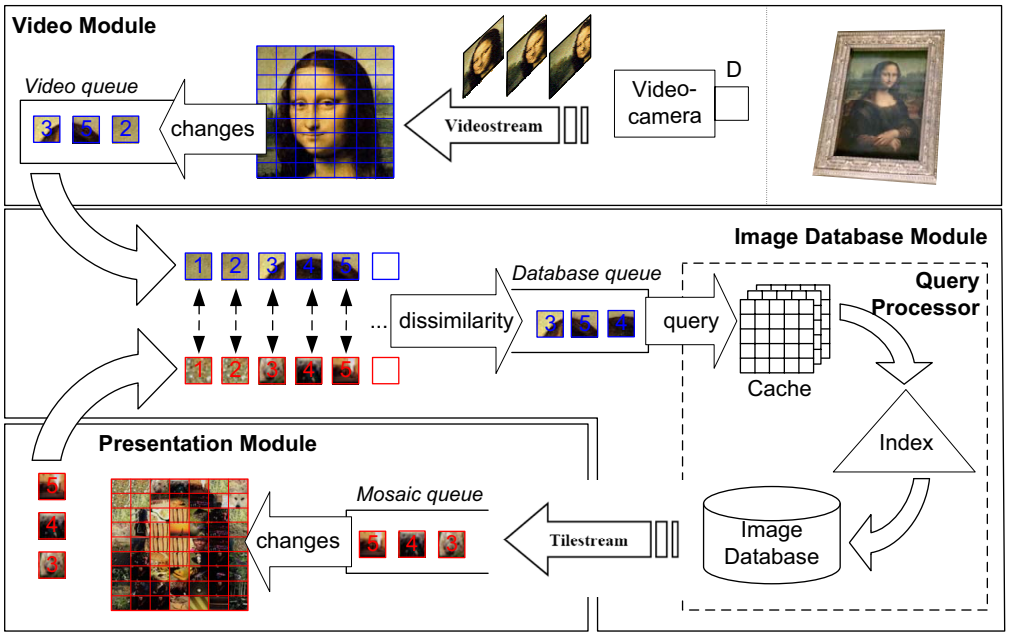


Figure 5.2: Architecture of the AttentionAttractor

noticed first by human spectators.

We use the previously discussed lower bounding multistep query processing of the Earth Mover's Distance for efficiency. The fact that several lower bounds can be used in sequence allows using different degrees of refinement during similarity search. That is, tiles can be processed only up to a certain level of filter function if tiles of higher priority arrive in the meantime.

5.2 System architecture

In order to attract attention it is important to process the video stream in real-time. To guarantee performance for interactive mosaic video the AttentionAttractor uses three priority queues to order the tiles according to their importance (see Figure 5.2). For human perception a mosaic tile should be

updated if the tile currently shown on screen is significantly different from the current actual tile. Thus the dissimilarity between the current tile and the new tile is used as the job priority.

Since the tiles in a video stream change constantly it is important to efficiently measure the dissimilarity between tiles. Thus, instead of only using the time consuming Earth Mover's Distance function the AttentionAttractor makes use of less time consuming lower bound distance functions.

Each priority queue is processed by the corresponding module:

1. The "Video Module" extracts the tiles from the live video stream. The extracted tiles are inserted into the video priority queue. Recall that the tile with the greatest change has the highest priority. Next the tiles added to the video priority queue are passed down to the image database module.

2. The "Image Database Module" consecutively receives tiles from the "Video Module". For each tile the dissimilarity between the tile and its corresponding counterpart received from the "Mosaic Module" is measured. Afterwards the tile is added to the database priority queue. The database queue is used as input for the query processor which seeks the most similar image from the image database. The query processor is described in detail in Section 5.3.

3. The job of the "Presentation Module" is to update the mosaic tiles. Thus it receives new tiles from the "Image Database Module", measures the dissimilarity between the new tile and the displayed counterpart and updates the tile priority queue. The tile with the highest priority is updated first. To illustrate updates of the mosaic image the modified tile can additionally be highlighted.

Each module of the AttentionAttractor can use a different distance function for determining the priority of tiles. Which distance function is best depends on the image database size and the performance of the computer. Figure 5.3 presents the setup dialog of the AttentionAttractor for choosing individual distance functions for each module.

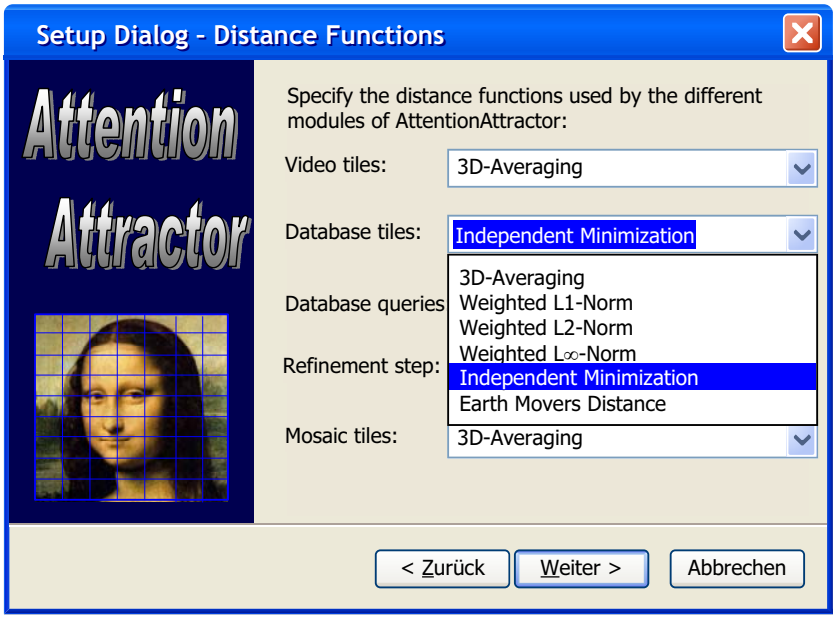


Figure 5.3: Setup Dialog of the AttentionAttractor

5.3 The Query Processor

The Query Processor consists of three components: the query cache, an index structure and the image database. The cache is important for storing tiles queried often by the image database module. Particularly tiles belonging to the background are queried repeatedly. The main memory cache significantly improves the response times for these tiles which is very important for the

interactive visualization of the mosaic video.

If the cache does not contain the tile queried an R*-Tree [BKSS90] indexing structure is used to search for the nearest neighbor. We use the multistep approach from the previous chapters. As it uses a ranking query to retrieve candidates from the index structure [SK98], we exploit this for priority based interruption. After a candidate is retrieved the query processor updates the priority distance for the tile queried. Before the query processor continues with the refinement step the queue is checked if another tile now has a higher priority. If so the query is suspended and stored with the current tile while the query processor continues with the tile now having the highest priority. Thus, our approach allows for interrupting the similarity search for one tile in favor of one with higher priority.

Even though a suspended candidate is not the correct nearest neighbor it is nevertheless a good approximation and passed down to the presentation module. It may be later refined further if no tiles of higher priority are queued. If a tile has been refined to the highest possible degree, it is removed from the database queue.

We implemented a prototype system that shows the aspects of video stream similarity search as described above. The video camera image is displayed along with its alienated version.

The AttentionAttractor prototype system has been successfully used on several occasions to demonstrate the efficiency and effectivity of our multistep Earth Mover's Distance algorithm in a practical application setting (see Figure 5.4).

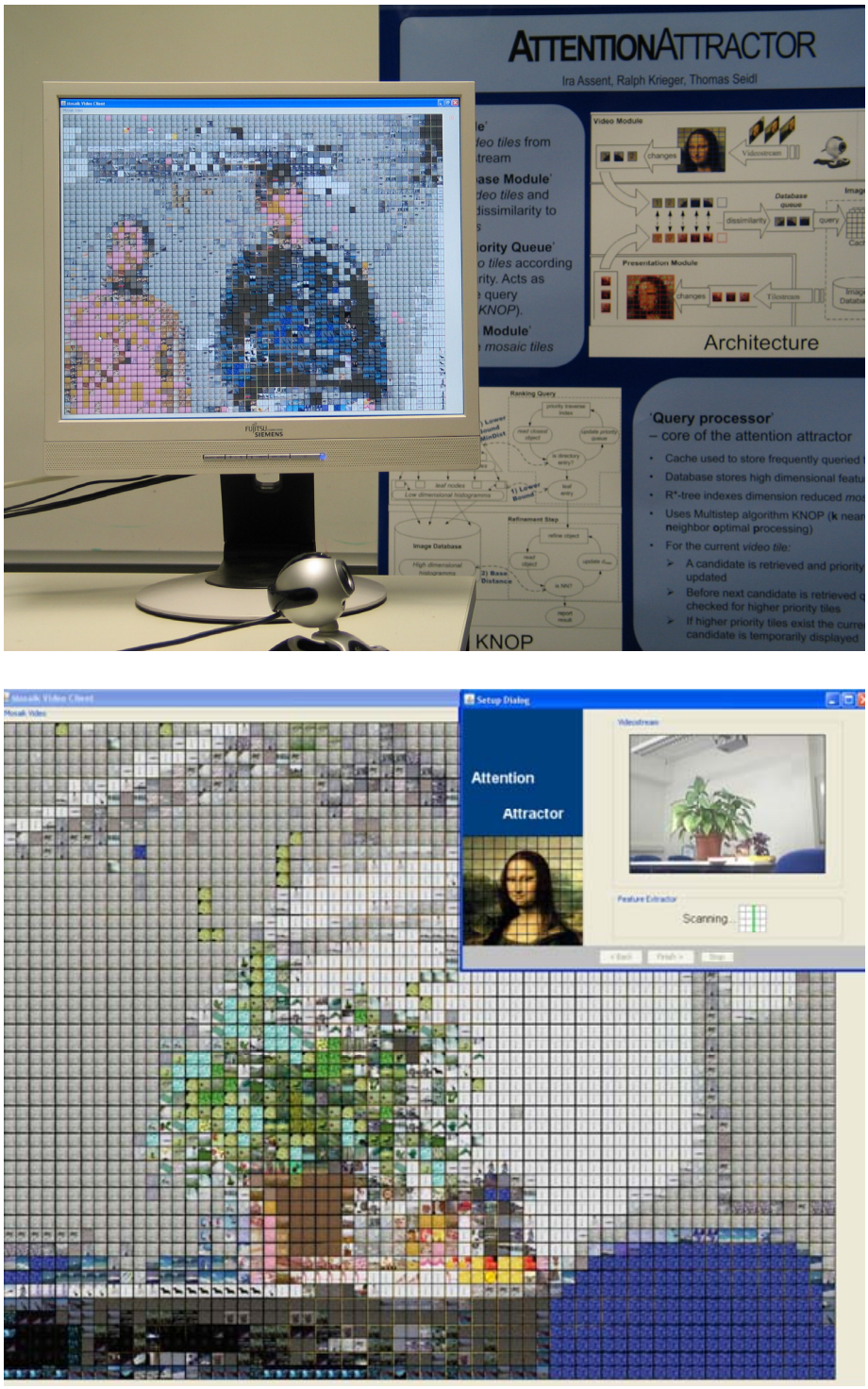


Figure 5.4: The demo system live (top) and screenshot of the demo (bottom)

Chapter 6

Exact indexing

Direct indexing of the Earth Mover's Distance without additional lower bounds requires computation of the so-called "*mindist*" function. This *mindist* is the minimum distance between query and index region (cf. Figure 6.1), and is used to decide which subtrees have to be accessed for similarity search.

We enable directly indexing the Earth Mover's Distance in structures such as the R-tree and the vector approximation file (VA-file) by providing the accurate *mindist* function to any bounding rectangle in the index. We exploit the computational structure of the new *mindist* to derive a new lower bound for the Earth Mover's Distance *mindist* for the vector approximation file which is assembled from quantized partial solutions yielding very fast query processing times. We prove completeness of our approach in a multistep scheme.

6.1 Introduction

We formally introduce the *mindist* function allowing for direct indexing of the metric Earth Mover's Distance on histogram data. While the results are

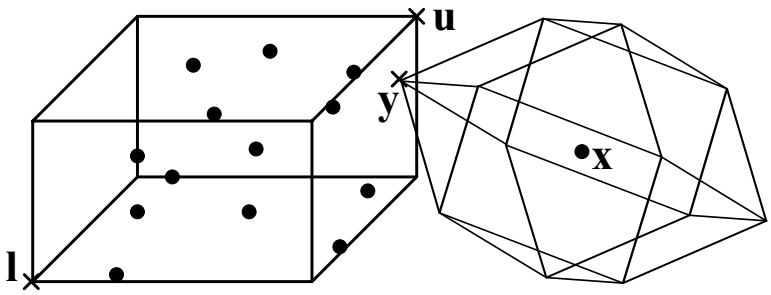


Figure 6.1: *mindist* for the Earth Mover's Distance

good for dimensionalities up to around ten, drastically faster query processing times for higher dimensional data are achieved through the analysis and decomposition of newly introduced *mindist* lower bounds. While the Earth Mover's Distance is intrinsically dimension-interdependent, we reduce our point-to-rectangle Earth Mover's Distance lower bound to dimension-wise distance components. Thus, we are able to transfer our results to the vector approximation file (VA-file) [WSB98], where the apriori knowledge of all occurring rectangle boundaries stemming from the quantized view on the data is exploited.

In summary, our contributions include:

- the exact indexing of the Earth Mover's Distance by introducing an exact point-to-rectangle *mindist* function and showing how this function can be computed algorithmically (Section 6.3)

- quantization-based optimization of Earth Mover's distance based retrieval by introducing lower bound of the *mindist* for histogram-based Earth Mover's Distance that is decomposed into dimension-wise components (Sections 7.1 and 7.2)

- a thorough empirical evaluation using real world data sets (Section 7.3)

6.2 Related work

As mentioned before, the Earth Mover's Distance is defined in terms of a linear optimization which can be solved using an adopted version of the simplex method [Dan51, HL90]. Even though the theoretically exponential complexity of the simplex method is not observed in practice, this method has at least quadratic complexity. For larger databases with higher dimensional histograms, our previous approach proposes indexing of filter approximations. In [LBS06], the authors compute lower bounds for homogeneous color images where differences in images are typically small [CPZ97]. However, exact indexing using R-trees or related indexing structures is not feasible without computation of the *mindist* function. Multidimensional indexes like the R-tree require computation of the distance between query histogram and MBRs (minimum bounding rectangles) which summarize the database histograms. In this chapter, we introduce this *mindist* to provide the means for direct indexing.

Indexing structures organize the data such that similarity search has to access only relevant parts of the database. Multi-dimensional indexing structures such as the R-tree or X-tree summarize histograms in minimum bounding rectangles [Gut84, BKK96]. During query processing, the query point is compared to the minimum bounding rectangles in a top-down manner to decide whether the subtrees summarized by the minimum bounding rectangle are relevant to the query. For high dimensionality, the minimum bounding rectangles tend to overlap due to the "curse of dimensionality" [BGRS99] which forces most minimum bounding rectangles to be examined, requiring random accesses to almost the entire database. Consequently, the sequential scan is eventually faster [WSB98].

This has spurred the search for alternative means of indexing. The vector approximation file quantizes the data space to provide compact representation of features for optimized sequential scans [WSB98]. In the next chapter, we show how vector approximation file based indexing can also be employed using our *mindist* definition. Moreover, we enhance efficiency of query processing by deriving *mindist* components which can be precomputed and stored in the vector approximation file lookup table to reduce online computation time.

6.3 Indexing of the Earth Mover's Distance

Exact indexing which relies on bounding regions to describe similar histograms on an abstract level requires computation of the *mindist* between a query and the bounding region stored in the index. In this chapter, we develop the *mindist* for direct indexing of the Earth Mover's Distance using any index based on rectangular bounding regions such as the R-tree family [Gut84, BKSS90].

The Earth Mover's Distance *mindist* for a query and an index bounding rectangle is the smallest Earth Mover's Distance value for any histogram in the bounding rectangle. This is formalized in the following minimization:

Definition 6.1 *mindist*
For a cost matrix C, a non-negative d-dimensional histogram x of mass m_x and a d-dimensional rectangle with lower and upper boundaries l and u with $m_l \leq m_x \leq m_u$, the EMD mindist is defined as

$$mindist_C(x, l, u) = \min_y \{EMD_C(x, y) | Cons_{mindist}\}$$

where $Cons_{mindist} = CMass \wedge CL \wedge CU$*:*

$$\begin{aligned} CMass: &\quad \textstyle\sum_{j=1}^{d} y_j = \quad \sum_{i=1}^{d} x_i \\ CL: &\quad \forall 1 \leq j \leq d: \quad y_j \geq l_j \\ CU: &\quad \forall 1 \leq j \leq d: \quad y_j \leq u_j \end{aligned}$$

$CMass$ ensures that only data points of suitable mass within the rectangle are considered.

Recall the definition of the Earth Mover's Distance as a minimization:

Definition 6.2 ***Earth Mover's Distance***

For two non-negative d-dimensional histograms x *and* y $(\forall 1 \leq i \leq d : x_i \geq 0 \wedge y_i \geq 0)$ *of equal total mass* $m_x := \sum_{i=1}^{d} x_i = m_y := \sum_{i=1}^{d} y_i = 1$ *and a cost matrix* $C = [c_{ij}]$*, the EMD is defined as a minimization over all possible flows* $F = [f_{ij}]$ *under positivity constraints* $CPos$*, source constraints* $CSource$ *and target constraints* $CTarget$*:*

$$EMD_C(x, y) = \min_F \{ \sum_{i=1}^{d} \sum_{j=1}^{d} c_{ij} f_{ij} | Constraints \} \tag{6.1}$$

with $Constraints = CPos \wedge CSource \wedge CTarget$ *:*

$$CPos: \quad \forall 1 \leq i, j \leq d: \quad f_{ij} \geq 0 \tag{6.2}$$

$$CSource: \quad \forall 1 \leq i \leq d: \quad \textstyle\sum_{j=1}^{d} f_{ij} = x_i \tag{6.3}$$

$$CTarget: \quad \forall 1 \leq j \leq d: \quad \textstyle\sum_{i=1}^{d} f_{ij} = y_j \tag{6.4}$$

For solving the Earth Mover's Distance *mindist* minimization in practice, we will restate the *mindist* as a flow minimization and then transform it into a transportation problem.

Theorem 6.1 *mindist* ***as flow minimization***

For a cost matrix C*, a non-negative d-dimensional histogram* x *of mass* m_x

and a d-dimensional rectangle with lower and upper boundaries l and u with $m_l \leq m_x \leq m_u$, *CPos and CSource according to Definition 6.2, the EMD mindist can be stated as*

$$mindist_C(x, l, u) = \min_F \{\sum_{i=1}^{d}\sum_{j=1}^{d} c_{ij} f_{ij} | Constraints'\}$$

with $Constraints' = CPos \wedge CSource \wedge CMin \wedge CMax$*:*

$$CMin: \quad \forall 1 \leq j \leq d: \quad \sum_{i=1}^{d} f_{ij} \geq l_j \tag{6.5}$$

$$CMax: \quad \forall 1 \leq j \leq d: \quad \sum_{i=1}^{d} f_{ij} \leq u_j \tag{6.6}$$

Proof.

We start by combining the *mindist* and EMD definition and merging the two minimizations.

$$\begin{aligned} & mindist_C(x, l, u) \\ &= \min_y\{\min_F\{\sum_{i=1}^{d}\sum_{j=1}^{d} c_{ij} f_{ij} | Constraints\} | Cons_{mindist}\} \\ &= \min_{y,F}\{\sum_{i=1}^{d}\sum_{j=1}^{d} c_{ij} f_{ij} | CPos \wedge CSource \wedge CTarget \wedge\ CMass \wedge CL \wedge CU\} \end{aligned}$$

We substitute y_j in $CMass$, CL and CU with the sum of flows going to y_j according to $CTarget$. Thus, we no longer search for an optimal flow F given some y conforming with $Cons_{mindist}$. Instead, we search for the optimal flow F that results in a target which conforms with $Cons_{mindist}$. The choice of F entirely determines y. Hence, we can eliminate $CTarget$ from the constraints and y from the minimization space.

$$\begin{aligned} = \min_F\{\sum_{i=1}^{d}\sum_{j=1}^{d} c_{ij} f_{ij} \mid CPos \ \wedge\ CSource \wedge\ \sum_{j=1}^{d}\sum_{i=1}^{d} f_{ij} = \sum_{i=1}^{d} x_i \wedge \\ \forall 1 \leq j \leq d: \sum_{i=1}^{d} f_{ij} \geq l_j \wedge\ \forall 1 \leq j \leq d: \sum_{i=1}^{d} f_{ij} \leq u_j\} \end{aligned}$$

The substituted $CMass$ is a relaxation of $CSource$ and thus reduntant while CL and CU become equivalent to $CMin$ and $CMax$ which concludes the proof.

$$= \min_F \{\sum_{i=1}^{d}\sum_{j=1}^{d} c_{ij}f_{ij} | CPos \wedge CSource \wedge CMin \wedge CMax\} \qquad \Diamond$$

While also a linear programming problem, this formulation is not in the form of a transportation problem due to the inequalities (6.5) and (6.6). For a transportation problem, which has a favorable structure that allows for more efficient solving, the total mass of both the source and the target must match exactly and all mass from the source must be moved to the target. This can be remedied according to [HL90] by transforming the problem such that each target dimension j is split into two parts that are each afforded their own target dimension j and $d + j$ in the transformed problem. The first part records how much flow has to at least be sent towards the old target dimension (l_j). The other tracks the amount of flow that additionally could have been allotted to the old target dimension ($u_j - l_j$). The source is extended with the additional mass required to match the mass of the transformed target.

Theorem 6.2 *mindist* ***as Transportation Problem***

For C, x, l and u according to Theorem 6.1, the EMD mindist can be stated as

$$mindist_{C''}(x'', y'') = \min_F \{\sum_{i=1}^{d+1}\sum_{j=1}^{2d} c''_{ij}f_{ij} | Constraints''\}$$

with $Constraints'' = CPos'' \wedge CSource'' \wedge CTarget''$ *and*

$$\begin{aligned} C'' &= \begin{bmatrix} C & C \\ \infty \cdots \infty & 0 \cdots 0 \end{bmatrix} \\ x'' &= (x_1, ..., x_d, \Delta m) \\ y'' &= (l_1, ..., l_d, (u_1 - l_1), ..., (u_d - l_d)) \end{aligned}$$

where $\Delta m := m_u - m_x$. $CPos''$, $CSource''$ *and* $CTarget''$ *as in Equations (6.2), (6.3) and (6.4) but for the adapted indexes* i, j *and the replacement of* x, y *with* x'', y''.

Proof. (Sketch)

To obtain a so-called augmented linear program that fits the transportation problem model, inequality constraints are converted to equivalent equalities using slack variables as discussed in detail in [HL90]. The augmented model represents exactly the same solution space, but provides favorable algebraic properties. The original variables (histogram dimensions of query and lower bound) are preserved. Slack variables are introduced for the variable source or target assignments (range between bounds). Columns of C are doubled in C'' since pairs of original and slack variables in the augmented model are the equivalent of original variables in the initial linear program.

Finally, the problem can be simplified by removing columns from C'' and y'' where either the lower flow limit is zero or the range of additional flow is zero or the mass available in a source dimension exceeds even the upper limit of the respective target dimension. ◊

The *mindist* allows for efficient query processing for two reasons. One, index structures using bounding rectangles can be utilized, and second, since

the *mindist* is formulated as a transportation problem just like the original EMD, existing lower bound filter distances can be used.

In Chapter 4, the independent minimization (LB_{IM}) filter was introduced and proven to be a lower bound to the Earth Mover's Distance. As opposed to the Earth Mover's Distance, LB_{IM} can be computed for each j independently as the flows must not exceed individual target masses. As the remaining constraints are untouched, the extension of this lower bound to the corresponding *mindist* is straightforward.

Chapter 7

Vector approximation

Our multistep approach using the independent minimization lower bound LB_{IM} as a filter in a filter-and-refine algorithm reduces the number of Earth Mover's Distance computations in query processing by reducing the *mindist* into dimension-wise distance components. However, as we will also see in the experiments, the overall runtime is still too high for applicability in very large multimedia databases. This is due to the fact that the Earth Mover's Distance *mindist* or LB_{IM}-*mindist* computations require repeated calculations of the query to the lower and upper bounds of the bounding rectangles in the index structure. This has to be done online while descending the hierarchy of the index.

We propose to reduce the computational overhead during query processing by precomputing distances between query and bounding rectangles offline. This requires three properties: one, the upper and lower bounds are known apriori, two, the *mindist* computation can be reduced to dimension-wise distance components, and three, these distance components can be efficiently recombined into the overall *mindist*. In the next section, we show how quantization provides apriori knowledge on upper and lower bounds.

7.1 Quantization and lookup table

Data-independent indexing refers to techniques which decompose the feature space. Features are indexed according to the intervals they fall into in the different dimensions. By contrast, in data-dependent indexes such as the R-tree, individual bounding rectangles around features describe the data. This hinders precomputation of distance components. In the following, we demonstrate how distance components are precomputed from the apriori knowledge of the boundaries in a quantization approach.

Quantization maps continuous feature values to fixed intervals in each dimension. Instead of the exact features, bit codes of intervals are stored. Given a histogram x with continuous values x_i and a quantization of each dimension into intervals $b_1, \ldots, b_n$ of range $[l_1, u_1), \ldots, [l_n, u_n)$, record b_j for dimension i if $x_i \in b_j$. The interval ranges $[l_1, u_1), \ldots, [l_n, u_n)$ constitute

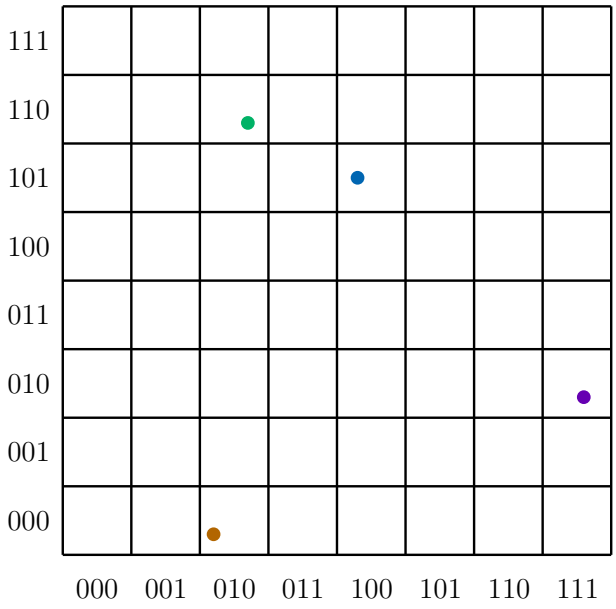


Figure 7.1: Quantization

apriori knowledge on possible upper and lower boundaries of all indexed features.

In Figure 7.1, a two-dimensional feature space of length one in each dimension is quantized using a regular decomposition with three bits, i.e. $2^3 = 8$ intervals. For example, $x = (0.5375, 0.6875)$ in $b_5 = 100$ in the first dimension and $b_6 = 101$ in the second dimension has a quantized representation $b_5 b_6 = 100101$. Its bounding rectangle is given by the corresponding interval ranges $[l_5, u_5), [l_6, u_6)$=$[0.5, 0.625), [0.625, 0.75)$.

Analogously, all possible upper and lower bounds are known apriori, i.e. regardless of the query or the data stored. This allows us to store the *mindist* components which correspond to any possible upper or lower bound per dimension in a lookup table as suggested in the original VA-file (vector approximation file) [WSB98]. These components can be efficiently recombined to generate the *mindist* to the bounding rectangle which consists of these intervals.

For each of the possible boundaries that are known apriori, the lookup table stores the distance component in this dimension. For example, for Euclidean distance, instead of computing the distance between query x and the bounding rectangle b as $\sqrt{\sum_i (x_i - b_i)^2}$, compute $\sqrt{\sum_i lookup_i(b_i)}$. This avoids repeated computation of the distance between the query and the same boundary limits over and over again. Please note that using a lookup table is straightforward for dimension-wise distances. However, for the Earth Mover's Distance which minimizes across rows and columns at the same time, direct usage of a lookup table is not possible. In the next section, we show which distance components may be stored in a lookup table.

7.2 Lookup table lower bound

As direct usage of a lookup-table is not possible for the Earth Mover's Distance *mindist*, we propose a new lower bound which contains exactly the lookup-table compliant components. The idea is to relax the source constraint to obtain a dimension-wise minimization. As the exact mass in each interval range is unknown apriori, we require that at least the mass of the lower bound is moved in those dimensions where the query is below the lower bound. Formally, the *lookup table lower bound* LB_{LT} is defined as follows:

Definition 7.1 *mindist* ***lookup table lower bound*** LB_{LT}**.**

For C, x'' and y'' according to Theorems 6.1 and 6.2, the lookup table lower bound LB_{LT} is defined as a minimization over all possible flows $F = [f_{ij}]$ under positivity constraints $CPos''$, source constraints $CSource_{LT}$ and target constraints $CTarget_{LT}$:

$$LB_{LT_C}(x'', y'') = \min_F \{\sum_{i=1}^{d+1} \sum_{j=1}^{2d} c''_{ij} f_{ij} | Constraints_{LT}\}$$

with $Constraints_{LT} = CPos'' \wedge CSource_{LT} \wedge CTarget_{LT}$:

$$CSource_{LT}: \quad \forall_{1\leq i\leq d} \forall_{1\leq j\leq d, j\neq i} \quad f_{ij} \leq x''_i$$

$$CTarget_{LT}: \quad \forall_{1\leq j\leq d} \quad x''_j < l_j : \sum_{i=1}^{d} f_{ij} = l_j$$

The following theorem states that the above definition indeed constitutes a lower bound of the Earth Mover's Distance *mindist*.

Theorem 7.1 *LB_{LT} lower bounds Earth Mover's Distance mindist.*

For C, x'' and y'' according to theorems 6.1 and 6.2:

$$LB_{LT_C}(x'', y'') \leq EMD_C(x'', y'').$$

Proof.

The difference in the definition of LB_{LT} and the *mindist* of the Earth Mover's Distance in Theorem 6.2 is a weaker set of constraints.

$$\begin{aligned}
& LB_{LT''_C}(x'', y'') \;\leq\; mindist_{C''}(x'', y'') \\
\Leftrightarrow\; & \min_F \{\sum_{i=1}^{d+1}\sum_{j=1}^{2d} c''_{ij} f_{ij} | Constraints_{LT}\} \\
& \leq \min_F \{\sum_{i=1}^{d+1}\sum_{j=1}^{2d} c''_{ij} f_{ij} | Constraints''\} \qquad (7.1) \\
\Leftrightarrow\; & Constraints_{LT} \supseteq Constraints'' \qquad (7.2) \\
\Leftrightarrow\; & CPos'' \wedge CSource_{LT} \wedge CTarget_{LT} \\
& \supseteq CPos'' \wedge CSource'' \wedge CTarget''
\end{aligned}$$

The positivity constraint is identical, and the source constraint relaxes the requirement that the sum of flows for any dimension i is exactly x_i to the weaker one that at most x_i has to be moved:

$$\forall_{1\leq i\leq d}\forall_{1\leq j\leq d, j\neq i}\; f_{ij} \leq x''_i \Leftarrow \forall_{1\leq i\leq d+1}\; \sum_{j=1}^{2d} f_{ij} = x''_i$$

Similarly, the target constraint is restricted to dimensions where $x''_j < l_j$:

$$\forall_{x''_j < l_j} \sum_{i=1}^{d} f_{ij} = l_j \Leftarrow \forall_{1\leq j\leq d}\; \sum_{i=1}^{d} f_{ij} = l_j$$

This means that the solution space for minimization grows, which can only result in a smaller or at most the same minimum (Equations 7.1 to 7.2). Consequently, LB_{LT} is a lower bound. ◇

The computation of an entry `delta[j][k]` in the lookup table is given in Algorithm 1. If x_j is less than the k^{th} lower boundary value in dimension j, the `mass` difference is successively minimized using an array `rows[j]` which

contains the row indices (`.idx`) and cost entries (`.cost`) of the j^{th} column in ascending cost order. If the mass in the next cheapest dimension of query `x` is larger than `mass`, `mass` is assigned to this dimension and the corresponding cost is recorded in `deltadist`. Otherwise, only the mass allowed in this dimension is assigned and the remainder is distributed iteratively to the next dimensions indicated by `rows`.

Algorithm 1: lookup table entry computation

```
1  deltadist = 0;
2  if x[j] < b[j][k] then
3      mass = b[j][k] - x[j];
4      for each row in rows[j] in ascending cost order do
5          if x[row.idx] > mass then
6              deltadist += row.cost * mass;
7          else
8              deltadist += row.cost * x[row.idx];
9          mass -= x[row.idx];
10         if mass <= 0 then
11             break;
12 delta[j][k] = deltadist;
```

The lookup table lower bound LB_{LT} uses the information available for offline computation. During online query processing, the mass distribution of the query with respect to the intervals is known. We incorporate knowledge of the actual query to further optimize our filter on the candidates generated by LB_{LT}.

7.2.1 Online query processing

Note that definiteness in the metric Earth Mover's Distance *mindist* means that diagonal entries in the cost matrix are zero: $c_{ii} = 0$. Consequently, the minimum assigns as much mass as possible to f_{ii}. There are three cases of mass distribution for each dimension i (Figure 7.2). The query mass in this dimension may be smaller than the lower bound $x_i < l_i$ (bottom), it may be inside the interval $l_i < x_i < u_i$ (center), or contain more mass than the interval $x_i > u_i$ (top). So far in minimization of LB_{LT}, only two constraints on these flows apply. $CTarget_{LT}$ restricts minimization to those cases where $x''_j < l_j$. Likewise, $CSource_{LT}$ constrains flows in dimension i to at most the mass in x''_i.

Figure 7.3 depicts the effect of these constraints for the minimization process. Consider the first case, dimension i with $x_i > u_i$: as LB_{LT} restricts flows to cases where the query mass is less than or equal to the lower bound, these dimensions are not taken into account. In the figure, we see that the minimum of x_i and u_i (i.e. u_i) is correctly assigned to the diagonal entries, leaving $x_i - u_i$. u_i is split up into the part that corresponds to the lower bound

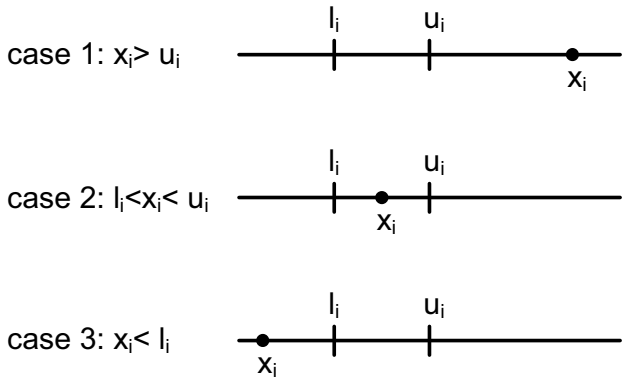


Figure 7.2: Mass distribution: three cases

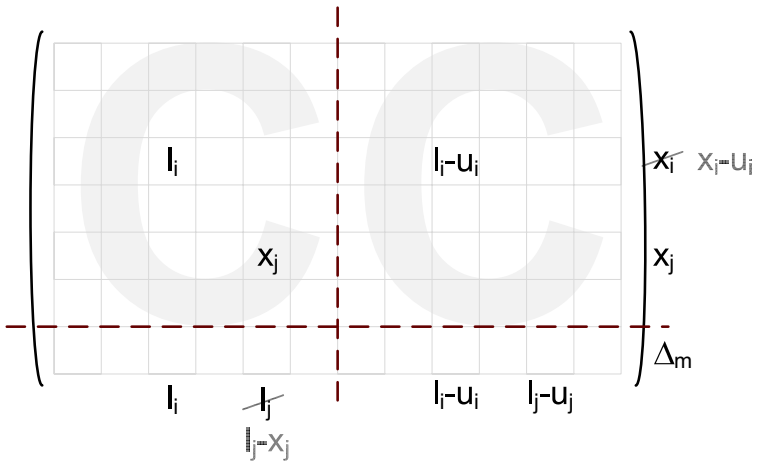


Figure 7.3: Flows in minimization

l_i (left part of the matrix) and the part that corresponds to the interval range $l_i - u_i$ (right part of the matrix). The remaining mass, depicted to the right as $x_i - u_i$ is assigned zero cost even though the Earth Mover's Distance *mindist* would yield non-zero cost for this mass as it would otherwise violate constraint $CTarget''$.

In the second case, just as in the first, the minimum of l_i and x_i has been considered so far, i.e. l_i. Their difference in mass, $x_i - l_i$, has not yet been taken into account. This is in agreement with the Earth Mover's Distance *mindist* computation, as only the boundaries l_i and u_i are known.

Consider the third case, illustrated for dimension j with $x''_j < l_j$. This is taken into account according to $CTarget_{LT}$, and the mass x_j is assigned to the entry f_{jj}. The mass still lacking to fill at least target l_j is assigned from elsewhere.

Thus, for $x''_i > u_i$ the mass $x''_i - u_i$ is not yet considered and for $x''_j \leq l_j$, the mass $l_j - x''_j$ has already been used. For closer approximation of the Earth Mover's Distance *mindist*, we use the actual mass distribution of the

query during candidate processing to further reduce the number of refinement computations. We do this simply by adding a second minimization of the objective function under the same constraints, but additionally restricted to the above mentioned flows:

$$\text{if } x_i'' > u_i : \ f_{ji} = 0 \ \forall \ \ j \neq i \tag{7.3}$$

$$m = \sum_{x_i'' > u_i} (x_i'' - u_i) - \sum_{x_i'' < l_i} (l_i - x_i'') \tag{7.4}$$

In Equation 7.3, remove all other column entries $f_{ij}, i \neq j$ from minimization where the source x_i already covers the maximum target u_i, i.e. $f_{ii} = x_i$. This ensures that these flows are constrained just as in the Earth Mover's Distance *mindist* computation. As analyzed above, flows which either exceed the target or source constrains, are exactly the ones which are now subject to an additional minimization (Equation 7.4). This second minimization is performed independently as an additional pruning step. As the objective function is the same, constraints are still weaker than the Earth Mover's Distance *mindist* ones, this optimization does not violate the lower bounding property.

7.2.2 Multistep algorithm

The lookup table lower bound LB_{LT} proposed allows for efficient query processing in a nearest neighbor multistep algorithm [SK98]. As illustrated in Figure 7.4, the lower bound LB_{LT+}, i.e. LB_{LT} on the lookup table plus our online optimization, is used as a filter on the database to quickly generate a small set of candidates. Additionally, LB_{IM} further reduces this candidate set. Finally, Earth Mover's Distance refinement ensures that the actual result set is obtained. Since lower bounds underestimate the true Earth

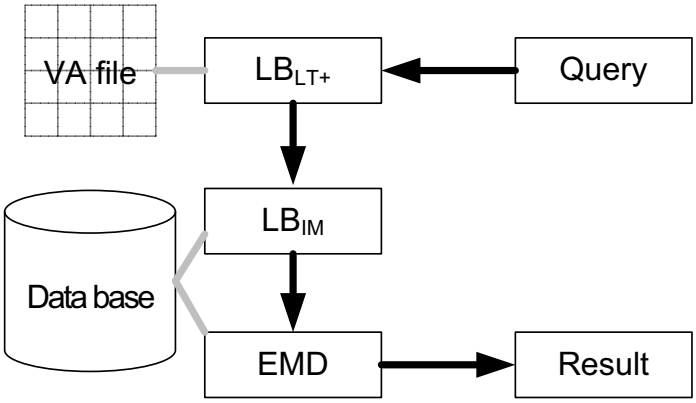


Figure 7.4: Multistep algorithm

Mover's Distance *mindist*, completeness in multistep processing is ensured [Fal96, SK98].

We begin by computation of a ranking according to the *mindist* of the lookup table lower bound LB_{LT}. As mentioned before, this ranking ensures that the number of computations in query processing is minimized [SK98]. The ranking is computed using an efficient sequential scan on quantized histograms of the compact VA-file [WSB98]. Starting with the nearest neighbors according to this filter distance, exact representations of the histograms are loaded and their distance is computed. This filter value is compared against the best true nearest neighbors according to the Earth Mover's Distance, and, as long as this filter value is smaller, the corresponding histogram is refined. If the Earth Mover's Distance value is smaller, the nearest neighbor set and the current *resultdist* are updated accordingly.

7.3 Experiments

Our main experiments use a database of 200,000 color images captured from several TV stations at an interval of at least 5 seconds. For a random sample

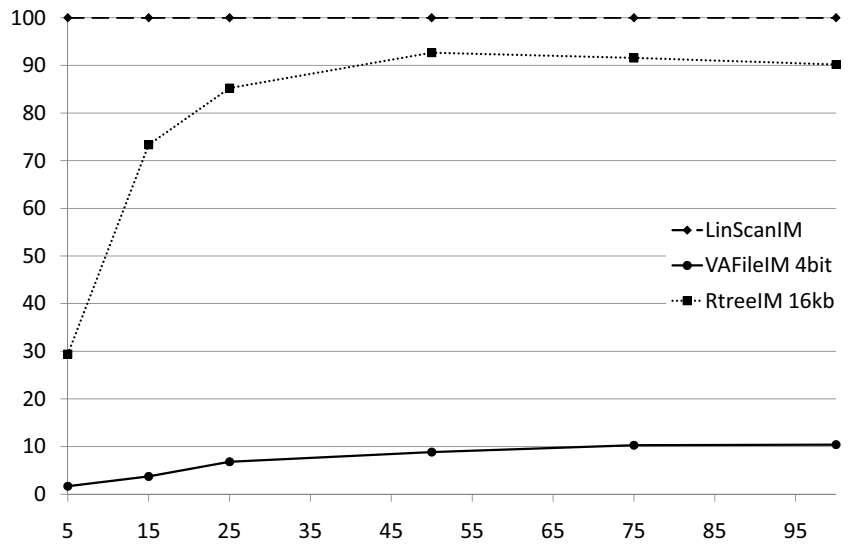


Figure 7.5: dimensionality vs selectivity

of the images (size 10,000), a random sample of pixels (size 500) was selected. For those pixels, relative x and y position, color features L, a and b according to CIE Lab and local texture measures contrast and coarseness were computed and normalized according to their standard deviation. The resulting five million seven-dimensional feature vectors were clustered using k-means to determine a given number of cluster centers which served as Voronoi centers for the subsequent histogram computation of all images. The Euclidean distance between those centers was used to compute the cost matrix for the Earth Mover's Distance. A second image data set consisted of 50,000 heterogeneous digital photos and was processed in the same manner. For each set, 50 separate images were selected as query images.

All experiments were executed on Pentium 4 2.4GHz work stations with 1GB of RAM running Windows XP where we simulated a warmed up system state by posing 25 queries and then recording the query processing times for 25 different queries.

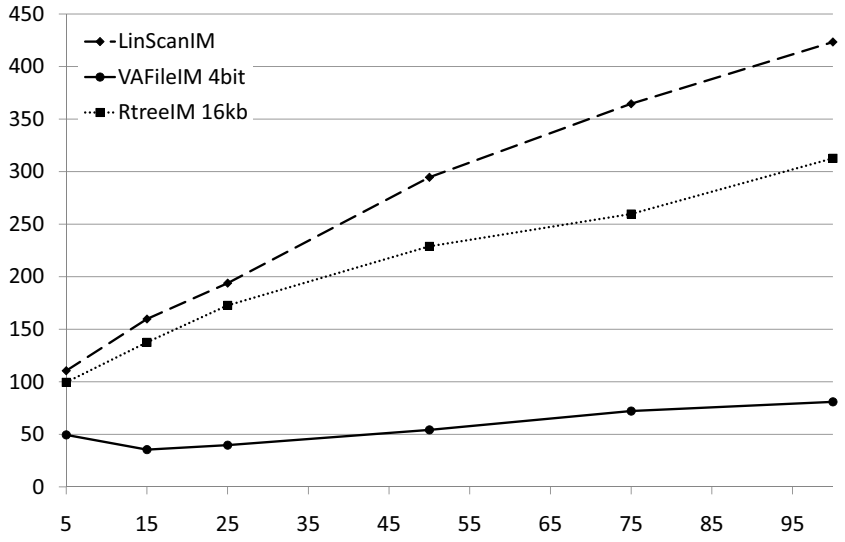


Figure 7.6: dimensionality vs total number of EMD computations

Besides the vector approximation file (VA file) using our extended new lower bound for the *mindist* for a regular 4 bit quantization grid, our *mindist* was utilized for the direct indexing of the Earth Mover's Distance in an R-Tree with a node block size of 16kb. For the vector approximation file with its ranking based nearest neighbor search, we count the number of times that a refinement step on the non-quantized data is required and record it as the selectivity after dividing by the database size. For the R-Tree the selectivity is the number of histograms read in a data node divided by the database size. In addition, we performed a linear scan over the database. In all three cases, the expensive Earth Mover's Distance in the refinement step is preceded by an LB_{IM} filter computation.

In our first set of experiments, we fixed the database size to 100,000 histograms and computed 10 nearest neighbors for dimensionalities between 5 and 100. As was expected and is shown in Figure 7.5, the selectivity of the R-Tree drops rapidly and crosses the 50 percent line at around 10 dimensions.

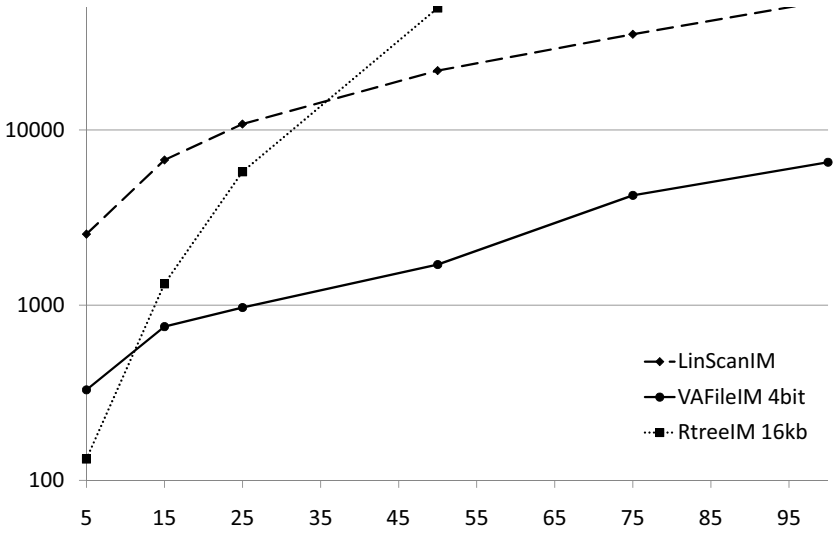


Figure 7.7: dimensionality vs query times

At the same time, the vector approximation file with LB_{LT+} manages to maintain a very good selectivity over the complete range of dimensionalities.

The low number of histograms that have to be examined by the LB_{IM} in the refinement step leads to a low number of expensive Earth Mover's Distance computations that have to be computed in the end. Figure 7.6 shows that out of 100,000 histograms on average no more than 70 extra Earth Mover's Distance computations have to be performed by the vector approximation file approach to find the 10 nearest neighbors. Even though the R-Tree does not prune many histograms from the search space for high dimensionalites, it still manages to beat the linear scan in the number of Earth Mover's Distance computations as the *mindist* order of accessing the data nodes guides the search to more quickly find histograms whose Earth Mover's Distance values are small resulting in more candidate dismissals by the LB_{IM} in the following refinement steps. However, the R-Tree uses additional expensive *mindist* computations during the search.

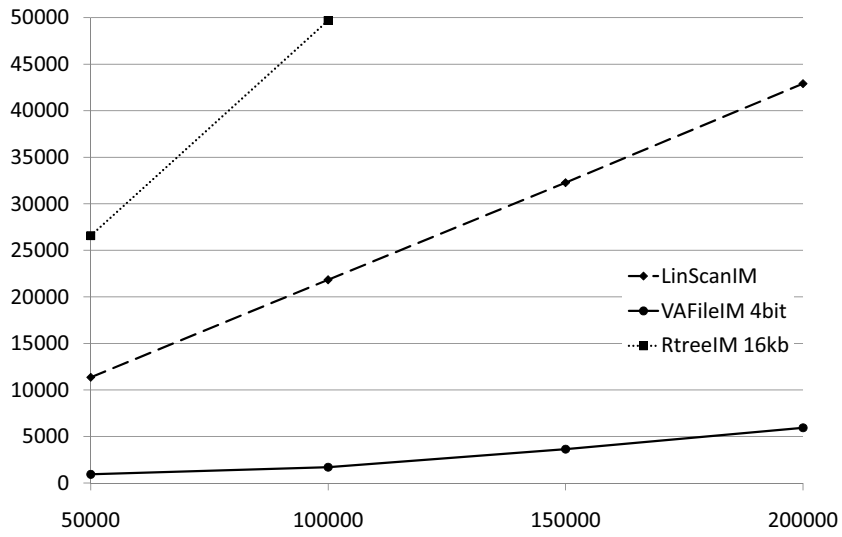


Figure 7.8: database size vs query times

Figure 7.7 shows the average query time for a ten nearest neighbor search with a logarithmic scale. Up to a dimensionality of ten, the R-Tree exhibited the fastest response times, making it our prefered choice for the low dimensional case. After that, the random accesses and the increasingly expensive *mindist* computations during the directory traversal make the R-Tree performance decrease fast. Contrary to that, from dimensionality ten onwards, the vector approximation file runs about one order of magintude faster than the linear scan utilizing LB_{IM} making it the clear favorite for the medium to high-dimensional case.

Next, we looked at the performance dependency of the vector approximation file approach for smaller and larger database sizes with a fixed dimensionality of 50. Figure 7.8 displays the result of that experiment, also including the linear scan and the R-Tree results. As discussed before, the R-Tree cannot compete in this 50-dimensional setting. The vector approximation file on the other hand scales very well. Its slightly super-linear behaviour can be

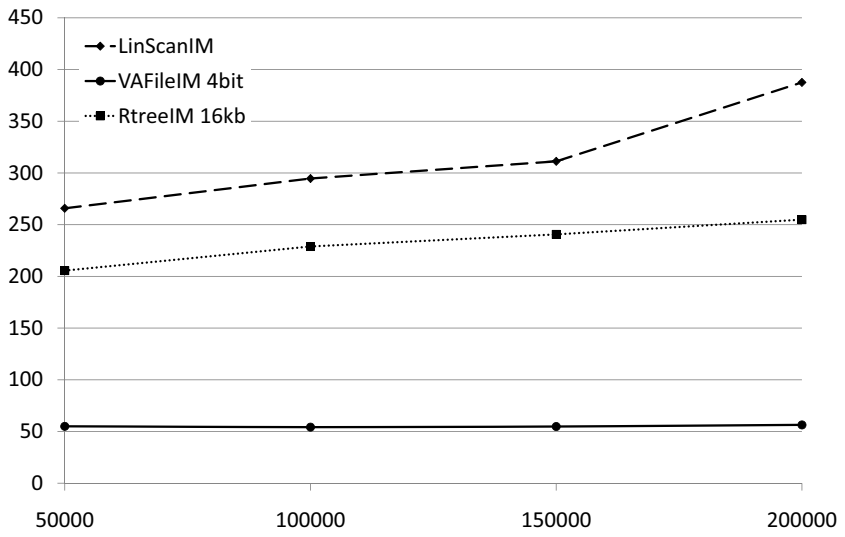


Figure 7.9: database size vs total number of EMD computations

explained by the logarithmic time operations on the larger heap it maintains during query processing.

With a slowly decreasing selectivity (11.4% at 50,000 histograms and 6.9% at 200,000 histograms) the average number of Earth Mover's Distance computations stayed virtually constant for the vector approximation file irrespective of database cardinality (cf. Figure 7.9). The bump at at database size of 150,000 for the linear scan was likely due to a slightly more fortunate ordering of the database as each smaller database was retrieved from the next larger set via random sampling. For the linear scan with a filter, close points at the start of the database translate to a low number of Earth Mover's Distance computations while an unfortunate ordering results in a larger number of Earth Mover's Distance computations.

The next parameter we varied was the number k of nearest neighbors that the queries were to return. Since a higher number k increases the maximum filter distance that still requires refinement, the selectivity is bound to

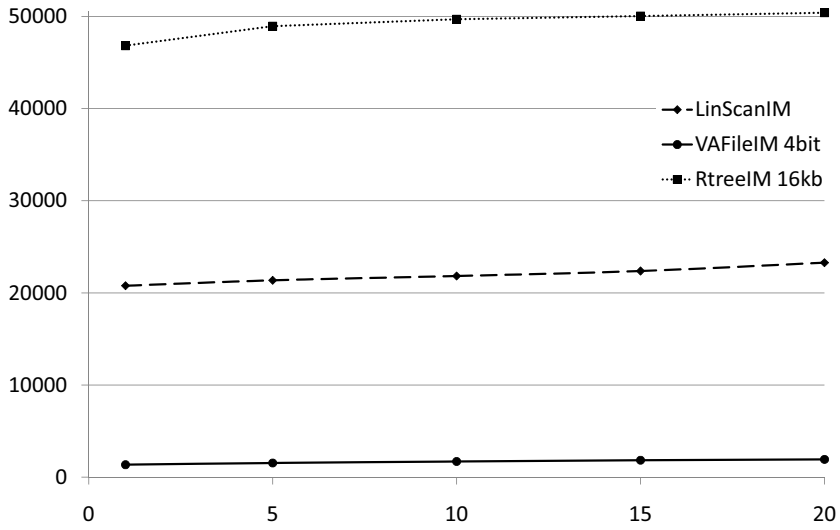


Figure 7.10: number of k nearest neighbors vs query times

increase. Still, at 100,000 histograms with 50 dimensions, the vector approximation file only had to refine up to 11.2% at 20 nearest neighbors, resulting in an average of 98.7 actual Earth Mover's Distance computations. The query response times are depicted in Figure 7.10 and show a stable behaviour for both the linear scan and the vector approximation file, the latter at a much lower level.

In order to see whether the behaviour of the three approaches was highly specific to the TV data set where some dense clusters exist (scenes with long lasting backgrounds), we also ran a subset of the experiments on a more heterogeneous data set consisting of a web collection of user-uploaded digital photos with a large variety of categories. As Figure 7.11 shows, there was hardly any difference in behaviour for identical dimensionalities.

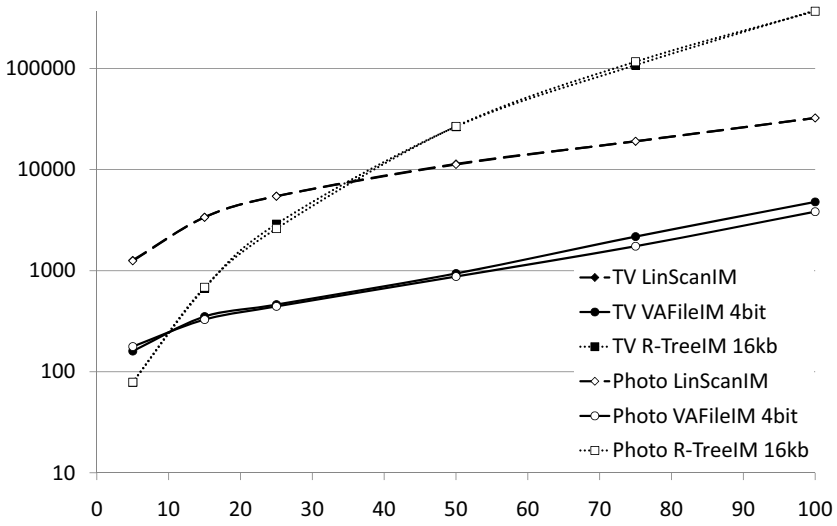


Figure 7.11: dimensionality vs query times for two data sets

7.4 Conclusion

For large and high dimensional multimedia databases, we provide the *mindist* function as a prerequisite for efficiency benefits of index structures based on multidimensional bounding rectangles. Moreover, we propose an optimized quantization-based indexing in the vector approximation file. Devising both offline precomputation in a lookup table and online conservative approximation of the *mindist*, effective pruning is obtained. We prove completeness of our multistep algorithm by showing that our filter is indeed a lower bound. We show the high efficiency of our approach in experiments on several real world data sets. Our multistep algorithm yields substantially better results by orders of magnitude and scales extremely well both in terms of dimensionality and in terms of database size.

Chapter 8

Clustering-based dimensionality reduction

In dimensionality reduction, the goal is efficient query processing through transformation of the features. Transformation of the orginal high dimensional histogram representation to a lower dimensional representation, allows using the original Earth Mover's Distance function, on the reduced histograms as a filter in a multistep architecture. For this filter, the ICES (index enabled, completeness, efficiency, selectivity) criteria as introduced in Chapter 3 apply.

Dimensionality reduction is especially helpful for efficiency gains in distance functions with superlinear complexity, like the Earth Mover's Distance. For smaller dimensionalities, distance calculations can be performed in reasonable time. Moreover, dimensionality reduction can be *chained* with existing filter functions for the Earth Mover's Distance, i.e an existing filter can additionally be applied to the reduced data as the result of the dimensionality reduction is again an Earth Mover's Distance.

For most bin-by-bin distances like L_p norms, simple dimensionality re-

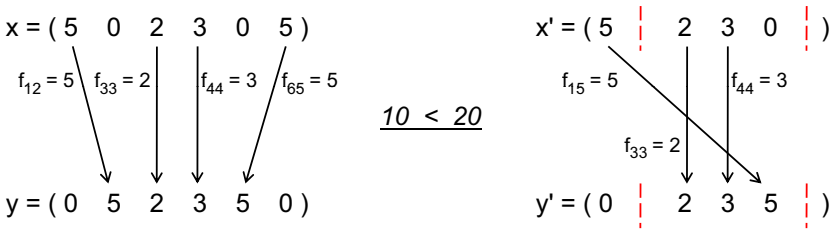


Figure 8.1: Dimensionality reduction and lower bounding

duction is straightforward. By discarding histogram dimensions, only non-negative addends are dropped, thus the resulting distance is guaranteed to be a lower bound. For the Earth Mover's Distance, discarding dimensions can result in larger distances, as the resulting match might be worse than the original one.

Consider the example in Figure 8.1: removing two dimensions results in a larger Earth Mover's Distance value. For example, on the left in the original histogram representation, x_1 may be moved at rather low cost to y_2. Discarding dimension two leads to higher cost for the reduced histograms on the right, as x'_1 has to be moved to y'_5. Overall, the distance between x and y is only 10, however, by discarding dimensions as in this example, it is 20.

Thus, discarding dimensions may violate the lower bounding property. Consequently, to avoid false dismissals, simply discarding dimensions is not a valid option. A special case of Earth Mover's Distance dimensionality reduction is discussed in [LBS06]. Focusing on bioinformatics image data, separate measures are computed for a tiling of the images. Each image is associated with independent features. For these features, the authors derive a hierarchy of filters, constructed by merging "neighboring" histogram bins, where "neighboring" means adjacent with respect to the tiling of the images. This merging constitutes a special case of dimensionality reduction. As additional "dummy tiles" are assumed, this approach does not generalize

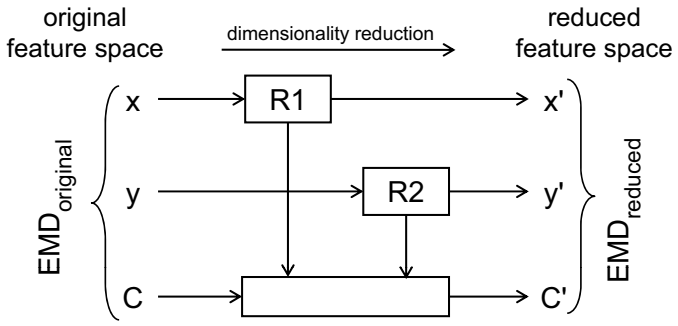


Figure 8.2: Dimensionality reduction

to arbitrary Earth Mover's Distance applications. We therefore propose a generalized Earth Mover's Distance dimensionality reduction.

In general, a reduced Earth Mover's Distance is defined via reduction matrices for the query and the database histograms (cf. Figure 8.2). Any reduction of the dimensionality of the Earth Mover's Distance requires specification of a corresponding reduced cost matrix:

Definition 8.1 ***Reduced Earth Mover's Distance.***

For two d-dimensional histograms x, y and a cost matrix C according to Definition 6.2 and for two reduction matrices $R1 \in \Re_{d,d1}$ and $R2 \in \Re_{d,d2}$, the lower-bounding reduced Earth Mover's Distance is defined as:

$$EMD_C^{R1,R2}(x, y) = EMD_{C'}(x \cdot R1, y \cdot R2) \tag{8.1}$$

where $C' \in \mathbb{R}^{d1 \times d2}$ is a lower-bounding reduced cost matrix.

The optimal cost matrix with respect to given reduction matrices is the one which provides the largest lower bound to the Earth Mover's Distance in the original dimensionality. As we will prove, the tightest possible reduced cost matrix consists of minima over the original cost entries.

To illustrate why those minima have to be chosen, we give an example of the worst case that leads to this condition: To ensure the lower bound property, underestimating the true distance means assuming the worst case, i.e. the original mass was transfered at minimum cost. Consider $x = (0, 1, 0, 0)$ and $y = (0, 0, 1, 0)$ and Manhattan (L_1 norm) ground distance. Their Earth Mover's Distance is then simply 1 (moving one unit of mass from dimension x_2 to y_3 at ground distance 1: $1 * 1$). Combining the first two and the last two dimensions, the reduced features are $x' = (1, 0)$ and $y' = (0, 1)$. The minimum cost entry from the original dimensions x_1 and x_2 to dimensions y_3 or y_4 is the cost from x_2 to y_3 which is indeed the 1 that was used in the original Earth Mover's Distance. If this value were to be exceeded, the lower bound property would be lost.

We formalize this for merging reduction matrices (i.e. binary entries):

Definition 8.2 ***Optimal Reduced Cost Matrix.***

For two d-dimensional histograms x, y and a cost matrix C according to Definition 6.2 and for a reduced $EMD_C^{R1,R2}$ according to Definition 8.1, the optimal reduced cost matrix $C' = [c'_{i'j'}]$ is defined by:

$$c'_{i'j'} = \min\{c_{ij} | r1_{ii'} = 1 \wedge r2_{jj'} = 1\}$$

Thus, the optimal reduced cost matrix entry $c_{i'j'}$ between reduced dimensions i' and j' is the one that takes the minimum cost over all dimensions i, j that were merged into i' and j', i.e. those dimensions where the corresponding reduction matrix entry is set to one.

8.1 Reduced Earth Mover's Distance

As discussed above, optimal reduction of the cost matrix in the Earth Mover's Distance depends on the reduction matrices. We define what would constitute an optimal choice of R. As that R is not attainable in practice, we then introduce a clustering-based approach for efficient reductions.

Given a d-dimensional query histogram x and a query distance ε, the optimal reduction $R \in \Re_{d,d'}$ to dimensionality d' can be defined in terms of the number of refinements required to answer an ε range query for a database DB:

$$R = \arg\min_{R' \in \Re_{d,d'}} |\{y \in DB \mid EMD_C^{R'}(x, y) \leq \varepsilon\}|$$

We propose a method that combines the original dimensions in such a way that the distances between the resulting reduced dimensions are as great as possible. At the same time the distance information that is lost shall be as small as possible. These two demands correspond to the well known goals of *maximum inter-class dissimilarity* and *minimum intra-class dissimilarity* aimed for by clustering algorithms.

Figure 8.3 gives an example for $d = 4$ and $d' = 2$, where the original dimensions $d1$, $d2$ are combined to the reduced dimension $d1'$ and $d3$, $d4$ to $d2'$. The lost distance information within $d1'$ is $c_{12} = c_{21} = 1$ and within $d2'$ it is $c_{34} = c_{43} = 1$. The distance that is preserved between $d1'$ and $d2'$ is $c_{23} = c_{32} = 2$ which is the minimum and thus consistent with Definition 8.2.

A further postulate is flexibility in terms of the number of reduced dimensions d'. This flexibility allows users full control of the trade-off between the quality of the approximation and the efficiency of the filter step computation. This is a unique advantage of our new method that none of the existing approximation techniques for the Earth Mover's Distance provide.

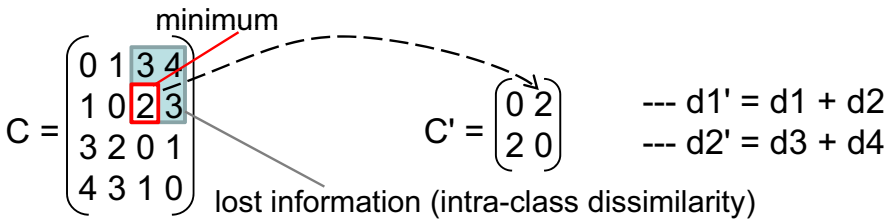


Figure 8.3: Clustering on EMD cost matrix entries.

We meet these demands for flexible data independent dimensionality reduction by clustering on the cost matrix entries between dimensions in the Earth Mover's Distance. Specification of d' is possible with partitioning clustering algorithms such as k-means or k-medoids [KR87, KR90]. Both of these algorithms start with an initial random partition of the data into k groups, where k is the user specified number of clusters. Working in an iterative manner, these algorithms assign points to the nearest cluster center and recompute these centers for the new partitioning until the clustering is stable with respect to a quality criterion. k-means uses the arithmetic mean as center of clusters, whereas k-medoids chooses a central point from the data set as representative. The points in our case refer to the original dimensions of the feature space.

We opt for the k-medoids algorithm, because, unlike k-means, it does not require an explicit distance function for the feature space. Thus we can handle any data set if the two inputs for the Earth Mover's Distance calculation are provided (histograms and cost matrix). This holds even if the ground distance function underlying the cost matrix is not explicitly known.

We sketch our k-medoids algorithm on the Earth Mover's Distance dimensions in the following, for details please refer to [KR87, KR90]. The algorithm starts by randomly choosing k representatives (medoids) from the

set of original dimensions and assigns the remaining ones to their nearest medoid according to the cost matrix. The quality of the clustering is determined via the total distance defined as:

$$TD = \sum_{i'=1}^{d'} \sum_{\{i|r_{ii'}=1\}} c_{i\,m_{i'}}$$

where $m_{i'}$ is the representing medoid of the cluster that corresponds to the reduced dimension i'. The total distance thus reflects the degree of dissimilarity within the clusters, i.e. the objective function that the algorithm tries to minimize. In the next step, the algorithm aims at improving the clustering. It determines the total distance that results when swapping a non-medoid with a medoid. In a greedy manner, the configuration with the lowest total distance is chosen and the corresponding pair is swapped. The algorithm terminates if no swapping leads to further improvement of the total distance. The result is a clustering into k partitions.

In our case each of the k clusters corresponds to one of the d' reduced dimensions. By setting the parameter k, the reduced dimensionality can thus be flexibly chosen. The elements contained in cluster i' are the original dimensions that were combined to i'. This approach requires no knowledge about the underlying data set.

8.2 Query processing algorithm

We use reduced Earth Mover's Distance in a multistep algorithm as discussed before, following the optimal (with respect to the number of refinements) framework [SK98]. For a k nearest neighbor query, k initial results are retrieved from the filter ranking. They are refined and inserted into the result set, sorted with respect to their actual distance from the query. Then, the

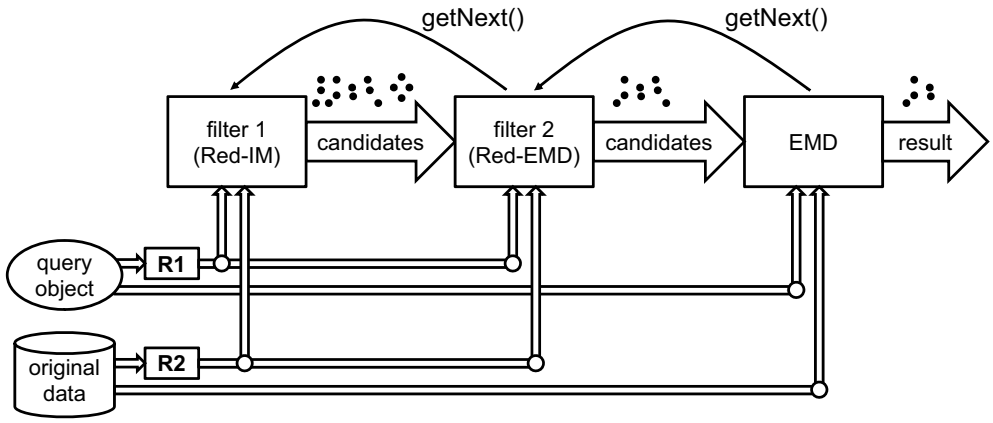


Figure 8.4: Multistep setup for dimensionality reduction

next best histograms with respect to the filter distance are retrieved. If the filter distance is smaller than the current k^{th} nearest neighbor, the corresponding histogram is refined and compared against the current k^{th} nearest neighbor with respect to the actual distance. If smaller, it is sorted into the result set, displacing the furthest one from the set. This is repeated until the filter distance is larger than the current k^{th} result. As soon as the filter distance is larger, none of the remaining histograms have a smaller filter distance. And since the filter distance is a lower bound of the actual distance, their actual distance is also larger. The result set now contains the actual k nearest neighbors.

8.3 Experiments

Setup.

Our experiments use a database of $10,000$ radiography images from the Image Retrieval in Medical Applications (IRMA) project [LGT+04, DKN05]. The data set is available for download from [LSOL05] and was part of the

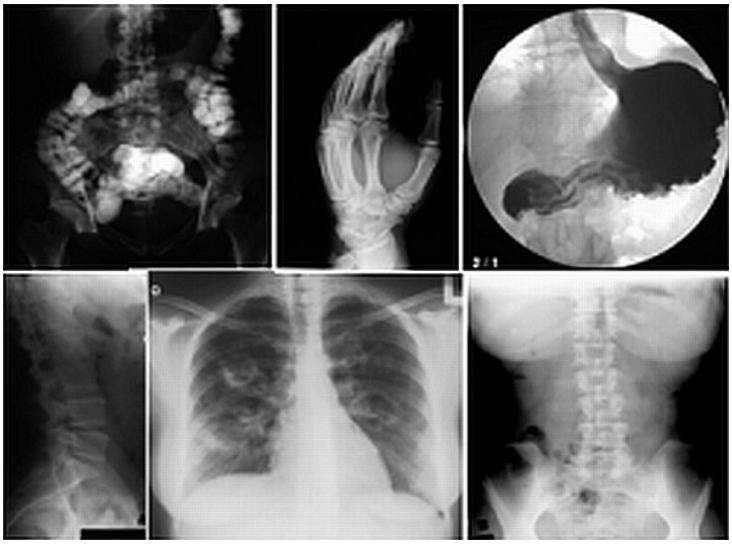


Figure 8.5: IRMA data

2005 ImageCLEFmed image retrieval competition [DMC+07]. The features are 199-dimensional histograms, and Euclidean distance is used for the cost matrix.

The reported results in this section are averages over a workload of 100 k-nearest neighbor queries. From the complete database, a query set of 100 query histograms was extracted. All experiments were executed on Pentium 4 2.4GHz work stations with 1GB of RAM running Windows XP.

Reduced dimensionality.

For 50 nearest neighbors, Figure 8.6 shows the runtime performance for varying degrees of dimensionality reduction, from using only 20 dimensions up to 90 dimensions. We can see that the two approaches which do not rely on a dimensionality reduction, the averaging lower bound LB_{Avg} and the independent minimization lower bound LB_{IM} (cf. Chapter 4), decrease the response times to between 3 and 4 minutes (depicted at the very left as single

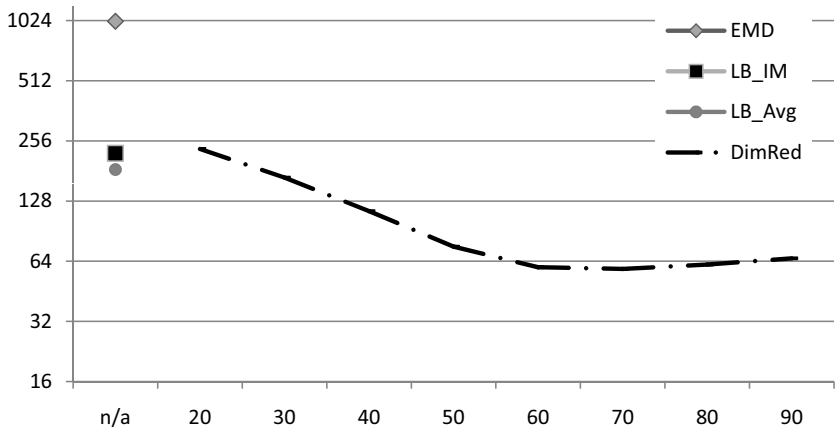


Figure 8.6: Computation time vs. reduced dimensionality

measurements). Our dimensionality reduction technique performs similarly well when reducing the dimensionality to between 10% and 15% of the original dimensionality and reaches its optimum of just below 1 minute around dimensionality 60. Compared to the 17 minutes of the sequential scan and to the 3 minutes of the next best competing approach this equates to a speedup of factor close to 22 and close to 4 respectively for our techniques.

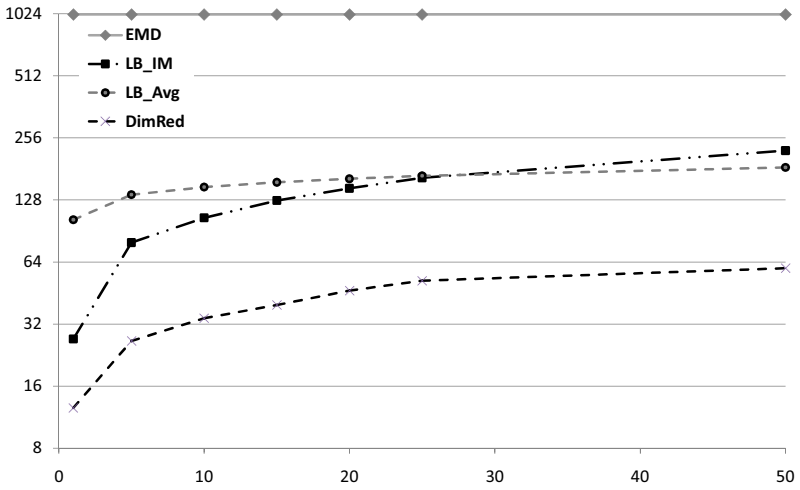


Figure 8.7: Computation time vs. number of nearest neighbors

Result size.

We varied the parameter k of nearest neighbors in the next experiment, illustrated in Figure 8.7. Of foremost interest in this experiment was how our approach fares compared to the independent minimization lower bound LB_{IM} when k is chosen to be lower than 50. While independent minimization LB_{IM} shows a comparatively low selectivity and a fast average response time, our reduction technique outperforms LB_{IM} by a factor of about two.

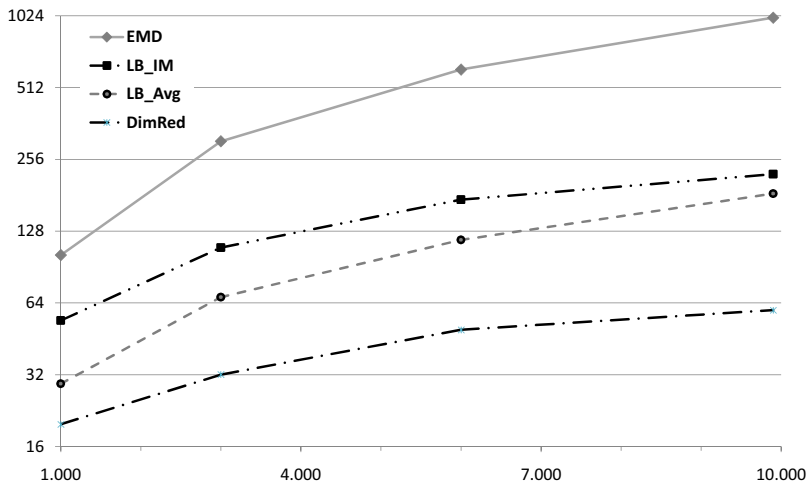


Figure 8.8: Computation time vs. cardinality

Scalability.

To asses whether our approach performs equally well over a range of database cardinalities, we subsampled the database in three steps, yielding databases of 1,000, 4,000, and 7,000 objects, respectively, as well as the entire database of 10,000 objects. As Figure 8.8 depicts, our approach scales well, and the querying improvements reported above transfer to these sets.

8.4 Conclusion

We propose clustering-based dimensionality reduction for the Earth Mover's Distance. Clustering based reduction relies only on the original cost matrix information to create reduced cost matrices. Our technique is flexible in the reduced dimensionality of both the data and the query, allows for chaining with existing approaches, and guarantees completeness in multistep query processing. Experimental evaluation on real world data demonstrates that dimensionality reduction yield substantial efficiency gains.

Part II

Retrieving time series data

Chapter 9

Efficient Time Series Search and Retrieval

Continuous growth in sensor data and other temporal data increases the importance of retrieval and similarity search in time series data. Efficient time series query processing is crucial for interactive applications. Existing multidimensional indexes like the R-tree provide efficient querying for only relatively few dimensions. Time series are typically long which corresponds to extremely high dimensional data in multidimensional indexes where time series are treated as histograms. Due to massive overlap of index descriptors, multidimensional indexes degenerate for high dimensions and access the entire data by random I/O. Consequently, the efficiency benefits of indexing are lost.

We propose the TS-tree (time series tree), an index structure for efficient time series retrieval and similarity search. Exploiting inherent properties of time series quantization and dimensionality reduction, the TS-tree indexes high-dimensional data in an overlap-free manner. During query processing, powerful pruning via quantized separator and meta data information greatly

reduces the number of pages which have to be accessed, resulting in substantial speed-up. In thorough experiments on synthetic and real world time series data we demonstrate that our TS-tree outperforms existing approaches like the R*-tree or the quantized A-tree.

9.1 Introduction

Many applications in science, e-commerce and surveillance monitoring rely on recording sensor information in regular time intervals. As a consequence, time series data has been growing tremendously. This flood of data requires data management for fast and easy storage and access. Similarity search on time series data is a typical requirement for such data bases. Similar patterns in sensor data might indicate a common cause or provide a prediction for future sensor values.

To enable efficient similarity search, R-trees and other multidimensional indexing structures have been used [Keo02]. These multi-purpose indexing structures, however, are designed for relatively low-dimensional data. For the special case of time series data, they are usually not adequate. Most time series are long, which corresponds to many dimensions in multidimensional indexes. Multidimensional indexes, however, are known to provide efficiency gains only up to a certain dimensionality [WSB98]. In the case of the R-tree, sequential scan is faster from about 16 dimensions, depending on the application [BKSS90]. One of the reasons why R-trees will eventually fail is that in high-dimensional spaces MBRs (minimum bounding rectangles) of the subtrees overlap to a high degree. Therefore more paths have to be accessed in query processing, requiring expensive random reads of index and data pages.

In time series data, this problem is often aggravated. Summarizing data in minimum bounding regions, the sequential nature of time series is not captured. For example, in temperature measurements, successive values are typically within a range of few degrees. Consequently, overlap in these dense areas is large, resulting in voluminous MBRs with large proportions of empty space. As typical queries intersect with many of these MBRs, the number of excess page reads increases further. Consequently, these indexes degenerate to random read of the entire data.

B-trees are another family of indexes that have been used for sequences e.g. in the Prefix B-tree, with further compression of blocks via Patricia tries in the String B-tree [BU77, FG99]. Prefixes are used as separators between subtrees. Since only the prefix is actually used to index the data this leads to poor pruning power for similarity search on long time series data. In turn, many paths have to be accessed as well, negating efficiency gains.

To formalize similarity such that sensor data can be searched accordingly, L_p norms and Dynamic Time Warping (DTW) are the most common models. L_p norms, such as Euclidean distance, compare time series one position at a time and have been shown to work well in a number of applications. To allow matching of slight shifts in time series, DTW stretches or squeezes the time series along the time axis to align their value patterns. The remaining differences between these best matches constitute the distance [BC94]. For indexing, envelope-style upper and lower bounds have been proposed [Keo02]. They allow for exact indexing of time series at the expense of additional relevant regions. Therefore, DTW-based similarity search typically entails more page reads.

We propose a novel index structure, the TS-tree (time series tree) that is specialized for efficient similarity search on time series. The TS-tree avoids

overlap and provides compact meta data information on the subtress, thus reducing the search space very effectively. To ensure high fanout, which in turn results in small and efficient trees, index entries are quantized and dimensionality reduced. Depending on the nature of the time series, e.g. smooth or bursty, quantization brings out the relevant characteristics via SAX (symbolic aggregate approximation) or dimensionality reduction techniques extract the most relevant dimensions using wavelet analysis, respectively [LKLC03, SZ04b]. For powerful pruning, we provide a formal *mindist* function between any query time series and the entire available meta data. This *mindist* enables efficient query processing without false dismissals.

Our contributions include

- a novel overlap-free index structure for time series
- efficient query processing on highly descriptive meta data
- support for the most common time series similarity models

This part is structured as follows: we review related work in Section 9.2. An analysis of inherent properties of time series and indexing paradigms is presented in Section 9.3. We define and discuss our novel TS-tree in Chapter 10. Query processing techniques and algorithms are discussed in Section 10.1. We demonstrate substantial efficiency gains in the Experiments, Section 10.2, and conclude in Section 10.3.

9.2 Related work

For efficient and effective time series indexing, multistep filter-and-refine approaches have been proposed, as discussed in Section 10.1.3 in greater detail. Due to the length of time series, dimensionality reduction is typically

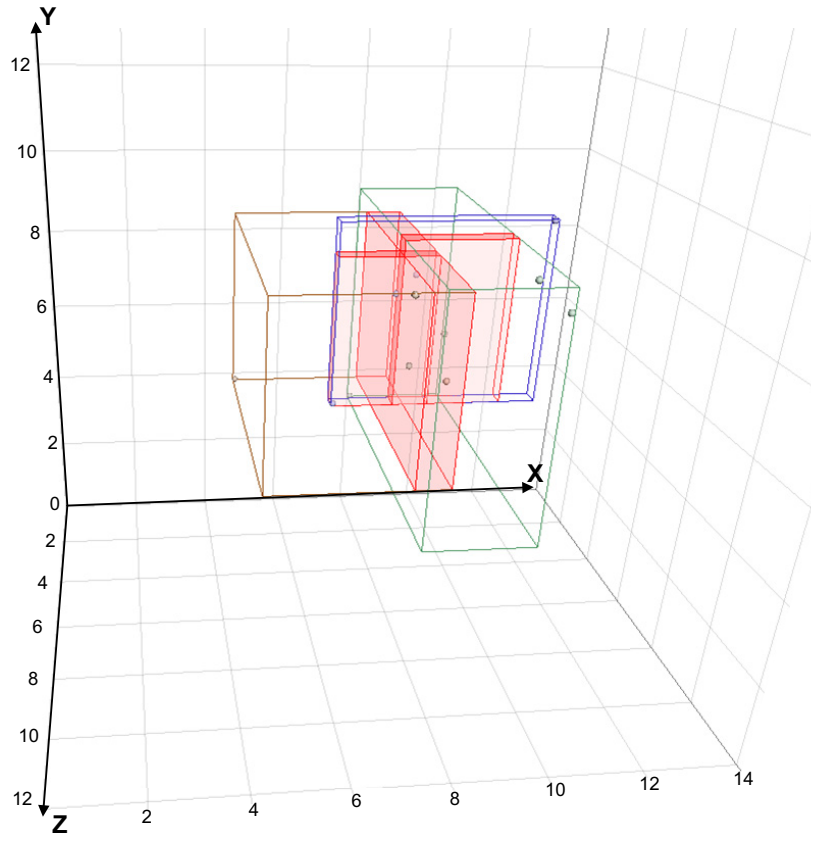


Figure 9.1: R-tree indexing of time series

required. Time series may contain hundreds of values which hinders applicability of multidimensional indexing structures as is. Piecewise aggregate approximation (PAA) and symbolic aggregate approximation (SAX) [KCPM01, YF00, LKLC03] outperform other dimensionality reduction techniques like singular value secomposition or discrete fourier transform (SVD, DFT) for time series data [SZ04b].

Multidimensional indexing structures like the R-tree or R*-tree organize data in minimum bounding rectangles (MBR), i.e. upper and lower bounds per dimension. This approach works well for spatial or point data in lower dimensional settings. For high dimensionality, as found in typical time series,

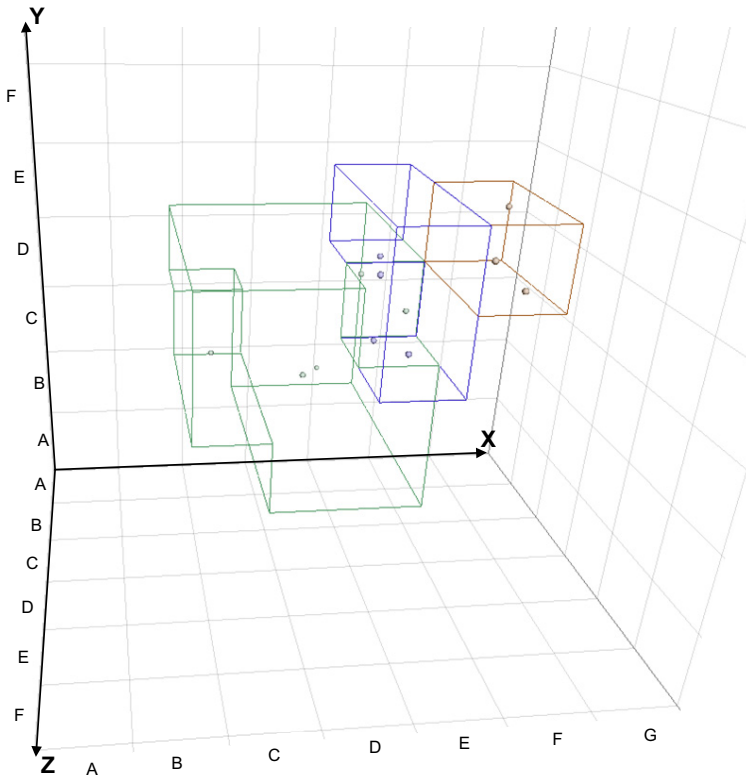


Figure 9.2: B-tree split for indexing of time series as used in our approach

R-trees and their variants fall prey to the curse of dimensionality, which leads to overlapping MBR regions [BGRS99, BBK01].

We illustrate this effect in Figure 9.1. We generated several time series of length six according to the random walk 2 model (see also the experiments in Section 10.2) and reduced them to a three-dimensional space for visualization purposes. They were inserted into a three-dimensional R-tree. As we can see from this toy example, time series data indexed in R-trees may create voluminous, largely empty, minimum bounding rectangles around the corresponding data points at the center of the data space. These effects on time series are discussed in more detail in Section 9.3.2.

B-trees [BM70, BM72], on which most hierarchical indexing structures

are based, were originally developed for one-dimensional data. They have been adapted for string data in [BU77, FG99]. B-trees use prefix separators, thus no overlap for unique data objects is guaranteed.

Using B-trees to index time series can be done using lexicographic order on their values. This way, time series are separated according to their common prefixes ("separator split"). This splitting strategy suits time series data very well, as we illustrate for our example time series data in Figure 9.2. As we can see, the central time series are very concisely summarized in the middle node, whereas those time series which are further away from the central diagonal are separated in the left and right nodes. Clearly, there is no overlap. This is highly advantageous for query processing, where many nodes may be pruned from the search. Thus, our approach is based on the B-tree separator split. Note however, that in traditional B-trees, the separators would span across the entire feature range between separator boundaries. Our approach combines B-tree separator split with meta data as in R-trees for the compact index node ranges as depicted in the figure. This concept is presented in Section 9.3.2.

Specialized indexing structures for high dimensions have been presented in the literature. The X-tree (extended node tree), for example, uses a different split strategy to reduce overlap [BKK96]. When low-overlap split is impossible, supernodes of double size are created, eventually degenerating into sequential scan. Our analysis of the disadvantages of MBR-structure in R-trees in principle generalizes to X-trees. The A-tree (approximation tree) uses VA-file-style (vector approximation file) quantization of the data space to store both MBR and VBR (virtual bounding rectangle) lower and upper bounds [BBK$^+$00, SYUK00, WSB98]. While VBRs provide additional pruning information about the subtrees, the overhead of additional meta data

reduces fanout. For time series, both MBR and VBR information suffer from the drawbacks as in R-trees. As the VBRs are quantized lower and upper bounds with respect to the MBRs, problems with overlap persist.

The TV-tree (telescopic vector tree) is an extension of the R-tree [LJF94]. It uses minimum bounding regions (spheres, rectangles or diamonds, depending on the type of L_p norm used) restricted to a subset of active dimensions. Active dimensions are the first few dimensions which allow for discrimination between subtrees. As the dimensionality in index nodes is reduced, fanout increases. However, overlap is still high, especially in the dimensions not considered during query processing. For similarity search, non-active dimensions have to assumed as infinite, resulting in poor pruning power.

The most popular similarity models for time series are L_p norms (typically Euclidean distance) and dynamic time warping (DTW). L_p norms compare values of corresponding points in time. DTW stretches or squeezes the time axis to compute the best match [SC71]. Using an envelope of upper and lower bounds around time series, indexing of short (about 16 dimensions) time series is efficient [Keo02, ZS03]. We discuss similarity models and indexing in greater detail in the following.

9.3 Time series similarity search

Sensor data or other types of measurements result in sequences of values called "time series". Examples include stock data (see Figure 9.3), temperature measurements and network traffic volume. Most time series are periodical recordings. This means that subsequent values are typically measured at regular intervals, e.g. on a daily or hourly basis.

9.3.1 Time series

Time series are sequences of values ordered with respect to the time associated with the values. Formally:

Definition 9.1 ***Time series.***

A time series t is a temporally ordered sequence of values

$$t = (t_1, \ldots, t_n), t_i \in \mathbb{R},$$

where time point $f(i)$ is before $f(i+1)$: $f(i) < f(i+1)$,
and $f : \mathbb{N} \rightarrow \mathbb{R}$ is a function mapping indices to time points.

Typically, time series are very long. Consider daily stock data measurements for periods of several years or even hourly temperature measurements for decades. Other applications of sensor based monitoring might record important values per hour or per minute. This is an important property which distinguishes time series data from many other multimedia data. Long time series data which is treated as histogram data, corresponds to very high

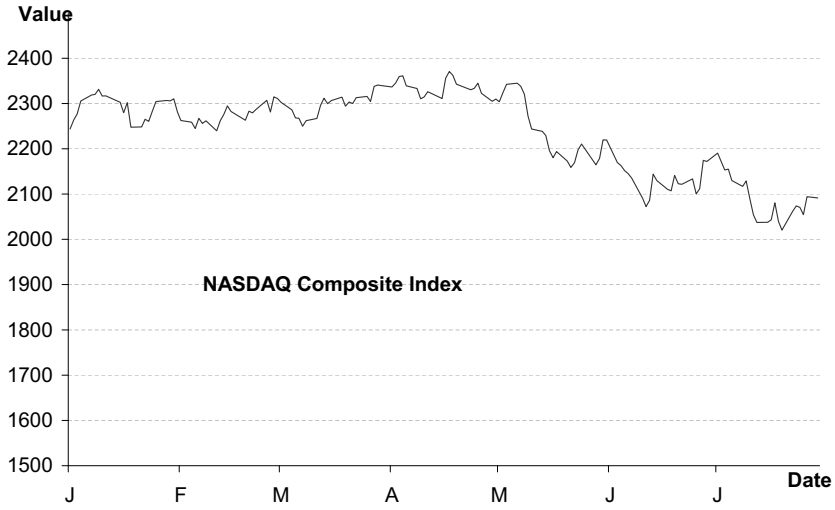


Figure 9.3: Time series data: stocks (2006)

dimensional feature spaces. This is a challenge for similarity search and indexing, as we will discuss later on.

In many applications, time series are *smooth*, i.e. subsequent values are within predictable ranges of one another [SZ04b]. For example, stock data as illustrated in Figure 9.3 typically exhibits only small changes within a few percentage points per day. Similarly, temperatures in a climate monitoring application do not jump from -20°C to +20°C. Rather, if one day's temperature is 10°C, we would expect the following day to have a temperature only a few degrees higher or lower. Thus, most time series data exhibits a correlation between subsequent values. This correlation is often used to mimic time series by synthetic random walk sequences [Cha96, AFS93].

9.3.2 Core concept

The design goal of the TS-tree is to create a compact yet detailed index structure for time series similarity search and retrieval. Compactness means that similar time series should be located in the same subtree using minimal storage space and directory overhead in overlap-free subtrees. Detailed information is necessary for powerful pruning during similarity search. The TS-tree exploits the inherent properties of time series. We describe the core concepts before giving formal definitions.

The TS-tree core concepts:

- **Hierarchical tree structure:** provides directory information on several levels for subtree data. As discussed above, we have to avoid degeneration into random read of the entire data base. Existing multidimensional indexing structure do not scale to high dimensions [BGRS99, WSB98, BBK01] and produce largely overlapping descriptors. We

therefore focus on compact, overlap-free descriptors for time series that provide substantial pruning power.

- **Large capacity:** is necessary for small trees with high fanout. Dimensionality reduction substantially shortens time series, requiring less memory while keeping the most relevant information. Similarly, quantization of the exact time series values to discrete symbols increases capacity of index nodes.

- **Overlap-free compact subtrees:** Like R-trees or B-trees, the TS-tree is a balanced tree that grows in a bottom-up fashion. Time series inserts into leaf nodes may lead to overflow, entailing node splits that may propagate up the tree. Splitting strategy is crucial for trees as it affects the compactness and utilization of the entire index. Most notably, overlap-free splitting is a major concern as time series are high-dimensional, which leads to degenerative overlap in R-tree family indexes.

  - **Overlap-free.** Overlap-free indexing is possible by exploiting lexicographic ordering on time series. Developed originally for one-dimensional data in B-trees, TS-tree *separators* are time series prefixes [BM70]. These prefixes separate time series smaller than the separator from large ones with respect to lexicographic order. Smaller time series are located in subtrees to the left of the separator, larger ones in subtrees to its right. This is unambiguous as lexicographic order is a total order. Splitting in lexicographic manner thus ensures overlap-free subtrees TS-tree descriptors.

  - **Compact.** Obviously, for real values common prefixes are rare. Short separators in the first few dimensions provide little infor-

mation. While this is desirable for large fan-outs, pruning power is poor. *Rough quantization* of separators using discretization to very few symbols yields longer separators which actually provide more information in more dimensions. Moreover, rough quantization maps similar time series to the same quantized representation. This means that they are very likely stored in the same subtrees, resulting in compact TS-trees.

  - **Descriptive.** Separators split one dimension at a time in lexicographic order. The TS-tree exploits the *order of dimensionality reduction* through DWT (discrete wavelet transform) or PCA (principle components analysis). Ordering is according to degree of information in dimensions, measured as variance or level of detail. In TS-tree this ordering means that dimensions with more descriptive content are split first, leading to descriptive separators.

- **Overlap-free detailed subtrees:** during query processing, long separators provide meaningful information for powerful pruning on the dimensions covered by the separators. More detailed information on those dimensions that have not (yet) been split, is provided by additional, and more finely quantized, *meta data* in TS-trees. They consist of upper and lower bounds per dimension of the time series values in the corresponding subtree, like MBRs (minimum bounding rectangles) as used in R-tree family indexes [Gut84]. In TS-trees, they provide additional descriptor information for query processing. Note that subtrees are still *overlap-free*, as we keep separator split and do not adopt MBR-split.

In summary, TS-trees exploit the inherent properties of time series. Lexico-

graphic separator split in coarsely quantized time series ordered with respect to the most descriptive reduced dimensions are coupled with finely quantized meta data information for powerful pruning. This allows for efficient and effective similarity queries for time series data. In the next chapter, we detail the structure of TS-trees in a formal manner before proceeding to query processing on the complex descriptor information.

Chapter 10

The TS-tree

In this chapter, we formalize the TS-tree. It is a hierarchical structure which extends the node structure of R-trees or B-trees and uses B-tree split ("separator split") to ensure balancing. The descriptors stored in the TS-tree nodes are lower and upper bounds of the dimensionality reduced time series data as well as quantized separators between subtrees. Separators are prefixes of time series, typically much shorter than the entries in data leafs, that are lexicographically larger than subtrees to the left and smaller than subtrees to the right:

Definition 10.1 ***Separator.***

A separator $\mathbf{S}$ *between two time series* t_l *and* t_r *is a time series with:*

- $t_l \leq \mathbf{S}$ *(lexicographically larger than left time series)*
- $\mathbf{S} \leq t_r$ *(lexicographically smaller than right time series)*
- $\nexists \mathbf{S}' : |\mathbf{S}'| \leq |\mathbf{S}| \wedge t_l \leq \mathbf{S}' \leq t_r$ *(as short as possible)*

where lexicographically smaller is defined as: $a \leq b$ *iff*

$$(\exists j \leq \min\{|a|, |b|\}\ \forall i \in \{1, \ldots, j-1\} : a_i = b_i \wedge a_j < b_j) \vee$$

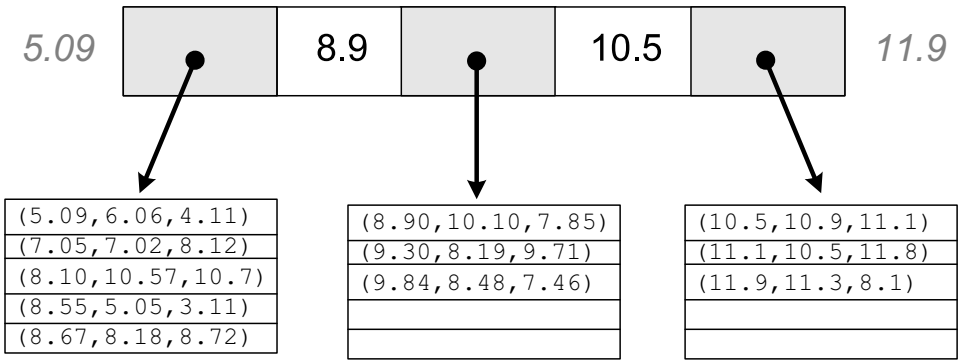


Figure 10.1: Separators

$$(|a| \leq |b| \wedge \ \forall i \in \{1, \ldots, |a|\} : a_i = b_i)$$

A separator is thus the shortest possible time series that differentiates between time series to the left and to the right in nodes. Lexicographic ordering is the numeric order in the first dimension in which time series differ (if any, else the shorter one is smaller). Separators are illustrated in Figure 10.1. The root node contains the one-dimensional time series separators 8.9 and 10.5, the left subtree's time series all begin with values smaller than 8.9, the time series to the right start with larger values. All elements in the middle subtree are time series which begin with a value between 8.9 and 10.5. As we can see from this small example, for continuous values, common prefixes are rare and separators are extremely short. Thus, there is only information on the very first dimension's value ranges. Comparing a query time series against short separators leads to very low distance values that are typically not sufficient for pruning.

The TS-tree uses quantization to group similar values. Mapping similar values to discrete symbols, common prefixes are more likely to occur. Consequently, separators are enlarged and pruning power is enhanced. Quantization is defined as any mapping of continuous time series values to time series

of discrete symbols:

Definition 10.2 ***Quantization.***

A time series $q = (q_1, \ldots, q_n)$ *is a quantization of a time series* $t = (t_1, \ldots, t_n)$ *with respect to a set of symbols* $\{s_1, \ldots, s_k\}$ *iff*

- *each symbol* s_j *represents an interval range*
  $s_j := [s_j^l \text{ to } s_j^u)$
- *symbol ranges are disjoint and adjacent*
  $s_j^u = s_{j+1}^l \; \forall j < k$
- *symbols range over the entire value range*
  $s_1^l = MINVALUE$ *and* $s_k^u = MAXVALUE$
- q *is a mapping of the values of* t *to covering symbol ranges*
  $q_i = s_j$ *iff* $s_j^l \leq t_i < s_j^u$

Quantization is thus a transformation of the continuous value range of the time series to relatively few symbolic representatives of value intervals. Different symbol quantization techniques may be used. The most common

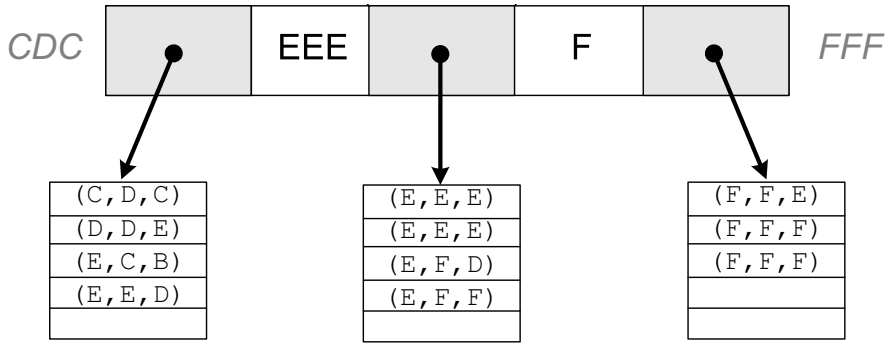


Figure 10.2: Quantized separators

ones are equiwidth and equidepth quantization. Equiwidth quantization divides the value range into intervals of equal length. Each of these intervals corresponds to one symbol. Equidepth quantization tries to create intervals with roughly the same number of entries each. SAX (symbolic approximate aggregation) uses the quantiles of the normal distribution [LKLC03].

Reconsider our separator example for quantization (Figure 10.2). Symbol ranges are A: 0-2, B: 2-4, C: 4-6, and so on. Mapping the bottom time series of the left subtree (8.67,8.18,8.72) to these symbols yields (E,E,D). To separate it from the first entry in the next subtree (8.90,10.10,7.85) $\rightarrow$ (E,E,E), more than the one dimension (8.9) is required (see also first example). The separator has to be extended to EEE, thus automatically providing information not only on the first time series dimension, but on three dimensions.

The TS-tree combines quantized separators with additional quantized meta data, i.e. upper and lower bounds per dimension as used in R-trees. Meta data $\mathbf{MD} = ((l_1, u_1), \ldots, (l_n, u_n))$ thus consists of lower bounds l_i, the smallest value in dimension i, and upper bounds u_i, the largest value in dimension i, of the respective subtree. These bounds provide additional information on the value ranges of the subtree for query processing, as discussed in Section 10.1. However, unlike R-trees, TS-trees do not use this meta data for splitting of nodes, but use separator split to ensure overlap-free subtrees.

The nodes in TS-trees thus store separators, meta data and pointers to subtrees. For compact nodes and splitting of more dimensions, separators are roughly quantized (cf. the experiments in Section 10.2). Additional detailed information for query processing is provided by finely quantized meta data and finely quantized leaf level information:

Definition 10.3 ***TS-tree inner node.***

An inner node of the TS-tree with branching factor m, rough quantization parameter r and fine quantization parameter f fulfills the following properties:

- *a inner node contains k entries ($m \leq k \leq 2m$) with k pointers to subtrees, $k-1$ separators and k meta data bounds.*
- *a root inner node contains k entries ($2 \leq k \leq 2m$) with k pointers to subtrees, $k-1$ separators and k meta data bounds.*
- *separators are roughly quantized to r symbols and prefix compressed, i.e. common prefixes are not materialized.*
- *meta data upper and lower bounds are finely quantized to f symbols.*

Definition 10.4 ***TS-tree leaf node.***

A leaf node of the TS-tree with branching factor n and fine quantization parameter f fulfills the following properties:

- *a leaf node contains i entries ($n \leq i \leq 2n$) with i pointers to time series.*
- *a root leaf node contains $1 \leq i \leq 2n$ entries.*
- *entries are finely quantized to f symbols.*

Thus, the TS-tree uses rough quantization for long separators on inner nodes while keeping more information through fine quantization of meta data and leaf node entries.

The combined meta data and separator descriptors are illustrated in Figure 10.3: meta data bounds are lower and upper bounds in each dimension.

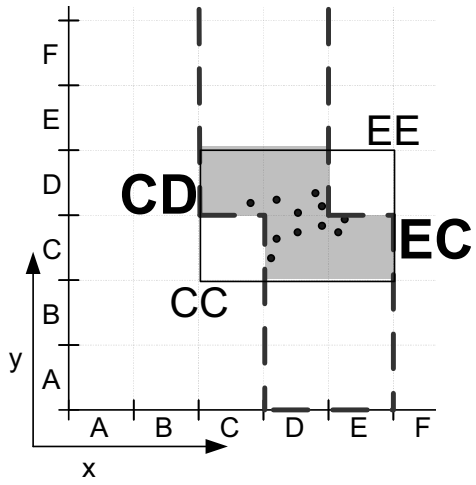


Figure 10.3: Descriptor in TS-trees geometrically

In this example, dimensions x and y both have lower bounds C and upper bounds E, corresponding geometrically to a (hyper-)rectangle. The separator information, CD to EC however corresponds to the area stretching to the end of the data range in the C interval of dimension x above D in dimension y, the entire data range in the D interval of dimension x and finally from the beginning of the data range in the E interval up to the C interval in

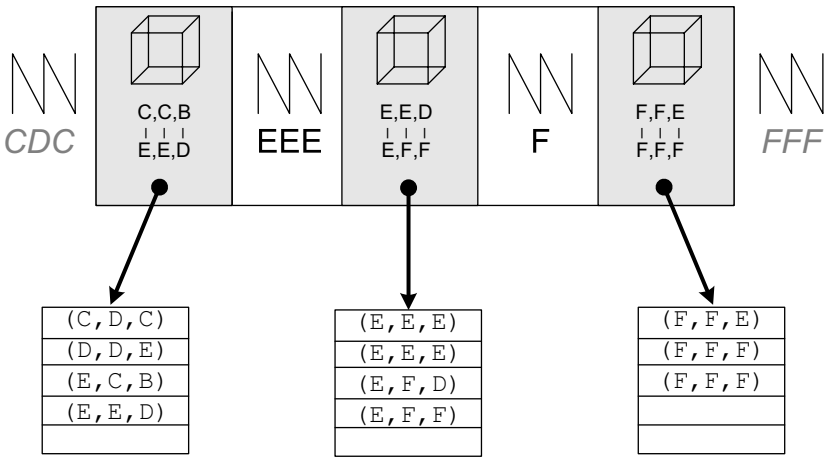


Figure 10.4: Structure of TS-trees

dimension y. The intersection of the two geometries, i.e. the indexed subtree range, is marked in gray.

The definition of TS-tree nodes is illustrated in Figure 10.4: nodes contain both quantized compressed separators as well as meta data information to describe the quantized time series in the subtrees.

Dimensionality reduction of time series is employed in TS-trees to ensure reasonable fanout. Typical time series are long, and dimensionality reduction limits storage requirements for time series. The basic idea is to represent the original time series by a shorter representation, keeping as much information as possible. Different approaches for generation of shorter representations may be used, e.g. PAA (piecewise aggregate approximation) a specialized approach for time series [KCPM01]. PAA replaces the original time series by piecewise averages of the time series values. The new time series length is thus the old length divided by the length of pieces. Another approach commonly used for time series is DWT (discrete wavelet transform). DWT with Haar wavelet basis builds successive averages and differences to these averages to aggregate the original dimensions. Alternatively, general purpose dimensionality reduction techniques like PCA (principal components analysis) may be used. PCA analyzes the statistical covariance in the dimensions. Using eigenvalue decomposition on the covariance matrix, the original dimensionality may be reduced to the first (with respect to their variance) of the resulting new dimensions. We analyze the performance of these three methods in the experiments. This concludes the formalization of the TS-tree. In the next section we discuss query processing based on the descriptor information.

10.1 Query processing in TS-trees

Comparing time series is usually done using the Euclidean distance or the Dynamic Time Warping Distance (DTW). While the former simply compares synchronous time series values, DTW allows for stretching and squeezing of the time series in the time dimension. In either case, k nearest neighbor processing in index structures requires computation of the *mindist*: at any node of the tree, the minimal distance between the query and the descriptor information in the node has to be calculated. This is crucial for pruning without loss of completeness of subtrees which are dissimilar and for ranking of potentially relevant nodes in k nearest neighbor search (or for discarding nodes exceeding the range threshold of range queries). We first give the *mindist* for TS-trees using Euclidean distance. The extension to DTW, which is based on the *mindist* for Euclidean distance, is discussed subsequently.

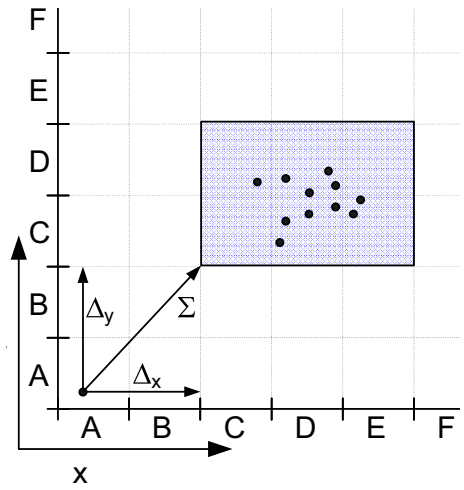


Figure 10.5: Mindist to meta data

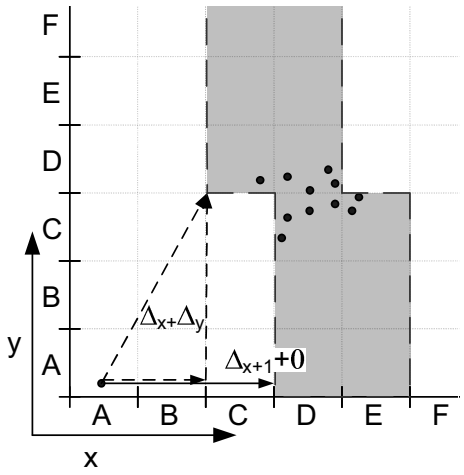


Figure 10.6: Mindist to separators

10.1.1 TS-tree *mindist*

During query processing in the TS-tree, the query $q = (q_1, \ldots, q_n)$ is compared against the descriptor information $\mathbf{D}$. The descriptor contains both separators and meta data $\mathbf{D} = (\mathbf{S}, \mathbf{MD})$, which both delimit the region indexed by the corresponding subtree. Consequently, the **S**-*mindist* to left and right separators $\mathbf{S_l}$ and $\mathbf{S_r}$, as well as the **MD**-*mindist* to meta data lower and upper bounds l and u, could be computed. We compute an even tighter overall *mindist* to the intersection of the two regions as illustrated in Figures 10.5 to 10.7. Figure 10.5 depicts the **MD**-*mindist* from query q, Figure 10.6 the **S**-*mindist*, and Figure 10.7 the overall *mindist*, respectively. As we can see, the overall *mindist* is neither the **MD**-*mindist* nor the **S**-*mindist*, but is actually located on the intersection of the regions.

The distance to upper and lower boundaries of the meta data can be computed in a straightforward manner as a sum over dimension wise differences:

Definition 10.5 MD-*mindist*.

The **MD**-*mindist between query* $q = (q_1, \ldots, q_n)$ *and meta data* $\mathbf{MD} =$

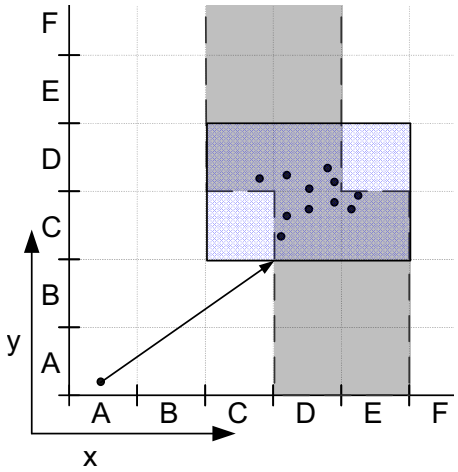


Figure 10.7: Mindist to intersection of meta data and separators

$((l_1, u_1), \ldots, (l_n, u_n))$ *is defined as:*

$$\mathbf{MD}\text{-}mindist(q, \mathbf{MD})^2 = \sum_n^{i=1} \begin{cases} (q_i - l_i)^2 & q_i < l_i \\ (q_i - u_i)^2 & u_i < q_i \\ 0 & else \end{cases}$$

This meta data *mindist* function is a lower bound to any time series in the corresponding subtree, as known from e.g. the *mindist* to MBRs in R-trees [Gut84]. The meta data is simply a dimension-wise information on lower and upper bounds, whereas the separator information is lexicographically ordered, which cannot be assessed in a dimension related manner. Consequently, *mindist* computation for these two descriptor types differs. Considering Figure 10.8, we see that the distance from the query to the separator is not the dimension wise addition of differences. Clearly, lexicographically, the query is smaller than the separator. Computing the distances dimension by dimension, the query would be considered larger than the separator in the second dimension. To compute the **S**-*mindist* correctly, we thus have to take previous dimension into account and cannot compute in a dimension

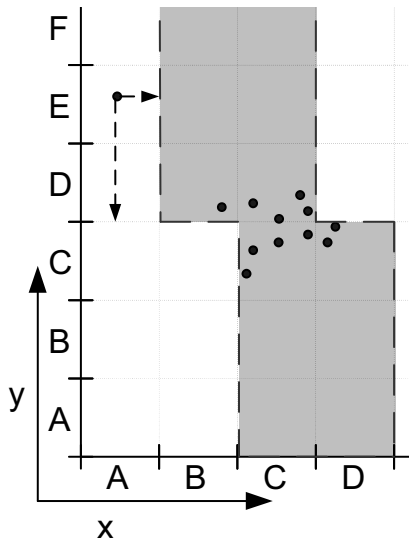


Figure 10.8: Lexicographic mindist computation

wise fashion.

As illustrated in Figure 10.6, the **S**-*mindist* is either the difference in the current dimension Δ_x plus the distance in the next dimension y or, if the separator in dimension y is larger than the query as is the case in this example, one step further in dimension x, i.e. $\Delta_x + 1$ to immediately reach the separator bounds.

Additionally, we have to take care not to exceed the right separator bounds. Figure 10.9 shows that the right separator in the next dimension may be actually smaller than the query. In this case, computing Δ_x+1 would yield a wrong result of the *mindist*. Summing up all of these cases, we give a recursive definition of the **S**-*mindist* where the query is smaller than the separator (cf. Figure 10.10). The **S**-*mindist* from the right is analogous.

Definition 10.6 S-*mindist*.

The **S**-*mindist between query* $q = (q_1, \ldots, q_n)$ *and left* $\mathbf{S_l} = (\mathbf{S_l}_1, \ldots, \mathbf{S_l}_n)$ *and right* $\mathbf{S_r} = (\mathbf{S_r}_1, \ldots, \mathbf{S_r}_n)$ *separators of a subtree is defined as the distance to*

the left separator if the query is smaller than the subtree or the distance to the right separator if it is larger. If the query is within the separator bounds, $\mathbf{S}$*-mindist is zero:*

$$\mathbf{S}\text{-}mindist(q,\mathbf{S}) = \begin{cases} \mathbf{S_l}\text{-}mindist(q_1,\mathbf{S}) & q \leq \mathbf{S_l} \quad (a) \\ \mathbf{S_r}\text{-}mindist(q_1,\mathbf{S}) & q \geq \mathbf{S_r} \quad (b) \\ 0 & else \quad (c) \end{cases}$$

where:

$$\mathbf{S_l}\text{-}mindist(q_i,\mathbf{S}) = \min \begin{cases} |q_i - \mathbf{S_l}_i| + \mathbf{S_l}\text{-}mindist(q_{i+1},\mathbf{S}) & (d) \\ |q_i - (\mathbf{S_l}_i + 1)| & (e) \end{cases}$$

$$\mathbf{S_r}\text{-}mindist(q_i,\mathbf{S}) = \min \begin{cases} |q_i - \mathbf{S_r}_i| + \mathbf{S_r}\text{-}mindist(q_{i+1},\mathbf{S}) & (d') \\ |q_i - (\mathbf{S_r}_i - 1)| & (e') \end{cases}$$

Note that in rare cases, the separator ends prematurely (cf. Figure 10.9) and the minimum is reduced to (d), (d'), i.e. (e) or (e') does not apply iff:

$$(e) \ : \ \mathbf{S_l}_i + 1 < \mathbf{S_r}_i \vee (\mathbf{S_l}_i + 1 \leq \mathbf{S_r}_i \wedge \mathbf{S_l}_{i+1} \leq \mathbf{S_r}_{i+1})$$
$$(e') : \ \mathbf{S_r}_i + 1 < \mathbf{S_l}_i \vee (\mathbf{S_r}_i + 1 \leq \mathbf{S_l}_i \wedge \mathbf{S_r}_{i+1} \leq \mathbf{S_l}_{i+1})$$

This separator *mindist* lower bounds the distance to any time series in the corresponding subtree, as in each recursive step, the closest possible time series on the separator bounds is used to compute the *mindist*. Thus, if the query is smaller than the left separator (a), the $\mathbf{S}$-*mindist* is the recursively defined left sided distance. Likewise, if the query is larger than the right separator (b), the analogous recursion from the right applies. If the query is inside the region delimited by the separators (c), the $\mathbf{S}$-*mindist* is clearly zero.

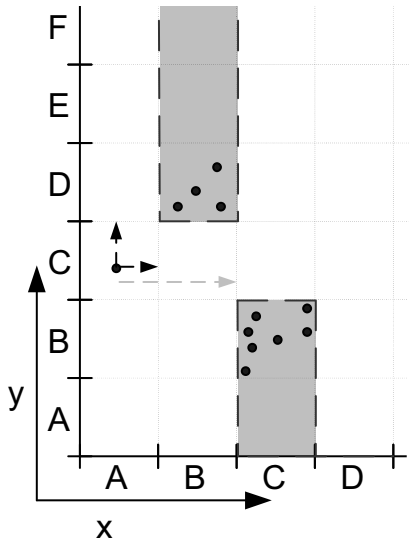


Figure 10.9: End of lexicographic separator

The recursive $\mathbf{S}_\mathrm{l}$-*mindist* takes two aspects into account: for one, as long as the separator bounds have not yet been reached, the difference in the current dimension may contribute to the overall distance and the same choice applies recursively for the next dimension (d). Second, as soon as the bounds are reached by going further an additional symbol, the remaining differences are zero (e).

(a) is a recursive choice denoted as $\mathbf{S}_\mathrm{l}$-$mindist(q_i, \mathbf{S})$. Moreover, care has to be taken, not to exceed the right separator. Likewise, (b) is the symmetric case of (a) for queries which are larger than the right separator. The $\mathbf{S}_\mathrm{l}$-*mindist* is then the minimum of these choices in reaching the separator boundaries.

Reconsidering Figure 10.7, we see that the overall *mindist* is based on both the **MD**-*mindist* and the **S**-*mindist*. In a recursive fashion, as for the **S**-*mindist*, we proceed until the separator bounds are reached. At this point, however, we may take the **MD**-*mindist* for the remaining dimensions into

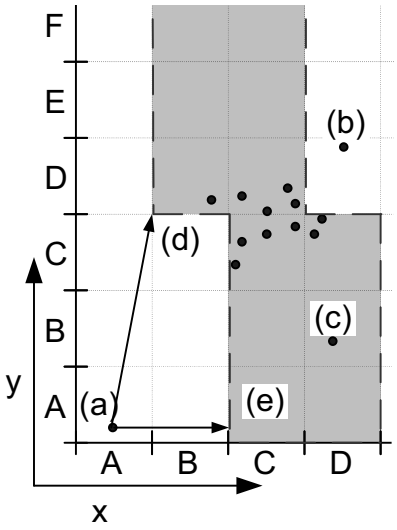


Figure 10.10: Lexicographic separator mindist

account to exploit the additional meta data information and reach a larger overall *mindist*. Note that a larger *mindist* is favorable as this allows for better pruning.

Definition 10.7 ***mindist.***

The mindist between query $q = (q_1, \ldots, q_n)$ *and descriptor* $\mathbf{D} = (\mathbf{MD}, \mathbf{S})$ *of a subtree is defined as:*

$$mindist(q, \mathbf{D}) = \begin{cases} l\text{-}mindist(q_1, \mathbf{D}) & q \leq \mathbf{S_l} \quad (a) \\ r\text{-}mindist(q_1, \mathbf{D}) & q \geq \mathbf{S_r} \quad (b) \\ 0 & else \quad (c) \end{cases}$$

where:

$$l\text{-}mindist(q_i, \mathbf{D}) = \min \begin{cases} |q_i - \mathbf{S_l}_i| + l\text{-}mindist(q_{i+1}, \mathbf{D}) & (d) \\ |q_i - (\mathbf{S_l}_i + 1)| + \mathbf{MD}\text{-}mindist((q_{i+1}, \ldots, q_n)) & (e) \end{cases}$$

$$r\text{-}mindist(q_i, \mathbf{D}) =$$
$$\min \begin{cases} |q_i - \mathbf{S}_{\mathbf{r}i}| + r\text{-}mindist(q_{i+1}, \mathbf{D}) & (d') \\ |q_i - (\mathbf{S}_{\mathbf{r}i} - 1)| + \mathbf{MD}\text{-}mindist((q_{i+1}, \ldots, q_n)) & (e') \end{cases}$$

as before, if the separator ends prematurely only (d) or (d') applies,

i.e. (e) or (e') does not apply iff:

$$(e) \ : \ \mathbf{S}_{\mathbf{l}i} + 1 < \mathbf{S}_{\mathbf{r}i} \vee (\mathbf{S}_{\mathbf{l}i} + 1 \leq \mathbf{S}_{\mathbf{r}i} \wedge \mathbf{S}_{\mathbf{l}i+1} \leq \mathbf{S}_{\mathbf{r}i+1})$$
$$(e') : \ \mathbf{S}_{\mathbf{r}i} + 1 < \mathbf{S}_{\mathbf{l}i} \vee (\mathbf{S}_{\mathbf{r}i} + 1 \leq \mathbf{S}_{\mathbf{l}i} \wedge \mathbf{S}_{\mathbf{r}i+1} \leq \mathbf{S}_{\mathbf{l}i+1})$$

Thus the overall *mindist* to the descriptors in TS-tree nodes is the distance to the intersection of left and right separators and the meta data. This *mindist* function constitutes a lower bound for the distance between any query q and any time series t in a subtree delimited by $\mathbf{D} = (\mathbf{MD}, \mathbf{S})$, i.e. $mindist(q, \mathbf{D}) \leq dist(q, t)$. Lower bounding follows from the derivation of the *mindist* in a straightforward manner. By taking the minimum of the distance to separator and meta data bounds in each dimension, the closest time series on the descriptor bounds is assumed. Note that this *mindist* function is also the largest such lower bound. Any larger function would violate the lower bounding property of the *mindist* since in each case of the recursive minimum computation, the bounds of separator or meta data may be reached, and thus potentially a time series in the respective subtree.

10.1.2 Extension to Dynamic Time Warping

The TS-tree is a flexible time series indexing structure that supports Euclidean distance, as seen before, as well as Dynamic Time Warping (DTW). Dynamic Time Warping allows for stretching and scaling of the time axis to detect slightly shifted or scaled patterns. An example is given in Figure 10.11: two time series are compared using the Euclidean Distance (left)

or Dynamic Time Warping (right). Horizontal lines indicate which values are matched by the respective distance functions. Both Euclidean distance and Dynamic Time Warping are commonly used for time series similarity search [Keo02]. We briefly review Dynamic Time Warping before detailing Dynamic Time Warping based query processing on TS-trees.

Dynamic Time Warping computes the best possible match between time series with respect to the overall warping cost. Typically, warping is restricted to some k-band neighborhood around a time series element to avoid degenerated matchings (e.g. all but one values of a time series are matched to a single element of the other). Formally, the definition of local k-band Dynamic Time Warping is:

Definition 10.8 ***k-band Dynamic Time Warping.***
The Dynamic Time Warping distance between two time series s, t with respect to a bandwidth k is defined as:

$$D^2_{DTW}(s,t) = D^2_{DTW}(band_k(s), band_k(t)) + \min \begin{cases} D^2_{DTW}(start(s), start(t)) \\ D^2_{DTW}(s, start(t)) \\ D^2_{DTW}(start(s), t) \end{cases}$$

with

$$D^2_{DTW}(band_k(s), band_k(t)) = \begin{cases} D^2_{DTW}(s_i, t_i) & |i-j| \leq k \\ \infty & else \end{cases}$$

Thus, Dynamic Time Warping is defined recursively on the minimal cost

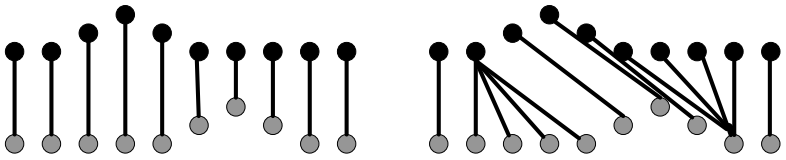


Figure 10.11: Euclidean Distance (left) and Dynamic Time Warping (right)

of possible matches of prefixes shorter by one element. There are three possibilities: either match prefixes of both s and t, or match s with the prefix of t, or vice versa. The difference between overall prefix lengths is restricted to a band of width k in the time dimension by setting the cost of all overstretched matches to infinity. Euclidean distance can be seen as a special case of Dynamic Time Warping with a band of width zero. Dynamic Time Warping can be computed via a dynamic programming algorithm in $O(mn)$ time and space, where m, n are the lengths of the time series. Using a k-band, this is reduced to $O(k * \max\{m, n\})$. For efficient query processing this is still a limiting factor. Indexing Dynamic Time Warping is possible by exploiting k-band restriction. Without this restriction, arbitrary stretching hinders pruning. Consequently, indexing k-Dynamic Time Warping using lower bounds has been proposed [Keo02, ZS03]. An "envelope" around the time series with respect to the k-band is defined. The squared Euclidean distance between values above or below the envelope of the other time series lower bound the actual k-Dynamic Time Warping distance. We review the closest existing lower bound:

$$DTW_{LB} = \sqrt{\sum_{i=1}^{n} \begin{cases} (s_i - U_i)^2 & s_i > U_i \\ (s_i - L_i)^2 & s_i < L_i \\ 0 & else \end{cases}}$$

$$\overline{U}_i = \frac{N}{n}(u_{\frac{n}{N}(i-1)+1} + \cdots + u_{\frac{n}{N}(i)}) \text{ and}$$
$$\overline{L}_i = \frac{N}{n}(l_{\frac{n}{N}(i-1)+1} + \cdots + l_{\frac{n}{N}(i)})$$

A detailed discussion of this bound can be found in [ZS03]. It is an extension of the original envelope for exact indexing of Dynamic Time Warping, LB_{Keogh} [Keo02].

As we can see from the definition of the lower bound, this approach is a dimension-wise computation of distances. Thus, the *mindist* function introduced in the previous section can be extended to Dynamic Time Warping in a straightforward manner by computing the *mindist* not to a single query time series, but to upper and lower envelope time series.

Lower bounds are useful as filters in multistep query processing to efficiently compute a set of candidates which is then refined using the original distance function.

10.1.3 Multistep query processing

Multistep query processing was introduced in the GEMINI framework (GEneric Multimedia object INdexIng, [Fal96]). In a first step, the set of candidates is generated using a filter distance on the index structure. These are then refined sequentially using the original distance (Figure 10.12).

To ensure completeness, filter distances must fulfill the lower bounding property. Dimensionality reduction and quantization as proposed above fullfil the lower bounding property for euclidean distances. Dynamic Time Warping band also fulfills the lower bounding property as proven in [ZS03].

The number of refinement steps is reduced to its minimum in KNOP (k nearest neighbor optimal query processing) [SK98] by a direct feedback loop between refinement and candidate generation: objects are ordered according

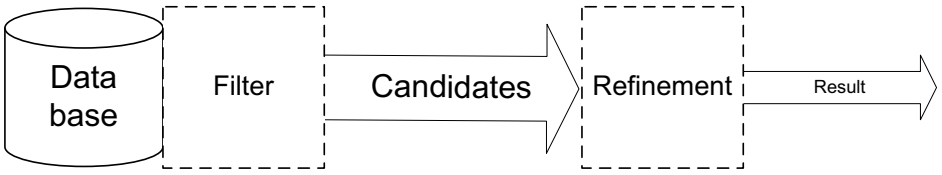


Figure 10.12: Multistep query processing

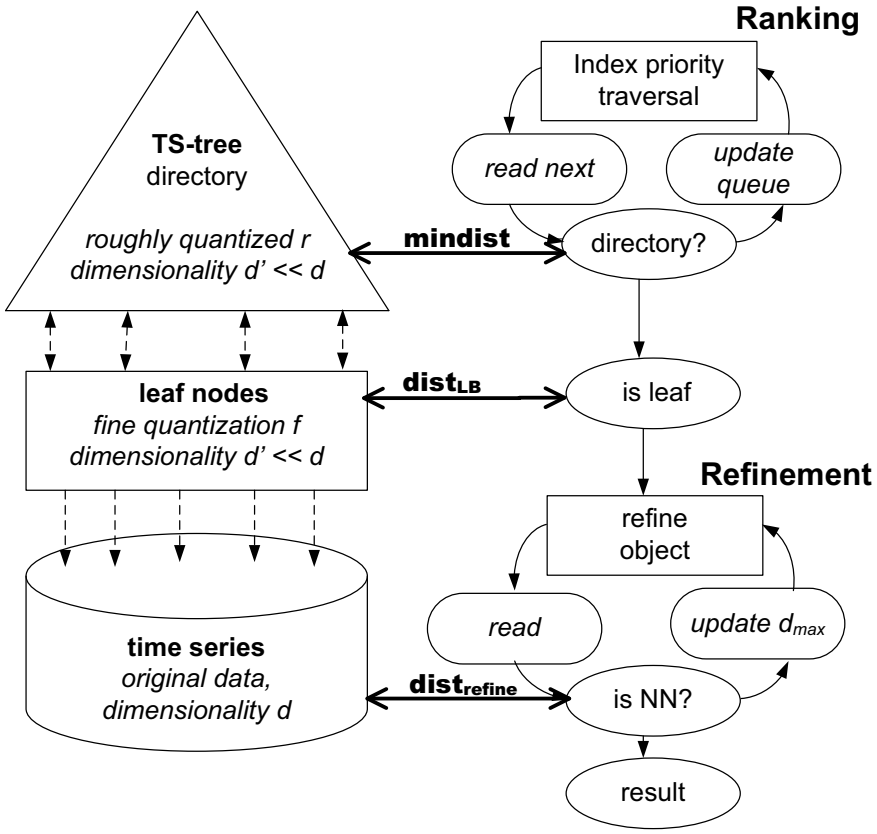


Figure 10.13: Optimal multistep query processing for time series

to their filter distance via a ranking query and refined until the filter distance is larger than the largest exact distance so far (Figure 10.13). Besides reduction of refinement computations, the KNOP approach is especially useful for our evaluation of different time series indexing structures: the number of time series which has to be refined using the original distance function is independent of the indexing structure used. This is due to the ranking query used on the index: since time series are processed in the same order regardless of the index, the number of time series which have to be refined is fixed. Thus, directory page accesses measure the performance of indexes for identical time series representations.

Theorem 10.1 ***Number of refinement computations.***

The number of refinement computations in KNOP query processing is constant for any index.

Proof.

Let q denote the query time series, t_i, t_j denote database time series with dimensionality reduced representations r_i, r_j, r_q, respectively. Then, as required in KNOP, filter distances underestimate the exact distance:

$$dist_f(r_i, r_q) \leq dist_e(t_i, q)$$

KNOP uses a ranking query, thus t_j is processed before t_i if its filter distance is smaller:

$$dist_f(r_i, r_q) \leq dist_f(r_j, r_q)$$

KNOP refines time series until the filter distance of the next time series t_n in the queue exceeds the kth nearest neighbor candidate refined so far:

$$dist_f(r_n) \geq d_{max}$$

with $d_{max} := \max\{dist_e(t_{c_1}), \ldots, dist_e(t_{c_k})\}$, where c_1 through c_k denote the current candidate k nearest neighbors. Thus, time series are only refined until the next reduced representation has a higher filter distance than the exact distance of the k best original representations. In summary, the set of time series refined Ref is

$$Ref := \{t_i \mid dist_f(t_i) \leq d_{max}\}$$

regardless of the underlying index structure.

Note that for different representations, e.g. quantizations or dimensionality reductions, the number of refinements may still vary.

Since the number of refinement steps given a certain time series representation cannot be reduced by any indexing structure, performance depends on the number of directory node accesses.

10.2 Experiments

The experiments are divided into three sections. First, we investigate the structure of TS-tree by measuring the average distances of the node entries and the capacity of the constructed nodes. The next section evaluates the performance of the TS-tree using Euclidean Distance based nearest neighbor queries on synthetic data followed by experiments on real world data sets. The last section evaluates Dynamic Time Warping (DTW) nearest neighbor queries.

Hierarchical indexing structures like R-trees or TS-trees grow very slowly in height when indexing more data. Thus very large data sets are used to evaluate indexing structures of appropriate height (four or five).

Data sets. Our experiments use the following data sets (two synthetic and three real world data sets):

1. **RW1**: A synthetic data set based on the random walk model one. The increments between two consecutive values are independent and identically distributed. We use a standard normal distribution $\mathcal{N}(0,1)$ for each increment: $t_{i+1} = \mathcal{N}(t_i, 1)$. We generated four different data sets each containing 250,000 time-series of length 256, 512, 1024 and 2048, respectively.

2. **RW2**: The second version of the random walk model uses independent but not identically distributed values for the increments. The increments of RW2 time series depend on the last increment: $t_{i+1} = \mathcal{N}(2t_i - t_{i-1}, 1)$. We also generated four different data sets each containing 250,000 time-series of length 256, 512, 1024 and 2048, respectively.

3. **Financial**: Historical time-series from

http://finance.yahoo.com. We use end of day stock quotes for more than 8,000 indices, corporate bonds, warrants etc. Overall the data set consists of 1,000,000 overlapping time series of length 256.

4. **Weather**: The weather data set contains 1,000,000 overlapping time series of length 128. The temperature data was retrieved from an interconnection of weather stations gathering agrarian meteorological data.

5. **EEG**: 128Hz-electroencephalographic data from the department of physiology (University of Bologna). Available from the UCR time series archive [Keo06]. We extracted 1,000,000 overlapping EEG time series of length 128.

The synthetic data experiments demonstrate the scalability of the TS-tree while the real world data illustrates the performance of the TS-tree in practical applications. The financial and random walk data set are mean and variance normalized [KK02]. These data sets are quantized using symbols based on the quantiles of the normal distribution (SAX [LKLC03]). The temperature and EEG data set is not normalized and thus indexed using equiwidth symbols. Disk page settings are 1kb and 4kb for synthetic and real world data sets, respectively. The temperature data set is very smooth while the financial data set contains some short changes. The last data set, the EEG data is bursty.

Experimental setup. We compare the TS-tree to the R*-tree [BKSS90], an extension of the R-tree [Gut84]. The R*-tree and R-tree both use minimum bounding rectangles (MBR) to describe the data contained in a subtree. We also investigated the performance of the R-tree with linear and quadratic split but as the R*-tree always outperformed the R-tree we only report results of the R*-tree.

The A-tree, a recent hierarchical indexing structure which approximates minimum bounding rectangles and data objects by quantized symbols, is also included in the experiments. Each node of the A-tree stores an exact representation of the subtree and quantized VBRs (virtual bounding rectangles) for subtrees. The A-tree originally uses $q = 6$ or $q = 12$ in its experimental evaluation. These quantizations would need bit shifting and masking which is very time consuming. We performed preliminary experiments with A-tree of $q = 8$ and $q = 16$, where the latter showed much worse performance due to lower fanout. We thus set $q = 8$, i.e. quantization with $2^8 = 256$ symbols.

The TS-tree is also set to 256 symbols for the leaf nodes. Where less symbols are used by TS-tree separators (this is a parameter we vary in the experiments) we still use one byte to store each symbol for simplicity.

The R*-tree implementation uses four bytes (floats) per dimension to store the continuous values of the minimum bounding rectangles.

All indexing structures (the R*-tree, the A-tree and the TS-tree) are implemented using Java 1.6. We use implementation invariant performance measure like the number of pages accessed or the size of the regions indexed by a subtree as well as the number of refinements necessary.

10.2.1 Evaluation of structural tree properties

This section studies the indexing structure of TS-trees, R*-trees and A-trees. We measure capacity and compactness of nodes.

Capacity. The first experiment investigates the average capacity of the index nodes. Figure 10.14 shows the storage capacity averaged for the data sets RW1 and RW2. Overall 250,000 time series of length 256 are stored by each index and reduced to dimensionality 16 to 32. In this experiment we use piecewise aggregate approximation to reduce the dimensionality.

Tree	Type	Dimension				
		16	20	24	28	32
R*-Tree	Non-Leaf	5.46	4.60	3.92	3.26	2.48
	Leaf	10.22	8.72	7.61	6.28	5.61
A-Tree	Non-Leaf	5.24	3.85	3.07	2.47	1.69
	Leaf	24.07	18.19	13.81	10.52	7.59
TS-Tree4	Non-Leaf	14,84	12.10	10.96	9.82	8.30
	Leaf	30,47	25.52	22.62	19.13	17.13
TS-Tree16	Non-Leaf	15.23	12.71	11.04	9.82	8.80
	Leaf	32.91	27.64	24.03	20.96	18.68
TS-Tree64	Non-Leaf	15.18	13.27	11.22	9.97	9.12
	Leaf	35.19	29.33	25.12	22.04	19.65

Figure 10.14: Capacity: average number of entries per node

Increasing resolution of the separators of TS-tree leads to longer separators, slightly reducing fanout. Consequently, using more symbols leads to better capacity results. Overall, the average capacity of all TS-trees inner nodes is much higher than that of the index structures. This is due to compact symbolic representation. The inner nodes of the A-tree do not show a significantly higher fanout than the nodes of the R*-tree. This is due to the storage overhead for exact minimum bounding rectangles and exact centroid per subtree in addition to quantized virtual bounding rectangles. The leaf nodes of the TS-trees also show the highest average storage capacity due to quantization and compression of common prefixes.

Compactness. The second experiment investigates compactness of subtrees. We measure the average width per indexed region, i.e. levelwise average distances of descriptors. We normalize the tree structures R*-tree, A-tree, and TS-tree to approximatively the same number of leaf nodes per

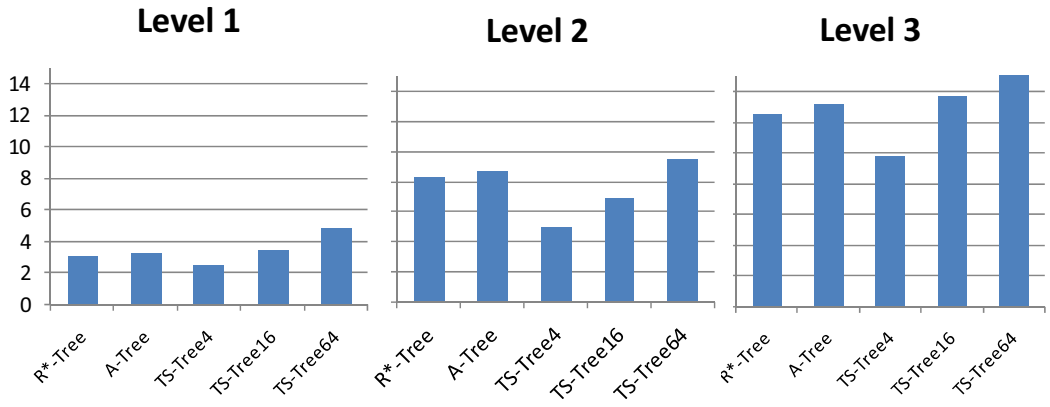


Figure 10.15: Compactness: average width per level

index in trees of height five. This normalization means that levelwise average widths are comparable across all three indexes. Figure 10.15 presents the result for the three inner levels of each index for the RW2 data set. "Level 1" denotes all nodes directly above leaves, "Level 2" two levels above leaves and "Level 3" three levels above leaves, respectively. The A-tree shows slightly higher average width than the R*-tree. This is due to the fact that the A-tree stores approximated virtual bounding rectangles, that are actually lower bounds of the exact minimum bounding rectangles. Both A-tree and R*-tree have greater width, which is probably due to overlap of large minimum bounding rectangle regions.

The compactness of the TS-tree depends on the resolution of the separators. As discussed before, rough quantization means that similar time series have the same quantization and are thus located in the same subtree. As we can see from the graphs, the roughest quantization (only 4 symbols) performs best. It is not only better than resolutions using 16 or 64 symbols, but is clearly more compact than R*-tree or A-tree. Small average width for

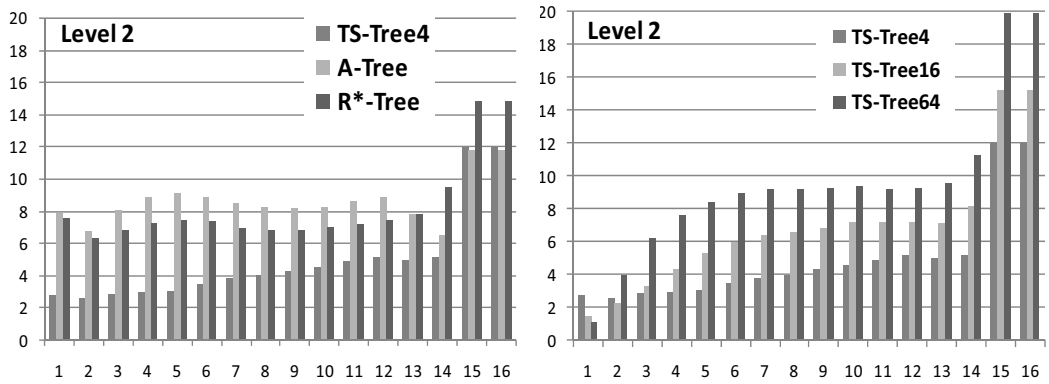


Figure 10.16: Compactness: average width per dimension

regions leads to better pruned during query processing, as each subtree is more clearly separated from the others.

Quantization. As claimed in Section 9.3.2, splitting of the most informative dimensions yields more compact descriptors. This is illustrated in a more detailed analysis of average width for individual dimensions in Figure 10.16 for "Level 2" nodes. The separator split in TS-trees splits dimensions successively. Thus, as can be seen in the left part of the figure, the TS-tree has much smaller average width in the first dimensions than in the later ones. In contrast to the TS-tree, both the A-tree's and the R*-tree's strategy to globally optimize the minimum bounding rectangles split falls behind. Even the average width of the TS-tree4's last dimension is lower than that of the R*-tree.

The right part of Figure 10.16 analyzes the effect of different separator resolutions for TS-trees. As discussed before, rough quantization indeed leads to longer separators, i.e. later dimensions are split as well. The TS-tree with 4 symbols shows lower average width for later dimensions, and slightly higher average width for the first dimensions. As seen in the previous experiment

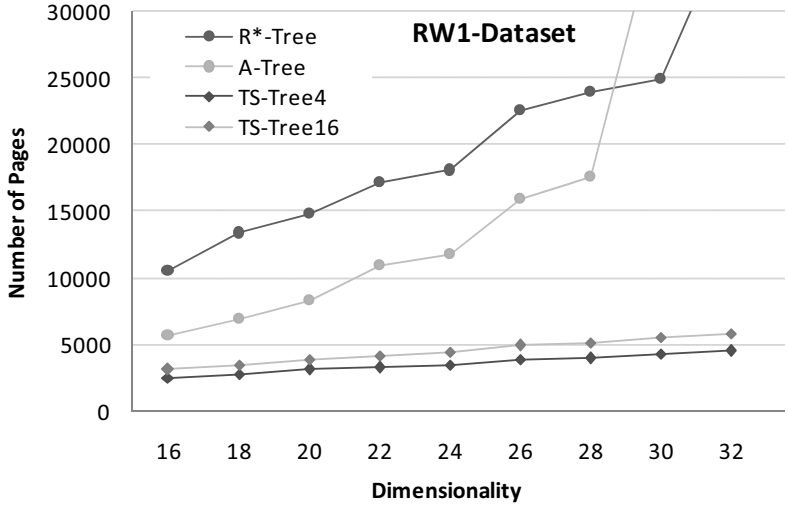


Figure 10.17: Number of page accesses for RW1 time series

in Figure 10.15, TS-tree4 performs best overall. As the TS-tree benefits from using less symbols for separators, we report only TS-trees with 4 or 16 symbols for separators in the following.

10.2.2 L_2 norm similarity search

In this section we investigate the scalability and the query performance of the TS-tree on synthetic and real world data sets using the L_2 norm, i.e. Euclidean distance.

Scalability with respect to dimensionality. To analyze scalability, we report the number of pages read averaged over 25 nearest neighbor queries. For each query we count both index and data pages. The more the dimensionality of a time series data set is reduced the more time series typically have to be refined in multistep query processing. Thus scalability in terms of dimensionality is very important for indexing time series.

Figures 10.17 and 10.18 illustrate the results for the RW1 and RW2 data

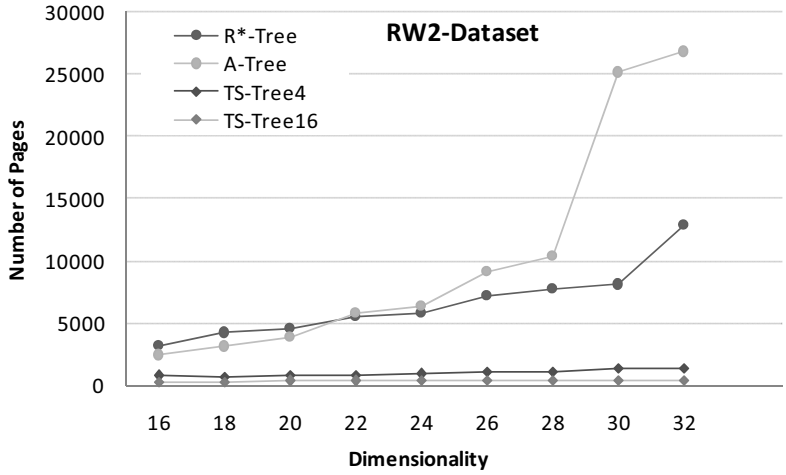


Figure 10.18: Number of page accesses for RW2 time series

sets, respectively, for time series length 256, reduced to dimensionality 16 to 32 using piecewise aggregate approximation. The TS-tree scales very well for both data sets. This is mainly due to the fact that its overlap-free separator split is not impaired by effects of high dimensional data spaces ("curse of dimensionality"). The R*-tree and A-tree, however, fall prey to the curse of dimensionality and degrade with increasing dimensionality (cf. also [BKSS90, BKK96, BBK$^+$00]). For the RW2 data set the TS-tree is only barely influenced by the dimensionality of the data. This is due to decreasing number of refinement steps. For 16 dimensions 8% of the pages read are refinement steps while only 4% are refinement steps if 32 dimensions are indexed. Thus, more dimensions require less refinement page reads whereas fewer dimensions require less directory page reads. Overall, the TS-tree outperforms both A-tree and R*-tree by nearly one order of magnitude.

Scalability with respect to time series length. We generated time series of different length (256, 512, 1024) and reduced them to 16 dimensions. As before, the TS-tree outperforms the R*-tree and A-tree (see Figure 10.19).

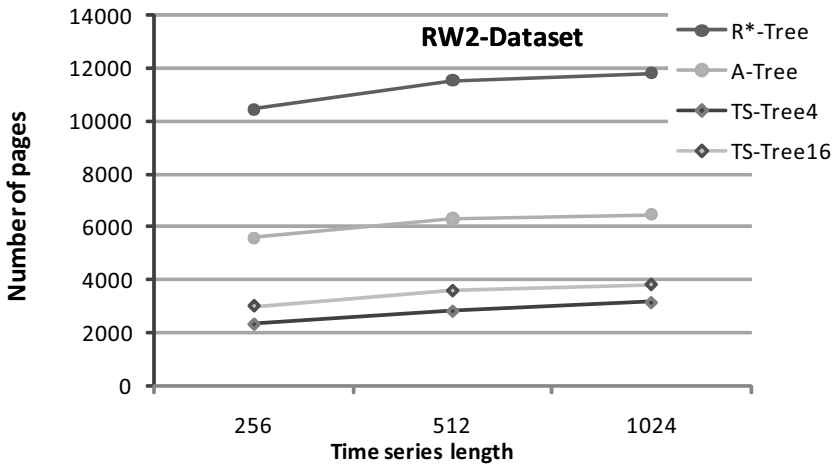


Figure 10.19: Number of page accesses for RW1 time series of varying length

With increasing length the number of refinement steps increases which influences all index structures in the same way, as proven in Section 10.1.3. As shown in the previous experiment, TS-trees may compensate this effect by indexing more dimensions.

Performance of dimensionality reduction techniques. We investigate three dimensionality reduction techniques: piecewise aggregate approximation (PAA), discrete wavelet transform with Haar basis (DWT) and principal component analysis (PCA). As mentioned before, principal components analysis orders new (reduced) dimensions in descending order of their variance. Discrete wavelet transform orders the dimensions according to their level of detail. Since TS-trees split the dimensions according to their order, it benefits from those dimensionality reduction techniques that provide an inherent ordering. Figure 10.20 shows the number of pages read, depicting data pages (lower bar), and index pages (upper bar) for the RW1 data set. As we can see, the R*-tree fails to capture the inherent dimensionality order. While the A-tree and TS-tree benefit from the ordering, the R*-tree actually

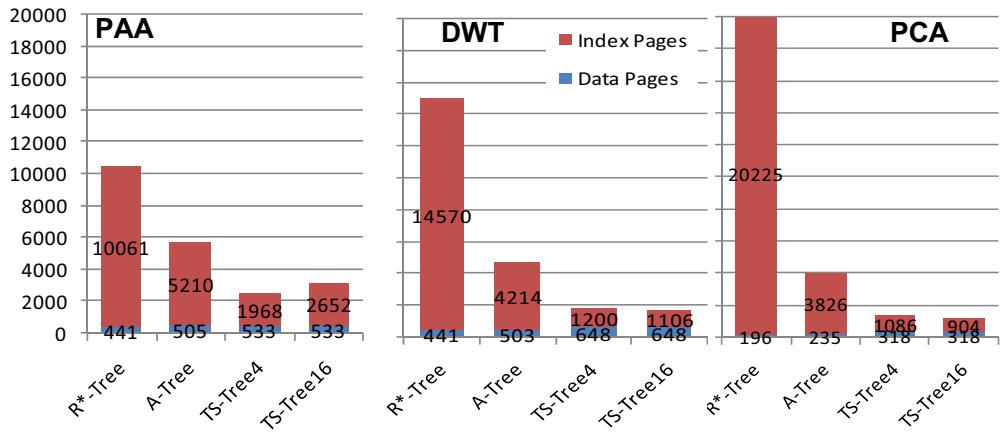


Figure 10.20: Page access vs. dimensionality reduction

degrades.

Note that the number of refinement steps is identical in discrete wavelet transform and piecewise aggregate approximation as both reduce to the same details in time series. However, the ordering is different, which results in different splits during index construction. Although the TS-tree accesses more pages for refinement than the other two indexes, it always performs more than twice as good as the A-tree, due to much more compact index pages. Using 16 instead of 4 symbols further reduces page access in TS-trees using discrete wavelet transform or principal components analysis. One reason for this is rough quantization. Using only four symbols, the first dimensions are split less often. As these first dimensions carry more information in principal components analysis or discrete wavelet transform, using more symbols makes better use of the inherent ordering by focusing split to the first dimensions. Thus, we use 16 symbols for the separators in the following.

Performance on real world data. We investigate the performance of the TS-tree on three real world data sets.

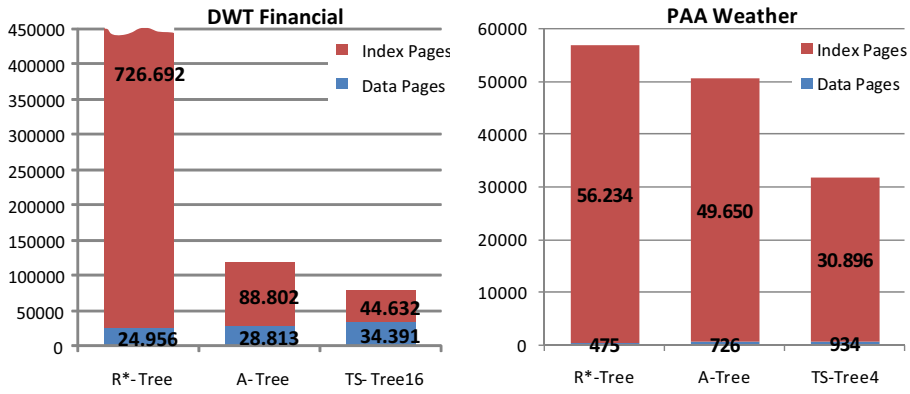


Figure 10.21: Page access for financial and weather data

Performance on smooth time series. The left part of Figure 10.21 shows the result for the financial data reduced to 16 dimensions using discrete wavelet transform. As discrete wavelet transform shows much better results than piecewise aggregate approximation we only report the result for discrete wavelet transform. Like before the R*-tree reads many index pages. Again the TS-tree needs more refinement steps than the A-tree but only half the number of index pages (the A-tree needs 3,552 and the TS-tree 1,785). The right part illustrates results for the time series weather data. Since the weather time series are very smooth, piecewise aggregate approximation is sufficient to reduce the dimensionality. Dimensionality reduction techniques like piecewise aggregate approximation capture the main characteristics of smooth time series very well. Hence, for the weather data set only a few refinement steps are necessary. The TS-tree again clearly outperforms the other two index structures.

Performance on bursty time series. Next, we analyze the query performance on bursty time series in the EEG data set. Bursts are typically smoothed away by dimensionality reduction techniques and more refinement

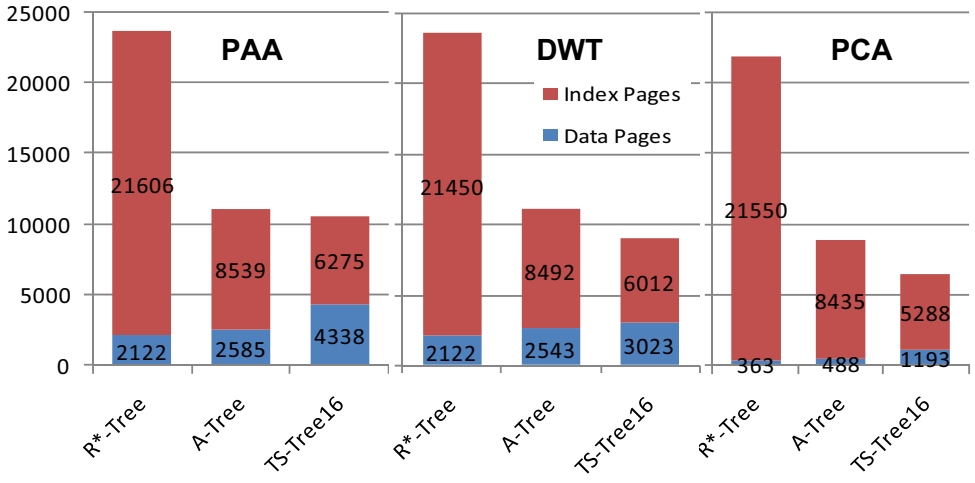


Figure 10.22: Number of pages accessed for the EEG data

steps are necessary for bursty data. As discussed, the number of refinement steps can be reduced by increasing the number of indexed dimensions. For the EEG data we obtained the best result by indexing 32 dimensions. Figure 10.22 shows the result for the number of refinement and index pages read for all three dimensionality reduction techniques. As already evaluated in the last section, the R*-tree slows down as more dimensions are indexed. For the EEG data principal components analysis is able to reduce the number of refinement steps by more than a factor of two. Considering the number of index pages read, the TS-tree also shows better results than the other index structures. Especially for the principal components analysis reduced data set, the TS-tree again reduces the number of index pages read.

10.2.3 Dynamic Time Warping similarity search

In this section we evaluate the applicability of Dynamic Time Warping queries using hierarchical index structures. In [ZS03] many linear transformations

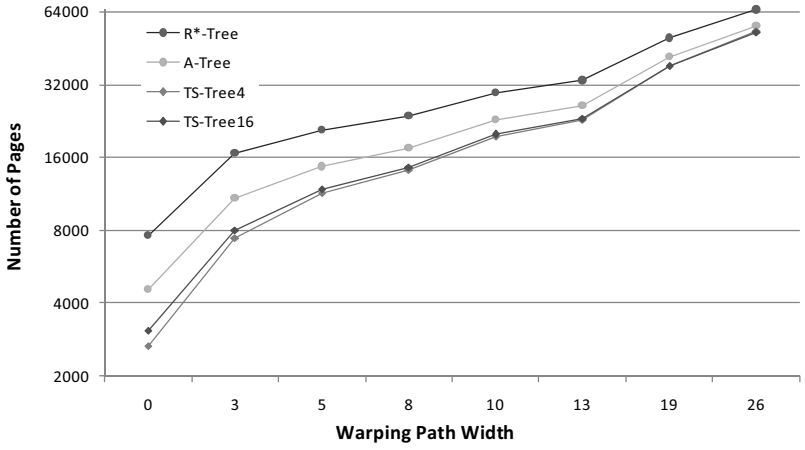


Figure 10.23: Pages access vs. warping width

have been tested but piecewise aggregate approximation has been shown to be the best dimensionality reduction technique for Dynamic Time Warping queries. Thus we use piecewise aggregate approximation to reduce the dimensionality of the data with the lower bound presented in Section 10.1.3.

The first experiment evaluates the influence of different Dynamic Time Warping band width on the query performance (Figure 10.23). For this experiment we use the RW1 data set. As the increments of the time series are independent in RW1 data, many time series have to be refined as the warping width is increased. Hence the total number disk pages read quickly increases with increasing warping width. As the number of refinement steps dominate the number of index pages read, the advantage of using the TS-tree is reduced for larger warping widths. For a small warping path the TS-tree clearly outperforms the R*-tree and A-tree and even with a larger warping width the TS-tree is still more efficient the the other two index structures.

The last experiment investigates the query performance of Dynamic Time Warping time series queries on real world data. We use a warping width of

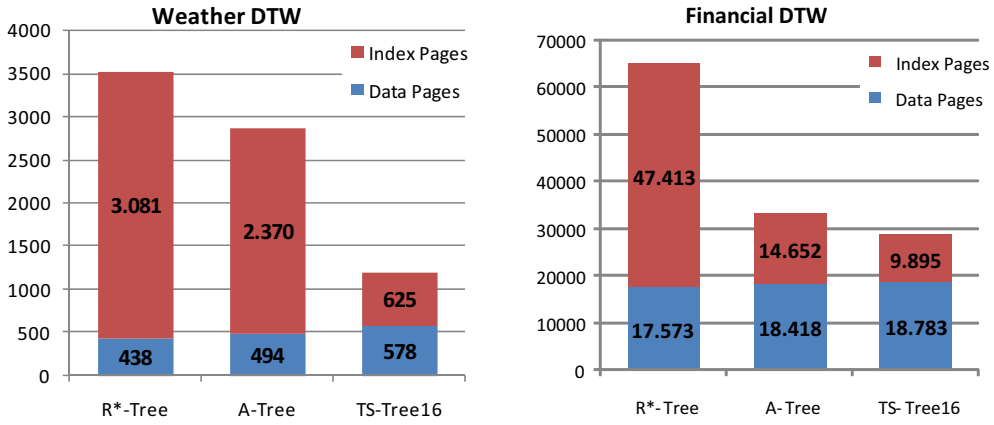


Figure 10.24: Page accesses for Dynamic Time Warping

3 and 5 for the weather and financial data set, respectively. For the weather data set, 32 dimensions are indexed and for the financial data set, 64 dimensions are necessary to reduce the number of refinement steps to a reasonable number. Figure 10.24 illustrates the result for both data sets. Again, the TS-tree works faster for both data sets. The TS-tree clearly outperforms both other index structures using Dynamic Time Warping queries, especially for the weather data set.

10.3 Conclusion

We demonstrated the importance of overlap-free tight bounding regions in hierarchical indexing structures for efficient time series query processing. We proposed the TS-tree which exploits the inherent properties of time series for compact descriptors and overlap-free indexing. Descriptors provide superior pruning power through quantization of separators. Quantization leads to automatic aggregation of similar time series, longer separators and better pruning power. Our *mindist* takes both separator and meta data into ac-

count to prune many index nodes as demonstrated in the experiments. The TS-tree clearly outperforms existing tree structures like the R*-tree and the quantization based A-tree.

Part III

Mining histogram data

Chapter 11

Subspace clustering

Clustering is a data mining task for summarizing data such that similar objects are grouped together while dissimilar ones are separated. In high-dimensional data, clusters are typically concealed by irrelevant attributes and do not show across the full space.

Subspace clustering mines clusters hidden in subspaces of high-dimensional data sets. Density-based approaches have been shown to successfully mine clusters of arbitrary shape even in the presence of noise in full space clustering. Direct extensions of density-based approaches to subspace clustering, however, entail runtimes which are infeasible for large high-dimensional data sets, since the number of possible subspace projections is exponential in the dimensionality of the data. Consequently, existing subspace clustering algorithms trade-off efficiency for accuracy. Grid-based space discretization or heuristics render subspace clustering efficient, yet these approaches lose clusters which were cut apart by the grid or were wrongfully pruned.

We propose lossless efficient detection of subspace clusters via a new density-conserving grid data structure which benefits from efficiency gains without losing any clusters. By detecting clusters in a depth-first manner,

our algorithm avoids excessive candidate generation found in existing apriori-based subspace clustering algorithms. In thorough experiments on synthetic and real world data sets, we demonstrate that our lossless approach yields accurate clusters and is faster than existing subspace clustering algorithms by orders of magnitude.

11.1 Introduction

Subspace clustering aims at automatically detecting clusters and their relevant attribute projections. The idea is to mine clusters in all of the possible subspaces of the high-dimensional data space. As the number of possible subspaces is exponential in the number of dimensions, this is a computationally challenging task.

In traditional full-space clustering, density-based approaches have shown to successfully mine clusters even in the presence of noise. They define clusters as dense areas separated by sparsely populated areas and have been extended to subspace clustering in previous work.

As naive re-running of full-space clustering algorithms for the exponentially number of subspace projections is usually infeasible, grid-based discretization of the space or other lossy approximations have been proposed. Whereas these algorithms show far better runtimes, they lose clusters which are cut apart by the grid.

Many subspace clustering algorithms are based on the apriori principle: assuming monotonicity of clusters with respect to the dimensionality of the subspace, higher dimensional projections of sparse regions are discarded as they are sparse regions as well. Apriori-based algorithms suffer from several drawbacks: first, they have to generate the entire set of one-

and two-dimensional subspace clusters before any pruning can be performed. Second, they require cluster monotonicity which ignores the effect of decreasing average densities for growing dimensionality of subspace clusters. Thus monotonous density thresholds detect either virtually no high-dimensional subspace clusters, or excessive low-dimensional subspace clusters. And finally, many redundant subspace clusters are reported: for all subspace clusters, all of their lower dimensional projections are included in the result set. Whereas this could be filtered out in a post-processing step, apriori-based algorithms cannot prune redundant subspace clusters in process, resulting in higher runtimes.

In this part, we propose a new concept for overcoming the existing trade-off between accuracy and efficiency. We present a subspace clustering algorithm which mines subspace clusters of any dimensionality by a novel efficient lossless algorithm. Our EDSC (Efficient Density-based Subspace Clustering) incorporates:

- **Accuracy**: lossless non-redundant density-based subspace clustering in arbitrary dimensionalities
- **Efficiency**: efficient depth-first subspace clustering algorithm

We review related work on subspace clustering in Section 11.2, before defining non-redundant density-based subspace clusters that take the effect of different subspace dimensionalities into account (Section 12). A novel lossless density-conserving grid is proposed which is indexed for depth-first mining, thus avoiding apriori-based candidate generation (Section 12.1). Our multistep algorithm quickly generates potential subspace clusters without false dismissals, i.e. completeness is guaranteed. We analyze our algorithm on both synthetic and real world data sets in the experiments in Section 12.2.

We demonstrate far better runtimes than existing subspace clustering approaches. Moreover, our efficient density-based subspace clustering approach clearly outperforms existing algorithms in terms of accuracy.

11.2 Related work

Full space density-based algorithms like DBSCAN (Density-Based Spatial Clustering of Applications with Noise) are capable of detecting arbitrarily shaped clusters even in noisy data [EKSX96, HK98, HK99]. These full space clustering algorithms do not scale to high-dimensional spaces. They suffer from the "curse of dimensionality", i.e. distances grow increasingly similar with increasing dimensionality and clusters can no longer be detected [BGRS99].

Dimensionality reduction aims at discarding irrelevant dimensions [Jol86]. In many practical applications, however, no globally irrelevant dimensions exist, but only locally irrelevant dimensions for each cluster are observed.

Projected clustering computes a partition into projected clusterings. Overlapping clusters in different projections cannot be detected [AY00, MSE06].

Subspace clustering mines clusters in arbitrary, possibly overlapping, subspaces. As the number of subspaces is exponential in the number of dimensions, existing algorithms trade-off efficiency for accuracy.

Grid-based approaches discretize the search space and typically mine the space in a bottom-up apriori-based algorithm [AGGR98]. Grids greatly reduce the computational complexity, yet clusters which spread across cells are missed and results are sensitive to the position of the grid. A general problem of these apriori-based algorithms is generation of huge numbers of candidates in low dimensional subspaces.

SUBCLU (SUBspace CLUstering) extends DBSCAN to subspace clustering via an apriori-based algorithm [KKK04]. Runtimes are better than for naive re-runs of DBSCAN, yet still not feasible for practical applications. Moreover, fixed thresholds either lose high-dimensional clusters or generate excessive low-dimensional subspace clusters.

SCHISM (Support and Chernoff-Hoeffding bound-based Interesting Subspace Miner) extends grid-based subspace clustering using a variable threshold adapted to the dimensionality of subspaces [SZ04a]. As the apriori property does not hold for variable thresholds, heuristics and a grid-based discretization are used for pruning. Consequently, clusters are lost as well.

FIRES (FIlter REfinement Subspace clustering) is another approximative approach which uses one-dimensional clusters only to estimate a set of potential high-dimensional subspace cluster candidates [KKRW05]. As this set is approximative, actual subspace clusters are lost.

RIS (Ranking Interesting Subspaces) only searches subspaces which might potentially contain clusters using a heuristic adaptation to the dimensionality of the subspace [KKKW03]. Clusters are mined in a second step using any traditional clustering algorithm. As there is no connection between these two steps, repeated computation of core properties of clusters results in high runtimes.

Chapter 12

EDSC: Efficient Density-based Subspace Clustering

In this chapter, we propose a novel efficient algorithm for mining of density-based subspace clusters of any dimensionality.

For notational convenience, let us assume the following terminology: the database DB contains d attributes $A_i \in [0, v]$ with N object features represented as histograms $o = (o_1 \dots o_d)$. A s-dimensional subspace is denoted by $S = \{k_1, ..., k_s\} \subseteq D = \{1, ..., d\}$. A projection of o to S is $o^S = (o_{k_1}, ..., o_{k_s})$. The distance between two objects o and p in subspace S is calculated by restricting the calculation to the respective dimensions of the subspace: $dist^S(o, p) = \sqrt{\sum_{i \in S}(o_i - p_i)^2}$.

In density-based clustering, *clusters* are defined as dense areas separated by sparsely populated areas. Density is evaluated for each object. In the original DBSCAN approach [EKSX96], objects are dense *core objects* if their neighborhood, specified by an ε-range around the object, contains more than

the threshold *minPoints* many objects. Chains of *density-connected* objects form the actual clusters. Consequently, arbitrarily shaped clusters can be successfully detected even in noisy settings [EKSX96]:

Definition 12.1 ***Density and Connectivity.***

An object o is ***dense*** *if at least minPoints objects are in its ε-neighborhood $N_\varepsilon(o) = \{p \in DB \mid dist(o,p) \leq \varepsilon\}$:*

$$o \text{ dense} : \Leftrightarrow |N_\varepsilon(o)| \geq minPoints$$

Two dense objects o and p are ***density-connected*** *if there is a chain of dense neighboring objects between them:*

$$\exists\, q_1 \ldots q_n \in DB : q_1 = p, q_n = o$$

$$\wedge\ \forall\ i = 1 \ldots n-1 :\ dense(q_i) \wedge\ dist(q_i, q_{i+1}) \leq \varepsilon.$$

Density-connected objects are thus elements of chains of objects which are mutually included in one another‘s neighborhoods.

SUBCLU extends the DBSCAN model to subspace clustering in a straightforward manner [KKK04]. In each subspace, clusters must exceed the same, fixed *minPoints* threshold according to the same ε neighborhood. This is useful for apriori-based pruning, as monotonicity on density holds.

However, with increasing dimensionality, distances grow and typical densities drop. In high dimensional subspaces, the expected density is smaller than in low dimensional spaces. Consequently, large thresholds are almost never exceeded in high dimensional spaces, resulting in cluster loss. On the other hand, small thresholds that find high dimensional clusters produce tremendous amounts of trivial low dimensional subspace clusters.

Hence, a constant threshold for all dimensionalities does not provide the means to compare subspace clusters of different dimensionalities and therefore it is virtually impossible to set up accurately. This effect is also discussed in [KKKW03, SZ04a]. Adapting to the changes in expected density in different dimensionalities increases the accuracy of the model. However, complexity increases and apriori-based pruning is no longer possible. These approaches therefore revert to heuristics to approximate subspace clusters.

We define subspace clusters with respect to the expected density of subspaces to ensure that the result is comparable across subspaces of different dimensionality. The expected density without a priori knowledge on the data distribution, i.e. assuming uniformly distributed data, is simply the average number of objects in the ε-neighborhood with respect to the dimensionality. This can be computed as the ratio of the volume of the ε-sphere to the volume of the subspace.

Definition 12.2 ***Normalized Density.***

An object o is ***dense*** *in subspace S of dimensionality s if its ε neighborhood in S, $N_\varepsilon^S(o) = \{p \in DB \mid dist^S(o,p) \leq \varepsilon\}$, contains more than $minPoints$ objects and exceeds the expected density in this subspace by at least a factor F:*

o *dense in* $S :\Leftrightarrow |N_\varepsilon^S(o)| \geq max\{minPoints, F \cdot expDen(s)\}$

with the expected density $expDen(s)$ in the s-dimensional subspace $expDen(s) = N \cdot \frac{c_s \cdot \varepsilon^s}{v^s}$, *and c_s the volume of the unit sphere in the s-dimensional subspace* $c_s = \frac{\pi^{\frac{s}{2}}}{\Gamma(\frac{s}{2}+1)}$ *defined via* $\Gamma(n+1) = n \cdot \Gamma(n), \Gamma(1) = 1, \Gamma(\frac{1}{2}) = \sqrt{\pi}$ *the Gamma function.*

Density in subspaces is thus normalized with respect to the dimensionality by adapting the threshold to the expected density. This ensures that subspace clusters of different dimensionalities are comparable and thus can be detected using a fixed threshold factor F.

Another effect to be taken into consideration in subspace clustering is the huge output generated by most algorithms. Subspace cluster projections to lower dimensional subspaces are likely to be subspace clusters as well. Removing these redundant projections is crucial in constraining the output to reasonable sizes. Moreover, meaningful subspace clusters should contain a minimum number of objects, which we model via a parameter $minSize$. Summarizing these requirements, density-based subspace clusters are defined as follows:

Definition 12.3 ***Subspace Cluster.***

A set of objects $C \subseteq DB$ in subspace $S \subseteq D$ with $|C| > minSize$ is a subspace cluster if:

- C ***density-connected****: $\forall o, p \in C$:*
  o^S, p^S density-connected with respect to normalized density (Definition 12.2).
- C ***maximal****: $\forall o, p \in DB$:*
  $o \in C$ and o^S, p^S density-connected $\Rightarrow p \in C$.
- C ***non-redundant****:*
  there is no higher-dimensional cluster containing objects in C.

Thus, subspace clusters are sets of density-connected objects where dimensionality of the subspace is taken into account for density estimation. These sets are maximal, i.e. all density-connected objects are assigned to the same subspace cluster. Additionally, we do not consider one-dimensional clusters

as they are typically uninteresting for the user. These clusters reflect merely the distribution of individual attributes which is usually known or analyzed using standard analytic methods. Hence, clusters of just one dimension, clusters containing less than $minSize$ objects or which are redundant projections are excluded from the result set.

12.1 Depth-first multistep algorithm

Subspace clustering is a highly complex task. The number of possible subspaces is exponential in the number of attributes. Hence pruning the search space is crucial. Existing subspace clustering algorithms use an apriori-based approach to prune possible subspace clusters. These algorithms in general use a breadth first search to identify sparse regions in low dimensional spaces. Those regions are then used to exclude higher-dimensional projections of these sparse regions from search. As discussed before, to identify higher-dimensional clusters the density threshold has to be decreased with respect to the expected density in higher dimensions. Thus in practical settings nearly no low-dimensional subspace projection can be pruned as this threshold is almost always exceeded. Consequently, in breadth-first traversal, nearly all low-dimensional subspaces are generated as candidate sets. This effect causes an extreme degeneration of the runtime behavior as the complete candidate sets of lower dimensions have to be available to apply the apriori property. Further on, redundancy pruning is not possible as information about higher-dimensional clusters is only available after all low-dimensional projections have been processed.

To the best of our knowledge, we propose the first algorithm for subspace clustering that proceeds in depth-first order. A cluster mined in a low dimen-

sional space is extended to the highest possible projection before the next low dimensional subspace cluster is analyzed. As not all clusters in all projections of the same dimensionality have to be determined simultaneously our algorithm overcomes the huge memory need of existing subspace clustering algorithms. Moreover, the depth-first approach makes redundancy pruning possible since information about high dimensional clusters is available before the next low dimensional subspace cluster is investigated.

A time consuming task in density-based subspace clustering is to identify density connected regions. As described in the last section, a neighborhood query is required to determine if an object is dense. Indexing according to all possible subspaces is not practicable, thus a sequential scan is necessary to determine the number of objects in an ε-neighborhood. Consequently, the complexity for clustering a subspace region is quadratic in the number of data objects contained in that region [EKSX96]. The number of subspaces which has to be investigated is typically very high and, consequently, the number of regions which have to be investigated is huge. Thus the computational cost of density-based clustering is problematic for large high dimensional data sets.

We reduce the computational cost by a multistep algorithm for subspace clustering with a filter step which efficiently determines candidate regions, i.e. regions that potentially contain density connected clusters. Multistep query processing uses approximations to efficiently determine candidate sets. By ensuring that each approximative step is optimistic, no true result is dropped. This guarantees completeness of the result. In the last step, candidate sets are refined to obtain the final result. Thus, the accuracy is preserved [FRM94, SK98].

We propose a filter step for subspace clustering based on a novel density

conserving grid. Traditional grid based clustering methods lose accuracy as the result is influenced by the position and resolution of the grid [Sil86]. Moreover, dense regions might be cut apart. Our novel density conserving grid does not lose any density connected cluster and still reduces the number of necessary scans for density connected regions. Hence, the good runtime performance of grid based algorithms is preserved.

In our multistep depth-first search we combine the following three methods to an Efficient Density-based Subspace Clustering (EDSC) algorithm:

(1) a density conserving grid which ensures completeness

(2) two effective filters for pruning the search space

(3) an in-process pruning of redundant subspace clusters

The rest of this section is organized as follows: we first give the definition of a density conserving grid before we prove monotonicity properties for pruning the search space. We then propose the EDSC algorithm which based on the density conserving grid efficiently mines density connected subspace clusters.

12.1.1 Density conserving grid

Density connected clusters might spread across multiple grid cells. To detect these clusters, we introduce a novel density conserving grid by devising connectivity barriers. Subspace clusters which are density-connected across cells can be detected along the connectivity barriers between cells. In the algorithm, these cells are merged until completely enclosing hypercubes for each subspace cluster are found.

Definition 12.4 ***Density-conserving grid.***

A ***density conserving grid*** *is a regular grid with connectivity barriers:*

- *A* ***regular grid*** *is a partition of the attribute range v into $g = \lceil \frac{v}{w} \rceil$ intervals p_i of equal width w and $p_i = [w \cdot (i-1), w \cdot i)$, $i = 1 \ldots g$.*
- ***connectivity barriers*** *are intervals of ε-width at the upper border of each cell in each dimension $b_i = [w \cdot i - \varepsilon, w \cdot i)$, $i = 1, \ldots, g$*
- *A s-dimensional* ***subspace cell*** *$C_{\alpha_1,\ldots,\alpha_d}$ is given by an index vector $\alpha_1, \ldots, \alpha_d$ of the corresponding intervals p_{α_j}, $\alpha_j \in \{1, \ldots, g, *\}$, where $(d-s)$ stars ("*") denote the unconstrained dimensions of the cell.*
- *A barrier $B_{\alpha_1,\ldots,\overline{\alpha_k},\ldots,\alpha_d}$ in dimension k is given by the index vector $\alpha_1, \ldots, \overline{\alpha_k}, \ldots, \alpha_d$ of its cell $C_{\alpha_1,\ldots,\alpha_d}$, where the dash ("-")denotes the barrier dimension k.*

In Figure 12.1 we illustrate our approach by an example of a two-dimensional space where each attribute range v is from 0 to 30. For example, cell $C_{3,4}$ contains two connectivity barriers $B_{\overline{3},4}$ and $B_{3,\overline{4}}$. In the example, $C_{3,4}$ is a 2-dimensional cell which is restricted in interval 3 for dimension 1 and in interval 4 for dimension 2. $C_{6,*}$ is a 1-dimensional cell restricted in interval 6 for dimension 1 and not constrained in dimension 2. Hypercubes consist of merged cells, given by interval ranges per dimension. $H_{[2,3][1,1]}$ is a 2-dimensional example hypercube.

$|H_{[a_1,b_1]\ldots[a_d,b_d]}|$ denotes the number of objects in a hypercube. A full space hypercube H is projected to a subspace H^S with $S = \{k_1, \ldots, k_s\}$ by removing the restriction in all other dimensions (i.e. setting the indices of $i \notin S$ of H to "$*$"). For example $|H_{[2,3][1,1]}|$ contains 9 objects, while its less con-

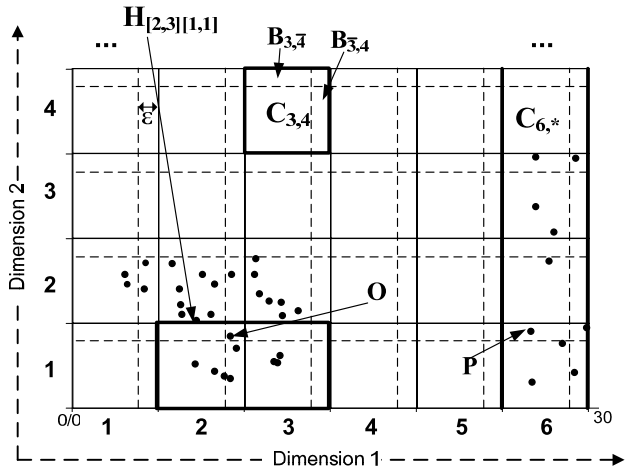


Figure 12.1: Density conserving grid in subspace of dimensions 1 and 2

strained projection to dimension one $H_{[2,3][*]}$ contains 35 objects (all objects contained in the 2nd and 3rd interval).

12.1.2 Monotonicity properties

For good runtime performance it is important to prune regions from the search space. Safely pruning without jeopardizing completeness requires a monotonicity on cluster properties. Monotonicity ensures that if a region which is not a subspace cluster in a subspace S implies that this region cannot be a subspace cluster in any higher dimensional subspace T ($S \subseteq T \subseteq D$), we may safely prune this region from further consideration.

We exploit two monotonicity properties in our EDSC algorithm. The first monotonicity property states that a higher dimensional projection of any hypercube contains at most the same number of objects.

(A) $|H^S_{[a_1,b_1]\ldots[a_d,b_d]}| \geq |H^T_{[a_1,b_1]\ldots[a_d,b_d]}|$

This monotonicity for hypercubes is used by grid-based subspace clus-

tering algorithms to prune the search space and proven e.g. in [AGGR98]. Our EDSC algorithm uses this monotonicity property in conjunction with the density conserving grid to prune sparse regions.

The second monotonicity property is used by traditional density-based subspace clustering algorithms to remove objects which violate the density constraint. It states that the number of objects contained in a neighborhood is monotonously decreasing with increasing dimensionality (proof in [KKK04]).

$(B)\ |N_\varepsilon^S(o)| \geq |N_\varepsilon^T(o)|$

Monotonicity does not hold for $|N_\varepsilon^S(o)|$ with respect to normalized density as the expected density *expDen* decreases with growing dimensionality (Definition 12.2). We device a weaker density criterion based on the expected density in the highest dimensionality d which can be used for pruning.

Definition 12.5 *Weak Density:*

An object $o \in DB$ is weak dense in S if:

$$|N_\varepsilon^S(o)| \geq max\{minPoints, F \cdot expDen(d)\}$$

Based on Definition 12.5 we derive the following pruning theorem.

Theorem 12.1 *Pruning weak dense objects: for all subspaces T with $S \subseteq T \subseteq D$ and $|T| = t$ holds:*

If an object is not weak dense in S it is not dense in T.

Following the weak density theorem, an object o can be pruned if it violates the weak density condition, as o is not dense in any higher dimensional subspace.

Proof:

$$|N_\varepsilon^S(o)| \leq max\{minPoints, F \cdot expDen(d)\} \tag{12.1}$$

$$\Rightarrow |N_\varepsilon^T(o)| \leq max\{minPoints, F \cdot expDen(d)\} \tag{12.2}$$

$$\Rightarrow |N_\varepsilon^T(o)| \leq max\{minPoints, F \cdot expDen(t)\} \tag{12.3}$$

Using monotonicity property (B) the left side of inequality (12.1 $\rightarrow$ 12.2) is monotonically decreasing with increasing dimensionality. As the expected density ($expDen$) is calculated as the ratio of the volume of the ε-sphere to the volume of the subspace, $expDen$ is monotonously decreasing with increasing dimensionality. Hence, the subspace of the highest dimensionality d has the lowest density: $expDen(d) \leq expDen(t)$. Thus the maximum on the right side of inequality (12.2) is less than or equal to the maximum in inequality (12.3). $\diamond$

12.1.3 The EDSC algorithm

Our multistep EDSC algorithm is based on two novel filters (Figure 12.2). The first filter step (*hypercube approximation filter*) approximates each density connected subspace cluster by a hypercube in the density conserving grid. Using the monotonicity criterion for hypercubes (A), a hypercube enclosing a density-connected set can be pruned if it contains less than $minSize$ objects. The second filter step (*weak density filter*) evaluates the hypercube according to weak density. Following Theorem 12.1, a region can be pruned if the objects in that region violate the weak density condition. Thus the multistep approach of the EDSC algorithm combines the pruning criterion of grid-based and density-based subspace clustering algorithms (Figure 12.2). Pruning based on these two properties is lossless in the sense that no cluster is wrongfully dropped from consideration.

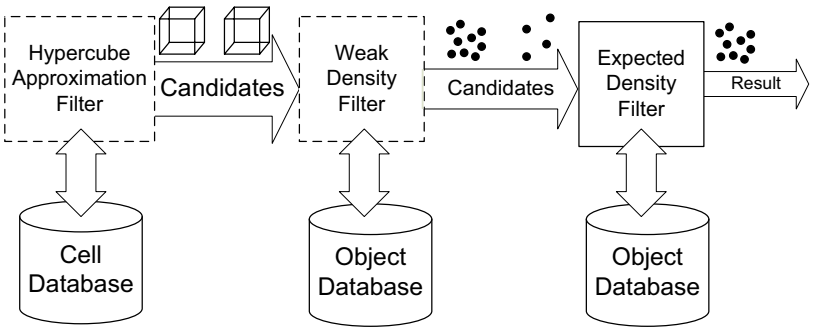


Figure 12.2: Multistep EDSC algorithm

As mentioned above, density connected clusters might extend into several grid cells. The hypercube approximation filter thus merges adjacent cells until an enclosing hypercube for each subspace cluster is found. For efficiency reasons we process cells in lexicographical order to ensure that each cell has to be processed only once. We mark future merges with respect to lexicographic order as *induced merges*; when these marked cells are actually processed, *performed merges* combine the cells. Lexicographically greater cells are processed in the future, hence we denote them as *future cells*, and lexicographically smaller cells as *past cells*. An enclosing hypercube is found if no more induced merges and performed merges are necessary from the current cell.

If a density-connected subspace cluster stretches from one cell into another, the connectivity barrier contains at least one object. Otherwise, as we set the barrier width exactly to the neighborhood range ε, objects in one cell cannot be in the ε-neighborhood of the other (Definition 12.1). When a cell is processed, the connectivity barriers of the cell are checked. For each barrier which contains an object a merge is *induced* into the corresponding adjacent cell. If both connectivity barriers contain an object a cluster might also extend into the diagonal cell, i.e. adjacent to the intersection of both

barriers, ensuring an induced merge to this diagonal cell.

When processing a cell, first merges into future cells are induced and then merges with past cells are *performed*. A merge is always performed with a past cell which induced a merge into the cell. When all induced merges of a hypercube have been performed the entire enclosing hypercube of a subspace cluster has been detected.

Using the method of induced and performed merges guarantees that we do not cut a cluster into two parts and that each subspace cluster induces exactly one enclosing hypercube (Theorem 12.2). The hypercube approximation filter is an optimistic filter, a conservative approximation which does not always contain a subspace cluster. The next filter steps remove all "false alarms", i.e., all hypercubes which do not contain a subspace cluster.

Figure 12.3 illustrates the merge steps performed to identify a hypercube enclosing the example cluster. We start by processing each cell of dimensionality two, beginning with cell $C_{1,1}$. The number of objects is determined, and if the cell is not empty, its connectivity barriers are tested. Cell $C_{1,1}$ is empty and hence no merge is induced. Next, cell $C_{2,1}$ is processed. As both connectivity barriers contain an object, cell $C_{2,1}$ induces a merge into $C_{3,1}$, $C_{2,2}$ and into $C_{3,2}$. Next, cell $C_{3,1}$ performs the merge with $C_{2,1}$, creating $H_{[2,3][1,1]}$. When $C_{2,2}$ is processed, it is merged with $C_{1,2}$ to $H_{[1,2][2,2]}$. After this, the next merge induced into $C_{2,2}$ is performed (center image in Figure 12.3). This merge step creates the hypercube $H_{[1,3][1,2]}$. Finally, cell $C_{3,2}$ is processed. As $C_{3,2}$ was already merged with both cells, only its counter for induced merges is decremented. After this step, no more merges are necessary and $H_{[1,3][1,2]}$ is completely detected.

Once an enclosing hypercube is complete, the EDSC algorithm checks if the hypercube contains more than a number of $minSize$ objects. If the region

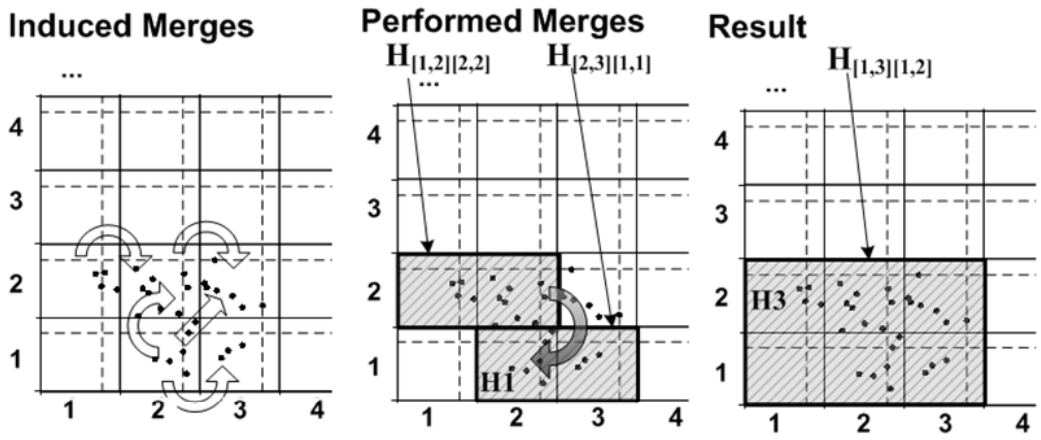


Figure 12.3: Induced and performed merges

does not contain enough objects, the hypercube and all higher dimensional projections of that hypercube are pruned (monotonicity property (A)). Thus the regions identified by the density conserving grid are used to reduce the number of time consuming scans for density connected clusters and to prune the search spaces.

Working depth-first, our EDSC analyzes only one hypercube at a time by successively extending it in all dimensions (Section 12.1.1). Hence the EDSC algorithm avoids generating more than one subspace cluster candidate at the same time. When a hypercube is extended to a new dimension it is successively restricted. For example, a two dimensional hypercube in a four dimensional space $H_{[1,3][1,2][*][*]}$ can be extended into dimensions three and four. We first extend it in dimension three and restrict it to cell one ($H_{[1,3][1,2][1,1][*]}$). Afterwards the merge procedure is applied recursively. Assume that the merge step mines $H_{[1,3][1,2][1,2][*]}$. It is next restricted in dimension four: $H_{[1,3][1,2][1,2][1,1]}$. After this step the next hypercube ($H_{[1,3][1,2][*][1,1]}$) is analyzed and processed accordingly.

For simplicity, we demonstrate the extension step using a one-dimensional

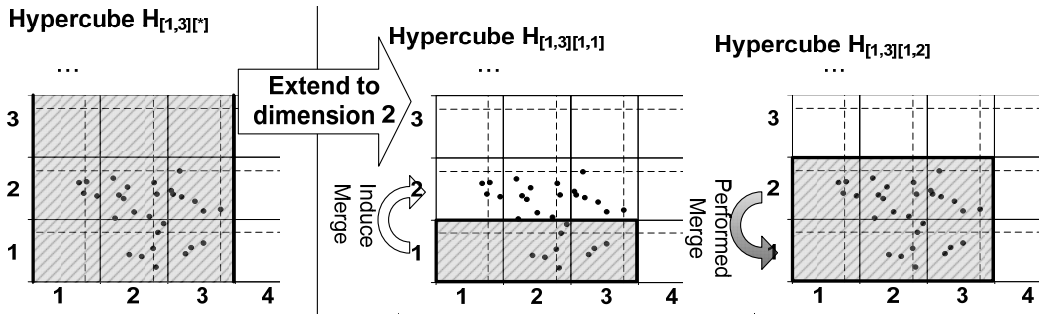


Figure 12.4: Extending a hypercube to next dimension

hypercube $H_{[1,3][*]}$. After extending the hypercube $H_{[1,3][*]}$ to dimensions two and restricting it to the first grid cell $H_{[1,3][1,1]}$ the merge procedure is applied again. As the corresponding connectivity barrier is not empty a merge is induced and performed directly afterwards, resulting in hypercube $H_{[1,3][1,2]}$. $H_{[1,3][1,2]}$ is complete in subspace $S = \{1, 2\}$ as its connectivity barrier is empty.

Density-conserving grid cells are a very compact representation of potential subspace clusters. By mapping cells to item sets, we can extend the frequent pattern tree (FP-tree) structure to support our EDSC algorithm. The FP-tree [HPY00] was originally devised for mining frequent item sets without candidate generation in association rule mining. We extend the FP-tree to manage both cells and connectivity barriers. Moreover, conditional FP-trees are merged efficiently to represent hypercubes. Our extended FP-tree uses cell-ids as item sets such that paths are annotated with the number of objects per cell/hypercube. The item sets are organized in lexicographic order, supporting the depth-first approach of the EDSC algorithm. All information about cells and connectivity barrier is stored in the tree. Thus the number of objects contained in a cell or connectivity barrier can be determined very efficiently without any additional database scans.

We now proof completeness by showing that each cluster induces exactly one enclosing hypercube.

Theorem 12.2 ***EDSC is complete:***

For each subspace cluster the enclosing hypercube is mined.

Proof: Each cluster induces exactly one enclosing hypercube:

- Start with the lexicographically first cell C_1 of a cluster. As C_1 is the first cell no past cell contains objects belonging to the cluster and hence no merge with a past cell is necessary.
- If the cluster is density-connected to a future cell C_f, the corresponding connectivity barrier contains at least one object. Thus the cell induces a merge into C_f.
- As C_f is processed in the future, the induced merge is performed later. After the merge step, the hypercube encloses cell C_f. Thus after processing a cell C_f no merge with a past cell is necessary.
- When the counter of induced merges is zero, the connectivity barriers are all empty. Consequently, no more future merges are necessary. Thus, the entire enclosing hypercube for the cluster is found. ◇

Algorithm 2 outlines the computation of the **hypercube approximation filter** (Line 1-12). The first filter step recursively calls the other filter steps of our multistep algorithm. Recall that each enclosing hypercubes are conservative approximations of subspace clusters (subspace cluster candidates). Using monotonicity property (A) hypercubes and all their higher dimensional projections can be pruned if they contain less than $minSize$ objects (Line 14).

Algorithm 2: EDSC(C, d_{first})

```
1  for k = d_first ... d do                          /* for each dimension and */
2      for i = 1 ... g do                        /* each cell in each dimension */
3          C = restrictCell(C, [k, i]);     /* restrict C in dimension k */
4          if restrictions(C) < 2 then                    /* no testing yet */
5              EDSC(C, k + 1) ;              /* continue EDSC recursion */
6          else
7              if restrictions(C) == 2 then
8                  I = testBarriers(C) ;                /* test all barriers */
9              else if restrictions(C) > 2 then
10                 I = testNewBarriers([k, i]); /* test only new barrier */
11             induceMerge(C, I) ;          /* for future neighbors of C */
12             C = performMerges(C) ;       /* for past neighbors of C */
13             if induceCounter(C)==0 then     /* hypercube complete */
14                 if objectCounter(C) > minSize then /* 1st filter step
                   */
15                     if WeakDensityScan(C) then  /* 2nd filter step
                       */
16                         EDSC(C,k+1);          /* further restriction */
17                         if isNonRedundant(C) then
18                             ExpDensityScan(C) /* 3rd filter step */
19                         else
20                             pruneCluster(C) ;          /* C is redundant */
21                     else
22                         prune(C) ; /* C and higher dim. projections
                           */
23                 else
24                     prune(C) ; /* C and higher dim. projections */
```

If an enclosing hypercube is a subspace cluster candidate, it is subsequently analyzed by the second filter step (**weak density filter**) of the EDSC algorithm (Line 15). The second filter step uses the weak density defined in Theorem 12.1 to determine if a hypercube contains a weak density connected region [EKSX96]. If the hypercube does not contain any weak density connected subspace cluster the corresponding hypercube and all its higher dimensional hypercube projections can be pruned. This is the second step of our multistep algorithm. Following a depth-first approach redundant subspace clusters can be pruned (Line 17).

The third step of the multistep algorithm removes false alarms (Line 18). If a region contains a weak density connected subspace cluster check if it contains a subspace cluster with respect to the expected density of the dimensionality of the subspace (Definition 12.2). Thus the method *ExpDensityScan* scans a region with respect to the expected density of a subspace. If the region does contain a subspace cluster the result is stored.

12.2 Experiments

In extensive experiments we evaluated both the efficiency and the accuracy of our EDSC algorithm. Competitors are SUBCLU [KKK04], a density-based approach and SCHISM [SZ04a], a recent grid-based algorithm. As discussed above, both algorithms use an apriori-based approach for pruning. To evaluate scalability we use synthetic data sets. Further on we show the performance of EDSC in terms of efficiency and accuracy on four real world data sets.

12.2.1 Setup and data sets

Based on properties of real world data sets we generate synthetic data for scalability experiments. We use a method proposed in SUBCLU [KKK04] to generate density-based clusters in arbitrary subspaces. Given the subspace and the number of objects for each cluster, the generator creates dense regions separated by sparsely populated noisy regions. In addition, our generator takes into account that objects can belong to multiple subspace clusters, just as in most real world data sets. We generate data of different dimensionalities and hide subspace clusters with a maximal dimensionality of 80% of the data dimensionality. Four subspace clusters are hidden in the synthetic data with two of them overlapping in 10% of their objects.

12.2.2 Measurements

Efficiency is simply measured in terms of runtime, accuracy is determined using the F1-value. The F1-value of a cluster is the harmonic mean of its precision and recall. The F1-value of the whole clustering averages the F1-values of all hidden clusters. To obtain a mapping between hidden and detected clusters, we assign each subspace cluster to the hidden cluster which has the highest support in the detected cluster. For each hidden cluster precision measures how accurately it is detected. Recall measures if a cluster is detected completely [MSE06].

12.2.3 Scalability

As the number of possible subspace clusters depends exponentially on the dimensionality of the subspace, scalability in term of dimensionality is crucial for any subspace clustering algorithm. As depicted in Figure 12.5, SUBCLU

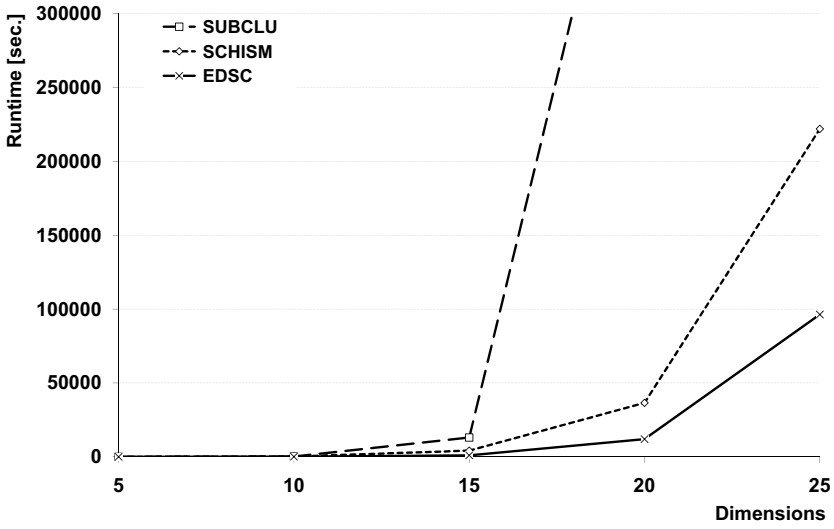


Figure 12.5: Scalability vs. dimensions

does not scale with respect to dimensionality. For the 20-dimensional data set the algorithm did not even finish after six days. The reason is the tremendous amount of 1 and 2-dimensional subspaces that have to be clustered before processing any higher dimensional subspace.

SCHISM as a grid-based approach has better runtimes but also suffers from apriori-based candidate generation overhead. The EDSC algorithm with its depth-first approach overcomes this problems. Further on EDSC is lossless because of the barriers in its density conserving grid. Moreover, EDSC clearly outperforms SUBCLU and SCHISM in terms of accuracy which can be seen in Figure 12.6. The reasons for its good accuracy are: (a) the lossless density-based clustering and (b) the new normalized density threshold. Thus, EDSC is able to separate all hidden subspace clusters from the synthetic noise. SUBCLU and SCHISM do not reach this accuracy because SCHISM suffers from the grid-based approach which mixes subspace clusters in one grid-cell or loses clusters cut apart by the grid. For SUBCLU the fixed density

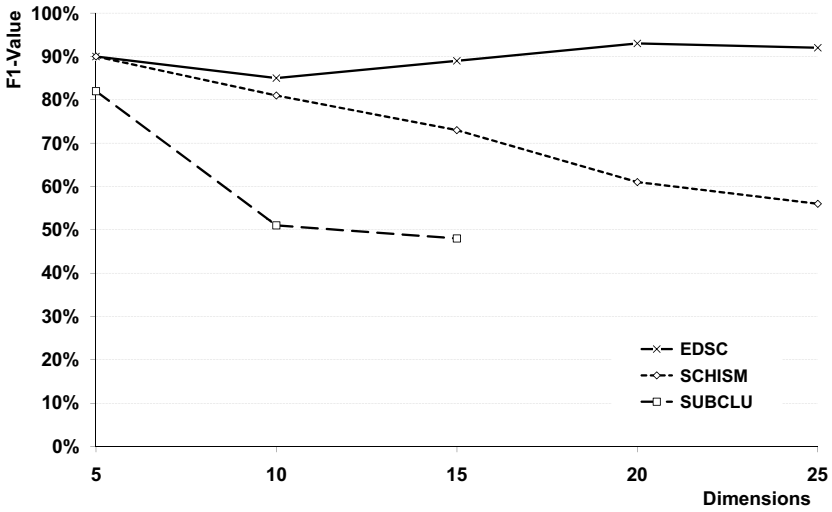


Figure 12.6: Accuracy vs. dimensions

threshold and huge runtimes hinder detection of subspace clusters hidden in high dimensional subspaces.

12.2.4 Database size scalability

In the next experiment we generate 15-dimensional data sets with different numbers of objects. As we can see in Figure 12.7, EDSC and SCHISM scale well with respect to the number of objects. However EDSC again outperforms SCHISM due to the new depth-first approach without candidate generation. SUBCLU also does not scale with increasing number of objects because of the time consuming database scans needed for density-based subspace clustering. In contrast EDSC uses the density conserving grid to avoid these scans and thus is only barely influenced by the database size.

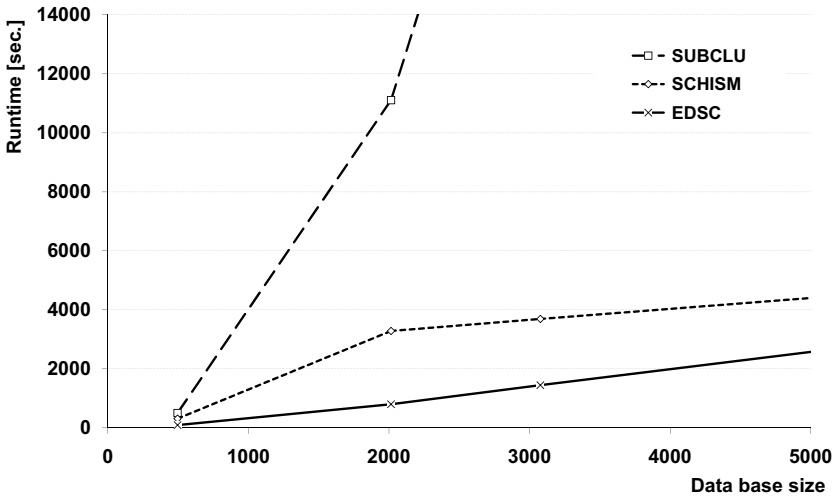


Figure 12.7: Scalability vs. database size

12.2.5 Selectivity

To illustrate the efficiency of the hypercube approximation filter we analyze its selectivity compared to the number of performed density-based scans in SUBCLU. Recall that density-based clustering has a quadratic complexity with respect to the number of objects. Thus avoiding density-based scans is an important contribution to the performance gain of EDSC.

In Figure 12.8 we see the number of required density-based clustering computations for the data from our first experiment with varying dimensionality (only for up to 15-dimensional data because SUBCLU does not scale to higher dimensional data). We see that indeed the main problem of SUBCLU is the huge amount of regions that have to be clustered. Using the multistep filtering in EDSC the number of density-based clustering computations is significantly lower. Many regions that have to be scanned in SUBCLU can be pruned by EDSC without any database scans only by the information of the grid and barrier cells.

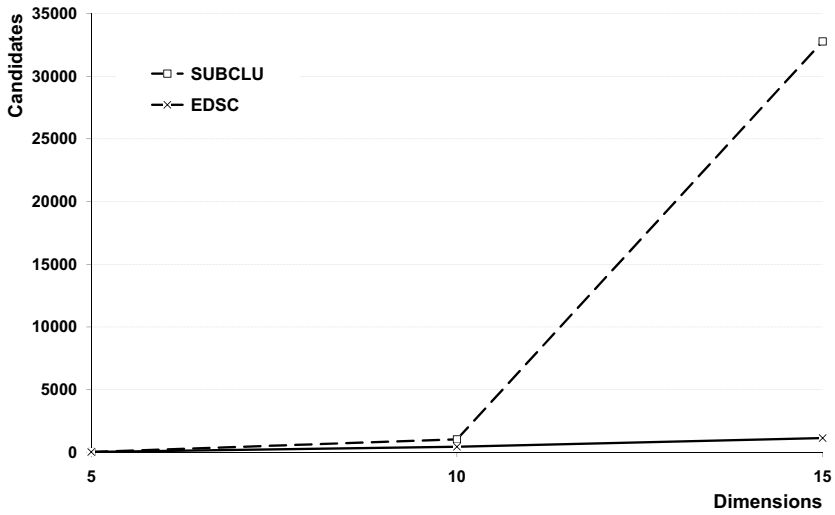


Figure 12.8: Selectivity vs. dimensionality

12.2.6 Parameters

As we already mentioned SUBCLU also suffers from the fixed density threshold $minPoints$. For different dimensionalities in subspaces the measured density is not comparable. To find high dimensional subspace clusters one has to use a lower value for $minPoints$ in SUBCLU. In EDSC this parameter is robust because of its normalized density with respect to the expected density of the dimensionality. As we can see in Figure 12.9 the problem is that with decreasing $minPoints$ the runtime of SUBCLU increases. Finding high dimensional subspace clusters is virtually infeasible because already for $minPoints$ set to a value of eight SUBCLU did not scale in the previous experiments.

Further experiments show that parameter F which reflects the factor by which the expected density should be exceeded is similarly robust in terms of runtime.

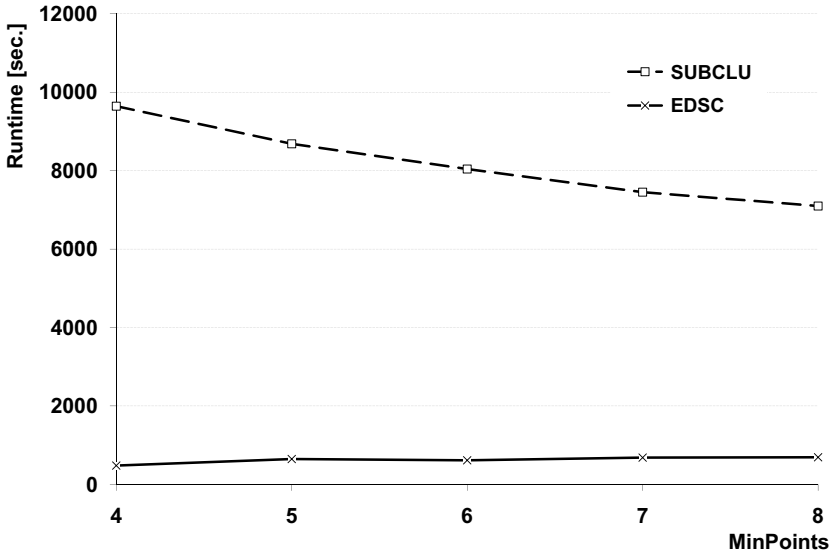


Figure 12.9: Parameter *minPoints* variation

12.2.7 Quality and runtimes on real world data

Real world data used in our experiments is characterized in Table 12.1. We evaluate the quality of EDSC by determining how accurate the hidden structure of the data given by the classes is identified.

For each data set efficiency and accuracy of each algorithm is presented in Table 12.2. We can see that the EDSC algorithm shows the best quality for all real world data sets. Thus EDSC indeed finds the intrinsic structure in the data. Compared with SUBCLU the EDSC algorithm shows significantly

	Objects	Dimensions	Classes	Source
Pendigits	7494	16	10	[AN07]
Vowel	990	10	11	[AN07]
Glass	214	9	6	[AN07]
Shapes	160	17	9	[KWX+06]

Table 12.1: Real world data sets

	EDSC		SCHISM		SUBCLU	
	F1	time [s]	F1	time [s]	F1	time [s]
Pendigits	48%	5743	25%	5278	24%	43825
Vowel	35%	47	21%	88	17%	310
Glass	57%	3	47%	8	52%	10
Shape	51%	34	2%	0.2	40%	1079

Table 12.2: Quality and runtimes on real world data

better runtimes. The grid-based SCHISM shows not only worse quality in all data sets but in some also higher runtimes. In the shape data set SCHISM only finds 2-dimensional clusters and thus cannot be compared with the other subspace clustering algorithms which also detect the higher dimensional clusters.

12.3 Conclusion

We introduced EDSC, an efficient density-based subspace clustering algorithm. EDSC uses a database inspired multistep approach which allows powerful pruning of the search space by exploiting two monotonicity properties. Based on a novel density-conserving grid a conservative approximation of density connected regions by hypercubes ensures completeness. A systematic processing order of the grid cells supported by an extended FP-tree structure guarantees high efficiency. Our experiments on large and high-dimensional synthetic and real world data sets show that EDSC outperforms recent subspace clustering algorithms by orders of magnitude.

Part IV

Mining time series data

Chapter 13

Clustering Multidimensional Time series

Many environmental, scientific, technical or medical database applications require effective and efficient mining of time series, sequences or trajectories of measurements taken at different time points and positions forming large temporal or spatial databases. Particularly the analysis of concurrent and multidimensional time series poses new challenges in finding clusters of arbitrary length and varying number of attributes.

We present a novel algorithm capable of finding parallel clusters in different subspaces and demonstrate our results for temporal and spatial applications. Our analysis of structural quality parameters in rivers is successfully used by hydrologists to develop measures for river quality improvements.

13.1 Introduction

Environmental sensors produce data streams at successive time points which are often archived for further analysis. Applications like stock market analy-

sis or weather stations gather ordered time series of different values in large temporal databases. Weather stations for example use multiple sensors to measure e.g. barometric pressure, temperature, humidity, rainfall. Many other scientific research fields like observatories and seismographic stations archive similar spatial or spatial-temporal time series.

As one application example, we focus on hydrological data. In a current project of the European Union on renaturation of rivers, the structural quality of river segments is analyzed. For a spatial database of German rivers, about 120.000 one-hundred-meter segments were evaluated according to 19 different structural criteria, e.g. quality of the riverbed [LUA03]. They were mapped to quality categories, where a value of "one" indicates perfect quality, while a value of "seven" indicates most severe damages. Figure 13.1 illustrates 8 of the 19 attributes of a sample river segment in GIS (geographic information system) representation. The sequence order of the segments is given by the flowing direction of the rivers.

As the project aims at a quality improvement over the next decades, packages of measures have been suggested for different structural damages. They have been formalized in rules specifying the attribute value constraints and the attributes influenced positively by execution of the respective measure. An example constraint might be that a certain segment has good quality (categories one to three) riverbed and riverbanks and poor river bending (categories five to seven). This could be improved by measures like adding deadwood to positively influence river bending.

Finding and analyzing these patterns helps hydrologists summarize the overall state of rivers, give compact representations of typical situations and review the extent to which these situations are covered by measures envisioned. They can identify those patterns which are problematic, i.e. have

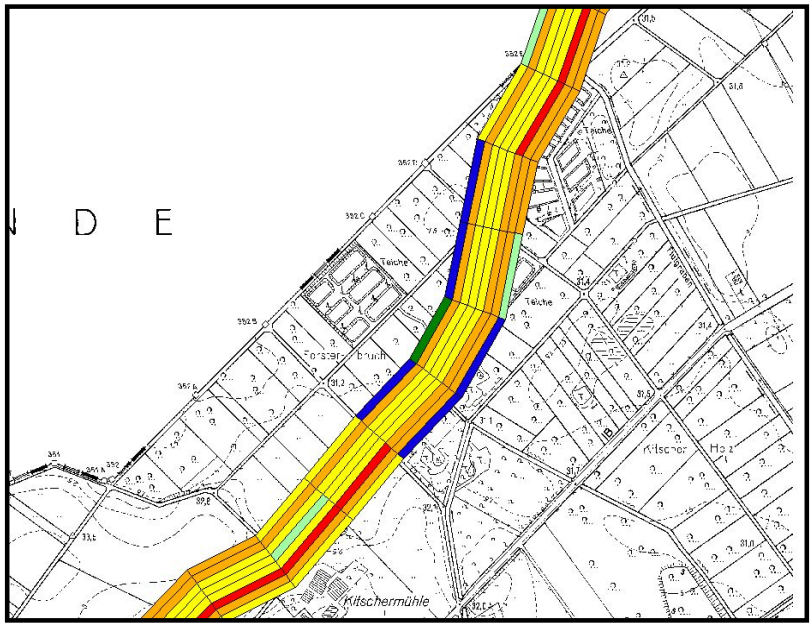


Figure 13.1: GIS illustration of eight out of 19 river attributes

low quality ratings, but are not yet covered by measures. In a follow-up step, these measures are annotated by time and cost information. This is used to generate an overview over the state of rivers as it might be in the near future if the measures are put into action.

From a computer science point of view, finding the intrinsic structure of these multidimensional time series is a two-fold task:

- detect frequent patterns within time series for all possible time series lengths (note that we cannot know the pattern length a priori),
- then detect parallel occurrences of these patterns.

Patterns are ranges of values (which correspond to several categories of river quality structure) found in several (sub-)time series. Pattern analysis has to take into account that the data is subjective and fuzzy, because struc-

tural quality of rivers was mapped by different individuals. Our approach is based on weighting by kernel densities in a density-based clustering approach. This effectively detects fuzzy time series patterns. These patterns are clustered efficiently for arbitrary lengths using monotonicity properties. We transform these time series pattern clusters into a cluster position space such that mining parallel patterns can be reduced to efficient FP-tree frequent itemset mining.

This part is organized as follows: we review related work in Section 13.2. Basic definitions in Section 13.3 lay a sound foundation for the proposed subspace clustering method in Section 14. We describe and disucss our algorithmic concept in Section 14.4. The experimental evaluation in Section 14.5 demonstrates the effectiveness of our approach on real world and synthetic datasets. Efficiency is shown and parametrization is evaluated. We conclude this part in Section 14.6, summarizing our results and anticipating future work.

13.2 Related work

Numerous clustering methods have been proposed in the literature, including partitioning clustering, e.g. the well-known k-means algorithm [Mac67]. These algorithms require the specification of the number of clusters to be found and can only detect convex cluster regions. Categorical clustering methods work well for categorical data where the notion of neighborhood is not meaningful [GRS99, ZPAS05, ZPAS07].

Density-based algorithms use a function to determine the density of the neighborhood of each object and use a connectivity-notion to assign similar points to the same cluster [EKSX96, HK98, HK99]. Density-based clustering

is robust to noise since it clusters only those objects or time series above some noise threshold as discussed in [Den04]. Moreover, it naturally incorporates neighboring objects into its cluster definition.

The analysis of time series data has recently gained a lot of attention. Recordings of data at successive time points have been studied in time series mining; e.g. [FRM94, KCMP01]. Most of these approaches, however, aim at finding patterns of values which do not have to directly follow one another, but may have other values in-between, as in sequential frequent itemset mining [AS95, AFGY02].

Motif mining searches those patterns which have the highest count of similar time series [PKLL02]. Matching within some range is used to determine the frequency. However, neighbors are not weighted and the range is fixed for all time series. Moreover, parallel patterns are not discussed since the application targeted is one-dimensional time series. While noise is removed in motif discovery as well, we found density-based clustering to be more useful in handling fuzzy data, because density-based clusters automatically adapt to different ranges of fuzziness.

13.3 Cluster model

In this section we formalize our notion of patterns in time series. We define a suitable cluster notion which reflects that our database consists of ordinal-valued time series with some degree of noise.

Density-based clustering, which tries to separate dense areas ("clusters") from sparse ones ("noise") reflects the requirements of applications such as the river data scenario in that arbitrary cluster shapes may be found, the number of clusters does not need to be fixed a-priori and noise is labeled as

such, i.e. it does not have to be assigned to any cluster [EKSX96, KKK04].

13.3.1 Time series patterns and clusters

As noted before, it is crucial to determine time series clusters of arbitrary lengths. We define time series patterns of arbitrary length in single attributes and subsequently model parallelism between these clusters.

Definition 13.1 ***Time series pattern:***

- *A tupel* $t = (t_1, \ldots, t_k)$ *of* k *subsequent values at positions* 1 *through* k *is called a time series of length* k*.*
- *A database* DB *is a set of time series* $\{t^1, ..., t^z\}$*.*
- *We denote a time series of* t *from position* i *to* j *by* $t[i, j] = (t_i, \ldots, t_s)$*.*
- *Whenever we are not interested in the concrete positions of a time series, but merely in its values, we call this a pattern* $p = \langle p_1, \ldots, p_k \rangle$*.*
- *A pattern* P *occurs in a database if there is a time series* $t \in DB$ *and a position* $i \in \mathbb{N}$ *with* $p = t[i, i+k\text{-}1]$*.*
- *The support of a pattern* p *is the number of its occurrences in the time series of the database* DB*:*
  $sup(p) = |\{i \in \mathbb{N} \text{ and } t \in DB, \text{ where } p = t[i, i+k\text{-}1]\}|$

When searching for prevailing patterns, it is important to note that merely counting of time series patterns is not sufficient in many scenarios. It is crucial to account for two factors: first, mapping of river structures may be blurred by people's subjective decisions on unclear category boundaries. Second, measures and their constraints may be applicable over several categories

and cannot always be fitted exactly to these categorical boundaries. Moreover, small deviations in few segments may be tolerable for the application of a certain measure if this results in longer river courses treatable by a single measure. Hydrologists are thus interested in including "similar" time series in frequency notions.

Density-based clustering tries to separate dense areas ("clusters") from sparse ones ("noise"). The density of a pattern is determined by evaluating its neighborhood according to some distance function.

Any L_p-Norm $\left(L_p(q,q') = \sqrt[p]{\sum_{i=1}^{k}(q_i - q'_i)^p}\right)$ between patterns q and q' can be used, yet the Manhattan norm (L_1) has shown to work well in preliminary experiments. We use a weighting function to ensure that with greater distance to the pattern evaluated, the influence of neighbors decreases. Any series of monotonously decreasing values can be used as a weighting function, since distances between nominal time series are always discrete. All kernel-estimators known from statistics [Sil86] are constantly falling functions. Experiments have shown that weighting functions based on Gaussian kernels ($W_\sigma^p(q) = \mathsf{exp}\left(-\mathsf{dist}(p,q)^2/2\sigma^2\right)$) perform well in many applications. Using weighting functions based on kernel estimators provides us with a natural way of defining the set of similar time series to be included in the density evaluation. Whenever the weights assigned drop below a certain significance threshold τ, these time series should not be considered in the density estimation. For example, Gaussian kernels assign (possibly very small) density values to all patterns in the database, which can be cut off below some tiny value, such as $\tau = 0.01$ or less. This way, excess density computations can be avoided.

Definition 13.2 ***Neighborhood and Density****:*

- *The τ-neighborhood of a pattern p, $N_\tau(p)$, is defined as the set of all patterns q which at least have the influence τ on the pattern p evaluated:* $N_\tau(p) = \{q, where\ W_\sigma^p(q) \geq \tau\}$.
- *The density of p is the weighted sum of patterns in the neighborhood*

$$\mathsf{density}(p) = \sum_{q \in N_\tau(p)} W_\sigma^p(q) * sup(q)$$

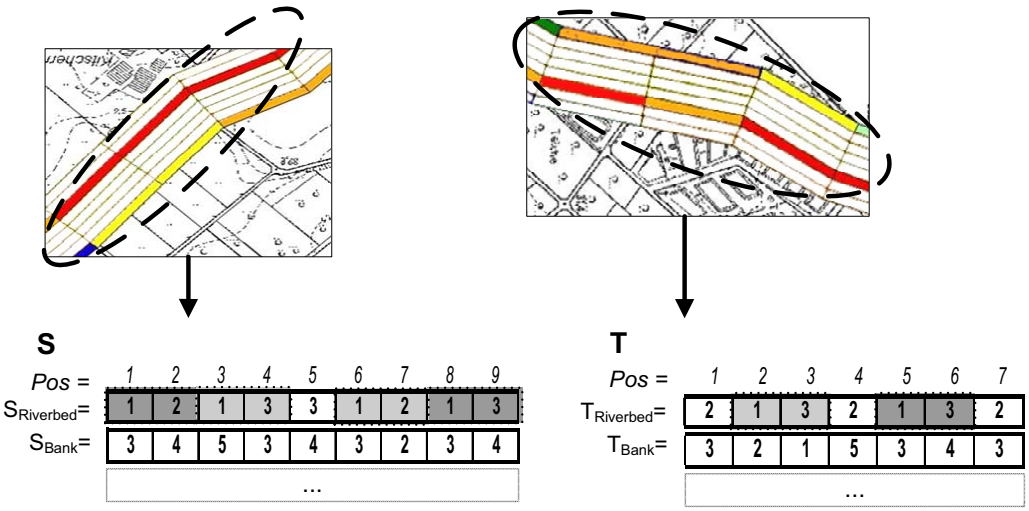
- *p is dense with respect to a density threshold δ iff* $\mathsf{density}(p) \geq \delta$.

We are now ready to formalize our cluster notion. As mentioned before, clusters should consist of similar, dense time series of the same length. Time series within one cluster should therefore be within certain parameterizable boundaries.

Definition 13.3 ***Cluster****:*
A set $C = \{p_1, \ldots p_m\}$ of m patterns p_i is a cluster of length k with respect to a density threshold δ and a compactness parameter γ iff:

- *for all patterns q of length k not in C: $C \cup \{q\}$ is not a cluster. (Maximality)*
- *for all patterns p_i :* $\mathsf{density}(p_i) \geq \delta$. *(Density)*
- *for any patterns $p_i, p_j \in C$ there is a chain of patterns $(q_1, \ldots, q_v) \in C$ such that $q_1 = p_i, q_v = p_j$ and $\forall\ r\ \mathsf{dist}(q_r, q_{r+1}) \leq \gamma$. (Compactness)*

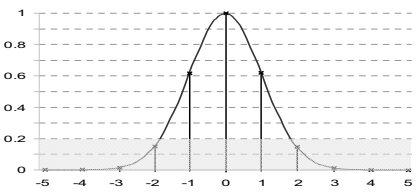
Put informally, we are thus looking for clusters which are as large as possible (maximality), whose elements are all dense (density) and at most γ apart from each other (compactness). We set γ to one in categorical settings.



Density Estimation of Pattern P=⟨1,2⟩:

$$\text{density}(\langle 1,2\rangle) = 2 * W_1^{\langle 1,2\rangle}(\langle 1,2\rangle) + 4 * W_1^{\langle 1,2\rangle}(\langle 1,3\rangle) + 4 * W_1^{\langle 1,2\rangle}(\langle 1,3\rangle)$$

Gaussian Weighting Function:



$$W_\sigma^P(Q) = e^{-\frac{dist(P,Q)^2}{2\sigma^2}}$$

Setup: $\sigma = 1$

$\tau = 0.2$

Figure 13.2: Example multicluster in the river database

Example.

Figure 13.2 illustrates the definition of density and gives an example for a density calculation of pattern $p = \langle 1, 2\rangle$. The upper part visualizes two time series S and T with two exemplary attributes, the riverbed and the bank. In the lower part of the figure the density-value for a pattern $\langle 1, 2\rangle$ is calculated. We assume a significance threshold of $\tau = 0.2$. The Gaussian weighting function drops below $\tau = 0.2$ for time series having a distance higher than 1.8 from the point evaluated. Thus in our ordinal setting only time series with a distances of or less 1 have significant influence: $\mathsf{exp}(-1^2/2) \approx 0.6 > \tau$ and $\mathsf{exp}(-2^2/2) \approx 0.13 < \tau$. In our example the τ-neighborhood of pattern $\langle 1, 2\rangle$ contains the time series $\langle 1, 2\rangle$ itself starting at two positions S_1 and S_6 (distance zero) and $\langle 1, 3\rangle$ starting at four positions S_3, S_8 and T_2, T_5 (distance one). Thus the density-value for $\langle 1, 2\rangle$ is $2 * W_\sigma^{\langle 1,2\rangle}(\langle 1, 2\rangle) + 4 * W_\sigma^{\langle 1,2\rangle}(\langle 1, 3\rangle) = 2 * 1 + 4 * 0.6 = 4.4$. For a density-threshold of e.g. $\delta = 3$ the pattern $\langle 1, 2\rangle$ is considered dense.

13.3.2 Multiclusters in parallel patterns

Single time series patterns give insight into the inherent structure in individual attributes. In multidimensional time series databases these patterns have to be extended to parallel patterns. River measures may affect several structural properties, e.g. the river bank on the left and the right as well as the river bending. Similarly, constraints are often formulated for several attributes as well. Likewise, in other applications, situations which require specific measures are typically described via several sensor values.

We define multiclusters as frequent parallel occurrences of single attribute clusters. Frequency is measured in terms of simultaneous occurrences, i.e.

the number of positions in any time series, where clusters are detected in different attributes. We formalize our notion of parallel clusters as follows:

Definition 13.4 ***Multicluster****:*
A set of time series clusters $C_1, \ldots, C_n$ *of length* k *from* n *different attributes is a multicluster* MC *iff:*

- $C_1, \ldots, C_n$ *are parallel at some position* i,
  i.e. $\forall j \in \{1 \ldots n\}$ *a pattern* $p_j \in C_j$ *occurs at position* i
- $C_1 \ldots C_n$ *occur frequently together,*
  i.e. $|\{\mathrm{i} \in \mathbb{N},\ C_1, \ldots, C_n$ *are parallel at position* $\mathrm{i}\}| \geq \phi$.

Put informally, we are thus looking for those positions which show a pattern contained in each of the clusters (parallelism), which have a count equal to or greater than the threshold given (frequency).

The frequency threshold ϕ reflects the number of positions where the multicluster is detected. It can be set in relation to the database size, i.e. the overall number of multidimensional time series segments (see Section 14.5). Hence, ϕ corresponds to the relative frequency of a multicluster (the support) and is a key parameter depending on the application. In general, increasing ϕ decreases the number of identified multiclusters, as more occurrences of the pattern are required. Multiclusters detected using a higher ϕ are far more typical for the data set. Our experiments suggest that an initial setting of about 1% of the data set serves as a good starting point for analysis. As more or less patterns are desired, the threshold is adapted accordingly.

Example.

In Figure 13.3 we present a multicluster in two attributes. The multicluster MC consists of two clusters $C_1 = \{\langle 1, 2\rangle, \langle 1, 3\rangle\}$ in the riverbed attribute and

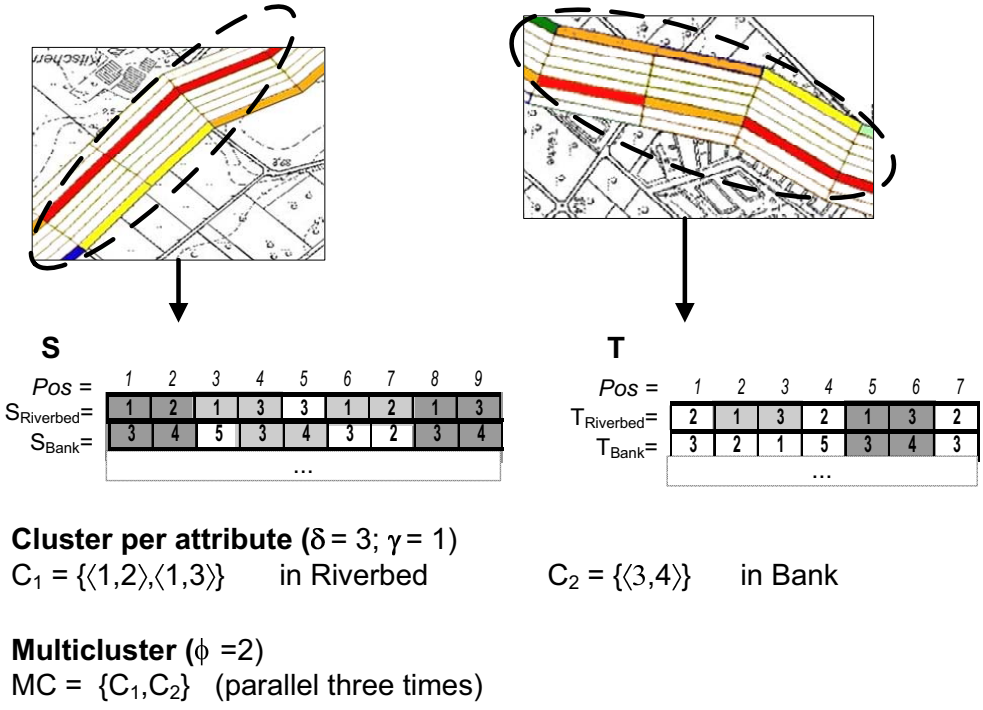


Figure 13.3: Example multicluster

$C_2 = \{\langle 3, 4\rangle\}$ in the bank attribute. They occur in positions S_1, S_8, T_5, thus three times. Assuming a frequency threshold of $\phi = 2$, MC is a multicluster.

Chapter 14

Efficient cluster mining

To detect clusters of arbitrary length, a naive approach might be to simply re-run a density-based clustering algorithm for each length value to detect all possible clusterings. Obviously, this leaves room for efficiency improvement.

14.1 Monotonicity

We avoid excess clustering steps by exploiting monotonicity properties of clusters. We show that patterns which are not dense, cannot be part of longer dense patterns. We may safely prune them from consideration in longer pattern clustering. We formalize this monotonicity first for patterns and then prove that this property holds for clusters themselves.

Theorem 14.1 ***Density Monotonicity****:*
For any two patterns p, q of length k and their respective prefix/suffix p', q' of length $k-1$ holds:

- $q \in N_\tau(p) \Rightarrow q' \in N_\tau(p')$
- $\mathsf{density}(p) \leq \mathsf{density}(p')$

For (1) we note that an L_p norm, $p \geq 1$ is the p-root of sum of absolute differences in time series values. This means that a reduced sum of $k-1$ of these differences is necessarily smaller than or at most equal to a sum of all k differences.

Proof: For a prefix/suffix q', p' of q, p we first show: $\mathsf{dist}(p,q) \geq \mathsf{dist}(p',q')$. Dropping the first or last summand in our distance sum for q, p, we obtain for the prefixes or suffixes q, p:

$$\sqrt[p]{\sum_{i=1}^{k} |p_i - q_i|^p} \geq \sqrt[p]{\sum_{i=1}^{k-1} |p_i - q_i|^p} \quad \text{dropping the last summand: prefix}$$

$$\sqrt[p]{\sum_{i=1}^{k} |p_i - q_i|^p} \geq \sqrt[p]{\sum_{i=2}^{k} |p_i - q_i|^p} \quad \text{dropping the first summand: suffix}$$

$\Rightarrow \mathsf{dist}(p,q) \geq \mathsf{dist}(p',q')$

Thus the distance between two patterns is greater than the distance between the respective prefix or suffix. Using this fact we can prove (1):

$\mathsf{dist}(p,q) \geq \mathsf{dist}(p',q')$

$\Rightarrow \quad W_\sigma^p(q) \leq W_\sigma^{p'}(q')$

(weighting functions are increasing with decreasing distance)

$\Rightarrow \quad (W_\sigma^p(q) \geq \tau \Rightarrow W_\sigma^{p'}(q') \geq \tau)$

(using definition of N_τ)

$\Rightarrow \quad (q \in N_\tau(p) \Rightarrow q' \in N_\tau(p'))$.

Part(2) is proven using a similar argument: the density is defined as a weighted sum of the support of patterns. Their shorter counterparts, prefix or suffix, are assigned smaller distance values (part 1) and therefore larger

weights.

$$
\begin{aligned}
&\mathsf{density}(p) = \\
&\quad \sum_{q \in N_\tau(p)} W_\sigma^p(q) * sup(q) \text{(definition of density)} \\
&\leq \sum_{q'} \in N_\tau(p') W_\sigma^p(q) * sup(q') \\
&\quad \text{(Part 1: } sup(q) \leq sup(q') \text{ as a time series for } q \text{ also matches } q') \\
&\leq \sum_{q'} \in N_\tau(p') W_\sigma^{p'}(q') * sup(q') \\
&\quad \text{(Part 1: distance of shorter patterns is smaller, weights are larger)} \\
&= \mathsf{density}(p') \text{ (definition of density)} \qquad \diamond
\end{aligned}
$$

Since density and neighborship are monotonous, we can conclude that clusters are monotonous as well:

Theorem 14.2 ***Cluster Monotonicity****:*
For any cluster C of length k, there is a cluster C' of length $k-1$ such that for any pattern p and its prefix/suffix p' holds:

$$p \in C \Rightarrow p' \in C'$$

Proof: For any p in C, we have that p' is dense (Lemma 14.1), and since the distance of shorter patterns is smaller (Proof of Lemma 14.1), there exists a chain of prefix/suffix patterns which guarantees compactness. This means that all prefixes/suffixes are part of a cluster (which might contain additional patterns due to the third requirement, completeness). $\diamond$

Summing up, we know that any pattern or cluster which does not satisfy the density criterion, can be safely pruned from the search for longer patterns or clusters.

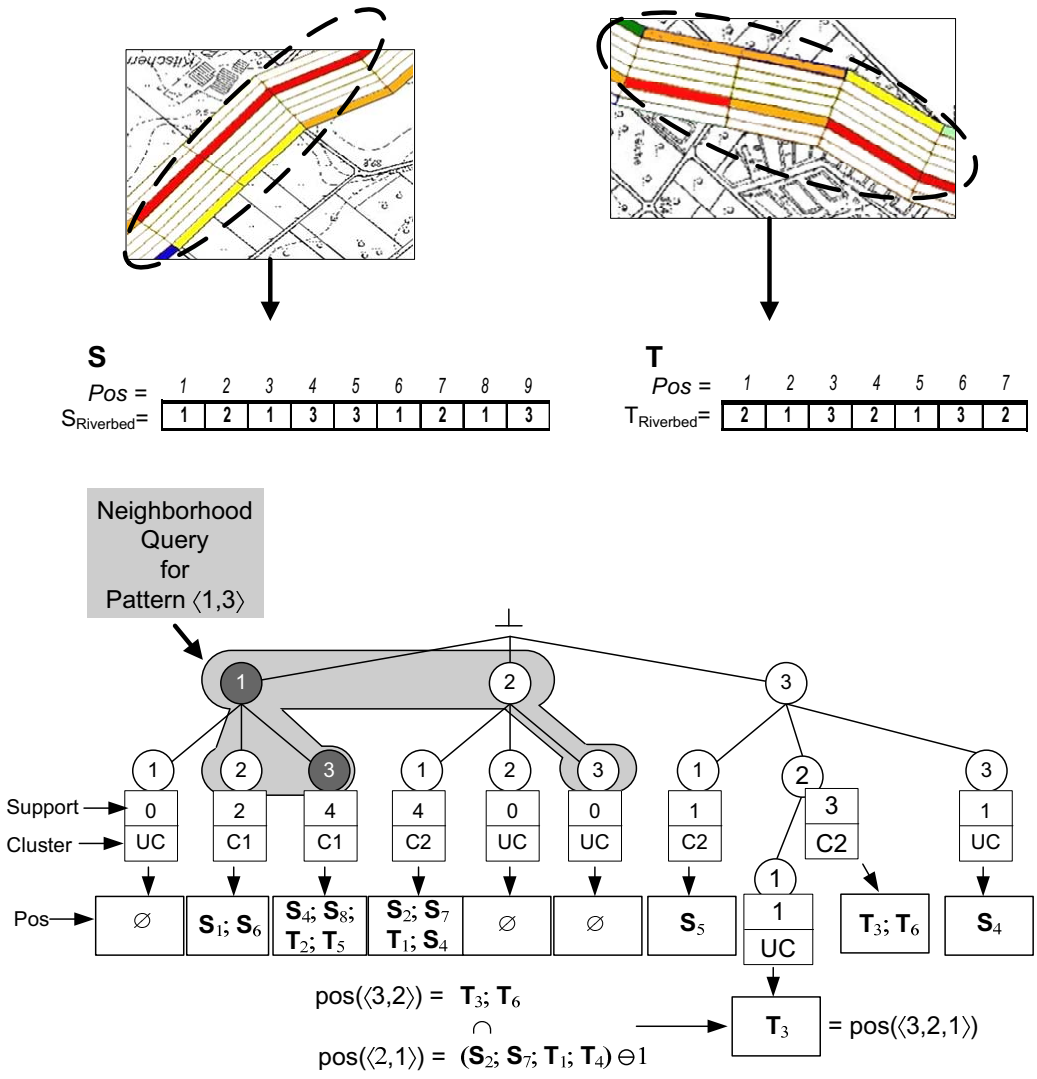


Figure 14.1: Hierarchical index structure

14.2 Indexing time series clusters

In order to efficiently calculate the density value for a pattern it is crucial to quickly retrieve the neighborhood of a pattern. We introduce an index structure for patterns which supports neighborhood queries and density estimators.

Since existing index structures do not work for patterns of different length, the hierarchical index structure illustrated in Figure 14.1 was developed. The index structure is tailored to the multicluster (MC) algorithm in that the bottom-up approach is supported: queried patterns always grow in length.

The index is constructed by scanning once over each time series. Each pattern of a fixed starting length determined is added to the index structure. A pattern is represented by a path of labeled nodes from the root to a leaf. To later determine the density value of a pattern the support is annotated at each corresponding leaf node. Further on the index structure stores the position list (all starting points) for each pattern. For scalability purposes the position list is stored on hard disk. In addition, a cluster identifier (e.g. $C1$) or the flag *unclassified* (UC) is stored at each leaf node (Figure 14.1). The multicluster algorithm uses this flag to efficiently calculate the transitive closure (compactness in Definition 13.3) for a dense pattern. The density value for a pattern can be calculated by summing up and weighting the support of all patterns in the corresponding neighborhood. The index structure efficiently supports neighborhood queries by selecting the appropriate node ranges while descending the tree.

Example.

In Figure 14.1 we assume a parameter setting of $\tau = 0.2$ with a weight of 1 for patterns having distance zero and a weight of 0.3 for patterns having

distance one. The density value for pattern $\langle 1,3\rangle$ can then be calculated in the following way: Starting at the root node the node range ("1″-″2″) must be considered. When processing node "2", for instance, all patterns containing this node have a distance of at least one (distance from $\langle 1,3\rangle$ to $\langle 2,*\rangle$ is greater than or equal to one). Since the maximal distance according to τ is bounded by 1, the only pattern which must be considered below node "2" is the pattern $\langle 2,3\rangle$. In the same way all other patterns within the specified range can be determined (in this example $\langle 1,2\rangle$, $\langle 1,3\rangle$ and $\langle 2,3\rangle$). Finally the density value for the pattern $\langle 1,3\rangle$ can be calculated by summing up the support of the leafs for each weighted time series:

$$\begin{aligned} density(\langle 1,3\rangle) &= W^{\langle 1,3\rangle}(\langle 1,2\rangle) * sup(\langle 1,2\rangle) + W^{\langle 1,3\rangle}(\langle 1,3\rangle) * sup(\langle 1,3\rangle) + \\ &\quad W^{\langle 1,3\rangle}(\langle 2,3\rangle) * sup(\langle 2,3\rangle) \\ &= 0.3 * 2 + 1.0 * 4 + 0.3 * 0 = 4.6. \end{aligned}$$

14.3 Clusters of arbitrary length

To identify clusters of arbitrary length the multicluster algorithm works bottom-up by first identifying short clusters and then successively searching for longer clusters. This section proposes a method to generate new longer cluster candidates by combining appropriate shorter clusters.

The multicluster algorithm elongates the investigated patterns by one in each step. Thus the index structure must be able to calculate the position list and support for constantly growing patterns. This is achieved by the algorithm: the position list for a pattern p of length k can be calculated by using its $k-1$ prefix and suffix. Intuitively, the pattern p can only occur in a time series if its prefix starts at exactly its starting position and its suffix ends where p ends. This means that no extra database scans are

necessary to determine its occurrence - we simply take a look at all the starting positions of its prefix and determine its intersection with its suffix shifted by one. Since both are shorter by one, they must have been processed earlier.

Example.

Figure 14.1 illustrates this algorithm for the pattern $P = \langle 3, 2, 1 \rangle$. Its prefix is the pattern $\langle 3, 2 \rangle$, its suffix $\langle 2, 1 \rangle$. We can compute the positions of pattern P simply by shifting the position list of the suffix one back and intersecting it with the position list of its prefix. The prefix $\langle 3, 2 \rangle$ occurs in positions T_3, T_6, while its suffix $\langle 2, 1 \rangle$ is found at starting positions S_2, S_7, T_1, T_4. Any time series elongated by one which contains this suffix has to start one position earlier to end with this suffix. Any occurrence of $\langle 3, 2, 1 \rangle$ will thus start at $\{T_3, T_6\} \cap \{S_1, S_6, T_0, T_3\} = \{T_3\}$. And indeed $\langle 3, 2, 1 \rangle$ is found at this position as we can verify in the illustration of T.

As we can see, patterns are efficiently extended on the fly when a longer pattern is accessed for the first time.

14.4 Multicluster algorithm

Our Multicluster algorithm exploits both monotonicity properties and density computation on the index. Working in two steps, each monotonicity property on time series patterns and clusters is used. The first step searches for clusters of arbitrary length while the second step combines parallel clusters.

14.4.1 Step one: time series clustering

The multicluster algorithm is presented in Algorithm 3. First, the index structure is created using a parameter $length_{start}$ (can be set to one to mine all patterns where desired). After the construction step the index structure contains one entry for each pattern of length $length_{start}$. Next the first cluster candidate is generated. The first candidate contains all patterns, since all of them might be dense and hence could belong to a cluster. A depth first search on the index structure efficiently retrieves all different patterns stored in the database (method *queryAll*).

Algorithm 3: MC-Clustering(data db, int $length_{start}$, int $length_{max}$)

```
1  Index index(db);
2  index.initialCreate(length_start);      /* index of length_start patterns */
3  ClusterSet candSet := index.queryAll();   /* initialize candidates */
4  ClusterSet clustSetResult := ∅;            /* initialize result set */
5  for length_start to length_max do         /* store new cluster sets */
6      ClusterSet clustSet := ∅;
       /* Generate new clusters based on each candidate cluster */
7      for each clustCand in candSet do
8          markUnclassified(clustCand);       /* mark all time series */
9          for each ts in clustCand do
                   /* If time series is dense, expand to cluster */
10             if isUnclassified(ts) and isDense(ts, index) then
11                 clustSet := clustSet ∪ ExpandClust(ts, clustCand, index);
12     index.prune(length-1); /* remove excess paths, position lists */
13     clustSetResult := clustSetResult ∪ clustSet;     /* store cluster */
           /* Create new candidates using monotonicity property */
14     candSet := CreateCandCluster(clustSet, length, index);
15 return clustSetResult;
```

Next the clustering loop starts which discovers all clusters from $length_{start}$ to $length_{max}$ (may be set to *infinity*). For all possible lengths all patterns of all cluster candidates are tested if they satisfy the density property. Each unclassified dense pattern is then expanded to a cluster by the method *ExpandClust*. This method creates a new cluster and assigns each dense pattern within the transitive closure (with respect to γ) to this new cluster.

Neighborhood queries are necessary to determine the density value (method *isDense*) and the transitive closure of a pattern (method *ExpandClust*). Since all patterns of shorter length are no longer necessary after each clustering step the position lists and flags indexed for these patterns can then be discarded (method *prune*).

After a set of clusters is discovered for a specific length the cluster set is stored in the result set *clustSetResult* and a new set of cluster candidates (*candSet*) is determined based on the old cluster set (*clustSet*).

Figure 14.1illustrates this generation of a cluster candidate. This step utilizes the monotonicity property of time series patterns in Lemma 14.2: only time series patterns whose prefix and suffix are member of an already discovered cluster (and hence are dense) must be considered as members of a new cluster candidate.

The algorithm to calculate the corresponding cluster candidates is based on the discovered clusters of the previous step (Algorithm 4). The method *CreateCandCluster* loops over all patterns of all clusters and tries to extend each pattern. For this purpose the suffix of $length - 1$ is extracted from each pattern and all patterns which start with the extracted suffix are queried from the index (*prefixQuery*). Each queried pattern which satisfies the density property is then used to create an elongated pattern by concatenating the appropriate prefix and suffix. These queries are also supported by the

proposed index structure. The elongated patterns are finally assigned to a new cluster candidate.

Algorithm 4:
CreateCandCluster(ClusterSet clustSet, int length, Index index)

```
1  ClusterSet resultSet := ∅;
2  for each clust in clustSet do
3      for each ts in clust do
           /* get all indexed patterns starting with suffix(ts) */
4          tsSet exttsSet := index.prefixQuery(suffix(ts));
5          ClusterSet clustCand := ∅;
6          for each extts in exttsSet do
               /* if extension is dense store it in result */
7              if isDense(extts, index) then
8                  clustCand := clustCand ∪ ts[1] · extts ;
9          resultSet := resultSet ∪ clustCand;
10 return resultSet ;
```

Example.

Consider a cluster containing only one pattern $P = \langle 5, 1, 3, 7 \rangle$. This cluster of length 4 is extended to a cluster candidate of length 5 in the following way: First a query using the suffix of P, $\langle 1, 3, 7 \rangle$, is performed. We assume that the intersection with all possible patterns of length 4 results in a set $\{\langle 1, 3, 7, 3 \rangle, \langle 1, 3, 7, 9 \rangle\}$. Next the patterns are checked whether they are dense or not. This can be achieved by simply evaluating the annotated cluster flag (a dense pattern must belong to a cluster). Let's assume $\langle 1, 3, 7, 9 \rangle$ is dense and $\langle 1, 3, 7, 3 \rangle$ is not dense. In this case the two patterns $\langle 5, 1, 3, 7 \rangle$ and $\langle 1, 3, 7, 9 \rangle$ are used to create the elongated pattern $\langle 5, 1, 3, 7, 9 \rangle$ of length 5. This pattern is assigned to a new cluster candidate and returned to the

multicluster clustering algorithm.

14.4.2 Step two: multiclustering

Having discovered dense patterns and combined them to clusters, we identify those clusters which are parallel in the dataset (see Definition 13.4). Since for any cluster of length k all corresponding clusters of length $k-1$ have previously been detected, we use the monotonicity property presented in Lemma 14.2.

An important property of multiclusters is that a time series of a specific length may belong to no more than one cluster. Hence any position in a time series is the starting point for at most one density based cluster of a fixed length. This property is used to transform the time series database from a value representation to a cluster representation. After this transformation it is possible to discover multiclusters by efficient frequent itemset mining techniques.

We use the following representation to apply the frequent itemset mining: clusters starting at the same position in different attributes are combined to one itemset. The clusters are identified by their cluster-ids. A frequent itemset contains often occurring multiclusters in which each cluster belongs to a different attribute.

Example.

Figure 14.2 illustrates the transformation process. In the upper part, three attributes of time series S are shown, with clusters highlighted in gray colors. Assumed are the following time series clusters: $C_1 = \{\langle 1,2\rangle, \langle 1,3\rangle\}$ in S_1, $C_2 = \{\langle 3,4\rangle\}$ in S_2, $C_3 = \{\langle 2,3\rangle\}$ in S_3. They are transformed according to their positions as follows: Position 1 in the time series S would yield a

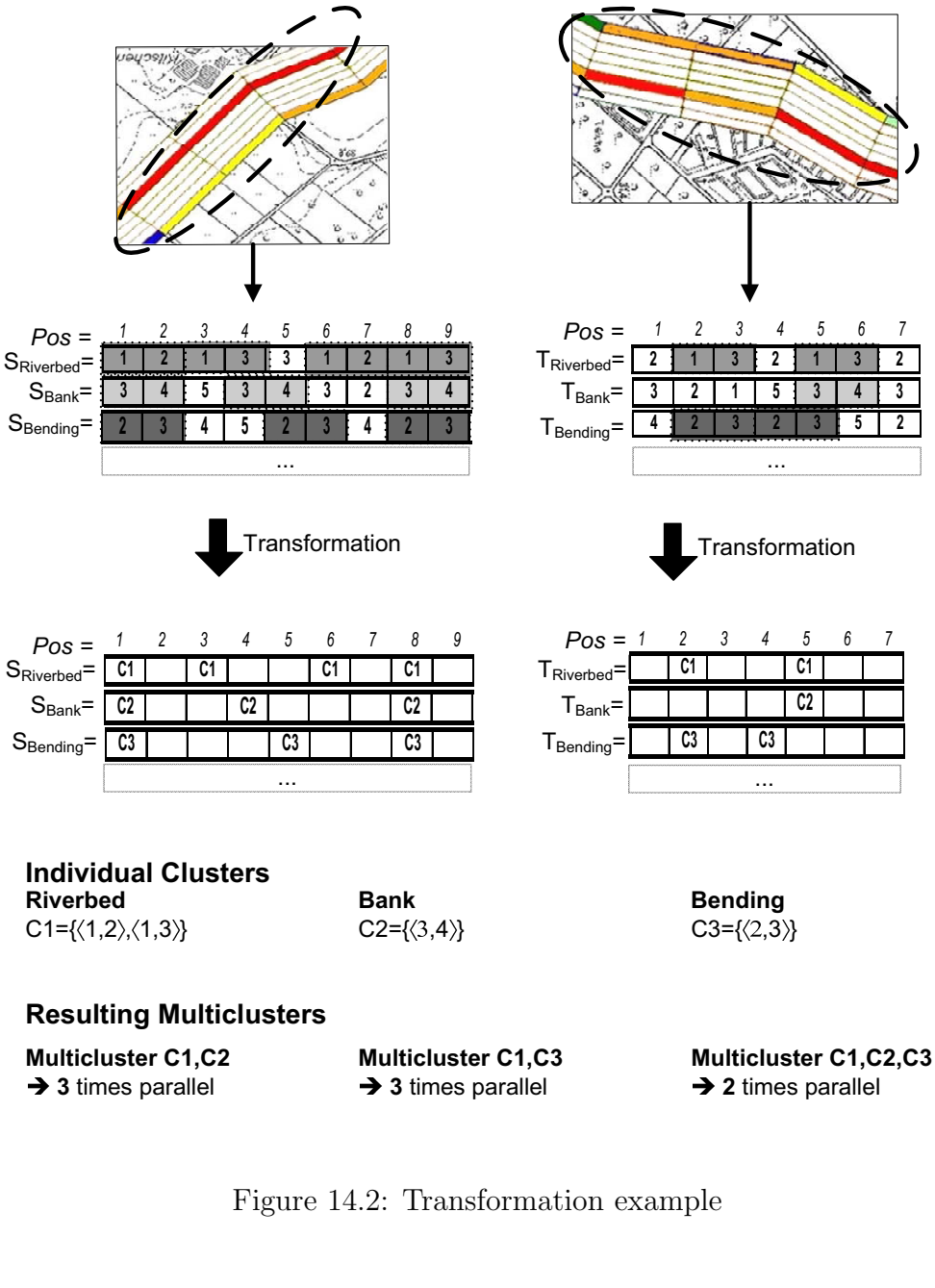


Figure 14.2: Transformation example

transaction $\{1, 2, 3\}$ since all three clusters are parallel. Position two has no starting clusters, position three only cluster C_1 and so on (lower part of Figure 14.2). We can therefore immediately derive the frequency of itemsets from the cluster information: the itemset $\{C_1\}$ occurs six times, $\{C_2\}$ four times, ..., and $\{C_1, C_2, C_3\}$ two times.

We use the Frequent Pattern (FP) growth algorithm proposed by Han et al. for the extraction of frequent itemsets [HPY00]. The tree representation is very suitable for our task since database scans are avoided. The FP-tree builds a path annotated by support values for all frequent items. To obtain frequent itemsets conditional FP-trees are constructed iteratively for each frequent item. As mentioned above we use the following representation to apply the FP-tree: one dimensional time series clusters starting at a certain position are itemsets containing cluster-ids (see Figure 14.2). The FP-tree efficiently supports determining frequent patterns with respect to ϕ. A frequent itemset contains often occurring correlated clusters in which each time series cluster belongs to a different dimension. Thus the result of the FP-tree algorithm contains multiclusters as described in Definition 13.4. In order to find multiclusters of any length the FP-tree algorithm is started for all different lengths for which clusters have been found. Since the FP-tree works extremely fast, clustering arbitrary lengths is efficient.

14.4.3 Analytical evaluation

Subspace clustering is a highly complex task, as the number of subspaces is exponential in the number of attributes. Further on, detecting multiclusters of arbitrary length is quadratic in the time series length. Consequently, multicluster detection is a challenging problem. To ensure efficiency, our algorithmic approach therefore bears on each of these aspects.

First, concerning arbitrary lengths of possible patterns, our index structure avoids re-building potentially frequent patterns from scratch. Only frequent patterns, not the actual time series nor the infrequent patterns, are actually processed when elongating clusters. For each of these patterns, compact representations are stored and position lists are kept on hard disk to reduce main memory usage. Second, only subspaces which contain clusters in single attributes are mined for multiclusters. This greatly reduces the number of potential combinations. Moreover, by mapping time series to cluster ids, the FP-growth algorithm allows for fast detection of multiclusters. Memory requirements could be further reduced by applying a secondary storage extension of the FP-tree method as suggested e.g. in [GZ04].

We validate this analysis in the experiments which show that our algorithm scales almost linearly in terms of both length and numbers of attributes on different data sets.

14.5 Experiments

We ran experiments on synthetic and real world data to evaluate the quality and performance. Synthetic data is used to evaluate different cluster parameters and to develop guidelines and heuristics for supporting users in setting up parameters. It also evidences the algorithm's scalability as well as its quality in detecting all generated clusters. Real world data demonstrates the usefulness of the results for domain experts. Our implementation is in Java, and experiments were run on Pentium 4 machines with 2.4 Ghz and 1 GB main memory.

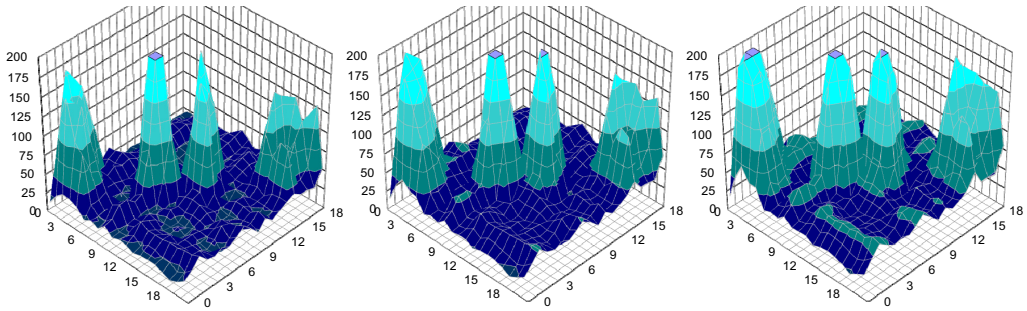


Figure 14.3: Influence of σ on density ($\sigma = 0.75, 0.9, 0.1$ from left to right)

14.5.1 Parameter setup

Several parameters can be used to tune our algorithm to domain specific needs. We therefore develop appropriate guidelines and heuristics for choosing reasonable parameter settings. To study the effect of different parameter setups on the clustering result we generated different synthetic data sets. For each data set we varied the number of time series clusters and multiclusters in the data.

As suggested in Section 13.3 we use a Gaussian kernel as a weighting function. Thus the first parameter to set is the value for σ. We propose using a density plot to determine the effect of different σ values on the density for length two patterns. The data set used to demonstrate this heuristic is eight dimensional and contains four multiclusters. Figure 14.3 illustrates the density plot for one attribute of the synthetic dataset using $\sigma = 0.75$, 0.9, and 1.0, respectively. Four separate clusters can be seen, which mainly consist of patterns of similar values (on the diagonal). Decreasing the value of σ corresponds to splitting the rightmost cluster around coordinates (16,16) into two separate clusters (leftmost illustration). As this would result in two clusters which would be a lot less frequent and distinct from the remaining three clusters, splitting is considered inappropriate and a higher value for

Param.	Usage	Setup	Exp. 1
δ	**Density Threshold** Separates noise from clusters.	see left	10
σ	**Expected Blur (Fuzziness)** Standard deviation in Gaussian Kernel. Value ~1 for categorical.	0.75-1.25	0.9
τ	**Significance Threshold** Cuts off insignificant weights. For very small value almost no error.	<<1	0.005
γ	**Compactness Parameter** Max. distance between patterns in cluster; ~1 for categorical.	1	1
ϕ	**Co-occurring Frequency** Correlates to data coverage of multicluster.	1% of the dataset	30

Figure 14.4: Parameter guidelines

σ should be chosen. On the other hand, further increasing the value of σ corresponds to merging the two clusters at the center into a single cluster (rightmost illustration). This would create a larger and more frequent cluster than the remaining two. Hence, a lower value for σ is needed to separate these clusters.

These plots, as well as further experiments, suggests a choice for σ of about one. This allows separation of clusters as well as aggregation of similar values. A choice of one for σ also reflects the fact that a deviation in one value is generally deemed tolerable in many ordinal settings. An appropriate minimum density value can also be derived from the density plot. The ordinate value which separates the clusters from the "noise floor" can be visually determined and extrapolated to longer patterns. This leads to appropriate values for the density threshold δ. (e.g. for this dataset $\delta = 10$). Note that the density plot can be created easily during the initialization of the index structure.

The remaining parameters can be determined in a straightforward man-

ner. τ is set to very small values (0.01 or less) to cut off insignificant density values. Similar patterns should be clustered together which leads to a compactness threshold of one ($\gamma = 1$) in ordinal settings like the river dataset. Finally, experts are only interested in multiclusters which cover a significant part of the dataset (e.g. one percent minimum frequency). Figure 14.4 summarizes our heuristics and parameter settings for the experiments.

Using this parameter setting the multicluster algorithm indeed finds all multiclusters in the synthetic dataset. This first experiment demonstrates the effectiveness of the multicluster algorithm and indicates that the developed heuristics help users find reasonable parameters.

14.5.2 Synthetic data

As mentioned in Section 14.1, a naive approach for detecting clusters of arbitrary length would be to re-run a density-based subspace clustering algorithm for all possible lengths. We compare the performance and the results of the multicluster algorithm with SUBCLU [KKK04], an extension of the density-based DBSCAN [EKSX96] algorithm to subspace clustering. Since SUBCLU was not developed to cluster time series we had to extend the original implementation by the density notion presented in Section 13.3.

To evaluate the scalability of our multicluster algorithm in comparison with existing approaches, we varied the length of the data time series and the number of time series in the data base. Additionally we investigated the influence of the number of different labels per time series domain on the performance of the multicluster algorithm. For this purpose we implemented a data generator which creates density connected clusters in each time series. The data generator gets the number of clusters, and the size and length of the data base as input parameters. To parameterize the position and size of

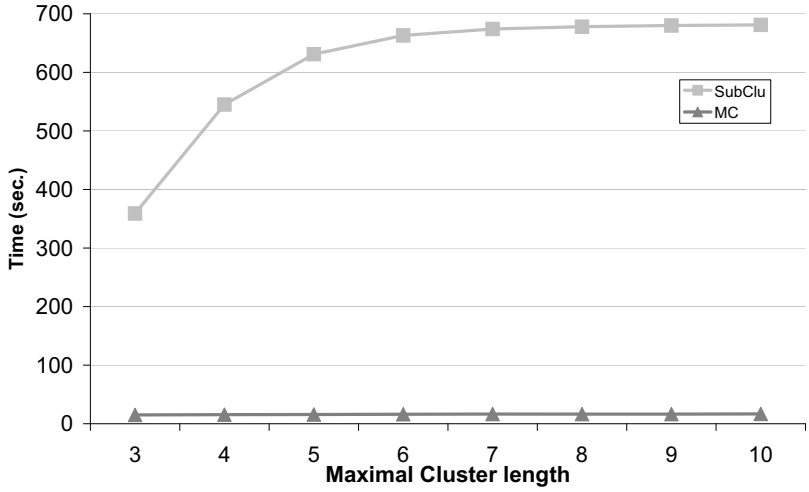


Figure 14.5: Runtime performance

a cluster, a starting time series and the number of pattern belonging to the cluster is specified for each cluster. To generate multiclusters some of these clusters are correlated to parallel multiclusters. All values not belonging to a time series cluster are uniformly distributed based on the size of the corresponding domain to generate noise.

In our first experiment we generated a data set consisting of eight attributes with twenty different labels. We hid three to six clusters of length five to ten in each time series. Each hidden multicluster has a minimum frequency of one percent and consists of three to six patterns. Figure 14.5 illustrates the runtime of both algorithms. Note that multicluster is faster by an order of magnitude. The runtime of both algorithms depends on the number of time series in a cluster. Since longer time series are less often dense the time to investigate longer multiclusters is nearly constant. As far as the quality of the result is concerned, both algorithms discovered the main patterns of the six multiclusters in the database.

In our second experiment we used the same hidden clusters as in our first

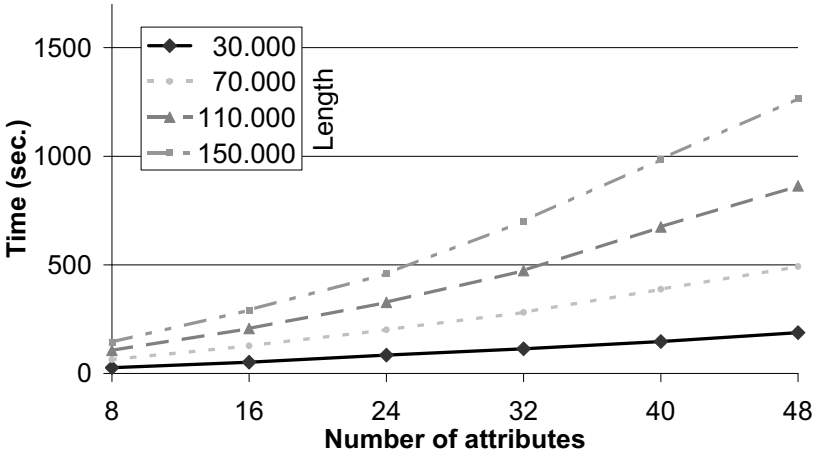


Figure 14.6: Scalability for number of attributes

experiment but varied the number of time series in the data base. Figure 14.6 shows the result for four different time series lengths (from 30,000 to 150,000). Even in 48 attributes the multicluster algorithm is capable of finding multiclusters in reasonable time. Once again, we have only slightly more than linear behavior for increasing number of time series.

To evaluate the scalability in terms of time series lengths we again used the eight-dimensional data set from our first experiment and recorded the time for time series lengths from 100,000 to 500,000. Additionally three different domains were used, depicted in Figure 14.7 as separate lines for 10, 20 and 40 ordinal values per time series attribute. In order to keep the experiments comparable we adjusted σ using our density plot heuristics from $0.6, 1.2, 2.4$, respectively. The multicluster algorithm shows linear behavior for 10 labels and slightly superlinear behavior for 20 and 40 labels. Even the largest setup with 40 different categories and $500,000$ time series segments takes less than $2,800$ seconds. This means that our algorithm is capable of handling very large databases with large domains.

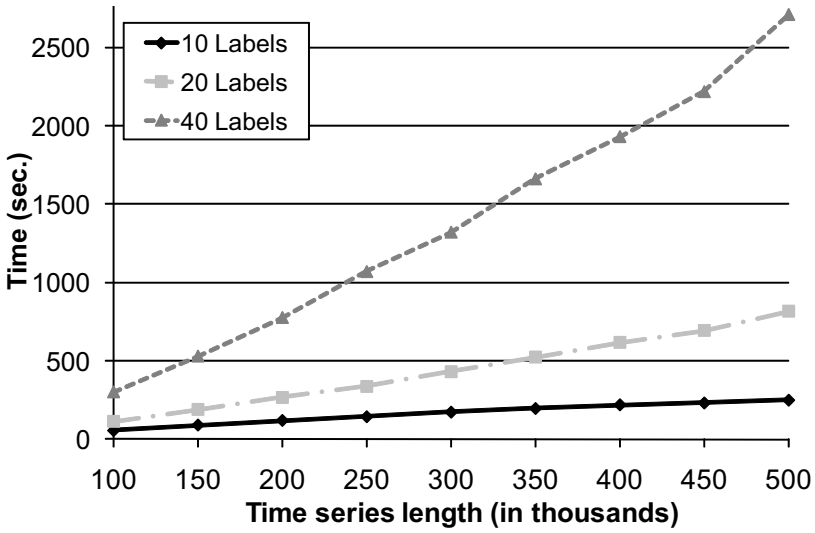


Figure 14.7: Scalability for time series length

14.5.3 Real world data

This section evaluates the effectiveness and efficiency of our algorithm on different real world datasets. Our analysis mainly focuses on the flowing water renaturation project of the European Union mentioned before. Additionally, we investigate the effectiveness of the multicluster algorithm on temporal data sets, using data from the "National Fire Danger Rating Retrieval System".

River data.

The river data set consists of 120,000 river segments describing the structural properties of a river, such as the structure of the riverbed. Experts working on renaturing rivers are interested in finding co-occurring clusters to determine those time series patterns which occur repeatedly in the river database and which allow to derive broad potential improvement through measures designated. As a final result, experts should be capable of summarizing river segments into measure packages, ranking them according to the qual-

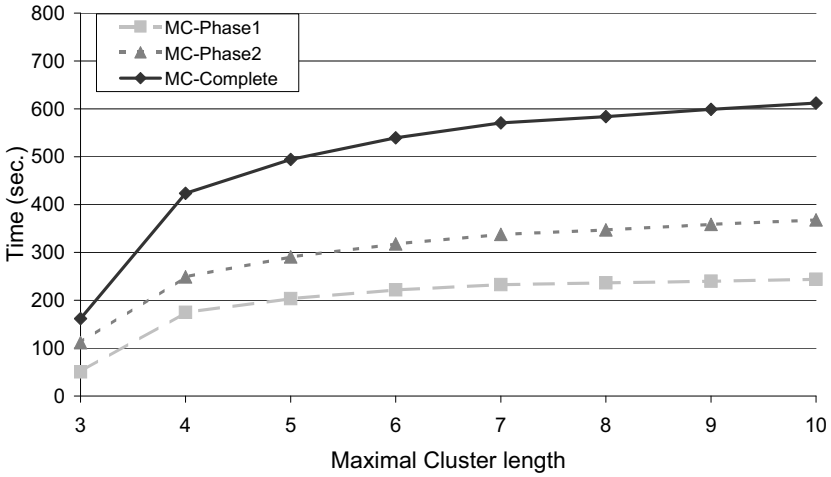


Figure 14.8: Scalability for maximal cluster length

ity enhancements expected. This can then be used to create supra-regional schedules for political decision-making.

For the river dataset a value for σ between 0.7 and 0.85 has shaped up as a reasonable parameter. The multicluster algorithm identifies more than a thousand clusters of length four by using a value for σ of 0.7 with a corresponding value for δ. Many of those clusters do not show when mining clusters of length five. By using a higher value for σ more patterns are considered similar and cluster count drops between lengths five and six. For this dataset experiments have shown that independent of the σ value, multiclusters in many attributes can be found only at length four to six. Beyond this length, parallel patterns are rare. This an interesting result for the hydrologists studying this data. They find that they should develop sensible packages of measures in this range and estimate costs for packages of about 500 meters river improvement.

Figure 14.8 illustrates the runtime of the multicluster algorithm for phase one (searching for clusters of arbitrary length) and for phase two (searching for multiclusters) separately as well as for both. As we can see, the time

requirements for mining all clusters of arbitrary length are distributed rather evenly between the two phases. The total time for mining multiclusters of arbitrary pattern length demonstrates the efficiency of our approach. Note that with increasing maximal length, few additional dense patterns are detected such that the increase in time consumption slows down. The steepest ascent is for lengths of up to six, which corresponds to our result findings.

Similar to the synthetic dataset, we also applied SUBCLU on this real world dataset. However, even the first iteration of SUBCLU for multiclusters of length ten did not finished after ten hours. One reason why multicluster works extremely faster on the investigated dataset than SUBCLU are the time consuming neighborhood queries on the nineteen-dimensional data points. Even the use of index structures like the R-Tree would not speed up these neighborhood queries in these high dimensionalities. Another reason is the efficient combination of dimensions using an FP-tree as done in our multicluster algorithm.

For hydrologists to see how the detected multiclusters are distributed and check in which areas the designed measures are applicable or where additional measure engineering is required, we visualize multiclusters in a geographical information system. An example for a cluster visualization is given in Figure 14.9. Those river segments which are part of the cluster are marked by dark lines. The corresponding cluster summary is visualized on the left side. It indicates the cluster length of five river sections (top to bottom) as well as the cluster range of ten out of nineteen attributes (left to right).

Additional tools for hydrologists give detailed information beyond this summary. Experts may browse clusters for an overview of the attributes contained in clusters, their value distributions as well as their occurrence in the rivers. Moreover, individual attributes and river values may be checked

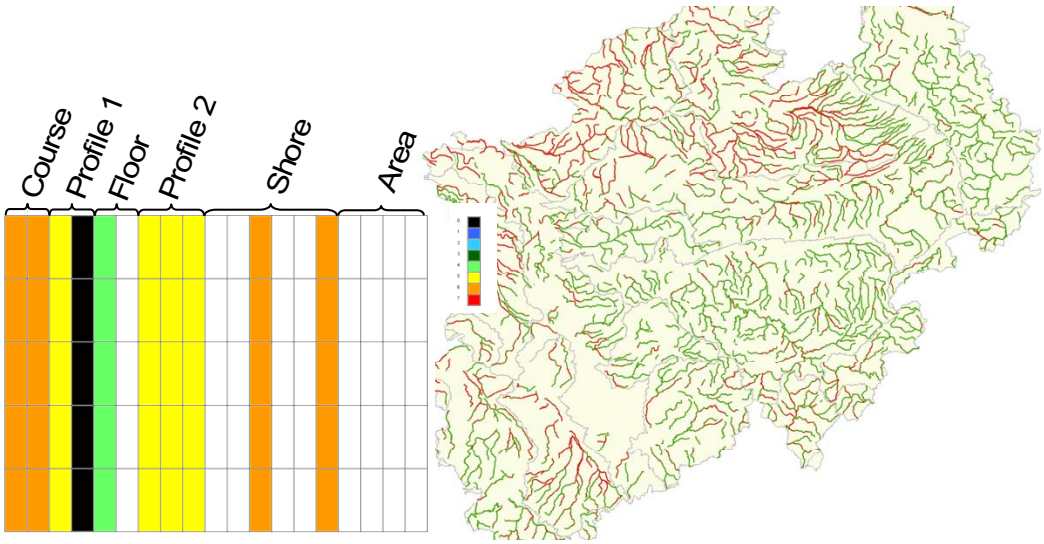


Figure 14.9: Cluster visualization for river data

for containment in clusters. By joining this information with the packages of measures designed, field experts can easily identify areas which are not yet met by measures.

Annotating the measures with cost and duration information, political decision making is supported. Hydrologists used the information derived from these clusters to build a decision support system [Bar05] that gives concise summaries as well as detailed inspection of the state of the rivers as it is now as well as a prognosis for future development depending on the packages of measures chosen.

Weather data.

The second real world dataset is retrieved from the "National Fire Danger Rating Retrieval System" which provides senor data from various weather stations. We use hourly weather variables from 1998 to 2005 about temperature, relative humidity, wind speed and direction, weather status etc.

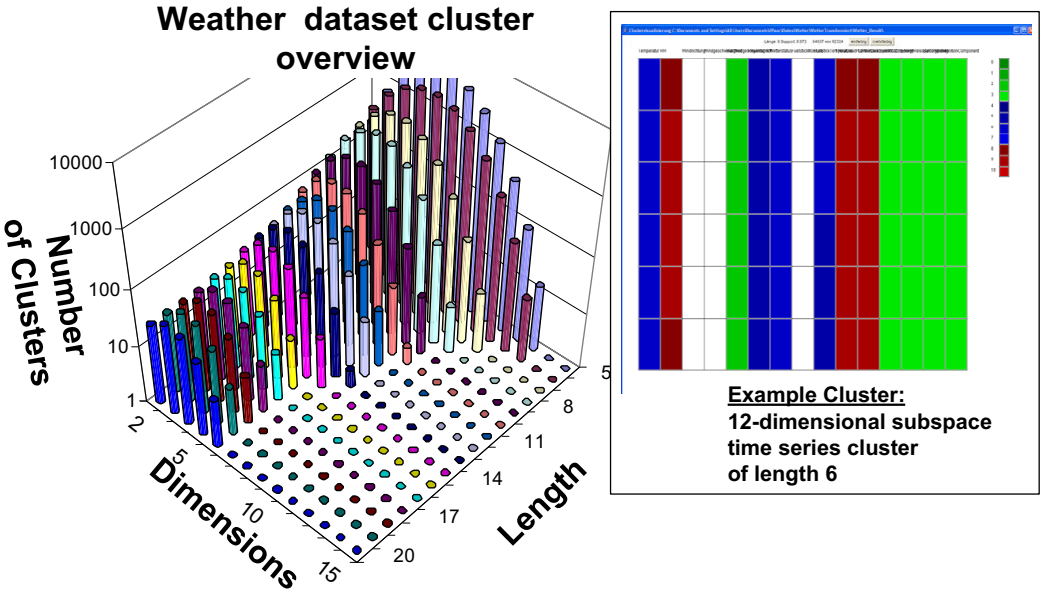


Figure 14.10: Cluster visualization for weather data

Overall the dataset contains fifteen variables measured at 27,048 time points [GA05]. This weather dataset contains a mixture of real valued and categorical variables. The continuous attributes are transformed into ordered categories using the transformation technique presented in [LKLC03]. After normalization we used ten symbols for the quantization.

The weather dataset allows us to determine the quality of the multicluster algorithm. Based on the weather variables the "National Fire Danger Rating System" calculates some indices like the "Ignition Component" which relates the probability that a fire that requires suppression action will result if a firebrand is introduced into a fine fuel complex. These indices are influenced by many variables measured in the rating system. They form natural parallel patterns in the data which we use to demonstrate that the multicluster algorithm does find these multiclusters. By inspecting the result we indeed find the correlation between the index variables and all measured sen-

sor data. Figure 14.10 (right side) illustrates an example cluster which nearly contains all attributes. The cluster overview (Figure 14.10 left side) shows many clusters containing nearly all attributes (13 and 14) with a length of up to seven. Thus the multicluster algorithm indeed finds the intrinsic structure in a continuous valued dataset.

14.6 Conclusion

Domain experts are supported in their need for pattern detection in time series databases such as river quality records. The information derived using our approach is incorporated in a decision support system devised by hydrologists. Future work will concentrate on integrating geographic information systems' data to handle non-sequential spatial information as well and to extend the model to graph structures [KKSS04].

Conclusion and Future Work

Conclusion

In this work, we proposed efficient and effective database techniques for similarity search and clustering in high dimensional large multimedia databases. Focusing on prevalent data types in many application domains, namely histograms and time series, we have discussed effective similarity models. As these models are computationally complex, direct usage in interactive applications on large and high dimensional databases is infeasible. We suggested efficient index based algorithms with novel filter functions for fast multistep retrieval and mining.

In Part I, we have presented the Earth Mover's Distance, an adaptable similarity model that incorporates knowledge on the feature space into a ground distance. The Earth Mover's Distance has been successfully used in a number of small scale applications, yet its computation complexity was a hindrance for larger scale settings. Our novel filter functions allow for substantial efficiency gains for multidimensional indexing structures from the R-tree family, as well as for efficient sequential scan based methods like the VA-file approach, or by dimensionality reduction of very high dimensional histogram features. By proving quality guarantees like the completeness of filter functions, we ensure that efficiency is not at the cost of effectivity. Parts of this thesis have been successfully published in conference proceedings or journals. In particular, excerpts from Chapters 2 to 4 have been presented at the 2006 IEEE international conference on data engineering [AWS06]. A demo based on the techniques described in Chapter 5 was shown at the 2007 IEEE international conference on data engineering [AKS07]. Chapters 6 to 7 have been accepted for the 2008 IEEE international conference on data engineering [AWMS08]. Further contents have been submitted for publication.

Part II presents a novel indexing structure for efficient time series similarity search, the TS-tree, that supports the most widely used distance functions for time series, Euclidean Distance and Dynamic Time Warping. Exploiting the inherent sequential nature of time series, our approach provides very compact and descriptive directory information that allows for pruning of substantial parts of the search space for very fast retrieval. The TS-tree approach has been accepted for the 2008 international conference on extending database technology [AKAS08].

We propose a novel effective and efficient approach for subspace clustering of high dimensional histogram data in Part III. Following the density-based clustering paradigm, we propose an efficient density-conserving grid structure that allows for lossless mining of clusters in arbitrary subspaces of the data at very fast processing runtimes.

Part IV introduces our new time series subspace clustering approach that is capable of efficiently detecting patterns in both arbitrary subspaces of the attributes as well as in arbitrary lengths of these patterns. Our novel indexing structure manages potential patterns and subspace clusters, thus successfully avoiding additional costly database scans. Some ideas introduced in this part have been presented at a workshop on spatial and spatio-temporal data mining at the 2007 IEEE international conference on data mining and have subsequently been accepted for publication in the international journal on knowledge and information systems [AKGS06, AKGS08].

Our work thus allows for retrieval and clustering of multimedia data where this was previously infeasible due to poor runtime performances. Our techniques provide the efficiency gains required for large scale mining and similarity search in a number of applications.

Future work

The investigations in this thesis create potential for future work in a number of areas:

Heterogeneous data

In a number of applications, multimedia data is best described using a combination of histograms and time series features. Typical examples include sensor data, where the actual measurements form natural time series, while the descriptive information on the type of measurements is usually recorded as tuples of attribute values. Our indexing and mining techniques in principle generalize to these heterogeneous data types, yet require novel generalized and consistent models. For example, the TS-tree method provides enormous efficiency gains for retrieval of long time series. Its compact directory information and powerful pruning have been successful in terms of efficiency benefits for time series data. This idea could be extended to histogram data or heterogeneous data using appropriate dimensionality reduction and quantization methods. For meaningful similarity search, the overall distance function has to reflect the respective attribute types. Such a generalized distance function for the different data types is likely to increase the complexity of retrieval and mining, generating novel requirements for efficient algorithmic processing, e.g. with respect to fast computation of overall mindist functions in query processing.

Video data

An even more complex type of heterogeneous data, namely video data, is increasingly prevalent in a number of applications. It can be modeled as

high dimensional time series. Each image can be represented via histogram features at the respective time points of the video. For histograms, the Earth Mover's Distance is an effective distance measure, for time series Dynamic Time Warping has been successfully used to model (dis-)similarity. Thus, the combination of Earth Mover's Distance and Dynamic Time Warping leads to a very promising overall distance function for video data. However, this combination is very costly to compute. Based on the results presented in this thesis, our filter functions could be unified to allow for feasibility of this model, providing a novel, potentially very effective video data distance function for large video databases. With respect to our filter quality criteria, the major issue in combination or extension of these filter functions is guaranteed lossless query processing.

Subspace mining

For data mining, we have demonstrated that subspace clustering is an effective method for mining in high dimensional databases for which we provide efficient algorithms. In many other data mining tasks, such as classification, high dimensional data poses challenges very similar to those found in clustering. As the number of attributes increases, patterns in the data are obscured by locally irrelevant attributes. Consequently, subspace mining could be beneficial for these data mining tasks as well. For classification, subspace clustering may identify locally relevant attributes that allow for locally adoptive classification. The challenge then lies in meaningful combination of subspace cluster information, possibly conflicting with respect to the class label and/or cluster membership of objects. A generalized model that combines this subspace information would be capable of classification in high dimensional data with respect to locally varying attribute relevance.

Subspace outlier detection

Likewise, in outlier detection, traditional approaches have successfully used full space clustering methods to identify outliers as those objects that do not follow the prevalent patterns in the data. For very high dimensional data, the clear distinction between clusters and outliers is lost, as discussed also in this thesis. Hence, focusing on locally relevant attributes may allow identification of deviating objects. This requires an outlier model that integrates possibly overlapping subspace cluster results such that those objects that agree with very few or very low dimensional patterns only can be clearly distinguished from those that agree with the main patterns as detected by subspace clustering.

In summary, this thesis provides novel techniques for effective and efficient retrieval and mining with substantial further research potential for other types of data as well as for other retrieval and mining tasks.

Bibliography

[ABKS98] Michael Ankerst, Bernhard Braunmüller, Hans-Peter Kriegel, and Thomas Seidl. Improving adaptable similarity query processing by using approximations. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB), New York, USA*, pages 206–217, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[AFGY02] Jay Ayres, Jason Flannik, Johannes Gehrke, and Tomi Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the ACM International Conference on Knowledge Discovery in Databases (SIGKDD)*, pages 429–435, New York, NY, USA, 2002. ACM.

[AFS93] Rakesh Agrawal, Christos Faloutsos, and Arun N. Swami. Efficient similarity search in sequence databases. In *Proceedings of the 4th International Conference of Foundations of Data Organization and Algorithms (FODO)*, pages 69–84, London,UK, 1993. Springer Verlag.

[AGGR98] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of*

the ACM SIGMOD International Conference on Management of Data, pages 94–105, New York, NY, USA, 1998. ACM.

[AKAS08] Ira Assent, Ralph Krieger, Farzad Afschari, and Thomas Seidl. The TS-tree: Efficient time series search and retrieval. In *Proceedings of the 11th International Conference on Extending Data Base Technology (EDBT), Nantes, France*, 2008.

[AKGS06] Ira Assent, Ralph Krieger, Boris Glavic, and Thomas Seidl. Spatial multidimensional sequence clustering. In *Proceedings of the 1st International Workshop on Spatial and Spatio-temporal Data Mining (SSTDM), in conjunction with the 2006 IEEE International Conference on Data Mining (ICDM)*, 2006.

[AKGS08] Ira Assent, Ralph Krieger, Boris Glavic, and Thomas Seidl. Clustering multidimensional sequences in spatial and temporal databases. *International Journal on Knowledge and Information Systems (KAIS) (to appear)*, 2008.

[AKKS99] Michael Ankerst, Gabi Kastenmüller, Hans-Peter Kriegel, and Thomas Seidl. Nearest neighbor classification in 3d protein databases. In *Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 34–43. Lecture Notes in Computer Science, Springer, 1999.

[AKS07] Ira Assent, Ralph Krieger, and Thomas Seidl. AttentionAttractor: Efficient video stream similarity query processing in real time. In *Proceedings of the IEEE 2007 International Conference on Data Engineering (ICDE)*, pages 1509–1510, 2007.

[AN07] Arthur Asuncion and David Newman. University of california machine learning repository, 2007.

[AS95] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proceedings 1995 IEEE International Conference on Data Engineering (ICDE)*, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.

[AWMS08] Ira Assent, Marc Wichterich, Tobias Meisen, and Thomas Seidl. Efficient similarity search using the Earth Mover's Distance for large multimedia databases. In *Proceedings of the IEEE 24th International Conference on Data Engineering (ICDE), Cancun, Mexico*, 2008.

[AWS06] Ira Assent, Andrea Wenning, and Thomas Seidl. Approximation techniques for indexing the Earth Mover's Distance in multimedia databases. In *Proceedings of the 2006 IEEE International Conference on Data Engineering (ICDE)*, 2006.

[AY00] Charu C. Aggarwal and Philip S. Yu. Finding generalized projected clusters in high dimensional spaces. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 70–81, New York, NY, USA, 2000. ACM.

[Bar05] Sabine Bartusseck. Regelbasiertes Entscheidungsunterstützungssystem (DSS) zur Bewertung von Maßnahmenplänen gemäß EG-WRRL. *Forum für Hydrologie und Wasserbewirtschaftung*, 10, 2005.

[BBK+00] Stefan Berchtold, Christian Böhm, Hans-Peter Kriegel, Jörg Sander, and H.V. Jagadish. Independent quantization: An in-

dex compression technique for high-dimensional data spaces. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pages 577–588, Washington, DC, USA, 2000. IEEE Computer Society.

[BBK01] Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Survey*, 33(3):322–373, 2001.

[BC94] Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *AAAI Workshop on Knowledge Discovery in Databases*, pages 229–248, 1994.

[BGRS99] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is nearest neighbors meaningful. In *Proceedings of the 7th International Conference onDatabase Theory (ICDT)*, pages 217–235, London, UK, January 1999. Springer Lecture Notes in Computer Science.

[BKK96] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB)*, pages 28–39, San Francisco, U.S.A., 1996. Morgan Kaufmann Publishers.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the ACM*

SIGMOD International Conference on Management of Data, pages 322–331, New York, NY, USA, 1990. ACM.

[BKSS94] Thomas Brinkhoff, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. Multi-step processing of spatial joins. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 197–208, New York, NY, USA, 1994. ACM.

[BM70] Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indexes. In *Record of the 1970 ACM SIGFIDET Workshop on Data Description and Access, November 15-16, 1970, Rice University, Houston, Texas, USA (Second Edition with an Appendix)*, pages 107–141. ACM, 1970.

[BM72] Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–189, 1972.

[BU77] Rudolf Bayer and Karl Unterauer. Prefix B-trees. *ACM Transactions on Database Systems*, 2(1):11–26, 1977.

[Cha96] Chris Chatfield. *The Analysis of Time Series: an Introduction*. Chapman and Hall, 1996.

[CN04] Yuhan Cai and Raymond Ng. Indexing spatio-temporal trajectories with chebyshev polynomials. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 599–610, New York, 2004. ACM Press.

[CPZ97] Paulo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of 23rd International Conference on Very Large Data Bases, Athens, Greece*, pages 426–435, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[Dan51] G. Dantzig. *Application of the simplex method to a transportation problem*, pages 359–373. John Wiley and Sons, 1951.

[Den04] Anne Denton. Density-based clustering of time series subsequences. In *Proceedings of the 2004 IEEE International Conference on Data Mining (ICDM)*, 2004.

[DKN05] Thomas Deselaers, Daniel Keysers, and Hermann Ney. Discriminative training for object recognition using image patches. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 157–162, Washington, DC, USA, 2005. IEEE Computer Society.

[DMC+07] Thomas Deselaers, Henning Müller, Paul Clogh, Hermann Ney, and Thomas M. Lehmann. The CLEF 2005 automatic medical image annotation task. *International Journal of Computer Vision*, 74(1):51–58, August 2007.

[DMHE+06] Laura Dempere-Marco, Xiao-Peng Hu, Stephen Ellis, David Hansell, and Guang-Zhong Yang. Analysis of visual search patterns with EMD metric in normalized anatomical space. *IEEE Transactions on Medical Imaging*, 25:1011– 1021, 2006.

[DSD+04] M. Fatih Demirci, Ali Shokoufandeh, Sven J. Dickinson, Yakov Keselman, and Lars Bretzner. Many-to-many feature matching using spherical coding of directed graphs. In *Proceedings of the 8th European Conference on Computer Vision (ECCV), Prague, Czech Republic*, pages 322–335. Lecture Notes in Computer Science, Springer, 2004.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases. In *Proceedings of the 2nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–231, Portland, Oregano, 1996. AAAI Press.

[Fal96] Christos Faloutsos. *Searching Multimedia Databases by Content*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[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*, 3:231–262, 1994.

[FG99] Paolo Ferragina and Roberto Grossi. The String B-tree: A new data structure for string search in external memory and its applications. *Journal of the ACM*, 46(2):236–280, 1999.

[FRM94] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 419–429, New York, NY, USA, May 1994. ACM Press.

[FWD06] Anthony Y. Fu, Liu Wenyin, and Xiaotie Deng. Detecting phishing web pages with visual similarity assessment based on Earth Mover's Distance (EMD). *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 3(4):301–311, 2006.

[GA05] Georgia forestry commission, weather data retrieval. *http://weather.gfc.state.ga.us*, 2005.

[GD04] Kristen Grauman and Trevor Darrell. Fast contour matching using approximate Earth Movers Distance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 220–227, 2004.

[GRS99] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. A robust clustering algorithm for categorical attributes. In *Proceedings of the 1999 IEEE International Conference on Data Engineering (ICDE)*, pages 512–521, Washington, DC, USA, 1999. IEEE Computer Society.

[Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 47–57, New York, NY, USA, 1984. ACM.

[GZ04] Gösta Grahne and Jianfei Zhu. Mining frequent itemsets from secondary memory. In *Proceedings of the 2004 IEEE International Conference on Data Mining (ICDM)*, pages 91– 98, 2004.

[Had02] Marios Hadjieleftheriou. The R-tree-portal. *http://www.cs.ucr.edu/ marioh/spatialindex/*, 2002.

[HK98] Alexander Hinneburg and Daniel A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *Proceedings of the 1998 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 58–65, 1998.

[HK99] Alexander Hinneburg and Daniel A. Keim. A general approach to clustering in large databases with noise. *Knowledge and Information Systems (KAIS)*, 5(4):387–415, 1999.

[HK01] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

[HL90] Frederick Hillier and Gerald Lieberman. *Introduction to Linear Programming*. McGraw-Hill, 1990.

[HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–12, New York, NY, USA, 2000. ACM.

[HSE+95] James Hafner, Harpreet S. Sawhney, Will Equitz, Myron Flickner, and Wayne Niblack. Efficient color histogram indexing for quadratic form distance functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(7):729–736, 1995.

[JLZZ04] Feng Jing, Mingjing Li, Hong-Jiang Zhang, and Bo Zhang. An efficient and effective region-based image retrieval framework. *IEEE Transactions on Image Processing*, 13(5):699–709, 2004.

[Jol86] Ian T. Joliffe. *Principal Component Analysis*. Springer, New York, 1986.

[KCMP01] Eamonn J. Keogh, Kaushik Chakrabarti, Sharad Mehrotra, and Michael J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 151–162, New York, NY, USA, 2001. ACM.

[KCPM01] Eamonn Keogh, Kaushik Chakrabarti, Michael J. Pazzani, and Sharad Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Journal of Knowledge and Information Systems*, 3(3):263–286, 2001.

[Keo02] Eamonn Keogh. Exact indexing of dynamic time warping. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB)*, pages 406–417. VLDB Endowment, 2002.

[Keo06] Eamonn Keogh. UCR time series archive, http://www.cs.ucr.edu/~ eamonn/TSDMA/datasets.html, 2006.

[KK02] Eamonn Keogh and Shruti Kasetty. On the need for time series data mining benchmarks: A survey and empirical demonstration. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery in Databases*, pages 102–111, New York, NY, USA, 2002. ACM.

[KKK04] Karin Kailing, Hans-Peter Kriegel, and Peer Kröger. Density-connected subspace clustering for high-dimensional data. In

Proceedings of the IEEE International Conference on Data Mining (ICDM), pages 246–257, 2004.

[KKKW03] Karin Kailing, Hans-Peter Kriegel, Peer Kröger, and Stefanie Wanka. Ranking interesting subspaces for clustering high dimensional data. In *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pages 241–252, 2003.

[KKRW05] Hans-Peter Kriegel, Peer Kröger, Matthias Renz, and Sebastian Wurst. A generic framework for efficient subspace clustering of high-dimensional data. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM)*, pages 250–257, Washington, DC, USA, 2005. IEEE Computer Society.

[KKSS04] Karin Kailing, Hans-Peter Kriegel, Stefan Schönauer, and Thomas Seidl. Efficient similarity search for hierarchical data in large databases. In *Proceedings of the 9th International Conference on Extending Database Technology (EDBT)*, pages 676–693, 2004.

[KR87] Leonard Kaufman and Peter J. Rousseeuw. Clustering by means of medoids. In Y. Dodge, editor, *Statistical data analysis based on the L1-norm and related methods*, pages 405–416. North-Holland, 1987.

[KR90] Leonard Kaufman and Peter J. Rousseeuw. *Finding groups in data. An introduction to cluster analysis.* Applied Probability and Statistics, John Wiley & Sons, New York, 1990.

[KSF+96] Flip Korn, Nikolaos Sidiropoulos, Christos Faloutsos, Eliot Siegel, and Zenon Protopapas. Fast nearest neighbor search in medical image databases. In *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB)*, pages 215–226, 1996.

[KUTR06] Shingo Kuroiwa, Yoshiyuki Umeda, Satoru Tsuge, and Fuji Ren. Nonparametric speaker recognition method using Earth Mover's Distance. *IEICE Transactions on Information Systems*, E89-D(3):1074–1081, 2006.

[KWX+06] Eamon Keogh, Li Wei, Xiaopeng Xi, Sang-Hee Lee, and Michail Vlachos. LB_Keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, pages 882–893, 2006.

[LBH98] Yingmei Lavin, Rajesh Batra, and Lambertus Hesselink. Feature comparisons of vector fields using Earth Mover's Distance. In *Proceedings of the IEEE conference on Visualization*, pages 103–109, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[LBS06] Vebjorn Ljosa, Arnab Bhattacharya, and Ambuj K. Singh. Indexing spatially sensitive distance measures using multi-resolution lower bounds. In *Proceedings of the 9th International Conference on Extending Database Technology (EDBT)*, pages 865–883, 2006.

[LGT+04] Thomas M. Lehmann, Mark O. Güld, Christian Thies, Benedikt Fischer, Klaus Spitzer, Daniel Keysers, Hermann Ney, Michael Kohnen, Henning Schubert, and Berthold B. Wein. Content-based image retrieval in medical applications. *Methods of Information in Medicine*, 43(4):354–361, April 2004.

[LJF94] King-Ip Lin, H. V. Jagadish, and Christos Faloutsos. The TV-tree: An index structure for high-dimensional data. *The International Journal on Very Large Databases (VLDB Journal)*, 3(4):517–542, 1994.

[LKLC03] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery (DMKD)*, pages 2–11, New York, NY, USA, 2003. ACM.

[LSOL05] Thomas M. Lehmann, Henning Schubert, Bastian Ott, and Marcel Leisten. *Download site IRMA project.* http://phobos.imib.rwth-aachen.de/irma/datasets_en.php?SELECTED=00004, 2005.

[LSP03] Svetlana Lazebnik, Cornelia Schmid, and Jean Ponce. Sparse texture representation using affine-invariant neighborhoods. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2003.

[LUA03] LUA NRW. River quality data. http://www.lua.nrw.de, 2003.

[Mac67] James B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th*

Berkeley Symposium on Mathemtical Statistics and Probability, pages 281–297, 1967.

[MNPT06] Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos N. Papadopoulos, and Yannis Theodoridis. *R-Trees: Theory and Applications*. Springer, London, 2006.

[MSE06] Gabriela Moise, Jörg Sander, and Martin Ester. P3C: A robust projected clustering algorithm. In *Proceedings of the 6th International Conference on Data Mining*, pages 414–425, Washington, DC, USA, 2006. IEEE Computer Society.

[NBE+93] Wayne Niblack, Ron Barber, Will Equitz, Myron Flickner, Eduardo H. Glasman, Dragutin Petkovic, Peter Yanker, Christos Faloutsos, and Gabriel Taubin. The QBIC project: Querying images by content, using color, texture, and shape. In *Proceedings of the SPIE Conference on Storage and Retrieval for Image and Video Databases*, volume 1908, pages 173–187, April 1993.

[PKLL02] Pranav Patel, Eamonn Keogh, Jessica Lin, and Stefano Lonardi. Mining motifs in massive time series databases. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM)*, page 370, Washington, DC, USA, 2002. IEEE Computer Society.

[Pra97] Balakrishnan Prabhakaran. *Multimedia Database Management Systems*. Kluwer Academic Publishers, 1997.

[RGT97] Yossi Rubner, Leonidas Guibas, and Carlo Tomasi. The Earth Mover's Distance, Multi-Dimensional Scaling, and color based

image retrieval. In *Proceedings of the ARPA Image Understanding Workshop*, pages 661–668, 1997.

[RPTB99] Yossi Rubner, Jan Puzicha, Carlo Tomasi, and Joachim M. Buhmann. Empirical evaluation of dissimilarity measures for color and texture. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1165–1173, 1999.

[RT01] Yossi Rubner and Carlo Tomasi. *Perceptual Metrics for Image Database Navigation*. Kluwer Academic Publishers, 2001.

[RTG98] Yossi Rubner, Carlo Tomasi, and Leonidas Guibas. A metric for distributions with applications to image databases. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 59–66, Washington, DC, USA, 1998. IEEE Computer Society.

[Rub98] Yossi Rubner. Earth Mover's Distance source code in C. *http://www.cs.duke.edu/ tomasi/software/emd.htm*, 1998.

[SC71] Hiroaki Sakoe and Seibi Chiba. A dynamic programming approach to continuous speech recognition. In *Proceedings 7th International Congress on Acoustics*, pages 65–68, August 1971.

[SH94] Harpreet S. Sawhney and James Hafner. Efficient color histogram indexing. In *Proceedings of the IEEE International Conference on Image Processing*, pages 66–70, 1994.

[Sil86] B.W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, London, 1986.

[SK97] Thomas Seidl and Hans-Peter Kriegel. Efficient user-adaptable similarity search in large multimedia databases. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB), Athens, Greece*, pages 506–515, 1997.

[SK98] Thomas Seidl and Hans-Peter Kriegel. Optimal multi-step k-nearest neighbor search. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 154–165, 1998.

[SYUK00] Yasushi Sakurai, Masatoshi Yoshikawa, Shunsuke Uemura, and Haruhiko Kojima. The A-tree: An index structure for high-dimensional spaces using relative approximation. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB)*, pages 516–526, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[SZ04a] Karlton Sequeira and Mohammed Zaki. SCHISM: A new approach for interesting subspace mining. In *Proceedings of the 2004 IEEE International Conference on Data Mining (ICDM)*, pages 186–193, 2004.

[SZ04b] Dennis Shasha and Yunyue Zhu. *High performance discovery in time series*. Springer, New York, 2004.

[TGV+03] Rainer Typke, Panos Giannopoulos, Remco C. Veltkamp, Frans Wiering, and Rene van Oostrum. Using transportation distances for measuring melodic similarity. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR)*, 2003.

[TVW04] Rainer Typke, Remco C. Veltkamp, and Frans Wiering. Searching notated polyphonic music using transportation distances. In *Proceedings of the 12th annual ACM international conference on Multimedia*, pages 128–135, 2004.

[UHMS06] Mats Uddenfeldt, Keiichiro Hoashi, Kazunori Matsumoto, and Fumiaki Sugaya. Adaptive video news story tracking based on Earth Mover's Distance. In *Proceedings of the 2006 IEEE International Conference on Multimedia and Expo*, pages 1029–1032, July 2006.

[WP05] Xiaojun Wan and Yuxin Peng. The Earth Mover's Distance as a semantic measure for document similarity. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 301–302, New York, NY, USA, 2005. ACM.

[WSB98] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB)*, pages 194–205, 1998.

[YF00] Byoung-Kee Yi and Christos Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB)*, pages 385–394, 2000.

[ZPAS05] Mohammed Zaki, Markus Peters, Ira Assent, and Thomas Seidl. CLICKS: An effective algorithm for mining subspace clusters in

categorical datasets. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 355–356, New York, NY, USA, 2005. ACM.

[ZPAS07] Mohammed Zaki, Markus Peters, Ira Assent, and Thomas Seidl. CLICKS: An effective algorithm for mining subspace clusters in categorical datasets. *Data & Knowledge Engineering (DKE)*, 60(1):51–70, January 2007.

[ZS03] Yunyue Zhu and Dennis Shasha. Warping indexes with envelope transforms for query by humming. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 181–192, New York, NY, USA, 2003. ACM Press.

Lebenslauf - Curriculum Vitae

Persönliche Daten

Dipl. Inform. Ira Assent

Frankenbergerstr. 33

52066 Aachen

Telefon: 0241-9010782

Email: assent@cs.rwth-aachen.de

Geburtsdatum und -ort: 30.04. 1975 in Aachen

Ausbildung

1981-1985	Gemeinschaftsgrundschule Zweifall
1985-1994	Ritzefeld-Gymnasium, Stolberg (Rhld.), Abitur
1991-1992	Summersville High School, Missouri, USA, High School Diploma
1994-1995	Mathematikstudium mit Nebenfach Informatik, RWTH Aachen
1995-2003	Informatikstudium mit Nebenfach Mathematik, RWTH Aachen, Diplom
Sommer 1999	Ausslandssemester, ULB, Brüssel, Belgien
Sommer 2000	Studiensemester, IFU, Hamburg
Seit 2003	Promotionsstudium Informatik, RWTH Aachen
Sommer 2006	Forschungsaufenthalt, Ochanomizu Universität, Tokio, Japan

Berufserfahrung

1996-1998	Studentische Hilfskraft, Hochschuldidaktisches Zentrum, RWTH Aachen
1996-2003	Selbständige Organisation und Durchführung von Informatikseminaren
1999-2000	Studentische Hilfskraft, Frauenbüro, RWTH Aachen
2001-2003	Studentische Hilfskraft, TorqControl GmbH, Herzogenrath
Seit 2003	Wissenschaftliche Mitarbeiterin, RWTH Aachen

Fremdsprachen

Deutsch (Muttersprache), Englisch, Französisch (beide fließend), Spanisch (fortgeschritten), Japanisch, Italienisch (beide Grundkenntnisse), Latinum.