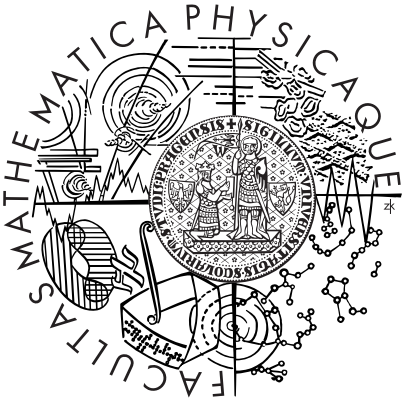
Charles University in Prague
Faculty of Mathematics and Physics

HABILITATION THESIS



RNDr. Jakub Lokoč, Ph.D.

Efficient Retrieval Using Feature Signatures

Prague, October 2015

Efficient Retrieval Using Feature Signatures

Habilitation thesis

Jakub Lokoč
October, 2015

lokoc@ksi.mff.cuni.cz
http://www.ksi.mff.cuni.cz/∼lokoc
http://siret.ms.mff.cuni.cz/lokoc

Charles University in Prague
Faculty of Mathematics and Physics
Department of Software Engineering
Malostranské nám. 25
118 00, Prague 1
Czech Republic

Acknowledgments

I would like to thank to all my colleagues and co-authors of my papers for their valuable time, influential discussions and intensive cooperation. In the first place my thanks go to members and Ph.D. students of the *Siret Research Group* (SRG), especially to the most involved members Tomáš Skopal, Přemysl Čech, Tomáš Grošup, Juraj Moško and Martin Kruliš, as well as all the excellent students whose Bachelor/Master theses and SW projects helped to investigate and form various research results. Secondly, I am very grateful for the opportunity to visit during my research stay Prof. Thomas Seidl from RWTH Aachen in Germany and Prof. Laszlo Böszörmenyi from Alpen Adria University in Klagenfurt, Austria and to get inspired with their interesting work and research topics. For helpful comments, suggestions, and ideas that enabled to improve our results I thank to all the anonymous reviewers of our papers, audience at our presentations, and all the scientists I had the pleasure to meet and talk to. For valuable comments regarding the commentary section, I would like to thank to Irena Holubová and Tomáš Skopal. And, last but not least, I am very thankful to my family whose support and understanding enables me to do the job I like.

Contents

Preface

The proposed thesis presents selected results of the author's research in the area of similarity retrieval using feature signatures and adaptive distance measures. The research has been carried out at the Faculty of Mathematics and Physics of the Charles University in Prague in years 2010–2015, mainly within the *Siret Research Group* (SRG)[1] lead by Associate Professor RNDr. Tomáš Skopal, Ph.D.

The results are presented as a collection of six selected papers [47, 53, 59, 86, 37, 49], where all of them focus on specific subproblems related to efficient retrieval using models based on feature signatures. The papers are presented in separate chapters (2–7) in their camera-ready forms (of *the Lecture Notes in Computer Science*, *the Information Systems journal*, *IEEE Transactions on Knowledge and Data Engineering journal*, and *the Distributed and Parallel Databases journal*), whereas the unifying commentary is provided in Chapter 1. Prior to summarizing the papers, the commentary provides a motivation, introduces the problematic and briefly surveys related state-of-the-art results. We conclude and outline directions of our current and future research in Chapter 8.

The research included in the selected papers has been supported by several grants, namely GAČR P202/11/0968, GAČR P202/12/P297, GAČR 201/09/0683 and GAUK 910913.

Prague, October 2015

Jakub Lokoč

[1] http://siret.ms.mff.cuni.cz

Chapter 1

Commentary

1.1 Introduction

The volume, complexity and diversity of digital data that can be automatically collected by devices measuring various physical effects (like sound, light, temperature, preasure, etc.) increase every year with new advancements of sensoring and recording technologies. The devices can work non-stop creating huge repositories of data collected with a specified recording precision, providing a detailed record for an observed event. Especially, the volume of the multimedia data like images and videos continuously grows exponentially forming a significant part of the digital universe, where the overall volume of the digital universe is reported to grow to 40,000 exabytes in 2020 [31]. The reason is that the multimedia data can be simply recorded and uploaded online by billions of users. Furthermore, also various industrial [8] or medical [50] projects start to include the data into their standard processes and workflows. However, the format of the collected multimedia data is quite simple, without any structure or semantic information, where the storage format is usually adapted just for displaying devices or limited storage capacity (e.g., JPEG compressed matrix of RGB pixels). Thus, the data comprising potentially a lot of useful but hidden information represents a true challenge for multimedia retrieval.

In order to design a successful multimedia retrieval system, the system architects have to address two orthogonal directions – *effectiveness* and *efficiency*. The effectiveness, corresponding to the precision of the retrieval, is probably the most crucial property of each retrieval system, because it

guarantees users a minimal level of relevancy of the results. The efficiency, representing the speed of query evaluation, is gaining in importance with the increasing size of the collections where sequential query evaluation can be too time consuming. In cases when it is impossible to design a sufficiently efficient solution, the systems often switch to approximate retrieval that can trade the effectiveness for efficiency. However, such trade-off is possible only if the effectiveness is not the most crucial part of the system.

Beside the classical trade-off between effectiveness and efficiency, the systems often rely on simple and intuitive query interfaces in order to let users to conveniently query and browse a multimedia database. Some systems go even further considering the *entertainment* of the retrieval, where the systems assume that some classes of users could spend more time on searching if the retrieval process is entertaining. Especially novel devices equipped with limited displays and control interfaces call for systems with simple, intuitive and entertaining novel interfaces.

In the following sections, we present two general approaches that have been successfully applied in the multimedia retrieval area during last decades.

1.1.1 Metadata-based retrieval

A popular option to search the multimedia data is by making use of metadata that can be associated with specific multimedia objects and thus the retrieval can be accomplished by searching in the metadata files. Usually, the metadata consist of keywords representing names or properties of potentially searched items, and/or the metadata can form a hierarchical/graph structure where data objects are linked to reference/relevant objects. The most straightforward way to create such metadata is to employ users (or domain experts in cases a deep domain knowledge is required) and let the users to annotate the data. However, such approach is not always feasible. Especially for immense collections and non-trivial annotation tasks a large crowd of experienced annotators is necessary. On the other hand, for simple annotation tasks (e.g., marking objects on photographs) there are already well-established web portals considering crowdsourcing as a powerful way to annotate data (e.g., the Amazon Mechanical Turk [77]). It has been also shown, that the power of the crowd can help experts to be more effective and efficient [28]. Furthermore, there are emerging specialized social networks that can connect a large number of experts, allowing them to annotate and discuss data related to their domain/expertise [88]. As a side effect, a lot of

metadata can be extracted from such social networks, enabling various forms of retrieval in the underlying multimedia data.

The World Wide Web, as one of the most general "social networks", is already effectively mined for automatic annotations of multimedia data appearing on web pages. Google[1], Bing[2], Yahoo[3] and other titans of the Internet searching have demonstrated that associating specific keywords from the surrounding text with the multimedia data can be used for effective and efficient retrieval (without analyzing the content of the data). As the most popular search portals have millions of users, the portals also analyze the behavior of the crowds during searching and use all the collected information to improve the rankings of the results. Although this approach seems to be promising for mainstream retrieval tasks where many users share the same search intents over general datasets, for domain-specific data without any preliminary annotations, with security restrictions, and without armies of authorized experts searching the data every day, this general search-engine-based approach cannot be employed.

1.1.2 Content-based retrieval

As an alternative to metadata-based retrieval approaches, *content-based retrieval* approaches try to analyze, understand and utilize raw contents of the multimedia data for effective retrieval [25]. Since raw contents of the data comprise just simple low level features (e.g., pixels), feature extraction techniques are often employed to represent data in the form of so-called *descriptors* that are more convenient for content-based multimedia retrieval. The descriptors can be created for specific retrieval tasks using different types of features present in raw data. For example, different descriptors are required to search for images with similar global distribution of colors and texture, images containing objects of the same class (e.g., images of all dogs), paintings of the same author (searching for a painting style), or medical images representing the same diagnosis. The descriptors for such tasks can be designed manually by domain experts or learned from training data.

Given a suitable descriptor for a particular retrieval task, a multimedia retrieval system has to provide a friendly and intuitive query formulation interface. Many recent systems employ the query-by-example paradigm, where

[1] www.google.com
[2] www.bing.com
[3] www.yahoo.com

users provide a query object to enter their search intents. The systems transform the query object into the descriptor which is employed in the retrieval process. Since searched objects are usually represented by descriptors that more or less differ from the query descriptor, similarity search using an effective similarity model is preferred.

The similarity model is defined as a pair $(\mathbb{U}, \sigma)$, where $\mathbb{U}$ represents an object descriptor universe and σ represents a similarity measure $\sigma : \mathbb{U} \times \mathbb{U} \rightarrow \mathbb{R}$, assigning a numerical similarity value to each pair of descriptors. The descriptor universe is usually determined by a feature extraction function that extracts representative features from multimedia objects to a feature space $\mathbb{F}$ and aggregates them. The similarity function is often modeled as a distance function $\delta : \mathbb{U} \times \mathbb{U} \rightarrow \mathbb{R}_0^+$, where the lower the distance between two descriptors, the higher their similarity, and vice versa. The distance function is used to rank and sort all database objects according to a query object $q \in \mathbb{U}$, where the top ranked objects (i.e., objects $o_i \in \mathbb{U}$ with the lowest distance to q) are usually returned as the answer of a similarity query[4].

There are several sources, where users can take a query object representing their search intents. In hybrid systems combining metadata-based and content-based retrieval, the query object can be obtained using previous keyword-based search. The query object can be also painted as a sketch, or uploaded by a user from his/her camera. In some retrieval scenarios, the query object does not have to necessarily represent a clear search intents of a user. For example, in multimedia exploration [11] users can just browse and investigate an unknown data collection without any form of annotation. In such scenarios, the query object can represent just a mediator (or link) to quickly browse to another view of the data. In specific retrieval tasks, the query can be formulated directly using descriptor features. Especially descriptors that are designed as sets of features with clear semantics for users can be used to formulate query sketches. For example, users may specify simple color regions in a key-frame sketch to search for key-frames (or a sequence of key-frames) represented by position-color feature signatures [54, 18, 17].

The success of the similarity search approach in a multimedia retrieval system depends on many factors, where the effectiveness of the employed similarity model plays one of the most crucial roles. Despite subjectivity of

[4]Given dataset $\mathbb{S} \subset \mathbb{U}$, a query object $q \in \mathbb{U}$ and $r_q \in \mathbb{R}_0^+, k \in \mathbb{N}$, two frequently used similarity queries are the range query $R(q, r_q) = \{o \in \mathbb{S}; \delta(q, o) \leq r_q\}$ and the k nearest neighbor query $kNN(q) = \{\mathbb{X} \subset \mathbb{S}; |\mathbb{X}| = k \wedge \forall x \in \mathbb{X}, y \in \mathbb{S} - \mathbb{X} : \delta(q, x) \leq \delta(q, y)\}$.

similarity perception, the effectiveness of a similarity model is often evaluated automatically using ground-truth datasets (e.g., [67, 76, 26]) representing specific retrieval tasks. Given a ground-truth dataset, the effectiveness of a similarity model can be evaluated by various performance indicators, for example, precision recall curve, top-k precision or mean average precision. Such indicators are often used to compare the effectiveness of two different similarity models on a particular ground truth dataset.

1.1.3 Thesis objectives

In this thesis, we investigate traditional approaches that model contents of multimedia data using manually designed descriptors representing distributions of selected features and where the similarity between two multimedia objects is modeled as a distance function on the distributions. The feature distributions are usually aggregated into *feature histograms* with predefined fixed bins. In this work, we focus on models based on more general *feature signatures* that can flexibly represent contents of modeled multimedia objects, depending on the complexity and specific contents of the objects. However, the flexibility comes at the cost of more expensive feature extraction and similarity evaluation, compared to models based on feature histograms. Therefore, the main objective of our work was to investigate and design novel efficient approaches for models based on feature signatures. During last five years, we have investigated methods for efficient extraction of feature signatures and efficient similarity search algorithms using expensive adaptive distance measures. The thesis is based on our 20 conference/journal papers focusing on effective, efficient or entertaining retrieval using models based on feature signatures, where four selected journal papers and two conference papers form chapters of this thesis. Although we have primarily focused on efficient multimedia retrieval using feature signatures, we have preferred the design of more general methods that can be applied with other similarity models. For example, for efficient query processing we have focused on new metric indexing approaches [86, 59] that can be used for arbitrary descriptors and metric distances. We have presented novel ptolemaic indexing approaches [37] that can be used for models employing popular Quadratic Form or Euclidean distance. We have investigated parallel processing using many-core CPU and multi-core GPU platforms for efficient metric filtering [49] and feature extraction based on k-means clustering [48]. We have also investigated applications for models based on feature signatures – multime-

dia exploration [55, 57] and video retrieval [54, 18, 17]. Especially in the video exploration and browsing area [81], we have demonstrated that feature signatures can significantly enhance the performance of the state-of-the-art approaches[5].

1.2 Models based on feature signatures

Multimedia objects consist of elements with various features. For example, a chromatic image consists of pixels, where each pixel contains position coordinates, color information or more complex features based on texture/gradient statistics from the neighborhood of the pixel. Considering traditional approaches to design object descriptors, a suitable set of features is usually manually selected to form so-called *feature space* $\mathbb{F}$ for a particular retrieval task. Given a suitable feature space, the elements of a multimedia object o can be mapped and aggregated in the feature space, forming a *feature signature function* $f^o : \mathbb{F} \to \mathbb{R}_0^+$. Generally, the feature signature function assigns importance (in the form of weights $w_i \in \mathbb{R}_0^+$) to representatives $r_i \in \mathbb{F}$ to model the contents of an object. Note that one object can be represented by various feature signature functions, depending on feature sampling and aggregation strategies. The concept of feature signatures is used in many areas of multimedia retrieval under different terminology. In this text, the terminology from works of Rubner et al. [78] and Beecks [7] is used.

The set of representatives with non-zero weights is usually restricted to be finite. The following definition represents a compact form of feature signatures where only representatives with non-zero weights are considered.

Definition 1 (Feature Signature) *Given a feature space $\mathbb{F}$, the feature signature S^o of a multimedia object o is defined as a finite set of tuples $\{\langle r_i^o, w_i^o \rangle\}_{i=1}^n$ from $\mathbb{F} \times \mathbb{R}^+$, consisting of representatives $r_i^o \in \mathbb{F}$ and weights $w_i^o \in \mathbb{R}^+$*

The feature signatures enable modeling of multimedia objects using object-specific representatives, resulting in a flexible representation of the contents of the objects. However, the flexibility of the representation comes at the cost of more complex feature extraction and similarity modeling. Therefore,

[5] Our signature-based video browsing tool has won the Video Browser Showdown competition (www.videobrowsershowdown.org) in 2014 and 2015.

the multimedia objects are often modeled using a finite set of shared representatives $\mathbb{X} \subset \mathbb{F}$ obtained for a particular database in a preprocessing phase. Given a set of shared representatives, each modeled object can be aggregated into a fixed-length vector (histogram), where each vector bin aggregates features aligned to one shared representative. Such representation enables efficient similarity evaluation using cheap bin-to-bin distance measures, because the vectors representing objects have the same length and corresponding bins have the same semantics.

Definition 2 (Feature Histogram) *Given a feature space $\mathbb{F}$ and an ordered set of n shared representatives $\{r_i\}_{i=1}^{n} = \mathbb{X} \subset \mathbb{F}$, the feature histogram h^o of a multimedia object o is defined as a vector $\{w_i^o\}_{i=1}^{n}$, consisting of weights $w_i^o \in \mathbb{R}_0^+$, where each weight w_i^o corresponds to one shared representative r_i.*

Both feature signatures and feature histograms represent a special case of a general mapping $f : \mathbb{F} \to \mathbb{R}_0^+$. Furthermore, if the union of all representatives from all feature signatures for a given dataset is used as the shared set of representatives, then the feature signatures can be represented as high-dimensional sparse feature histograms. Contrary, sparse feature histograms with zero bins can be represented in a compact form as feature signatures. However, data representations, feature extraction approaches and similarity measures for these two types of descriptors usually differ (see following sections). So which descriptor is a better choice? Generally, the number of required shared representatives and the knowledge of the database matters. If a small set of shared representatives is sufficient to create discriminative descriptors for all database objects, then feature histograms are the preferred efficient choice. If the number of shared representatives has to be too large to fit the minimal level of necessary details in all database objects or the dataset is dynamically changing over time, then the feature signatures can be used to flexibly represent the contents of the objects using object specific representatives.

The difference between feature histograms and feature signatures gets remarkable in high-dimensional feature spaces. In Figure 1.1, there are depicted three ways to represent an image using 7-dimensional position-color-texture feature space – a feature signature (Figure 1.1a), compared to feature histograms based on 10000 and 1000 shared representatives (Figure 1.1bc). Whereas feature signatures employing object-specific representatives

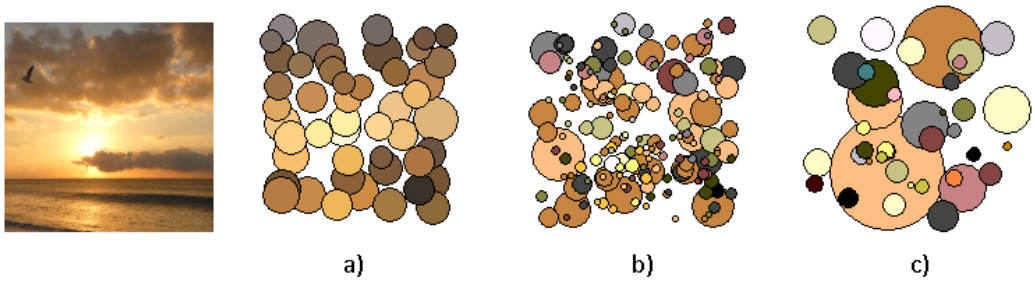


Figure 1.1: An image represented using position-color-texture feature space and **a)** feature signature, compared to feature histograms based on **b)** 10000 and **c)** 1000 shared representatives (only representatives with a non-zero weight are displayed).

can flexibly aggregate the contents of the original image, the expressiveness of feature histograms can suffer from the usage of shared representatives, especially for unique images in the database. In Figure 1.1b, the expressiveness of the feature histogram is improved at the cost of the high number of shared representatives, while in Figure 1.1c, the original image contents is almost lost when aggregated using the small number of shared representatives not corresponding to the modeled image. For more details about models based on feature signatures we refer readers to [78, 7].

1.2.1 Feature extraction

In order to create an object descriptor, a suitable feature extraction function aggregating extracted features has to be utilized. In the following, we present just a basic introduction and terminology used by traditional feature extraction approaches. For more advanced feature extraction techniques like feature selection [34], feature learning [42] or novel deep learning techniques [16, 45, 89], we refer readers to the corresponding literature.

When designing a feature extraction function, first a suitable feature space has to be selected with respect to a particular retrieval task [27, 65]. For example, when searching for color-specific images, the feature space could be based on chromatic information obtained from pixels. If the color is not characteristic for searched images, the feature space often considers other features like texture [90], edges, shapes, pixel masks or complex summarizations of gradient distributions around a selected set of salient points [62]. The feature

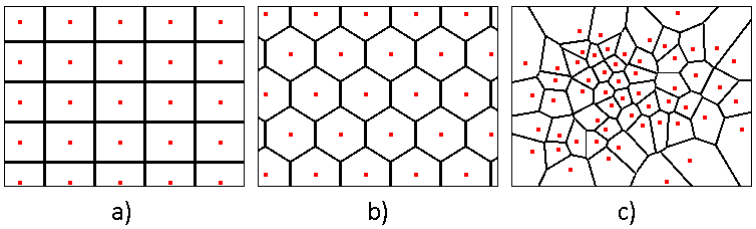


Figure 1.2: Two dimensional feature space partitioning based on uniformly distributed representatives **a, b)** and representatives corresponding to a nonuniform data distribution **c)**.

space can be also a composition of heterogeneous properties, for example, a feature space modeling joint distribution of color and texture [21].

Given a suitable feature space, a set of descriptive representatives $\mathbb{X} \subset \mathbb{F}$ has to be selected for a modeled multimedia object. In case of feature signatures, the set of representatives has to be detected for each multimedia object separately, while in case of feature histograms the set of representatives is obtained for a particular database in a preprocessing phase. Having the set of representatives, a feature extraction function maps elements from modeled multimedia objects to features $f_i^o \in F^o \subset \mathbb{F}$ and aggregates them[6]. In order to align and aggregate the extracted features to representatives $r_i \in \mathbb{X}$, the feature space is usually partitioned to subsets S_i. Such partitioning is denoted as *feature space dictionary*. For example, each subset S_i can be defined as a Voronoi cell, considering seeds r_i and a total feature space distance function $\mathbb{F} \times \mathbb{F} \rightarrow \mathbb{R}_0^+$. Several feature space dictionaries employing different sets of representatives and Euclidean distance are depicted in Figure 1.2. The partitions can also overlap, which leads to a soft assignment coding [51].

Using the feature space dictionary over feature space $\mathbb{F}$, the extracted features $F^o \subset \mathbb{F}$ of multimedia object o are assigned to representatives r_i, where weight $w_i \in \mathbb{R}_0^+$ corresponds to $|S_i \cap F^o|$. In each object descriptor, the weights are often normalized such that their sum is equal to one.

In the following two sections, similarity models based on feature histograms and feature signatures are described in more detail.

[6]Note that the aggregation is meaningful only if the new aggregated features are still discriminative.

1.2.2 Feature histograms

In order to extract a feature histogram for a multimedia object, a proper shared feature space dictionary has to be selected or created. If a representative subset of a multimedia database is available, the shared feature space dictionary can be precomputed from features from the objects in a preprocessing phase using a clustering technique (e.g., k-means). However, for unknown or dynamically changing databases the feature histogram extraction function can result in non-discriminative feature histograms. In such cases, a domain knowledge is necessary to prepare a suitable shared feature space dictionary, or object-specific dictionary has to be employed (see the next section).

During last decades, there have been designed and even standardized many types of similarity models based on feature histograms over various feature spaces (e.g., the MPEG-7 standard [20, 66]). The similarity models use either a linear combination of feature histograms modeling each feature independently, or use a feature space dictionary corresponding to a joint distribution of features. The feature space dictionary is often used in computer vision area to aggregate features in high-dimensional feature spaces (e.g., SIFT [62] or SURF [6] providing more descriptive local features for advanced retrieval tasks like object detection or image registration). In the computer vision area, the representation using the feature space dictionary is known as *bag of features model* [83]. The bag of features model enables efficient retrieval using inverted files, a well established technique in the text-based retrieval area [100]. In the bag of features model, the precomputed feature space dictionary is called *codebook*, and the representatives are called *codewords*. In the model, each multimedia object is represented as a frequency histogram of codewords present in the object, where all the objects in the database share one codebook. Whereas the efficiency of the bag of features model is sufficient for large scale multimedia retrieval, the practical effectiveness of the model is limited by the shared feature space dictionary. Therefore, the bag of features model has been improved using more advanced techniques like re-ranking using spatial information [76], semantic preserving models [97], Hamming embedding [40], compressed Fisher vectors [75] or vectors of locally aggregated features [41]. We may observe that the general trend is to use the bag of features model just for a preliminary filtering, while more effective representations are considered for re-ranking of the results.

Given multimedia objects represented by feature histograms, a suitable

distance measure has to be utilized to rank the similarity between each pair of the objects. The distances utilize either a simple bin-to-bin matching strategy or more complex bin-to-many-bins matching strategies considering also perceptual relations between bins of a homogeneous domain. As probably the most popular distance measures, Minkowski metrics

$$L_p(x, y) = (\sum_{i=1}^{d} |x_i - y_i|^p)^{\frac{1}{p}},$$

are the frequently used class of cheap bin-to-bin distances for d-dimensional vectors x, y, given $p \in \mathbb{R}, p \geq 1$. The Minkowski metrics can perform well as long as features from similar multimedia objects are aggregated to the same histogram bins. However, if the features are aggregated among several neighboring cells of the shared feature space dictionary, it may turn out that two similar objects can have dissimilar feature histograms considering just bin-to-bin distances. In such situations, the quadratic form distance

$$QFD_A(x, y) = \sqrt{(x - y)A(x - y)^T},$$

using a $d \times d$ positive-definite correlation matrix A is employed to straighten the ambiguity of the feature extraction process on homogeneous domains [35]. If the quadratic form distance is utilized just to model fixed correlations between the histogram bins (i.e., matrix A is fixed), the costly retrieval model employing quadratic form distance can be transformed to an equivalent but much cheaper retrieval model employing the euclidean distance [85]. The quadratic form distance can be also used to model user preferences changing over time [39], however, such a dynamic model can be efficiently indexed just using spatial access methods [80] that can partition the descriptor space independently of a distance measure. Another approach to model similarity between two normalized feature histograms with correlated bins is the Earth Mover's Distance [79] that interprets the similarity as a transportation problem.

In the following section, we discuss models based on feature signatures that adaptively represent contents of the multimedia objects. As demonstrated in [9], models based on feature signatures can be an effective alternative to the bag of visual words approaches, especially in several retrieval tasks.

1.2.3 Feature signatures

Unlike feature histograms represented by simple vectors referring to a shared feature space dictionary, each feature signature has its own object-specific feature space dictionary enabling more flexible representation. On the other hand, the object specific dictionary becomes the part of the descriptor and the dictionary has to be determined for each multimedia object separately.

The feature signatures for a given feature space can be obtained using various feature extraction techniques. For example, a position-color feature signature for an image can be created using an image resize operation (a convolution operation, respectively). In such case, each representative with a constant weight corresponds to one pixel of the small image thumbnail, where the resolution of the thumbnail has to be fixed in advance (see Figure 1.3c). Another approach to extract a feature signature is the adaptive k-means clustering of (sampled) pixels from an image (see Figure 1.3ab). Although the adaptive k-means clustering can adaptively fit to the content of the image, the clustering is costly operation and thus an efficient parallel implementation of feature signature extraction is necessary for huge multimedia collections. Our technique enabling extraction of thousands of position-color-texture feature signatures per second is presented in Chapter 2.

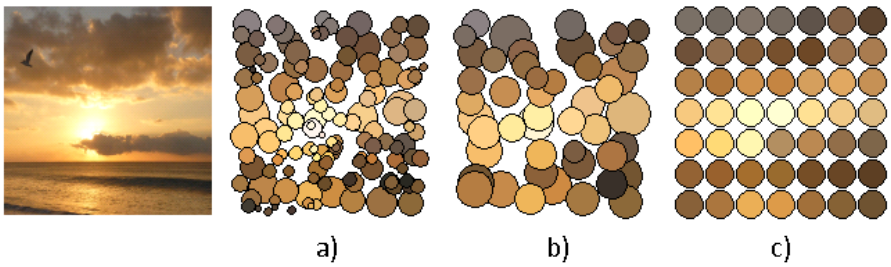


Figure 1.3: Examples of feature signature extraction techniques of an image – **a, b)** two position-color-texture feature signatures obtained by two variants of adaptive k-means clustering of the same set of sampled points and **c)** position-color feature signature obtained by image resize operation.

In order to define a similarity model based on feature signatures, distance measures capable to compare two feature signatures with different number of representatives have to be employed. Such distances are denoted as adaptive distance measures. Let us note that since two feature signatures do not (have to) share the same set of feature space representatives, the adaptive

distances have to consider all pairwise distances between all the representatives, resulting in at least quadratic time complexity of similarity evaluation. During last decades, there have been designed several adaptive distance measures like Hausdorff Distance [38], Earth Mover's Distance [79], Perceptually Modified Hausdorff Distance [73], Signature Quadratic Form Distance [14], or recently introduced Signature Matching Distance [9], and there have been also evaluated several studies comparing the distances [12, 9].

In our work, we have mainly focused on distance spaces based on the Signature Quadratic Form Distance, because the distance spaces are effective and also enable efficient indexing. The Signature Quadratic Form Distance is defined as follows:

Definition 3 (Signature Quadratic Form Distance) *Given two feature signatures* $S^o = \{\langle r_i^o, w_i^o \rangle\}_{i=1}^n$ *and* $S^p = \{\langle r_i^p, w_i^p \rangle\}_{i=1}^m$ *and a similarity function* $f_s : \mathbb{F} \times \mathbb{F} \to \mathbb{R}$ *over a feature space* $\mathbb{F}$*, the signature quadratic form distance* SQFD_{f_s} *between* S^o *and* S^p *is defined as:*

$$\mathrm{SQFD}_{f_s}(S^o, S^p) = \sqrt{(w_o \mid -w_p) \cdot A_{f_s} \cdot (w_o \mid -w_p)^T}\,,$$

where $A_{f_s} \in \mathbb{R}^{(n+m)\times(n+m)}$ *is the similarity matrix arising from applying the similarity function* f_s *to the corresponding feature representatives, i.e.,* $a_{ij} = f_s(r_i, r_j)$*. Furthermore,* $w_o = (w_1^o, \ldots, w_n^o)$ *and* $w_p = (w_1^p, \ldots, w_m^p)$ *form weight vectors, and* $(w_o \mid -w_p) = (w_1^o, \ldots, w_n^o, -w_1^p, \ldots, -w_m^p)$ *denotes the concatenation of weight vectors* w_o *and* $-w_p$*.*

To determine similarity values between all pairs of representatives from the feature signatures, the Gaussian similarity function $f_{gauss}(r_i, r_j) = e^{-\alpha L_2^2(r_i, r_j)}$ or the Heuristic similarity function $f_{heuristic}(r_i, r_j) = 1/(\alpha + L_2(r_i, r_j))$ can be utilized, where α is a parameter for controlling the precision, and L_2 denotes the Euclidean distance.

The Signature Quadratic Form Distance has not only proved to be an effective distance measure, but also a distance suitable for efficient retrieval. Although the distance has quadratic time complexity, the distance satisfies metric/ptolemaic postulates necessary for efficient metric/ptolemaic indexing [37]. Furthermore, the α parameter of the distance affects not only effectiveness, but also the intrinsic dimensionality property[7] of the corresponding

[7]The intrinsic dimensionality is a crucial property for efficiency of the metric/ptolemaic indexing, for more details see Section 1.3.

distance space [10]. Last but not least, we have also demonstrated that Signature Quadratic Form Distance represents a suitable task for GPU devices [46].

1.2.4 Discussion

The models based on feature signatures and feature histograms represent traditional approaches with manually modeled descriptors. Although the recent developments in machine learning seem to outperform the traditional approaches in many classification tasks (e.g., image classification using convolutional neural networks in connection with deep learning [45]), we believe that the traditional approaches can still find many applications. In the following, we present several arguments supporting models based on feature signatures:

- The feature signatures in connection with adaptive distance measures constitute a strong formal framework [78, 7], yet simple enough to prove many properties analytically. The framework supports both flexible representation of multimedia objects and also nontrivial perceptual similarity measures. Compared to histogram-based approaches, the framework has also demonstrated its effectiveness in several retrieval tasks [60, 9, 30]. Recently, feature signatures have been employed also for effective content-based near-duplicate video detection [92].

- The feature signatures can be used in combination with other retrieval models [55, 60] as an alternative retrieval model or for re-ranking of the results returned by a primary retrieval model. For example, images of the same class returned by an effective model based on convolutional neural networks (e.g., [45]) could be rearranged using a model based on feature signatures that could distinguish minor differences between the images.

- The feature signatures are intuitive enough to be used for simple sketch based retrieval [54, 18] or multimedia exploration. For example, using position-color feature signatures, users can utilize just a color distribution of an image to control browsing of an unknown database [55]. More specifically, a user can select an actually displayed image of a sunset to find an image of a fireplace representing different concept but having similar feature signature.

- The feature signatures can flexibly represent contents of multimedia objects. Especially, in cases when a representative part of the database cannot be analyzed beforehand to design a database-specific similarity model, the feature signatures can be employed to design at least descriptive object representations independent on the database.

- The feature signatures can be utilized for low resolution images often comprising only limited number of features. An effective model for small thumbnail images can be useful from both performance (faster feature extraction and data transfers) and privacy issues (only low-resolution images can be available, similarity search can be performed in a cloud environment [44]).

However, the applicability of the models based on feature signatures is still limited by at least quadratic time complexity of involved similarity measures and thus the main focus of our work was to find efficient retrieval techniques for such models. Nevertheless, recent advancements presented in the following section demonstrate that models based on feature signatures could be utilized also in large-scale multimedia retrieval systems.

1.3 Efficient retrieval using feature signatures

Given a similarity model $(\mathbb{U}, \delta)$ based on feature signatures and a similarity search task defined by a query object $q \in \mathbb{U}$ and a query constraint ϕ, there can be utilized several orthogonal approaches to process such task more efficiently than just simple sequential search using the original expensive model. The approaches differ in assumptions about the similarity model and the database, whether the model is static or dynamic (i.e., descriptors and the distance can be changed), and whether the database is static or dynamic. Also the number of query objects affects the choice of the optimal solution. Despite their differences, most of the techniques share one principle for efficient filtering of non-relevant objects – *lower-bounding principle*, where a lower-bound distance $LB(\delta(q, o)) \leq \delta(q, o)$ between a query object $q \in \mathbb{U}$ and a database object $o \in \mathbb{U}$ is expected to be much cheaper than the original distance $\delta(q, o)$. Using the lower-bound distance, the query can be processed using a filter and refine approach, where the original distance is evaluated only on a fraction of the database. The lower-bound distance can be approximated, determined for a specific domains rigorously or determined using general properties of the distance measure (e.g., metric/ptolemaic properties).

1.3.1 Distance-specific approaches

During the last decade, there have been presented many attempts to find rigorously efficient lower-bound distances for various adaptive distance measures. However, many of the methods are usually restricted to feature histograms or the lower-bound is not too tight.

For example, a simple and not too tight lower-bound for the Earth Mover's Distance is the Rubner filter [79] that evaluates the ground distance between mean centroids of two compared feature signatures S^o, S^q. The Rubner filter holds only if the sum of weights is the same for both feature signatures, otherwise, it becomes an approximate method. In [3, 4], the authors have presented novel dimensionality reduction techniques for the Earth Mover's Distance in a two-step filter-and-refine architecture for efficient exact search. However, the authors assume feature histograms and metric ground distances. In [96], the authors utilized a dimensionality reduction technique to improve the time of Earth Mover's Distance evaluation and proved that Earth Mover's Distance evaluated in a low-dimensional sub-

space lower-bounds the Earth Mover's Distance in the original space. Again, the technique is restricted just to feature histograms. In [82], the authors presented a linear-time algorithm for approximating the Earth Mover's Distance for low-dimensional histograms using the sum of absolute values of the weighted wavelet coefficients of the difference histogram. Recently, a new lower-bound called Independent Minimization for Signatures [95, 94] has been presented for more efficient retrieval using the Earth Mover's Distance.

There have been also attempts to find cheap distances approximating the Signature Quadratic Form Distance. In [15], the authors have demonstrated that if similarity function $f_{L_2}(r_i, r_j) = -L_2^2(r_i, r_j)/2$ is utilized, then the Signature Quadratic Form Distance becomes L_2-Signature Quadratic Form Distance suffering from worse effectiveness but computable in linear time. In [13], the authors proposed a simple feature signature reduction technique based on removal of tuples with small weights. The authors also defined a signature quadratic form filter distance, applicable for approximate filter and refine retrieval. The filter distance just evaluates the signature quadratic form distance using reduced feature signatures. However, according to our results, the filter distances do not provide too tight approximations of the lower-bounds, because the reduction technique removing tuples with small weights significantly deteriorates the original feature signatures. On the other hand, the idea of feature signatures reduction is a general approach that enables retrieval using an arbitrary adaptive distance measure.

Therefore, we have investigated advanced feature signature reduction techniques that can significantly improve the efficiency of the retrieval[8] (see Chapter 3). In [53], we have presented scalable feature signatures, a class of feature signature reduction techniques based on agglomerative hierarchical clustering [33], [29]. We have experimentally demonstrated that feature signatures can be significantly reduced at the cost of just a small loss of quality. In Figure 1.4, we may observe an example of a feature signature for the image of a sunrise and corresponding reduced feature signatures. Whereas the original feature signature flexibly approximates the contents of the original image, the reduced feature signatures at least preserve the general color layout of the image. Furthermore, the reduced feature signatures can be compared with an arbitrary adaptive distance measure, in other words, this approach is not restricted just to metric/ptolemaic distances discussed in the

[8]The time complexity of adaptive distance measures depends quadratically on the size of the feature signature vocabulary, i.e., the number of tuples.

following sections.

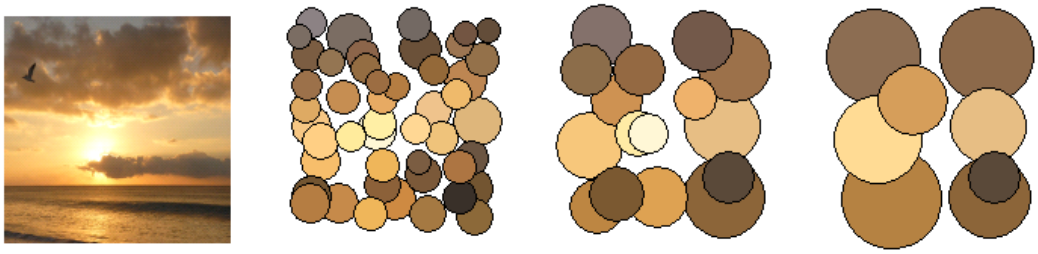


Figure 1.4: An example of a feature signature reduction, starting from left – original image, original feature signature, and reduced feature signatures comprising 32 and 16 tuples.

1.3.2 Metric indexing

In this section, we overview the metric space approach [23, 98, 80] that can be used to efficiently process similarity queries employing similarity models based on feature signatures and metric adaptive distance measures. In order to process the queries efficiently, the metric space approach utilizes lower-bounding techniques that employ precomputed distances between database objects and a set of reference points $p_i \in \mathbb{P} \subset \mathbb{U}$, so-called pivots. The metric space approach assumes that the utilized distance function satisfies reflexivity, non-negativity, symmetry and triangle inequality axioms. Especially the triangle inequality axiom $(\forall x, y, z \in \mathbb{U} : \delta(x, y) \leq \delta(x, z) + \delta(y, z))$ is necessary for the correctness of the lower-bounding based on precomputed distances. More precisely, given a query object $q \in \mathbb{U}$, a database object $o \in \mathbb{U}$ and a pivot $p \in \mathbb{U}$, the lower-bound distance between o and q can be directly derived from the triangle inequality using precomputed distances as $LB_{\triangle}(\delta(q, o)) = |\delta(o, p) - \delta(q, p)|$, where $\delta(q, p)$ is evaluated just once before query processing and $\delta(o, p)$ is the precomputed distance stored in a metric index (see Figure 1.5). Furthermore, the metric space approach provides also partitioning mechanisms enabling grouping of similar objects into partitions so the whole groups of objects can be filtered during query processing. There have been designed a lot of indexing techniques for metric spaces, so-called metric access methods [23, 98, 80], that differ in the way they partition database objects, store precomputed distances and process similarity queries [64, 24, 87, 91, 22, 2]. Furthermore, there still appear new approaches that

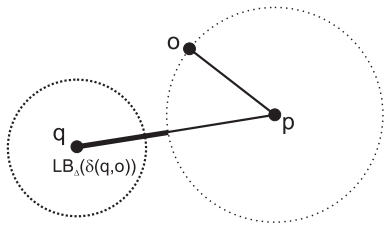


Figure 1.5: Pivot-based lower-bounding using triangle inequality.

can be used to enhance many of the well-established metric access methods. For example, in [59] we have presented Cut-regions [59] that represent compact metric regions suitable for more efficient indexing and retrieval (see Chapter 4).

However, not only the metric axioms but also the distribution of the distances between database objects plays a significant role in the efficiency of the metric space indexing. In [23], the authors have proposed the intrinsic dimensionality measure that indicates whether the data can be efficiently indexed using a given distance space. The lower values of intrinsic dimensionality indicate that the data form clusters in the distance space and thus metric indexes based on metric space partitioning can be utilized [68, 69, 61]. On the other hand, high values of intrinsic dimensionality indicate no clusters, which means only the sequential processing using just simple query-to-object lower-bounding can be utilized to filter at least some costly distance computations [64]. The problem of high intrinsic dimensionality can be also addressed by approximate search strategies that can provide interesting precision-speedup trade-offs [74, 84, 1, 70]. For example, the Signature Quadratic Form Distance trained for maximal effectiveness often suffers from high intrinsic dimensionality. Nevertheless, we have experimentally demonstrated [56] that using approximate k-NN search strategies designed for M-Index [68, 69], the effectiveness can be still competitive even if just a fraction of the database is visited.

Although metric indexes can significantly speed up query processing, the methods still assume that the number of query objects is high enough to compensate the indexing costs. Furthermore, the techniques assume that indexed data are not changed too often, so once data are indexed, they are often queried. These assumptions are not always satisfied, considering for example multimedia streams, where just few queries can be issued for data stored in a search window. In such cases, the cost of updating the index

would highly overcome the benefits of the indexing, while a sequential scan using a multi-query processing strategy (e.g., [19]) could be more efficient. If the queries are issued independently and no delays for query collection are allowed, the D-Cache structure [86] can be used to efficiently process a few number of independently issued similarity queries (see Chapter 5).

The structure of the D-Cache is simple – it is just a simple block of memory where distances evaluated during previous queries are hashed and stored. As in other cache types (disk, processor), the space for cached distances is limited and thus the new distances can replace the original ones. In order to filter non-relevant objects, each actually processed query object q_i considers several previously issued query objects $q_j, j < i$ as pivots, and thus using $\delta(q_i, q_j)$ and $\delta(o_k, q_j)$ potentially stored in D-Cache, the lower-bound $LB_{\triangle}(\delta(o_k, q_i)) = |\delta(q_i, q_j) - \delta(o_k, q_j)| \leq \delta(o_k, q_i)$ can be evaluated and used for filtering. If $\delta(o_k, q_j)$ was not stored in the D-Cache or was already replaced, distance $\delta(o_k, q_i)$ has to be evaluated. Although the lower-bounding is the same as the one used by metric access methods, the D-cache does not have to create an index structure in advance, thus it can be used instantly and starting from the second query object the metric filtering can be employed. The D-Cache can be employed also for dynamically changing similarity models, for example, if the alpha parameter of the Signature Quadratic Form Distance is changed to improve efficiency of the filtering. We have also demonstrated that standard metric access methods can be enhanced by D-Cache for more efficient indexing and retrieval [86].

1.3.3 Ptolemaic indexing

The metric space approach is not the only way to efficiently index adaptive distance measures. Recently, we have proved that the Signature Quadratic Form Distance is a ptolemaic metric [37] (see Chapter 6), which means it satisfies metric properties and also the ptolemaic inequality stating that for any quadrilateral, the pairwise products of opposing sides sum to more than the product of the diagonals. Formally, for any four points x, y, u, $v \in \mathbb{U}$, we have the following:

$$\delta(x, v) \cdot \delta(y, u) \leq \delta(x, y) \cdot \delta(u, v) + \delta(x, u) \cdot \delta(y, v) \tag{1.1}$$

As for the triangle inequality, the ptolemaic inequality can be used for distance-based indexing to construct a pivot-based lower bound. For a query

q, object o, and pivots p and s, we get the *candidate bound*:

$$\delta_{\mathrm{C}}(q, o, p, s) = \frac{|\delta(q, p) \cdot \delta(o, s) - \delta(q, s) \cdot \delta(o, p)|}{\delta(p, s)} \tag{1.2}$$

For simplicity, we let $\delta_{\mathrm{C}}(q, o, p, s) = 0$ if $\delta(p, s) = 0$. As for triangular lower-bounding, one would normally have a set of pivots $\mathbb{P}$, and the bound can then be maximized over all (ordered) pairs of distinct pivots drawn from this set, giving us the final Ptolemaic bound [36, 58]:

$$\delta(q, o) \geq \mathrm{LB}_{\mathrm{ptol}}(\delta(q, o)) = \max_{p,s \in \mathbb{P}} \delta_{\mathrm{C}}(q, o, p, s) \tag{1.3}$$

Using all pairs of pivots results in the optimal lower-bound for a given set of pivots $\mathbb{P}$, however, the quadratic time complexity (based on $|\mathbb{P}|$) of the lower-bound estimation can slow-down the retrieval, especially for cheap distances. In order to avoid using all pairs of pivots, heuristics selecting a specific set of pairs can be utilized. Given a ptolemaic metric, another question is whether to use lower-bounding based on triangle inequality or ptolemaic inequality. In other words, whether the ptolemaic lower bounding can improve the triangle lower bounding, and vice versa.

In Figure 1.6, we have visualized points in 2D euclidean space, that can be filtered just by $LB_{\triangle}$ (blue points), LB_{ptol} (green points), by both lower bounding techniques (gray points) and points that cannot be filtered by any of the two techniques (white points). We may observe that both filtering techniques can contribute to the filtering, where the filtering power of each technique depends on the query radius and also on the constellation of pivots and the query object. The filtering power of cheap triangle lower bounding could be increased by higher number of pivots. On the other hand, the ptolemaic lower bounding can improve the filtering power of a given pivot set. Both techniques can be also combined, where first the cheap triangle lower bound is evaluated and used for filtering. If the triangle lower bound is not sufficient, more expensive ptolemaic lower bound is employed for a given object. As for triangle lower-bounding, the ptolemaic lower bound can be evaluated also for whole regions and thus, given a ptolemaic metric, the filtering rules of many metric access methods can be simply extended [37].

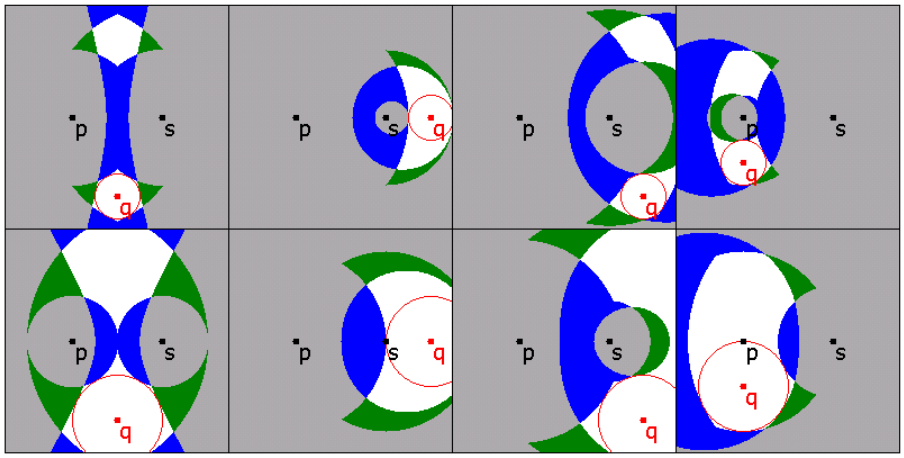


Figure 1.6: Triangle versus ptolemaic filtering in 2D euclidean space using two pivots p, s and range query (q, r_q), where the blue points can be filtered just using $LB_{\triangle}$, green points can be filtered just using LB_{ptol}, gray points can be filtered by both techniques, while white points cannot be filtered by neither of the two techniques.

1.3.4 Parallel computing

Although domain specific approaches, feature signature reduction techniques, and distance based indexing methods can significantly improve the efficiency of multimedia retrieval using feature signatures and adaptive distance measures, the techniques alone cannot make the model applicable for immense databases comprising billions of multimedia objects. In such cases, approaches like distributed computing and/or massively parallel computing have to be employed as well [5, 32, 63, 99]. For distributed computing, there have been already developed several approaches that can be directly applied for models based on feature signatures and metric adaptive distance measures. For example, in [69] the authors propose a distributed metric index (M-Index) that can organize the database into a large number of nodes according to a metric distance. The index is suitable both for exact and approximate search, where especially the approximate search strategies can prune a significant part of the searched database. Furthermore, each node can utilize a centralized index structure and/or massively parallel computing to improve the efficiency of the distributed index.

The parallel computing (especially new GPU architectures [72, 71]) rep-

resents another promising approach for evaluation of costly adaptive distance measures that constitute a serious bottleneck of a multimedia retrieval system based on feature signatures. In recent days, novel many-core devices with specific hardware architectures are designed for various computing tasks. Hence, one of the goals of the research in this area is to find suitable computation tasks for existing many-core devices and to adapt the existing algorithms to better utilize properties of the devices. The typical example are GPU cards that provide thousands of cores. However, their hardware architecture and programming model significantly differ from traditional CPUs. Especially different memory organization and thread execution in GPU cards require different algorithms. The CPU and GPU approaches can be also efficiently combined in hybrid systems that better utilize available hardware.

In Chapter 7, we have focused on parallel processing of adaptive distances and compared the efficiency of the retrieval when using two parallel environments with different architectures and also prices. More precisely, we have compared a cheap desktop PC comprising two GPU cards with CUDA architecture and an expensive high-end NUMA server. As parallel computing tasks, we have investigated the efficiency of the retrieval when using parallel computing for batches of distance computations, or even parallel processing using a simple metric index structure [49]. More specifically, we have designed two algorithms considering utilization balance between CPU and many-core GPUs for efficient similarity search with the Signature Quadratic Form Distance. We have shown how to process multiple distance computations and other parts of the search procedure (e.g., lower-bound estimation) in parallel, achieving maximal performance of the combined CPU/GPU system. We have experimentally demonstrated that using GPU cards for models based on feature signatures represents an order of magnitude faster and cheaper solution than a high-end many core NUMA server, despite the memory organization and thread execution specifics of the GPU architectures. Similar results have been achieved also for feature signature extraction process, where we have reached the throughput of approximately 8000 extracted feature signatures per second [47].

1.4 Author's contributions

Reaching goals of the presented research could be compared to a puzzle, where the way to solve the overall problem can be seen only after the problem is solved. However, in practice there is no direct and clear way to solve such complex tasks like content-based multimedia retrieval and it is also beyond the skills of an individual to solve such problems. As a logic consequence, usually only a joint research with the immense support of research institutions, hours of influential discussions with colleagues (including involved students), brainstormings with members of particular teams, and weeks spent with programming and running experiments can make any progress, and thus can place a new puzzle piece at the correct place.

This is also the case of the presented work that is based on six papers where five of them have more than one author. Furthermore, in three cases the collaboration was international and thus also long email conversations were the important part of the research process. Each of the authors has significantly contributed to the papers, starting with influential discussions forming the core of the contributions and ending with programming, running experiments and writing the papers. Since it is hard to find clear borders between particular contributions (all authors contribute to many subtasks and influence each other), not to mention finding the author of the first idea, the contribution of each author of the presented papers can be just approximately estimated as $1/x$, where x is the number of authors of a particular paper.

Chapter 2

Efficient Extraction of Feature Signatures Using Multi-GPU Architecture

Martin Kruliš
Jakub Lokoč
Tomáš Skopal

Published in the proceedings of the 19th International Conference on Multi-Media Modelling MMM 2013, LNCS, ISSN 0302-9743.
dx.doi.org/10.1007/978-3-642-35728-2_43

The extended version of this paper [48] was accepted to Multimedia Tools and Applications journal (IF in 2014: 1.346) and is in press now.

Efficient Extraction of Feature Signatures Using Multi-GPU Architecture

Martin Kruliš, Jakub Lokoč, and Tomáš Skopal

SIRET research group, Dept. of Software Engineering,
Faculty of Mathematics and Physics, Charles University in Prague
{krulis,lokoc,skopal}@ksi.mff.cuni.cz

Abstract. Recent popular applications like online video analysis or image exploration techniques utilizing content-based retrieval create a serious demand for fast and scalable feature extraction implementations. One of the promising content-based retrieval models is based on the feature signatures and the signature quadratic form distance. Although the model proved its competitiveness in terms of the effectiveness, the slow feature extraction comprising costly k-means clustering limits the model only for preprocessing steps. In this paper, we present a highly efficient multi-GPU implementation of the feature extraction process, reaching more than two orders of magnitude speedup with respect to classical CPU platform and the peak throughput that exceeds 8 thousand signatures per second. Such an implementation allows to extract requested batches of frames or images online without annoying delays. Moreover, besides online extraction tasks, our GPU implementation can be used also in a traditional preprocessing and training phase. For example, fast extraction allows indexing of huge databases or inspecting significantly larger parameter space when searching for an optimal similarity model configuration that is optimal according to both efficiency and effectiveness.

Keywords: similarity search, feature extraction, GPU, parallel.

1 Introduction

The traditional approaches to the multimedia retrieval rely on the well-established fulltext search. However, as the amount of new multimedia data grows immensely nowadays, there often appear scenarios where data annotations cannot be provided and so the content-based retrieval techniques in connection with the similarity search paradigm is the only viable possibility for computer-aided image retrieval [8]. The content-based retrieval approach requires an effective similarity model that should mimic a user's perception of which images are similar and which are not [22]. More specifically, the similarity model consist of features extracted from the original images (formed into image descriptors) and a total similarity function (often modeled as a distance function) providing a similarity ranking (ordering) on the descriptors. The similarity model is effective if the ranking corresponds to user preferences, that are often provided in the form of so-called ground truth testbed.

S. Li et al. (Eds.): MMM 2013, Part II, LNCS 7733, pp. 446–456, 2013.

For example, the ground truth can be represented as an annotated sample of the database, and is often utilized to train or verify the similarity model [7].

While the traditional content-based retrieval applications use a slow preprocessing phase to prepare feature representations, recent applications calls for faster feature extraction tools enabling immediate online analysis of the picture content. Examples of such systems can be, e.g., camera systems producing a lot of video streams that have to be processed frame by frame or image exploration techniques that create exploration structures from a given set of images during the interaction with users [14]. Furthermore, the fast feature extraction can be utilized also in traditional tasks, e.g., for fast indexing of huge datasets or for training of similarity models, where significantly larger parameter space can be considered and tested.

In this paper, we focus on the similarity models based on feature signatures, that can be effectively compared via the *signature quadratic form distance* (SQFD) [4,6]. This model is competitive in the terms of the effectiveness and at the same time provides many parameters to tune. We provide efficient GPU implementation of the expensive feature extraction process resulting in two orders of magnitude speed up. The contributions of this paper can be summarized as:

- Description of the feature signatures extraction process with emphasis on performance issues.
- Proposal of a fast GPU implementation of this extraction process.
- Experimental results expressing the power of our GPU feature extraction.

The paper is organized as follows. In Section 2, we describe the similarity model based on feature signatures and the basics of feature extraction process. In Section 3, we recall GPU platform basics and describe details of our GPU implementation of the feature extraction process. The experimental results are presented and discussed in Section 4 and Section 5 concludes the paper.

2 Similarity Model Based on Feature Signatures

In this section, we sketch the recently studied similarity model based on feature signatures and the Signature Quadratic Form Distance, that can be utilized to solve the content-based image retrieval tasks [5,3]. Especially, we describe in detail the employed feature extraction method, where we highlight parameters influencing many aspects of the resulting similarity space. We also remember recent works focusing on efficient query processing in this model – the metric and ptolemaic indexing of the Signature Quadratic Form Distance.

2.1 Feature Signatures

The traditional object representation approaches utilizing feature histograms aggregate features within predefined bins of fixed-sized vectors. Unlike the feature histograms, the feature signatures allow a more flexible object representation

within a utilized feature space [9,18,8], where the size of resulting feature signatures is not fixed. Hence, complex multimedia objects can be represented by a feature signature consisting of many centroids, while simple multimedia objects have just few centroids in their feature signatures.

Definition 1 (Feature Signature). *Given a feature space* $\mathbb{F}$*, the feature signature* S^o *of a multimedia object* o *is defined as a set of tuples from* $\mathbb{F} \times \mathbb{R}^+$ *consisting of representatives* $r^o \in \mathbb{F}$ *and weights* $w^o \in \mathbb{R}^+$

In order to compare feature signatures, the SQFD, which is a generalization of the conventional QFD, is employed. In contrast to the well-known Earth Mover's Distance, the SQFD makes it possible to balance the tradeoff between indexability and retrieval quality [2]. The authors have demonstrated that the parameters of the similarity functions affect the indexability of the underlying data space, thus allowing to balance the tradeoff between indexability and retrieval quality. It was shown that even a very simple metric pivot table approach [23] can reach a speedup factor of up to 170 with respect to the sequential scan. In addition, the combination of the SQFD and ptolemaic pivot tables has shown a speedup factor of up to 300 [15]. In the meantime, Krulis et al. [13] came up with the idea of processing the SQFD on many-core GPU architectures. By implementing the query evaluation process on many-core GPUs and also multi-core CPUs, they have shown a significant improvement in efficiency compared to the serial approaches.

2.2 Extraction of the Feature Signatures

The feature extraction process determining a feature signature from an image consists of several consecutive steps, each of them providing several options with various sets of parameters. The basic overall schema for this process is depicted in Figure 1. The image is preprocessed first by common image algorithms. After that, a suitable sampling method is selected and several features are extracted for every sampled point. Finally, all features are clustered via the k-means clustering. In the following paragraphs, we explain details of the feature extraction and the k-means algorithm used in our implementation.

In the feature extraction step, the sampled points are mapped into the requested feature space $\mathbb{F}$. We utilize seven-dimensional representatives $f_i^o = (x, y, L, a, b, c, e) \in \mathbb{F} \subseteq \mathbb{R}^7$, where (x, y) are the coordinates of the sampled point, (L, a, b) represent the color of the sampled point mapped into the CIE Lab color space [12], and (c, e) are contrast and entropy values computed from the neighborhood of the point in the corresponding gray-scale image. We compute texture information from the gray level co-occurrence matrix G [10] extracted from the neighborhood of the point, where each point is assigned an intensity $i \in I$[1]. Since the value ranges of utilized features differ significantly, we also normalize values from each dimension into $[0, 1]$ interval.

[1] More specifically, the contrast c and entropy e are evaluated as $c = \sum_{i,j \in I} (i-j)^2 \times G(i,j)/n$ and $e = -\sum_{i,j \in I} (G(i,j)/n) \times log(G(i,j)/n)$, where $n = \sum_{i,j \in I} G(i,j)$.

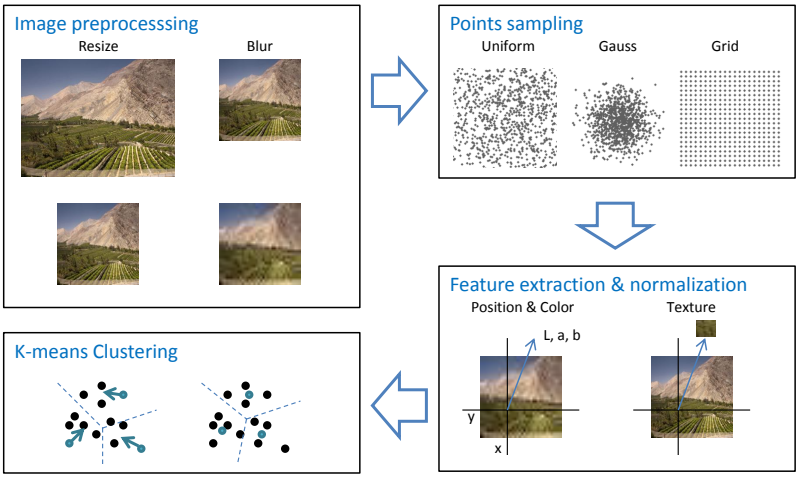


Fig. 1. Extraction schema for feature signatures

In the last step of the overall feature extraction process, all the extracted representatives $f_i^o \in \mathbb{F}$ are aggregated using the k-means clustering algorithm [17] employing the weighted L_p norm distance. The weights change the impact of each utilized feature and thus fundamentally influence the result of the clustering. By setting a weight to zero, we can even totally ignore the effect of a particular feature. The k-means clustering is an iterative method, where the number of iterations is defined by the user. In each iteration, all representatives $f_i^o \in \mathbb{F}$ are distributed within the actual set of centroids $r_i^o \in \mathbb{F}$ of the clusters $\mathcal{C}_i^o \subseteq \mathbb{F}$. Then, for each cluster a new centroid is created by averaging all the points in the cluster, which can be depicted as a shift of the old centroid to the clusters center of gravity. When using an adaptive version of the k-means clustering, we can also remove some too close or too small clusters and thus influence the number of the resulting clusters. As a result of the k-means clustering algorithm, the feature signature consisting of representatives $r_i^o \in \mathbb{F}$ and weights $w_i^o \in \mathbb{R}^+$ is created, where each representative $r_i^o \in \mathbb{F}$ corresponds to the centroid of the cluster $\mathcal{C}_i^o \subseteq \mathbb{F}$ obtained in the last iteration of the k-means, i.e., $r_i^o = \frac{\sum_{f \in \mathcal{C}_i^o} f}{|\mathcal{C}_i^o|}$, with relative frequency $w_i^o = \frac{|\mathcal{C}_i^o|}{\sum_i |\mathcal{C}_i^o|}$.

3 Implementation

Before we introduce the details of our GPU extractor, let us briefly revise current GPU architecture. Our implementation was developed and tested on the NVIDIA Fermi architecture [20], however, it should work on the new Kepler architecture as well as on the current AMD GPU devices. GPU architectures differ from CPU architectures in multiple ways. The most important two are rather specific thread execution model and complex memory model. The CPU is designed so that each core process one independent thread at a time. Threads

running on GPU all execute the same program and small groups of threads even execute the same instruction at a time (Single Instruction Multiple Threads).

The GPU is a rather independent device, so it has its own memory. This means that all input data must be transferred to the device and computed results must be transferred back to memory of the host system. Furthermore, there are multiple types of memories – *the global memory* (of several GBs), *the local (or shared) memory* (tens of kBs), and the *private memory* (registers of each core). Each memory has some specific limitations which are inevitably inherited from the parallel nature of the architecture.

We would like to summarize some of the architecture implications and best programming practices suggested by the vendor [19]:

- The latency of data transfers between the host system and the GPU devices needs to be inhibited. Therefore, we should bulk the transfers and try to overlap them with GPU computations.
- Data structures must be designed according to memory limitations of the GPU. The data placement must be considered carefully as different types of memories have different properties (especially the size and speed).
- The algorithm must embrace the SIMT execution model, at least for the parts of the work being processed by one thread group. This usually requires significant modifications of the algorithm or selection of a different algorithm solving the same problem.
- A multitude of threads (at least thousands) needs to be spawned in order to utilize all available cores and balance the load efficiently.

3.1 GPU Extractor

Our GPU extractor implementation is quite complex. In this section, we will focus on the key details that allowed us to achieve such excellent performance.

The extractor exploits two approaches to parallelism. Each signature is computed by a SIMT parallel algorithm and multiple signatures are computed concurrently. The CPU code ensures the loading of images from persistent data store, create blocks of images of appropriate size, and dispatch these blocks to the available GPUs. Each block is transferred to the GPU, then the GPU computes signatures for all images in the block, and the block of signatures is transferred back to the host memory. The blocks are dispatched so there are always two blocks assigned to one GPU. One block is being computed while the data of the other block are transferred. Furthermore, there are two CPU threads allocated for each GPU device. These threads are responsible for feeding the GPU, waiting for the GPU to terminate, and consolidation of the results.

Images in the block are processed as depicted in Figure 2. Each thread group (assigned to one SMP[2]) processes one image and all threads in the group synergically cooperates to compute the signature. The optimal block size was empirically determined as $2\times$ number of SMPs on the GPU (in our case $2 \times 16 = 32$).

[2] Symmetric Multi-Processor unit, which contains 32 synchronously running cores.

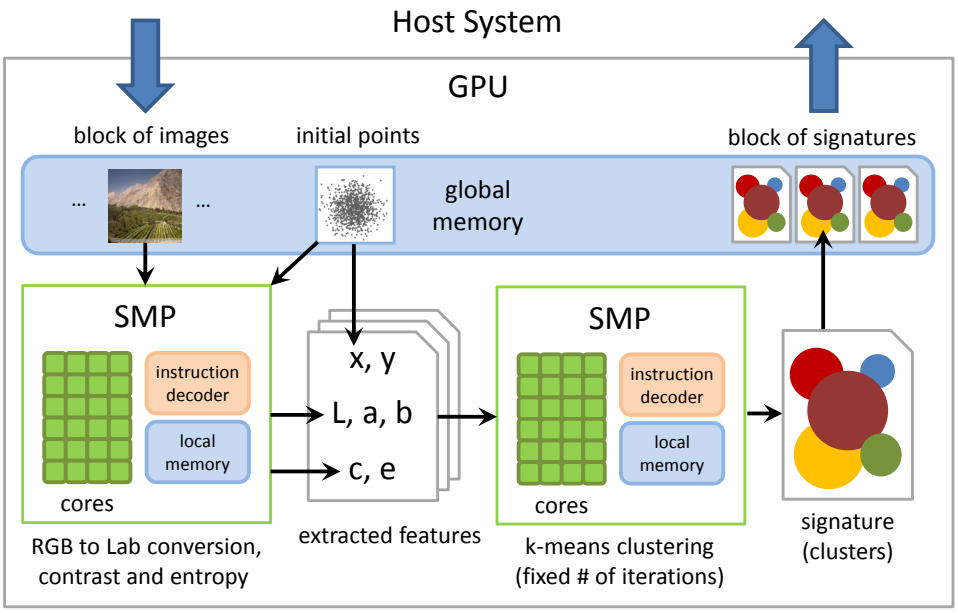


Fig. 2. Schema of the extraction process of one image

Different approaches are clearly suboptimal. Computing multiple signatures by one thread group would be highly complicated as the size of the local memory is very limited and its utilization is very important to overall performance of both feature extraction and k-means clustering. Computing one signature by multiple groups would be even more complicated as the groups have limited means of communication and synchronization.

Feature Extraction Process. The first phase of the signature creation is the feature extraction and normalization process (see Section 2.2). The sampling points (whole set) are provided by CPU and uploaded into constant global memory before the extraction is started. The extraction of the first 5 dimensions of the feature space (x, y, L, a, b) is quite straightforward. The (x, y) coordinates are computed from the sampling points coordinates as a simple linear combination. The RGB value of the corresponding pixel is taken and converted into the CIE Lab color space by transformation equations from the CIE Lab specification.

The computation of contrast and entropy features is slightly more complicated. The bitmap is converted to gray-scale using all threads in the group. Since each pixel is represented with only a few bits and the GPU natively processes data in 32-bit words, we use simple bit-packing technique that stores multiple pixels in one word. We convert the entire bitmap and keep it in the local memory of the SMP for the sake of simplicity and parallelism even though only sampled pixels and their surroundings are required for the computation.

To compute contrast and entropy, the co-occurrence matrix G must be constructed for each pixel. The matrix is rectangular $|I| \times |I|$, where I is the set of

possible intensities. Since we use 4-bit gray-scale in our experiments, the $|I| = 16$ and the matrix has 16×16 items. We allocate as many matrices as possible in the local memory and assign one thread to each matrix. These threads iteratively process initial points, construct corresponding co-occurrence matrices, and compute contrast and entropy. The time complexity of this step depends on the size of the matrix G and size of the neighborhood of the pixel, which are constant for all points. Therefore, each thread performs almost the same amount of work.

Described algorithm was designed under the assumption that it is possible to fit at least as many G matrices to local memory (along with the gs-bitmap), as there are cores on SMP, or (better) as there are threads in the corresponding group. This assumption holds in our case[3] as we are able to accommodate approximately 150 matrices in the local memory. On the other hand, if we use more bits per pixel in the gray-scale bitmap, the bitmap itself and the matrices will be significantly larger and a different algorithm could be more suitable for the problem.

K-means Clustering. The second phase is the k-means clustering performed on the points from a feature space (see Section 2.2). Since the k-means algorithm has many variations, we need to specify several details:

- We use fixed number of iterations and this number is a configurable parameter of the extractor. This way more complex images end up with more centroids in their signatures than the simple images.
- The clusters that have centroids closer than specified threshold (which is also a parameter) are merged together.
- After each iteration, clusters that are smaller than $s \times i$, where i is the number of the iteration and s is a parameter of the extractor, are thrown away. Points from these clusters are not dismissed, but rather reassigned in next iteration.
- Our algorithm does not care for the final assignment of points to clusters, but only for the final centroids and weights (number of points in each cluster).

All the threads in the group follow the algorithm steps together waiting on an explicit barrier after each step of the algorithm. One k-means iteration consists of the following steps:

1. The closest centroid is found for each point and coordinates of the point are atomically added to the new centroid coordinates (per dimension). Also the weight of the closest centroid is atomically incremented.
2. New centroid coordinates are computed dividing sums from previous step by number of points in the corresponding cluster (computing an average).
3. The clusters with centroids closer than joining threshold are merged.
4. The clusters smaller than $s \times i$ limit are disposed of.

First two steps are embarrassingly parallel. Each point may be processed independently and we assume that there are more points than threads in a group.

[3] SMP has 48 kB of local memory, thumbnails are 150×150 px in 4-bit gray-scale.

The only interesting issue is the optimal data representation. We represent each set of N d-dimensional points[4] as d arrays of N values rather than an array of N structures with d values. This representation better fits the properties of both global memory and local memory where the points and centroids are stored.

It is possible to use kd-trees or other geometric data structures to accelerate the nearest neighbour problem (the first step) [11]. We can also use an approximative approach to k-means [21]. However, these techniques are faster only for large number of centroids, and kd-trees do not perform well in the GPU memory. As we use only hundreds of centroids, empirical results show that it is better to pursue raw power of parallelism with the simplest algorithm.

The third step tests the distance of every centroid pair. The centroid pairs are iterated using the nested two-level for-loop, where only the inner loop is parallelized and the explicit barrier synchronization is performed after each iteration of the outer loop. This way the parallelism is slightly reduced, however, we do not require any means of data synchronization. In order to avoid expensive merging and array compacting, this step just sets the weight of one of the merged clusters to 0, so the cluster will be disposed of by the last step. More elaborate methods did not show any measurable speedup as the third step is significantly cheaper than the remaining steps.

The last step filters out small clusters and compacts the set of centroids, so the arrays does not contain empty elements. First, the compacting step computes new offset for each nonempty element. The offsets are computed by standard binary reduction tree algorithm performed by all available threads. Finally, all nonempty elements are copied into new compacted array at their new positions.

4 Experimental Results

In the experiments we focused on the efficiency of the extraction and the MAP evaluation process using GPU architecture, where we measured the speed up according to the CPU platform.

4.1 The Testbed

The *Thematic Web Images Collection* (TWIC) database of $11,555$ images divided in 200 classes [16] was employed for basic experiments and approximately 17.5 mil. images from the profimedia dataset [1] were used for large-data tests.

Our experiments were performed on a server with special motherboard (FT72-B7015) designed to embrace up to 8 GPUs. The server was equipped with Xeon E5645 processor that contains 6 physical (12 logical) cores running at 2.4 GHz, 96 GB of DDR3-1333 RAM, and 4 NVIDIA Tesla M2090 GPU cards (Fermi architecture). Each GPU chip has 512 cores (32 cores per 16 SMPs) and 6 GB of memory. We also tested the implementation on commodity PC with two NVIDIA GTX 580 which have also 512 cores, but only 1.5 GB of memory. We have found that in our case the gaming GTX 580 cards have similar performance as the much more expensive Tesla cards, thus we do not provide detailed comparison.

[4] As described in 2.2, the $d = 7$ in our case.

4.2 Performance Tests

Since the major contribution of this paper is a fast GPU extraction implementation, we provide performance test results and comparison to the CPU extractor. All the times were measured using the system real-time clock. We are aware that this method is not entirely precise and there are many both technical and philosophical issues regarding performance benchmarks. However, these tests are designed to give the reader a general idea about the performance rather than provide an accurate comparison. All tests were conducted using parameters that produced the highest precision results.

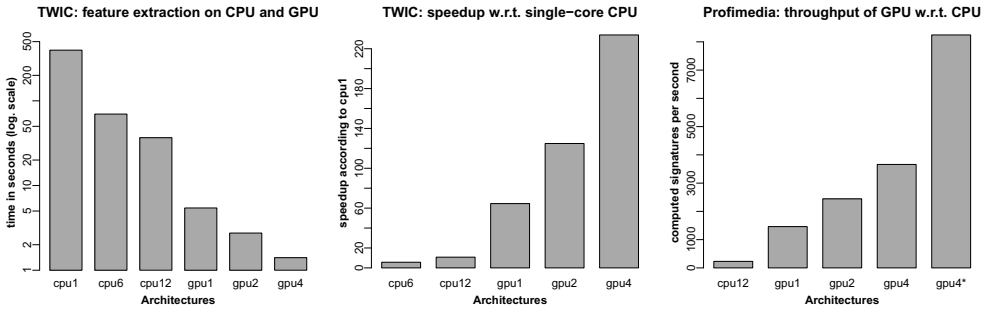


Fig. 3. Time, speedup, and throughput comparisons

Figure 3 summarizes the times and speedup of the experiments conducted on CPU (using different numbers of threads) and GPU (using different numbers of devices). The cpu t methods designate tests running on CPU with t threads[5] and the gpu d methods designate tests running on d GPU devices. The times depicted in the first graph are separated into two columns – the extraction process of $11,555$ images (TWIC dataset [16]), which we used for tests designed to explore the parameter space of the extractor.

The tests were evaluated 233× faster on 4 GPUs than on single-core CPU and 21.7× faster than on 12 core CPU. Thanks to this speedup, we were able to conduct all experiments presented in previous section in the matter of hours. The same experiments would take days on 12 CPU cores and weeks on single-core.

We have also considered using the GPU extractor for the indexing and video stream processing techniques. For testing purposes, we have extracted a database of $1,000,000$ images [1] and created signature index by both GPU and CPU extractors. The extractor running on 4 GPUs can extract 8244 signatures per second while on 12 CPU cores the throughput is only 303 signatures per second. These tests were conducted so that all images were pre-cached in RAM, hence the extractor has not been slowed down by loading data from persistent storage. Since each image has size of approximately 33.9 KB, the system would require a persistent storage that is capable of reading data at the minimum rate of 273.3 MB/s, which cannot be easily achieved by common hard disk drives. Previous

[5] The tests run up to 12 threads as we have 12 core CPU.

tests were focused on the speed of the extractor. If the extractor is employed as an indexing service, the performance of the persistent data storage cannot be ignored. We have equipped our server with two RAID 0 arrays both containing two common hard disks One array kept the input images and the other array was used for storing signatures. We have used the same data source as for the previous experiment, but we took 17.5 millions of images to ensure that the data nor the result will fit the RAM. The throughput of the extractor is depicted in the third graph of the Figure 3. The speed of the extractor has dropped to 3661 signatures per second on four GPUs. It is 2.25× slower than previous experiment, in which the images were cached in RAM (denoted gpu4* in the graph). Based on the empirical data, we speculate that the feature extraction system would require at least 4 modern SSD drives connected to RAID 0 in order to match the speed of 4 GPU devices.

5 Conclusions

In this paper, we present a highly efficient GPU implementation of the feature extraction of image signatures, reaching more than two orders of magnitude speedup with respect to classical CPU platform. It also achieves a throughput of 8244 signatures per second, which is far beyond the throughput of common hard drives or network devices. Fast feature extraction implementation is critical in many recent applications like video stream processing or image exploration techniques, where for user interaction scenarios the low response times are essential. Our GPU implementation can be used also in traditional indexing or training tasks, where huge datasets have to be extracted or broad parameter spaces have to be inspected.

Acknowledgments. This research has been supported in part by Czech Science Foundation projects P202/11/0968, P202/12/P297 and Charles University grant agency (GAUK) project 277911.

References

1. Profimedia Image Database, `http://www.profimedia.cz/`
2. Beecks, C., Lokoč, J., Seidl, T., Skopal, T.: Indexing the signature quadratic form distance for efficient content-based multimedia retrieval. In: Proc. ACM Int. Conf. on Multimedia Retrieval, pp. 24:1–24:8 (2011)
3. Beecks, C., Seidl, T.: Analyzing the inner workings of the signature quadratic form distance. In: Proc. IEEE Int. Conference on Multimedia and Expo. (2011)
4. Beecks, C., Uysal, M.S., Seidl, T.: Signature quadratic form distances for content-based similarity. In: Proc. 17th ACM Int. Conference on Multimedia (2009)
5. Beecks, C., Uysal, M.S., Seidl, T.: A comparative study of similarity measures for content-based multimedia retrieval. In: Proc. IEEE International Conference on Multimedia & Expo. (2010)
6. Beecks, C., Uysal, M.S., Seidl, T.: Signature quadratic form distance. In: Proc. ACM International Conference on Image and Video Retrieval, pp. 438–445 (2010)

7. Chechik, G., Sharma, V., Shalit, U., Bengio, S.: Large scale online learning of image similarity through ranking. J. Mach. Learn. Res. 11, 1109–1135 (2010)
8. Datta, R., Joshi, D., Li, J., Wang, J.Z.: Image retrieval: Ideas, influences, and trends of the new age. ACM Comput. Surv. 40(2), 5:1–5:60 (2008)
9. Deselaers, T., Keysers, D., Ney, H.: Features for image retrieval: an experimental comparison. Information Retrieval 11(2), 77–107 (2008)
10. Gotlieb, C.C., Kreyszig, H.E.: Texture descriptors based on co-occurrence matrices. Comput. Vision Graph. Image Process. 51(1), 70–86 (1990)
11. Kanungo, T., Mount, D., Netanyahu, N., Piatko, C., Silverman, R., Wu, A.: An efficient k-means clustering algorithm: Analysis and implementation. IEEE Transactions on Pattern Analysis and Machine Intelligence 24(7), 881–892 (2002)
12. Kasson, J.M., Plouffe, W.: An analysis of selected computer interchange color spaces. ACM Trans. Graph. 11(4), 373–405 (1992)
13. Krulis, M., Lokoc, J., Beecks, C., Skopal, T., Seidl, T.: Processing the signature quadratic form distance on many-core gpu architectures. In: CIKM, pp. 2373–2376 (2011)
14. Lokoč, J., Grošup, T., Skopal, T.: Image exploration using online feature extraction and reranking. In: Proceedings of the 2nd ACM Int. Conference on Multimedia Retrieval, ICMR 2012, pp. 66:1–66:2. ACM, New York (2012)
15. Lokoč, J., Hetland, M.L., Skopal, T., Beecks, C.: Ptolemaic indexing of the signature quadratic form distance. In: Proceedings of the Fourth International Conference on Similarity Search and Applications (SISAP). Springer (2011)
16. Lokoč, J., Novák, D., Skopal, T., Sibirkina, N.: Thematic Web Images Collection, SIRET Research Group (2012), http://siret.ms.mff.cuni.cz/twic
17. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: Proc. Fifth Berkeley Sympos. Math. Statist. and Probability (1967)
18. Mikolajczyk, K., Schmid, C.: A performance evaluation of local descriptors. IEEE Transactions on Pattern Analysis and Machine Intelligence 27(10), 1615–1630 (2005)
19. NVIDIA. CUDA Programming Guide, http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf
20. NVIDIA. Fermi GPU Architecture, http://www.nvidia.com/object/fermi_architecture.html
21. Philbin, J., Chum, O., Isard, M., Sivic, J., Zisserman, A.: Object retrieval with large vocabularies and fast spatial matching. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2007, pp. 1–8. IEEE (2007)
22. Tversky, A.: Features of similarity. Psychological Review 84(4), 327–352 (1977)
23. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. In: Advances in Database Systems, Springer-Verlag New York, Inc., Secaucus (2005)

Chapter 3

Approximating the Signature Quadratic Form Distance Using Scalable Feature Signatures

Jakub Lokoč

Published in the proceedings of the 20th International Conference on Multi-Media Modelling MMM 2014, LNCS, ISSN 0302-9743.
dx.doi.org/10.1007/978-3-319-04114-8_8

The extended version of this paper [52] was accepted to Multimedia Tools and Applications journal (IF in 2014: 1.346) and is in press now.

Approximating the Signature Quadratic Form Distance Using Scalable Feature Signatures*

Jakub Lokoč

SIRET research group, Department of Software Engineering,
Faculty of Mathematics and Physics, Charles University in Prague
lokoc@ksi.mff.cuni.cz

Abstract. The feature signatures in connection with the signature quadratic form distance have become a respected similarity model for effective multimedia retrieval. However, the efficiency of the model is still a challenging task because the signature quadratic form distance has quadratic time complexity according to the number of tuples in feature signatures. In order to reduce the number of tuples in feature signatures, we introduce the scalable feature signatures, a new formal framework based on hierarchical clustering enabling definition of various feature signature reduction techniques. As an example, we use the framework to define a new feature signature reduction technique based on joining of the tuples. We experimentally demonstrate our new feature signature reduction technique can be used to implement more efficient yet effective filter distances approximating the original signature quadratic form distance. We also show the filter distances using our new feature signature reduction technique significantly outperform the filter distances based on the related maximal component feature signatures.

Keywords: Similarity Search, Approximate Search, Content-based Retrieval, Signature Quadratic Form Distance, Scalable Descriptor.

1 Introduction and Related Work

The content-based multimedia retrieval [6] has become an integral part of various information systems managing multimedia data (e.g., e-shops, image banks, industry and medical systems), providing users an alternative to the keyword-based retrieval approaches. In order to search the multimedia data in the content-based way, the systems often employ a similarity model enabling ranking of the database objects according to a query object, where the similarity model comprises multimedia data descriptors and a suitable similarity measure defined for the utilized descriptors. The selection of a proper similarity model then belongs among key tasks when designing an effective and efficient content-based multimedia retrieval system. During the last decades, many types of similarity models

* This research has been supported by Czech Science Foundation project GAČR P202/12/P297.

C. Gurrin et al. (Eds.): MMM 2014, Part I, LNCS 8325, pp. 86–97, 2014.

have been designed and even standardized for a particular multimedia retrieval tasks (e.g., the MPEG-7 standard [15]). One of the most popular similarity models investigated during the last decade is the bag of visual words (BoVW) model [18], utilizing a statically-created vocabulary of codewords, so called codebook. In the BoVW model, each object is represented as a frequency histogram of codewords present in the object, where all the objects in a database share one codebook. Such representation enables efficient retrieval using inverted files, a well established technique for the text-based retrieval area. Whereas the efficiency of the BoVW model is sufficient for large scale multimedia retrieval, the practical effectiveness of the model is still an open problem. While recent works have tried to improve the effectiveness of the BoVW model using semantic preserving models [19], Hamming embedding [10], compressed Fisher vectors [16] or vectors of locally aggregated features [11], several new approaches have relaxed from a common static vocabulary and investigated more general similarity models based on the feature signatures [17] and the adaptive distance measures (e.g., Signature Quadratic Form Distance [3] or Signature Matching Distance [1]). The signature-based models utilize an object specific vocabulary and thus can flexibly represent the contents of an object. Hence, the feature signatures can capture more disparities in the data, which can be beneficial in dynamic databases rapidly changing in content (e.g., multimedia streams). As recently shown, several signature-based models can outperform the BoVW approaches in the terms of effectiveness [1], however, the efficiency of the signature-based models is still a challenging task, especially for feature signatures comprising a high number of tuples. In [9], the authors employ metric/ptolemaic indexing to improve the efficiency of the retrieval, however, the approach is restricted only to distances satisfying metric/ptolemaic postulates. In [13], the authors show signature-based models can be utilized for effective re-ranking when obtaining a candidate result set using an efficient model based on a subset of the MPEG-7 descriptors. In this paper, we focus on new feature signature reduction techniques enabling more efficient yet still effective retrieval. Furthermore, we consider also scalability of the reduction techniques enabling adjusting the size of the feature signatures according to the actual system load. Let us now recall several basic concepts and definitions referred in this paper.

1.1 Feature Signatures and Signature Quadratic Form Distance

Feature signatures [17] have been introduced to flexibly aggregate and represent the contents of a multimedia object mapped into a feature space $\mathbb{F}$. Whether the requested feature space $\mathbb{F}$ comprises color, position, texture information, SIFT gradient vectors or other complex features [7,14], the feature signatures are often obtained by an adaptive variant of the k-means clustering selecting the most significant centroids. In Figure 1, we depict an example of image feature signatures according to a CPT feature space[1]. The feature signatures were extracted using the GPU extractor [12] employing an adaptive k-means clustering

[1] Color $\langle L, a, b\rangle$, position $\langle x, y\rangle$ and texture information $\langle contrast, entropy\rangle$, $\mathbb{F} \subseteq \mathbb{R}^7$.

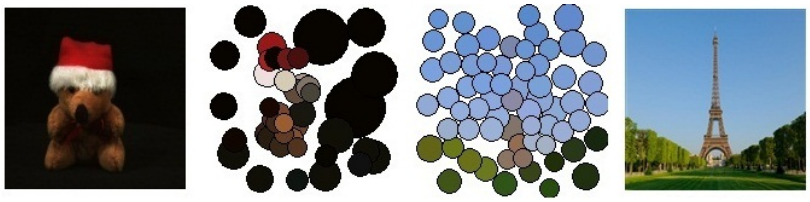


Fig. 1. Example of feature signatures

algorithm, where the extraction of the first image in Figure 1 has put stress on the color, while the extraction of the second image in the figure has put stress on the position. The representatives $r_i \in \mathbb{F}$ corresponding to the selected centroids are depicted by circles in the corresponding position and color, while the weights $w_i \in \mathbb{R}^+$ corresponding to the size of the cluster determine the diameter of the circles (texture information is not depicted). Formally, the feature signatures are defined as:

Definition 1 (Feature Signature). *Given a feature space $\mathbb{F}$, the feature signature S^o of a multimedia object o is defined as a set of tuples $\{\langle r_i^o, w_i^o \rangle\}_{i=1}^n$ from $\mathbb{F} \times \mathbb{R}^+$, consisting of representatives $r_i^o \in \mathbb{F}$ and weights $w_i^o \in \mathbb{R}^+$*

The number of tuples in a feature signature can vary depending on a complexity of a corresponding multimedia object and the parameters used for the extraction. As a consequence, a feature signature can comprise tens or hundreds of tuples, which significantly affects the time for similarity computations. In [2], the authors propose a simple feature signature reduction technique based on maximal components of a feature signature O, where the maximal component feature signature O_{MC} with c components is defined as: $O_{MC} \subseteq O, |O_{MC}| = c$, such that $\forall \langle r_i^o, w_i^o \rangle \in O_{MC}, \forall \langle r_j^o, w_j^o \rangle \in O - O_{MC} : w_i^o \geq w_j^o$. In other words, the maximal component feature signature contains c tuples with the highest weights. The authors also define a signature quadratic form filter distance, applicable for approximate filter and refine retrieval, where the filter distance just evaluates the signature quadratic form distance using maximal component feature signatures. In Figure 2, we depict an example of maximal component feature signatures with 10 and 20 components. We may observe the maximal component feature signatures can omit representative tuples from the original feature signatures (the rightmost signatures in Figure 2) when few maximal components are utilized.

Let us now shortly recall the Signature Quadratic Form Distance [3], an effective adaptive distance measure generalizing the quadratic form distance.

Definition 2 (SQFD). *Given two feature signatures $S^o = \{\langle r_i^o, w_i^o \rangle\}_{i=1}^n$ and $S^p = \{\langle r_i^p, w_i^p \rangle\}_{i=1}^m$ and a similarity function $f_s : \mathbb{F} \times \mathbb{F} \to \mathbb{R}$ over a feature space $\mathbb{F}$, the signature quadratic form distance SQFD_{f_s} between S^o and S^p is defined as:*

$$\mathrm{SQFD}_{f_s}(S^o, S^p) = \sqrt{(w_o \mid -w_p) \cdot A_{f_s} \cdot (w_o \mid -w_p)^T},$$

where $A_{f_s} \in \mathbb{R}^{(n+m)\times(n+m)}$ *is the similarity matrix arising from applying the similarity function* f_s *to the corresponding feature representatives, i.e.,* $a_{ij} = f_s(r_i, r_j)$*. Furthermore,* $w_o = (w_1^o, \ldots, w_n^o)$ *and* $w_p = (w_1^p, \ldots, w_m^p)$ *form weight vectors, and* $(w_o \mid -w_p) = (w_1^o, \ldots, w_n^o, -w_1^p, \ldots, -w_m^p)$ *denotes the concatenation of weight vectors* w_o *and* $-w_p$*.*

To determine similarity values between all pairs of representatives from the feature signatures, the Gaussian similarity function $f_{gauss}(r_i, r_j) = e^{-\alpha L_2^2(r_i, r_j)}$ or the Heuristic similarity function $f_{heuristic}(r_i, r_j) = 1/(\alpha + L_2(r_i, r_j))$ can be utilized, where α is a parameter for controlling the precision, and L_2 denotes the Euclidean distance. If we utilize similarity function $f_{L_2}(r_i, r_j) = -L_2^2(r_i, r_j)/2$, we obtain the L_2-Signature quadratic form distance [4] suffering from worse effectiveness but computable in linear time.

The rest of the paper is structured as follows: we present the scalable feature signatures and our new reduction technique in the following section, then we experimentally demonstrate in section 3 our new reduction technique can be employed for effective approximate search with the signature quadratic form distance, and finally we conclude the paper and point on the future work in section 4.

2 Scalable Feature Signatures

In this section, we introduce the *scalable feature signatures* – a formal framework based on hierarchical clustering enabling definition of sophisticated reduction strategies for feature signatures. The framework extends and generalizes the maximal component feature signatures [2] primarily designed for approximate filter and refine retrieval. As we experimentally demonstrate, the filter signature quadratic form distance employing the maximal component feature signatures does not approximate the original distance (or its lower bound) well, and thus we focus on new filter distances using new feature signature reduction techniques providing better approximations of the original feature signatures. Unlike the maximal components approach that just removes tuples with small

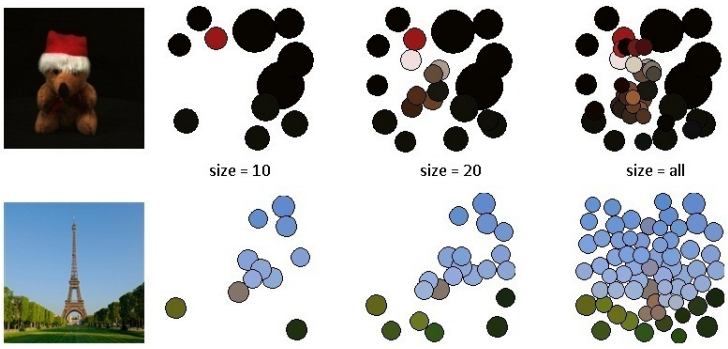


Fig. 2. Maximal component feature signatures

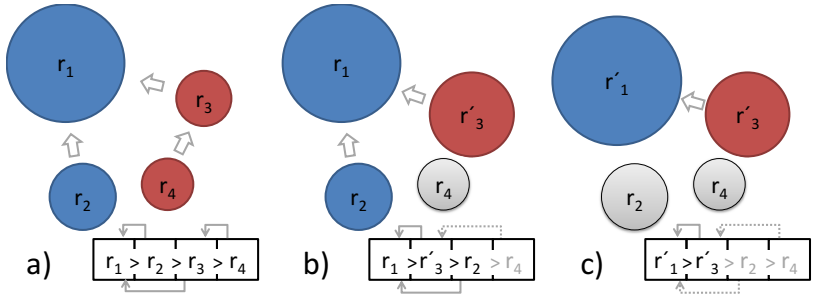


Fig. 3. Scaling feature signature

weights, our new approach aggregates the tuples during the reduction of the feature signatures. Our new approach is motivated by the feature signature extraction process [12], where the adaptive k-means clustering removes centroids with small weights, while the points distributed within the removed centroids are assigned to the remaining centroids. However, after the extraction process is finished and new feature signatures are stored, the points are no longer available and thus only information in the stored tuples can be used for the reduction of feature signatures.

Before we proceed to formal definitions, let us describe a motivation example depicted in Figure 3 where the feature signature FS in Figure 3a is consecutively reduced to the half of the original size in Figure 3c. If the maximal components approach was used, the reduced feature signature would contain only two blue tuples, which would not correspond to the original image. Therefore, instead of removing tuples, we can join them using an aggregation function τ to keep the original information at least in the aggregated form. To determine which tuples are joined in each step, we expect a total ordering $>$ defined over all the tuples in $\mathbb{F} \times \mathbb{R}^+$ and a mapping function ϕ^{FS} defined for all tuples in FS depicted as gray arrows in Figure 3. Using a suitable $>$, ϕ^{FS} and τ, we may observe the reduced feature signature can be a good approximation of the original feature signature (as depicted in Figure 3). Furthermore, if we store one of the original tuples and a pointer to the joined tuple after each join operation, we can later utilize a reverse split operation to reconstruct the original feature signature (or just less reduced feature signature). Such scalability property of the descriptor can be beneficial because we can balance the actual size of the feature signatures according to actual performance needs of a multimedia retrieval system. Let us also emphasize, the reduction process should be deterministic in order to enable preprocessing optimizations for a particular distance functions.

In the following paragraphs we provide definitions formalizing the key concepts described in the motivation example, starting with the definition of the scalable feature signatures.

Definition 3 (Scalable Feature Signature). *Given a feature signature FS over a feature space $\mathbb{F}$, a total ordering $>$ defined over all tuples in $\mathbb{F} \times \mathbb{R}^+$, a total*

mapping function $\phi^{FS} : FS \to FS$ *and an aggregation function* $\tau : (\mathbb{F} \times \mathbb{R}^+)^2 \to \mathbb{F} \times \mathbb{R}^+$*, then the tuple* $(FS, >, \phi^{FS}, \tau)$ *is called scalable feature signature.*

In the following paragraphs, we show an example of the total ordering and several examples of mapping and aggregation functions. Let $\mathbb{F}$ be the euclidean space over field $\mathbb{R}^n$ and FS be a feature signature over that feature space. We can define a total ordering $>$ using weights and the lexicographic ordering $>_{lex}$ over vectors in $\mathbb{R}^n$ as: $\forall \langle r_i, w_i \rangle, \langle r_j, w_j \rangle \in \mathbb{F} \times \mathbb{R}^+ : \langle r_i, w_i \rangle >_{wl} \langle r_j, w_j \rangle$ if and only if $w_i > w_j \vee (w_i = w_j \wedge r_i >_{lex} r_j)$.

The mapping function ϕ^{FS} can utilize the total ordering $>_{wl}$ and can be defined for each tuple $\langle r_i, w_i \rangle \in FS$ as:

$\phi^{FS}_{min}(\langle r_i, w_i \rangle) = \langle r_i, w_i \rangle$ for $\langle r_i, w_i \rangle = \max_{>_{wl}} FS$, and otherwise as:

$\phi^{FS}_{min}(\langle r_i, w_i \rangle) = \min_{>_{wl}} \{\langle r_j, w_j \rangle : \langle r_j, w_j \rangle \in FS \wedge \langle r_j, w_j \rangle >_{wl} \langle r_i, w_i \rangle\}$.

The mapping function ϕ^{FS}_{min} just maps each tuple from FS to the first greater tuple in FS, except for the maximal tuple that is mapped to itself. The mapping function can consider also a Minkowski distance L_p between the representatives in FS as follows:

$\phi^{FS}_{L_p}(\langle r_i, w_i \rangle) = \langle r_i, w_i \rangle$ for $\langle r_i, w_i \rangle = \max_{>_{wl}} FS$, and otherwise as:

$\phi^{FS}_{L_p}(\langle r_i, w_i \rangle) = \langle r_j, w_j \rangle$ such that $\langle r_j, w_j \rangle \in FS \wedge \langle r_j, w_j \rangle >_{wl} \langle r_i, w_i \rangle \wedge (\forall \langle r_k, w_k \rangle \in FS, k \neq i \neq j : \langle r_k, w_k \rangle >_{wl} \langle r_i, w_i \rangle \implies (L_p(r_j, r_i) < L_p(r_k, r_i) \vee (L_p(r_j, r_i) = L_p(r_k, r_i) \wedge \langle r_k, w_k \rangle >_{wl} \langle r_j, w_j \rangle)))$.

The aggregation operation τ can be defined trivially as a projection:

$\tau_{first}(\langle r_i, w_i \rangle, \langle r_j, w_j \rangle) = \langle r_i, w_i \rangle$,

or as a more complex aggregation:

$\tau_{avg}(\langle r_i, w_i \rangle, \langle r_j, w_j \rangle) = \langle r_i \cdot w_i/(w_i + w_j) + r_j \cdot w_j/(w_i + w_j), w_i + w_j \rangle$.

Having defined scalable feature signature $(FS, >, \phi^{FS}, \tau)$, we can now define an unary reduction operation that replaces the minimum $\langle r, w \rangle$ in FS and the corresponding tuple $\phi^{FS}(\langle r, w \rangle)$ by the join tuple $\tau(\langle r, w \rangle, \phi^{FS}(\langle r, w \rangle))$.

Definition 4 (Scalable Feature Signature Reduction). *Given a scalable feature signature* $SFS = (FS, >, \phi^{FS}, \tau)$*, let* $\langle r, w \rangle = \min_{>} FS$*, let* $FS' = (FS - \{\phi^{FS}(\langle r, w \rangle), \langle r, w \rangle\})$ *and let* $\langle r_k, w_k \rangle = \tau(\phi^{FS}(\langle r, w \rangle), \langle r, w \rangle)$*, then the reduction of scalable feature signature* SFS *denoted as* $\otimes SFS$ *is defined as* $\otimes SFS = (FS_r, >, \phi^{FS_r}, \tau)$*, where* $FS_r = FS' \cup \{\langle r_k, w_k \rangle\}$ *for* $\langle r_k, w_k \rangle \notin FS'$*, and* $FS_r = FS' - \{\langle r_k, w_k \rangle\} \cup \{\langle r_k, 2 \cdot w_k \rangle\}$ *otherwise.*

Let as denote ϕ^{FS_r} is defined in the same way as ϕ^{FS}, but in the context of new feature signature FS_r. We provide also a simple lemma to emphasize the unary reduction operation creates another scalable feature signature. The lemma is without proof because it is a direct consequence of the previous definitions.

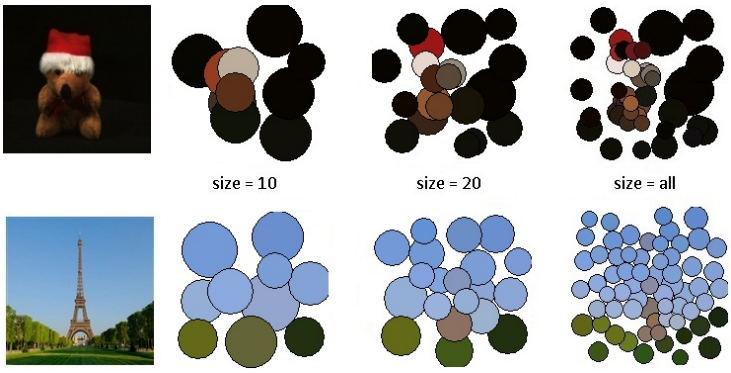


Fig. 4. Scalable feature signatures using $\phi^{FS}_{L_2}$ and τ_{avg}

Lemma 1. *Let* $(FS, >, \phi^{FS}, \tau)$ *be a scalable feature signature over a feature space* $\mathbb{F}$*, then* $\otimes(FS, >, \phi^{FS}, \tau)$ *is also a scalable feature signature over the feature space* $\mathbb{F}$*.*

So far we have provided a formal framework enabling definition of various feature signature reduction techniques. Using the framework, we can simply define our new reduction technique based on joining of the tuples as a quintuplet $(FS, >_{wl}, \phi^{FS}_{L_2}, \tau_{avg}, \otimes)$, consisting of the scalable feature signature $(FS, >_{wl}, \phi^{FS}_{L_2}, \tau_{avg})$ and the reduction operation $\otimes$. In Figure 4, we may observe our new reduction technique can approximate the distribution of the tuples in the original feature signature well even for smaller number of tuples.

Let us now provide several notes, for the lack of the space without proofs. First, the $\otimes(FS, >, \phi^{FS}, \tau)$ can create new scalable feature signature with $|FS|$, $|FS| - 1$ or $|FS| - 2$ tuples depending on the result of ϕ^{FS} and τ. Second, in case $|FS| = 1$, the reduction operation does not have to be identity, for example, if the aggregation function τ penalizes one of the arguments. Third, $(FS, >_{wl}, \phi^{FS}_{min}, \tau_{first})$ with the reduction operation $\otimes$ corresponds, except for minor differences[2], to the maximal component feature signatures. However, in the text we will strictly use the label *maximal component feature signatures* in order to distinguish the related work from the scalable feature signatures based on joining of the tuples.

Having defined an operation reducing the size of a scalable feature signature, we can now proceed to the definition of a new signature quadratic form filter distance, generalizing the filter signature quadratic form distance $SQFD_{filter}$ defined in [2], where the filter distance is utilized for the approximate search in a filter and refine architecture. In order to express multiple superpositions of the unary operation $\otimes$, we use in the following definition $\otimes^n(FS, >, \phi^{FS}, \tau)$ notation as a shortcut for $\underbrace{\otimes \cdots \otimes}_{n-times}(FS, >, \phi^{FS}, \tau)$.

[2] The maximal component feature signatures do not assume a total ordering of the tuples.

Definition 5 (Signature Quadratic Form Filter Distance). *Given two reduced scalable feature signatures* $(FS_{r_1}, >, \phi^{FS_{r_1}}, \tau) = \otimes^{|FS_1|-n}(FS_1, >, \phi^{FS_1}, \tau)$ *and* $(FS_{r_2}, >, \phi^{FS_{r_2}}, \tau) = \otimes^{|FS_2|-n}(FS_2, >, \phi^{FS_2}, \tau)$ *over a feature space* $\mathbb{F}$*, and let* $SQFD$ *be the signature quadratic form distance, then the distance* $SQFD_f^n = SQFD(FS_{r_1}, FS_{r_2})$ *is called the signature quadratic form filter distance according to* $SQFD(FS_1, FS_2)$*.*

The filter distance just simply reduces the original scalable feature signatures to a requested size and evaluates the original distance measure for the two reduced feature signatures. If we define the scalable feature signatures using $>_{wl}$, ϕ_{min}^{FS} and τ_{first} (which corresponds to the maximal component feature signatures), then the signature quadratic form filter distance $SQFD_f^n$ corresponds to the filter distance $SQFD_{filter}$ presented in [2]. The new signature quadratic form filter distance can be also utilized for the approximate search in a filter and refine schemes, where the reduced scalable feature signatures can be either cached or evaluated every time the filter distance is requested. Furthermore, such retrieval system can decide to temporarily use a reduced version of the scalable feature signatures also for the refinement step. In order to prevent from storing multiple versions of the scalable feature signatures, we can implement the reduction operation as a reversible update of the original scalable feature signatures enabling to keep just one actual version of the scalable feature signatures.

For example, the reduction operation in Figure 3ab replaces $\phi_{L_2}^{FS}(\langle r_4, w_4\rangle) = \langle r_3, w_3\rangle$ by $\langle r_{3'}, w_{3'}\rangle = \tau_{avg}(\langle r_3, w_3\rangle, \langle r_4, w_4\rangle)$, removes $\langle r_4, w_4\rangle$, inserts pair ($\langle r_4, w_4\rangle$, pointer to $\langle r_{3'}, w_{3'}\rangle$) into a stack and sorts the tuples. The corresponding reverse operation removes pair ($\langle r_4, w_4\rangle$, pointer to $\langle r_{3'}, w_{3'}\rangle$) from the stack, inserts $\langle r_4, w_4\rangle$ into the feature signature, replaces $\langle r_{3'}, w_{3'}\rangle$ by $\tau_{avg}^{rev}(\langle r_{3'}, w_{3'}\rangle$, $\langle r_4, w_4\rangle) = \langle r_3, w_3\rangle$ and sorts the tuples, where τ_{avg}^{rev} is derived from τ_{avg} as:

$$\tau_{avg}^{rev}(\langle r_i, w_i\rangle, \langle r_j, w_j\rangle) = \langle (r_i - r_j \cdot w_j/w_i) \cdot w_i/(w_i - w_j), w_i - w_j\rangle.$$

3 Experimental Evaluation

For the experiments, we make use of the three different datasets, each with different source of ground truth. Specifically, we use a subset of the ALOI dataset [8] comprising 12,000 images divided into 1,000 classes, each class contain 12 images of a 3D object rotated by 30 degrees; a subset of the Profimedia dataset [5] comprising 21,993 images divided into 100 classes, where the ground truth was collected semi-automatically and verified by users; the TWIC dataset [13] comprising 11,555 images forming 197 classes, where each class represents images obtained by a keyword query to the google images search engine. Each TWIC class was further manually filtered by users. The feature signatures were extracted using a GPU extractor tool [12]. For all three datasets we have used the same extractor parameters except the multiplicative vector that was adjusted to each dataset separately. The average number of tuples in feature signatures was 33 for ALOI dataset and 66 for TWIC and Profimedia datasets. As the query

Table 1. The time (in milliseconds) needed to evaluate the filter distances using various number of tuples (1, 2, 4, ..., 64) and the signature quadratic form distance (all)

	1	2	4	8	16	32	64	all
Gaussian	0.0006	0.0007	0.0015	0.0039	0.013	0.050	0.193	0.205
Heuristic	0.0004	0.0006	0.0013	0.0031	0.010	0.038	0.149	0.159

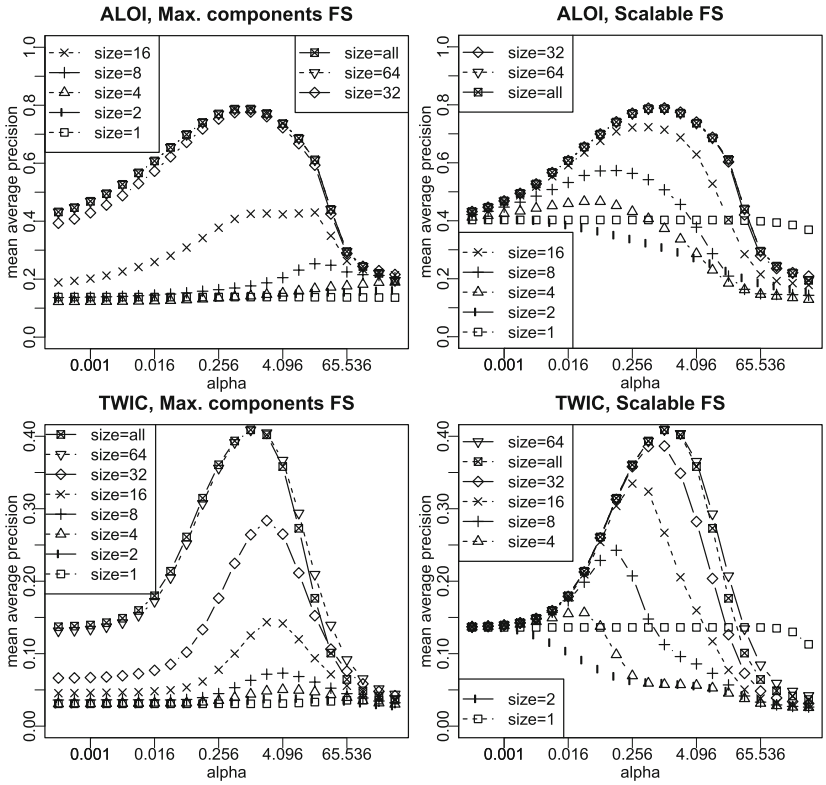


Fig. 5. Mean average precision of the filter distance $SQFD_f^{size}$ utilizing Gaussian similarity function

objects, one representative from each class was selected for all three datasets[3], resulting in 1000 query objects for ALOI, 100 query objects for Profimedia and 197 query objects for TWIC. The experiments have run on 64-bit Windows Server 2008 R2 Standard with Intel Xeon CPU X5660, 2.8 GHz.

In the experiments, we use $(FS, >_{wl}, \phi_{min}^{FS}, \tau_{first})$ as an implementation of the maximal component feature signatures and compare them to the scalable feature signatures using joining of the tuples $(FS, >_{wl}, \phi_{L_2}^{FS}, \tau_{avg})$. For each reduction technique, we utilize six variants of the filter distance $SQFD_f^{size}$ using $size \in \{1, 2, 4, 8, 16, 32, 64\}$ and compare them to the original distance denoted

[3] Profimedia dataset is already provided with a set of query objects.

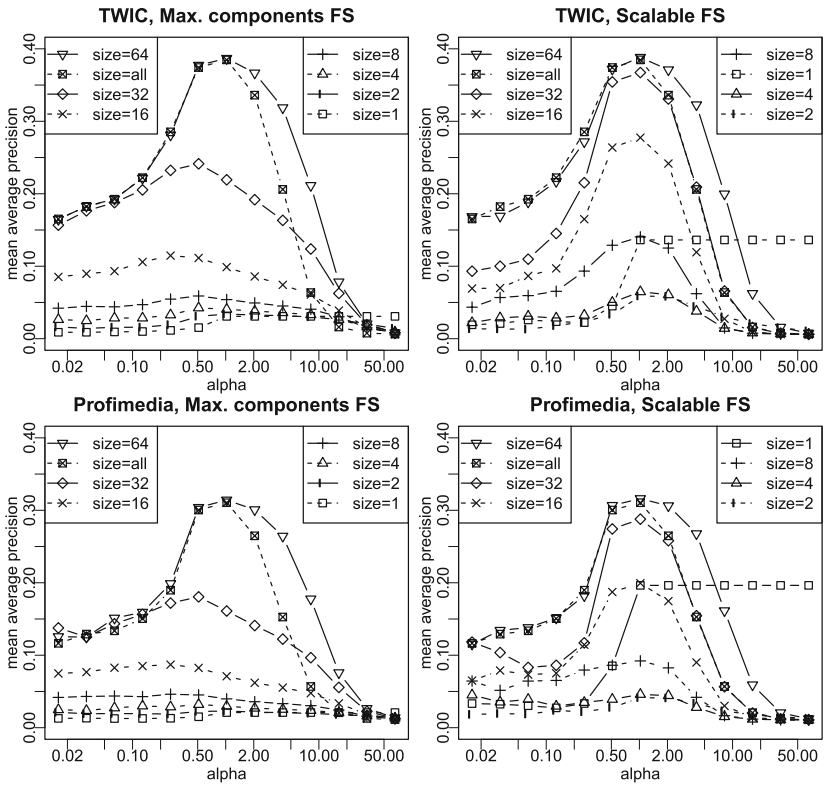


Fig. 6. Mean average precision of the filter distance $SQFD_f^{size}$ utilizing Heuristic similarity function

as *size = all*. Before we proceed to the experiments comparing the two reduction techniques, we present a table of average times (in milliseconds) needed to evaluate the utilized filter distances and the original distance, measured for the TWIC dataset for both Gaussian and Heuristic similarity functions. In Table 1, we may observe the Heuristic similarity function is slightly faster then the Gaussian similarity function. We may also observe the expected quadratic time dependency of the signature quadratic form filter distance on the number of tuples in the reduced feature signatures.

Let us now proceed to the following two figures, where the filter distances utilizing the Gaussian similarity function are depicted in Figure 5 and the filter distances utilizing the heuristic similarity function are depicted in Figure 6. In both figures, we have focused on the mean average precision (y-axis) measured for varying parameter α (x-axis). The figures are organized into two columns, where the first column contains results for the maximal component feature signatures (denoted as Max. components FS), while the second column contains the results for the scalable feature signatures using joining of the tuples (denoted simply as Scalable FS). Let us also denote, we have unified the y-axis scaling for

each row and thus the reader can directly compare the effectiveness of two corresponding filter distances. In all the graphs we may observe the similar behavior – when decreasing the size of the reduced feature signatures, the filter distances using maximal component feature signatures loose the effectiveness more rapidly than the filter distances using scalable feature signatures based on joining of the tuples. For example, in the second row of Figure 5, we may observe a markable difference between the corresponding pairs of filter distances for signatures comprising 32 and less tuples, where for scalable feature signatures using joining of the tuples the mean average precision is over 30% even for just 16 tuples, while for the same number of tuples and the maximal component feature signatures the mean average precision is just 15%. From the experiments, we may conclude the scalable feature signatures using joining of the tuples provide better filter distances than the maximal component feature signatures.

4 Conclusions and Future Work

In this paper, we have introduced the scalable feature signatures, a formal framework enabling definition of various reduction techniques for feature signatures. As an example, we have defined a new feature signature reduction technique employing joining of the tuples and utilized the technique for definition of effective signature quadratic form filter distances. We have also experimentally demonstrated the filter distances using our new reduction technique significantly outperform the filter distances using maximal component feature signatures. In the future, we plan to examine the scalable feature signatures with other adaptive distance measures and measure the effectiveness of the corresponding similarity models. We would also like to design more complex mapping and joining functions in order to provide more options for the reduction of the scalable feature signatures. We also plan to investigate the performance of the scalable feature signatures on various different features extracted from the images (e.g., SIFT or color SIFT descriptors). We would also like to utilize the scalable feature signatures for more efficient retrieval using new filter and refine schemes or metric/ptolemaic access methods.

References

1. Beecks, C., Kirchhoff, S., Seidl, T.: Signature matching distance for content-based image retrieval. In: Proc. ACM International Conference on Multimedia Retrieval (ICMR 2013), Dallas, Texas, USA, pp. 41–48. ACM, New York (2013)
2. Beecks, C., Uysal, M.S., Seidl, T.: Efficient k-nearest neighbor queries with the signature quadratic form distance. In: Proc. 4th International Workshop on Ranking in Databases (DBRank 2010) in Conjunction with IEEE 26th International Conference on Data Engineering (ICDE 2010), Long Beach, California, USA, pp. 10–15. IEEE, Washington (2010)
3. Beecks, C., Uysal, M.S., Seidl, T.: Signature quadratic form distance. In: Proc. ACM CIVR, pp. 438–445 (2010)

4. Beecks, C., Uysal, M.S., Seidl, T.: $L2$-signature quadratic form distance for efficient query processing in very large multimedia databases. In: Lee, K.-T., Tsai, W.-H., Liao, H.-Y.M., Chen, T., Hsieh, J.-W., Tseng, C.-C. (eds.) MMM 2011 Part I. LNCS, vol. 6523, pp. 381–391. Springer, Heidelberg (2011)
5. Budikova, P., Batko, M., Zezula, P.: Evaluation platform for content-based image retrieval systems. In: Gradmann, S., Borri, F., Meghini, C., Schuldt, H. (eds.) TPDL 2011. LNCS, vol. 6966, pp. 130–142. Springer, Heidelberg (2011)
6. Datta, R., Joshi, D., Li, J., Wang, J.Z.: Image retrieval: Ideas, influences, and trends of the new age. ACM Comput. Surv. 40(2), 5:1–5:60 (2008)
7. Deselaers, T., Keysers, D., Ney, H.: Features for image retrieval: an experimental comparison. Information Retrieval 11(2), 77–107 (2008)
8. Geusebroek, J.-M., Burghouts, G.J., Smeulders, A.W.M.: The Amsterdam Library of Object Images. IJCV 61(1), 103–112 (2005)
9. Hetland, M.L., Skopal, T., Lokoč, J., Beecks, C.: Ptolemaic access methods: Challenging the reign of the metric space model. Inf. Syst. 38(7), 989–1006 (2013)
10. Jegou, H., Douze, M., Schmid, C.: Hamming embedding and weak geometric consistency for large scale image search. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008, Part I. LNCS, vol. 5302, pp. 304–317. Springer, Heidelberg (2008)
11. Jegou, H., Perronnin, F., Douze, M., Sánchez, J., Perez, P., Schmid, C.: Aggregating local image descriptors into compact codes. IEEE Trans. Pattern Anal. Mach. Intell. 34(9), 1704–1716 (2012)
12. Kruliš, M., Lokoč, J., Skopal, T.: Efficient extraction of feature signatures using multi-GPU architecture. In: Li, S., El Saddik, A., Wang, M., Mei, T., Sebe, N., Yan, S., Hong, R., Gurrin, C. (eds.) MMM 2013, Part II. LNCS, vol. 7733, pp. 446–456. Springer, Heidelberg (2013)
13. Lokoč, J., Novák, D., Batko, M., Skopal, T.: Visual image search: Feature signatures or/and global descriptors. In: Navarro, G., Pestov, V. (eds.) SISAP 2012. LNCS, vol. 7404, pp. 177–191. Springer, Heidelberg (2012)
14. Mikolajczyk, K., Schmid, C.: A performance evaluation of local descriptors. IEEE Transactions on Pattern Analysis and Machine Intelligence 27(10), 1615–1630 (2005)
15. MPEG-7. Multimedia content description interfaces. Part 3: Visual. ISO/IEC 15938-3:2002 (2002)
16. Perronnin, F., Liu, Y., Sánchez, J., Poirier, H.: Large-scale image retrieval with compressed fisher vectors. In: 2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3384–3391 (2010)
17. Rubner, Y., Tomasi, C.: Perceptual Metrics for Image Database Navigation. Kluwer Academic Publishers, Norwell (2001)
18. Sivic, J., Zisserman, A.: Video google: A text retrieval approach to object matching in videos. In: Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV 2003), vol. 2, p. 1470. IEEE Computer Society, Washington, DC (2003)
19. Wu, L., Hoi, S.C.H., Yu, N.: Semantics-preserving bag-of-words models and applications. Trans. Img. Proc. 19(7), 1908–1920 (2010)

Chapter 4

On Indexing Metric Spaces Using Cut-regions

Jakub Lokoč
Juraj Moško
Přemysl Čech
Tomáš Skopal

Published in the *Information Systems* journal, volume 43, pages 1–19. Elsevier, July 2014. ISSN 0306-4379.
dx.doi.org/10.1016/j.is.2014.01.007

Impact Factor in 2014: 1.456
5-Year Impact Factor in 2014: 1.618

Information Systems 43 (2014) 1–19

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

On indexing metric spaces using cut-regions ☆

Jakub Lokoč *, Juraj Moško, Přemysl Čech, Tomáš Skopal

Charles University in Prague, Faculty of Mathematics and Physics, SIRET Research Group, Malostranské nám., 11800 Prague, Czech Republic [1]

ARTICLE INFO

Article history:
Received 10 July 2013
Received in revised form
21 January 2014
Accepted 24 January 2014
Recommended by: F. Korn
Available online 1 February 2014

Keywords:
Indexing methods
Multimedia databases
Metric access methods
PM-tree
M-Index
List of Clusters

ABSTRACT

After two decades of research, the techniques for efficient similarity search in metric spaces have combined virtually all the available tricks resulting in many structural index designs. As the representative state-of-the-art metric access methods (also called metric indexes) that vary in the usage of filtering rules and in structural designs, we could mention the M-tree, the M-Index and the List of Clusters, to name a few. In this paper, we present the concept of *cut-regions* that could heavily improve the performance of metric indexes that were originally designed to employ simple ball-regions. We show that the shape of cut-regions is far more compact than that of ball-regions, yet preserving simple and concise representation. We present three re-designed metric indexes originating from the above-mentioned ones but utilizing cut-regions instead of ball-regions. We show that cut-regions can be fully utilized in the index structure, positively affecting not only query processing but also the index construction. In the experiments we show that the re-designed metric indexes significantly outperform their original versions.

1. Introduction

Although there have been many *metric access methods* (or metric indexes) [1–4] developed in the past decades, there still emerge new metric access method (MAM) designs and other approaches addressing the problem of efficient processing of similarity queries. In the last years we observe a trend towards even more complex MAM structures represented by, e.g., the M-Index [5], the D-file [6], the pivot table (and all its variants) [7], the permutation indexes [8], and others that are often based on transformation of the metric space model into another geometric model. The "good old" indexing structures that directly partition the metric space, e.g., the M-tree [9], the (m)vp-tree, the GNAT [10,11], etc., are often outperformed by the new MAMs. From this perspective, it might seem that the MAMs relying on direct hierarchical partitioning of the metric space bring an unnecessary overhead and so they should be abandoned. However, although the partitioning-based MAMs exhibit worse performance in traditional queries, such as the range query or the k nearest neighbor query [6], for modern retrieval modalities the compact hierarchies of metric regions could perform much better. For instance, various iterative queries within the *multimedia exploration* area [12] could benefit from the native hierarchy of metric regions where a continuous traversal in the metric space is required. Another application proving the benefits of metric partitioning is demonstrated by the M-Index that efficiently combines partitioning with the iDistance [13] mapping approach. Here a compact hierarchy is crucial if the M-Index is distributed among many machines [14].

In this paper, we define a new formalism for construction of compact metric regions – the *cut-regions*. The formalism enables to simplify the adaptation of complex MAM algorithms using ball-regions to employ the cut-regions instead. In particular, we show how the formalism can be used for re-definition of the PM-tree structure and

☆ This paper is an extended version of a previous paper by Lokoč et al., [31].

* Corresponding author.
E-mail addresses: lokoc@ksi.mff.cuni.cz (J. Lokoč), mosko@ksi.mff.cuni.cz (J. Moško), Premekcech@seznam.cz (P. Čech), skopal@ksi.mff.cuni.cz (T. Skopal).

[1] http://www.siret.cz.

0306-4379/$ - see front matter © 2014 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.is.2014.01.007

its construction algorithms. Based on the cut-regions we introduce new PM-tree construction algorithm that leads to more compact PM-tree hierarchies (and so faster similarity search). Note that compact hierarchy of metric regions is not only beneficial for efficiency of traditional queries (range or k NN), but it can also better serve as a hierarchy of clusters that can be used in exploration queries, data mining, and other tasks. We also implement cut-regions to other two state-of-the-art MAMs – the M-Index and the List of Clusters.

1.1. Paper contributions

The paper contributions can be summarized into four main points:

- The new cut-region formalism that is suitable for simplified description of compact metric regions. Cut-regions can be utilized in new or existing metric indexing structures and algorithms, as demonstrated on the PM-tree.
- New cheap dynamic construction techniques for the PM-tree that can compete with expensive strategies of the original PM-tree (e.g., multi-way leaf selection).
- Adaptation of M-Index and List of Clusters to operate with cut-regions.
- Thorough experimental evaluation also including comparison with the state-of-the-art MAMs.

The rest of the paper is organized as follows. For readers not familiar with the metric search approach we provide a quick overview in Section 2. As the cut-region is the key concept used in this paper, we precisely define the cut-regions and basic operations on them in the following Section 3. Then, we redefine the PM-tree index using cut-regions in Section 4. In Section 5, we present new dynamic PM-tree construction techniques, while in Section 6 we describe other MAMs that can benefit from cut-regions. We provide thorough experimental evaluation in Section 7 and, finally, we conclude the paper and sum up its main contributions in Section 8.

2. Similarity search essentials

Having a collection of complex unstructured objects (like multimedia documents, texts, time series, 3D models, etc.), the search in such collection can hardly be based on traditional query models that assume the user is familiar with an explicit structure of the data (e.g., relational schema used by SQL). Instead, the unstructured objects have to be transformed into structured *feature descriptors* (or descriptor objects)[2] by means of a feature extraction procedure. In this step a similarity model is established, consisting of a universe $\mathcal{D}$ of feature descriptors and a function δ for measuring dissimilarity/distance of any pair of feature descriptors. Using a similarity model, the collection of descriptors can be searched using the query-by-example paradigm (e.g., return the 5 most similar images to my image of a dog).

[2] In the rest of the text the term *object* is used in the meaning of descriptor object/feature descriptor.

2.1. Similarity search

Similarity queries are an intuitive way of how to express search intent on some objects in a given domain. Usually, we want to find the k most similar objects to a given query object q or just find all objects within a distance r from the query object q. These types of queries are called the k nearest neighbor (k NN) query and the range query, respectively. Although different query types were designed and utilized for image retrieval problems (Section 1.4 in [2]), these two are the most common ones.

Definition 1 (*Range query*). Let $q \in \mathcal{D}$ be a query object and $r \in \mathbb{R}_0^+$ be a query radius (or a distance threshold). Range query is defined as $R(q,r) = \{o \in X, \delta(o,q) \leq r\}$, where δ is a distance function on domain $\mathcal{D}$ and $X \subseteq \mathcal{D}$ is a dataset to be searched.

Definition 2 (k *nearest neighbor query*). Let $q \in \mathcal{D}$ be a query object and $k \in \mathbb{N}$ be a number of requested nearest neighbors. The k nearest neighbor query is defined as $k-NN(q) = \{R \subseteq X, |R| = k \wedge \forall x \in R, y \in X - R : \delta(q,x) \leq \delta(q,y)\}$, where δ is a distance function on domain $\mathcal{D}$ and $X \subseteq \mathcal{D}$. If multiple such sets R exist, one is chosen arbitrarily.

2.2. Metric space model for similarity search

In order to search the data collections efficiently (quickly), the similarity function δ in the similarity model is often restricted to be a metric distance, hence obtaining the *metric space model* (see Definition 3). The properties of metric space enable to construct cheap lower bounds of the original (computationally expensive) similarity function which, in turn, are the basis for efficient similarity search. For more details on construction of lower bounds and on principles of metric indexing in general we refer the reader to a monograph [2] or survey [1].

Definition 3 (*Metric space*). Let $\mathcal{D}$ be a domain of feature descriptors, δ: $\mathcal{D} \times \mathcal{D} \mapsto \mathbb{R}$ a pairwise distance function on $\mathcal{D}$. Then $\mathcal{M} = (\mathcal{D}, \delta)$ is called a metric space, if the following postulates hold $\forall x, y, z \in \mathcal{D}$:

(p1) $\delta(x,x) = 0$	reflexivity
(p2) $x \neq y \Rightarrow \delta(x,y) > 0$	positiveness
(p3) $\delta(x,y) = \delta(y,x)$	symmetry
(p4) $\delta(x,z) \leq \delta(x,y) + \delta(y,z)$	triangle inequality

Despite the restriction of the similarity function to a metric distance the metric space model still remains extensible enough to fit the needs of domain experts (i.e., practitioners from various domains managing large data volumes). The metric space approach enables to utilize not only vector spaces for modeling data, but also nonvectorial descriptor types, like strings, sets, time series, etc., and appropriate nonvectorial distance

functions. As popular distance functions we could name, for example, traditional L_p metrics (vectors), edit distance (strings), Hausdorff distance (sets), Signature Quadratic Form Distance (sets) [15] or variants of Jaccard's coefficient (sets).[3] For our experiments we have chosen both, the cheap L_2 metric (Euclidean distance) and the more expensive Signature Quadratic Form Distance, applying them on different domains. For more details about specific domains (datasets), metric distances and experimental results, see Section 7.

2.2.1. Ball-region

One of the most popular means of metric space region description is a simple "ball" defined just by one object and a radius, the same way as the range query is defined. As the cut-region introduced in the following section is based on "balls" as well, we define the ball-region and thoroughly discuss its fundamental properties.

Definition 4 (*Ball-region*). Let o be an object in the domain $\mathcal{D}$ and $r_o \in \mathbb{R}_0^+$ be a distance threshold (radius). Then $B = Ball(o, r_o)$ is called a ball-region. An object $x \in \mathcal{D}$ is covered by ball-region B (denoted as $x \in B$) iff $\delta(o, x) \leq r_o$.

The ball-region is a basic metric space unit – it can be recognized in the definition of the two most popular similarity queries and it is utilized as a data partitioning technique in popular metric indexes (e.g., in M-tree, PM-tree, M-Index and List of Clusters). To create a ball-region partition P_B, the indexes use a preselected database object o as the region center, while the radius r_o is adjusted according to the objects x_i assigned to the partition P_B (i.e., $\delta(o, x_i) \leq r_o, \forall x_i \in P_B$). Since the partitions are often only subsets of the corresponding ball-regions,[4] the ball-region partitions are mainly used for efficient pruning of whole groups of irrelevant objects when searching. The following lemma guarantees the correctness of the ball-region based filtering.

Lemma 1 (*Overlap test of two ball-regions*). *Let* $B = Ball(o, r_o)$ *and* $Q = R(q, r_q)$ *be two ball-regions. If* $\delta(o, q) > r_o + r_q$ *then B and Q do not share any object.* (*Proof: Trivially implied by metric postulates.*)

For example of nonoverlapping ball-regions see Fig. 1.

2.3. Intrinsic dimensionality

Some topological properties (such as the metric postulates, see Definition 3) satisfied by a distance measure are necessary but not sufficient for design of a successful access method. Other information based on statistical analysis over a particular database is needed. The distribution of pair-wise distances between objects from the database can reveal whether there exist clusters of objects and how tight the clusters may be. The *intrinsic dimensionality* [1] can indicate efficiency limits of any access method and is defined as

$$\rho(\mathcal{S}, \delta) = \frac{\mu^2}{2\sigma^2}$$

Basically, the intrinsic dimensionality of the data space is a global characteristics related to the mean μ and variance σ computed on the set of pair-wise distances within the data space. A high intrinsic dimensionality of the data leads to poor partitioning/indexing by any access method (resulting in slower searching), and vice versa. The problem of high intrinsic dimensionality can be considered as a generalization of the well-known *curse of dimensionality* into metric spaces [16,1].

3. Cut-regions

The popular simple ball-regions, defined only by the center object and the covering radius, have one main drawback – they cannot cover tightly a cluster of similar objects in a sparse metric space (see the sparse ball-region (o,r) in Fig. 1). Moreover, for metric spaces suffering from high intrinsic dimensionality [1], the ball-regions become useless. The reason is simple – since only the center object is considered as a reference object (pivot), there is no additional information describing relations between the remaining objects in the region. In consequence, in high-dimensional spaces almost all non-empty balls overlap each other,[5] so that efficient filtering using Lemma 1 is not possible.

However, if a static set of k global pivot objects is employed, the original ball-region can be further cut off by rings (where a ring is an annulus centered in a pivot), forming thus a cut-region. In particular, the ring for a given pivot is determined by the distances from the closest and the farthest objects in the ball-region to the pivot. The definition can be further extended to support list of rings for each pivot. An example of the difference between cut-region and the ball-region is depicted in Fig. 3.

3.1. Definition of cut-region

The idea of cut-regions was first used in the PM-tree [17,18], though there it was not described as a standalone formalism but as a part of the PM-tree structure itself. Since cut-regions with their operations can be utilized also in other metric indexes, we have decided to separate this compact metric unit from the PM-tree into the following definition.

Definition 5 (*Cut-Region*). Let $(\mathcal{D}, \delta)$ be a metric space, $Ball(o, r_o)$ be a ball-region, $p_i \in \mathbb{P} \subset \mathcal{D}$ be k global pivots from an ordered pivot set $\mathbb{P}$, and hr be an ordered set of k intervals $hr_i = \langle hr_i^{min}, hr_i^{max} \rangle$, the tuple $CR(o, r_o, \mathbb{P}, hr)$ is called a *cut-region*. An object $x \in \mathcal{D}$ is covered by the cut-region (denoted as $x \in CR(o, r_o, \mathbb{P}, hr)$) iff $x \in Ball(o, r_o) \wedge \forall i_{\in (1,k)} : \delta(p_i, x) \in hr_i$.

[3] For more details see Section 1.3 in [2].

[4] Ball-regions may overlap, thus not all objects lying inside a ball-region B have to be stored within the corresponding data partition P_B.

[5] Note that in high-dimensional spaces almost all objects have the same (large) distance to each other.

If $r_o=0$ then all the intervals hr are set to $hr_i^{min}=hr_i^{max}$ and the cut-region represents only a simple point. Such cut-region is denoted as $CR(o,0,\mathbb{P},hr^o)$. For an illustration of a cut-region see Fig. 2a.

Definition 6 (*Minimal cut-region for a set X*). Let X be a subset of a database $\mathbb{S}\subset\mathcal{D}$. Then a cut-region $CR(o,r_o,\mathbb{P},hr)$ is called minimal cut-region for X (denoted as $CR(o,r_o,\mathbb{P},hr,X)$) iff $r_o=\max_{x\in X}\{\delta(o,x)\}\wedge\forall_i\in(1,k):hr_i^{min}=\min_{x\in X}\{\delta(p_i,x)\}\wedge hr_i^{max}=\max_{x\in X}\{\delta(p_i,x)\}$.

For an example of minimal cut-region for a set $X=\{o_1,o_2,o_3,o_4,o_5\}$ see Fig. 2b.

The cut-region in combination with an appropriate set of global pivots is supposed to be a core representation of a metric space region. First, the cut-region allows to determine the center o of a cluster and to control mutual proximity of the objects via the radius r_o. Second, the cut-region utilizes the rings to cut off the "empty space" of the original ball-region. In the task of metric space clustering and indexing, the cut-region is a suitable unit for a compact cluster description and representation. It is comparable to permutation-based regions, where the proximity of two objects is approximated by the similarity of their permutations [8], however, the cut-region further controls the object locality via the ball-region center o and radius r_o.

In the following paragraphs, we propose definitions and lemmas enabling the formalization of index operations employing cut-regions. For correct filtering during range or k NN query processing, we propose the following lemma.

Lemma 2 (*Overlap of cut-region and ball-region*). *Let* $CR(o,r_o,\mathbb{P},hr)$ *be a cut-region and* $Ball(q,r_q)$ *be a ball-region, if* $\delta(o,q)>r_o+r_q$ *or for some interval* hr_i *it holds that* $hr_i\cap\langle\delta(p_i,q)-r_q,\delta(p_i,q)+r_q\rangle=\varnothing$ *then the cut-region and ball-region do not share any object* (*they do not overlap*). (*Proof*: *Trivially implied by cut-region and ball-region definitions, metric postulates.*)

For an example of non-overlapping cut-region and ball-region, see Fig. 3.

Note 1. The overlap test combines ball-region test and pivot rings test with disjunction (logical OR). Therefore, in some cases, we can successfully apply this filtering rule without explicit evaluation of $\delta(o,q)$.

During indexing a database, new objects could be inserted into existing cut-regions (into data buckets associated with them, respectively). Since a new object represents a trivial cut-region, the following definition formalizing the inclusion of two cut-regions is proposed. Let us also note the inclusion test (or a weaker form of the test) is necessary for two nontrivial cut-regions ($r>0$) in hierarchically organized indexing structures.

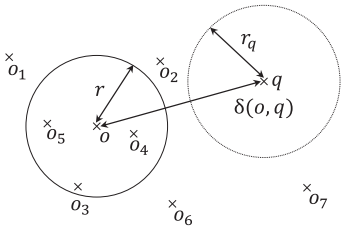


Fig. 1. Overlap test of two ball-regions.

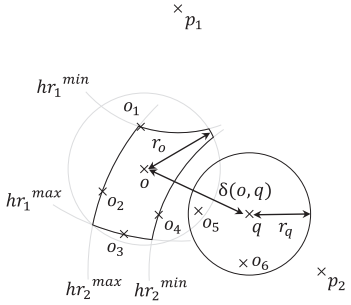


Fig. 3. Overlap of cut-region and ball-region. The balls of cut- and ball-region do overlap, but the actual regions cannot share any object because of cut-region's hr_2 boundaries.

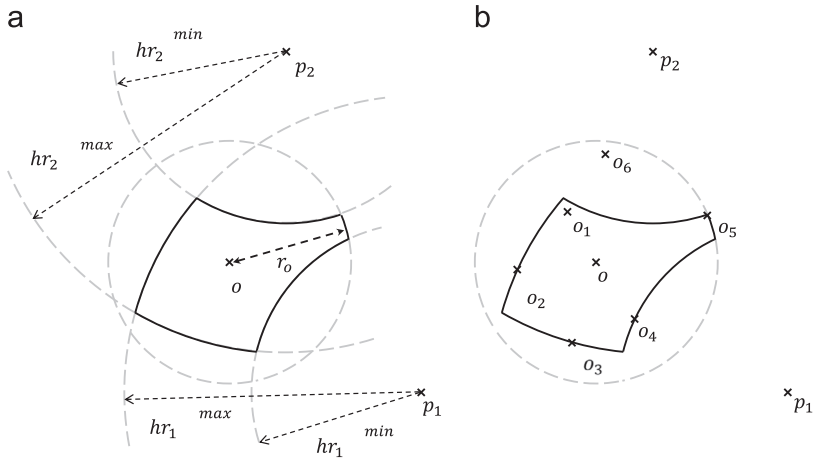


Fig. 2. (a) Cut-region and (b) minimal cut-region for a set $\mathbb{X}=\{o_1,o_2,o_3,o_4,o_5\}$.

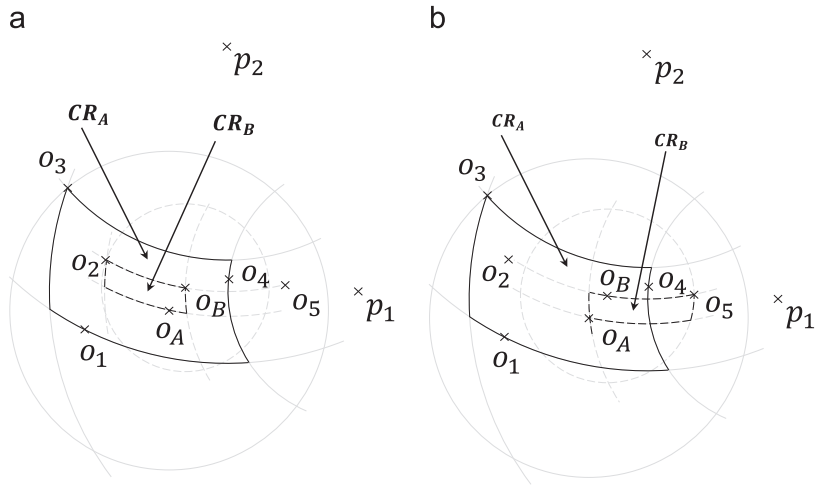


Fig. 4. In (a) $CR_B \subseteq CR_A$ but in (b) it is not. In (b), the condition with radii is fulfilled but $hr_1^B min < hr_1^A min$.

Definition 7 (*Cut-regions inclusion*). The cut-region $CR_A = CR(o_A, r_A, \mathbb{P}, hr^A)$ includes a cut-region $CR_B = CR(o_B, r_B, \mathbb{P}, hr^B)$ iff $\delta(o_A, o_B) + r_B \leq r_A \wedge \forall_{i \in (1,k)} : hr_i^B \subseteq hr_i^A$. Formally we write $CR_B \subseteq CR_A$.

The cut-region inclusion is a geometric relation between two cut-regions CR_A and CR_B (depicted in Fig. 4 and implemented in Algorithm 1).

Algorithm1. CutRegionInclusion(CR_A, CR_B, *pivotCount*)↦ *bool*.

```
1: if CR_A.r_o < δ(CR_A.o, CR_B.o)+CR_B.r_o then
2:    return false//CR_B is not inside CR_A
3: end if

4: for (i = 0; i < pivotCount; i++) do
5:    if CR_A.hr^min[i] > CR_B.hr^min[i] or CR_A.hr^max[i] < CR_B.hr^max[i] then
6:       return false//CR_B is not inside CR_A
7:    end if
8: end for

9: return true    //CR_B is inside CR_A
```

3.2. Operations on cut-regions

In this section we define two operations, the cut-region extension and the cut-region reduction, that are supposed to be frequently employed operations by dynamic indexing techniques.

At some point of indexing it is impossible to fit the object into an existing cut-region. Then some suitable cut-region has to be selected and extended by the new inserted object (again treated as the cut-region) or even by a new inserted cut-region in the case of a bulk-loading operation. For such reasons, we define the cut-region extension as follows.

Definition 8 (*Cut-region extension*). Let $CR_A = CR(o_A, r_A, \mathbb{P}, hr^A)$ and $CR_B = CR(o_B, r_B, \mathbb{P}, hr^B)$ be cut-regions. Then $CR_E = CR(o_A, r_E, \mathbb{P}, hr^E)$, where $r_E = \max\{\delta(o_A, o_B) + r_B, r_A\} \wedge \forall_{i \in (1,k)} : hr_i^E = \langle \min\{hr_i^A min, hr_i^B min\}, \max\{hr_i^A max, hr_i^B max\}\rangle$ represents the extension of the cut-region CR_A by the cut-region CR_B. Formally, we write $CR_E = CR_A \oplus CR_B$.

In Fig. 5 see an example of cut-region extension.

Note 2. The cut-region extension $\oplus$ is not a commutative operation, as the ball-region of CR_E is centered in o_A. For indexing purposes we expect that CR_A represents an index node and CR_B is absorbed by CR_A.

The implementation of cut-region extension is described in Algorithm 2.

Algorithm 2. CutRegionExtension(CR_A, CR_B, *pivotCount*).

```
1: if CR_A.r_o < δ(CR_A.o, CR_B.o)+CR_B.r_o then
2:    CR_A.r_o = δ(CR_A.o, CR_B.o)+CR_B.r_o
3: end if

4: for (i=0; i < pivotCount; i++) do
5:    if CR_A.hr^min[i] > CR_B.hr^min[i] then
6:       CR_A.hr^min[i] = CR_B.hr^min[i]
7:    end if
8:    if CR_A.hr^max[i] < CR_B.hr^max[i] then
9:       CR_A.hr^max[i] = CR_B.hr^max[i]
10:   end if
11: end for
```

The dynamic rearrangements of the objects within an index can significantly improve the index performance. For such reasons, some objects from a set X associated with a cut-region can be removed and reinserted elsewhere. Thus, we discuss also the cut-region reduction operation. The reduction operation creates constructively the reduced cut-region from the scratch.

Definition 9 (*Cut-region reduction*). Let $CR(o, r_o, \mathbb{P}, hr, X)$ be a minimal cut-region for a set X and $Y \subset X$ be set of objects to be excluded from X. Then $CR(o, r_o, \mathbb{P}, hr, X - Y)$ is called cut-region reduced by Y.

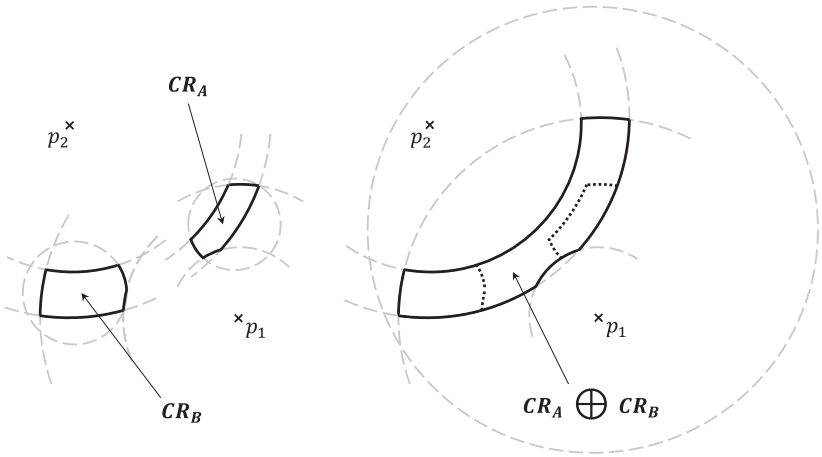


Fig. 5. Cut-region extension of CR_A by CR_A (i.e., $CR_A \oplus CR_B$).

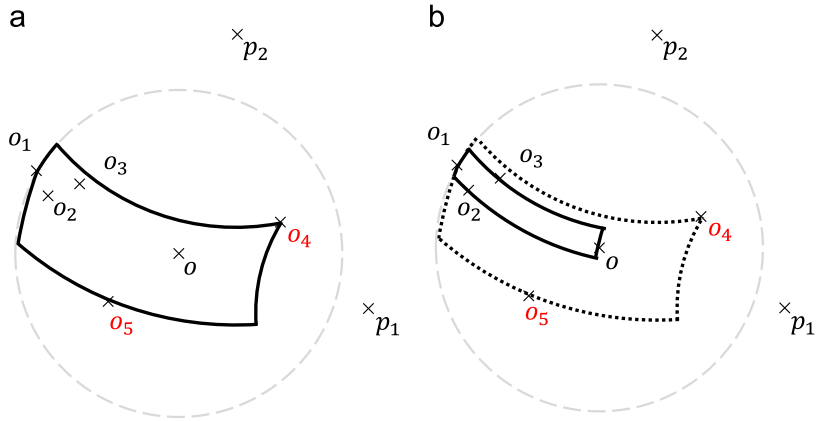


Fig. 6. (a) Minimal cut-region for $\{o, o_1, o_2, o_3, o_4, o_5\}$ and (b) cut-region reduced by $\{o_4, o_5\}$.

See an example of cut-region reduction in Fig. 6. In the following definitions we describe "helper vectors" that allow to quantify the differences between cut-regions.

Definition 10 (*Change vector for cut-region extension*). Let $CR_A = CR(o_A, r_A, \mathbb{P}, hr^A)$ and $CR_B = CR(o_B, r_B, \mathbb{P}, hr^B)$ be cut-regions. Then $k+1$ dimensional change vector cv_e for extension $CR_A \oplus CR_B$ is defined as $cv_e = \langle \max(\delta(o_A, o_B) + r_B - r_A, 0),\ \max(hr_1^{A\ min} - hr_1^{B\ min}, 0) + \max(hr_1^{B\ max} - hr_1^{A\ max}, 0), \ldots, \max(hr_k^{A\ min} - hr_k^{B\ min}, 0) + \max(hr_k^{B\ max} - hr_k^{A\ max}, 0)\rangle$.

Definition 11 (*Change vector for cut-region reduction*). Let $CR(o, r_o, \mathbb{P}, hr, X)$ be minimal cut-region for a set X, $Y \subset X$ be set of objects to be excluded from X. Let CR_R be $CR(o, r_o, \mathbb{P}, hr, X)$ reduced by Y. Then $k+1$ dimensional change vector cv_r for cut-region CR reduction is defined as change vector cv_e of the cut-region extension $CR_R \oplus CR$.

The change vectors provide a basic quantitative information about the transformation cost of a cut-region. The first dimension refers to the change in the radius of the ball-region, while the other k dimensions refer to the change in intervals defining individual rings. Depending on whether the change is extension (change from smaller

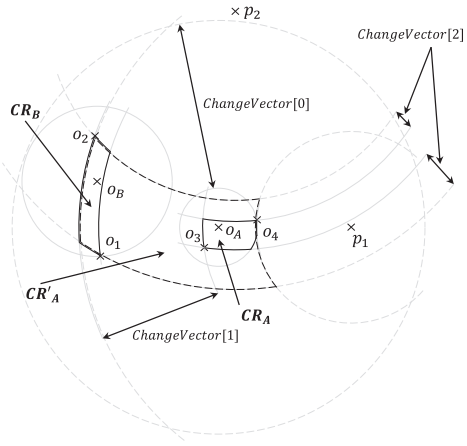


Fig. 7. Change vector of the cut-region extension $CR'_A = CR_A \oplus CR_B$.

cut-region to larger) or reduction (change from larger cut-region to a smaller), the respective change vector is applied. As an example of change vector for cut-region extension see Fig. 7.

The change vector stores information useful for indexing algorithms that need to select cut-regions based on some criteria. The criteria are often based on an aggregation of the change vector components. In Algorithm 3 we demonstrate how to determine and aggregate (see line 12) the change vector of the cut-region extension (to obtain a numeric stat, e.g., the cut-region growth). The result can serve as a criterion in node selection strategies, node split functions using cut-regions, reinsertion strategies employing cut-regions, etc.

Algorithm 3. GrowthOfCutR egionExtension($\boldsymbol{CR_A}$, $\boldsymbol{CR_B}$, ***pivotCount***, ***bestResultSoFar***, ***agr***)↦*Result*.

```
1: Let CV be a (pivotCount+1) dimensional vector of zeros // CV is a ChangeVector
2: if CR_A.r_o < δ(CR_A.o, CR_B.o)+CR_B.r_o then
3:    CV[0] = δ(CR_A.o, CR_B.o)+CR_B.r_o − CR_A.r_o
4: end if

5: for (i=0; i<pivotCount; i++) do
6:    if CR_A.hr^min[i] > CR_B.hr^min[i] then
7:       CV[i+1] += CR_A.hr^min[i] − CR_B.hr^min[i]
8:    end if
9:    if CR_A.hr^max[i] < CR_B.hr^max[i] then
10:      CV[i+1] += CR_B.hr^max[i] − CR_A.hr^max[i]
11:   end if

12:   Result = aggr(CV) // aggregate values of CV
13:   if Result > bestResultSoFar then
14:      break
15:   end if
16: end for

17: return Result
```

4. Revisiting PM-tree

The PM-tree [17,18] is a metric index that conceptually merges the pivot table [7] with the M-tree [9]. More specifically, the PM-tree enhances the original M-tree hierarchy by the information related to a static set of k global pivots $p_i \in \mathbb{P} \subset \mathcal{D}$. Thus, the ground entries (representing data) contain also distances to the global pivots, while the original ball-region (inherited from M-tree) is further cut off by a set of rings (centered in the global pivots), so the region "volume" becomes more compact. If we look carefully at the definition of the PM-tree routing entry [17,18], we observe that the PM-tree has already introduced the cut-regions, however, not as a formalism for independent regions in metric space but as a part of the PM-tree data structure itself.

In the following subsections we quickly review the basic PM-tree principles and slightly redefine the structure of the PM-tree routing and ground entries to comply with the cut-region notation used in this paper. In the original papers the PM-tree substructure representing the minimal cut-region for a set X is defined as

$$CR(o) = [o, r_o, hr^{min}, hr^{max}],$$

where o is the center object of the ball-region, r_o is the radius of ball-region constructed as maximal distance $\delta(o, x_i), \forall x_i \in X$, and hr^{min}/hr^{max} are k-dimensional arrays of min/max distances from x_i to k global pivots. The set X is supposed the contain objects of a PM-tree node (inner node or leaf).

4.1. PM-tree structure

The PM-tree consists of inner nodes with routing entries and leaf nodes with ground entries. A routing entry in a PM-tree inner node is defined as

$$rout_{PM}(o) = [CR(o), \delta(o, \mathrm{Par}(o)), ptr(T(o))],$$

where $CR(o)$ is minimal cut-region for objects stored in the child node, $\delta(o, \mathrm{Par}(o))$ is the distance from o to the parent routing object, and $ptr(T(o))$ is a pointer to the child node. A ground entry in a PM-tree leaf is defined as

$$grnd_{PM}(o) = [CR(o), id(o), \delta(o, \mathrm{Par}(o))],$$

where $CR(o)$ is a point cut-region,[6] $\delta(o, \mathrm{Par}(o))$ is the distance from o to the parent routing object, and $id(o)$ is a unique identifier of object o. For a fragment of PM-tree hierarchy, see Fig. 8.

The combination of all the k entries' ranges (stored in the cut-regions) produces a k-dimensional minimum bounding rectangle (MBR), and hence the global pivots actually map the metric regions/data into a pivot space of dimensionality k. The number of pivots can be defined separately for routing and ground entries—we typically choose fewer pivots for ground entries to reduce storage cost (i.e., $k = k_{innerNode} > k_{leafNode}$). The pivot space mapping abstraction is much like that one used in pivot tables [7]; however, in the PM-tree case the pivot space also includes the hierarchy of MBRs (similar to R-tree).

4.2. PM-tree querying and construction

When issuing a range or k NN query, the query object is mapped into the global pivot space—this requires p extra distance computations $\delta(q, p_i), \forall p_i \in \mathbb{P}$. The query processing starts in the root node and checks all routing entries' cut-regions for overlap with the query ball applying Lemma 2. If a cut-region cannot be filtered, the child node of the corresponding routing entry is visited. If a leaf node is reached, all stored cut-regions are checked using Lemma 2 and all non-filtered objects are included in the (candidate) result set. Besides Lemma 2, also the parent filtering rule [9] can be utilized to improve the filtering power of the PM-tree. Note that applying Lemma 2 does not require many explicit distance computations, so the PM-tree usually achieves significantly lower query cost when compared with the M-tree [17–20].

The PM-tree construction algorithm is a simple extension of the original M-tree algorithm. The only difference is the maintenance of the cut-region's hr^{min}/hr^{max} arrays.

[6] Minimal cut-region for set $X = \{o\}$.

5. Improving the PM-tree construction

In this section, we apply the basic cut-region operations to the new indexing techniques in PM-tree. This comprises the cut-region inclusion, cut-region extension, reduction and how to determine the change vector of the cut-region extension (all defined in Section 3.2). Then we consecutively introduce our new dynamic PM-tree construction techniques that use the cut-regions (leaf selection strategies, node splitting and forced reinsertions). Let us also emphasize we consider the newly inserted object as a point cut-region. The pivot count is an input parameter in most of the algorithms, but it is supposed a static value for the whole lifetime of the structure and cannot be dynamically changed.

5.1. Leaf selection strategies using cut-regions

The PM-tree [18] is generally built in the bottom-up manner, so a suitable leaf selection strategy has a crucial impact on the quality of the resulting hierarchy as has been shown for its predecessor M-tree in [21,22]. The original PM-tree construction technique is very simple – as in the original M-tree construction technique, the PM-tree employs only the ball-regions determined by its routing entries. In fact, the PM-tree construction algorithms ignore the rings used to cut empty space in the covering ball-regions, thus a newly inserted object can drastically increase the volume of the corresponding cut-regions (see SW_BR arrow in Fig. 9). Therefore, our new leaf selection strategy considers also the rings delimiting the borders of the cut-regions (see SW_CR arrow in Fig. 9). In other aspects, the new leaf selection strategy follows the necessary rules that preserve the original PM-tree invariants [18].

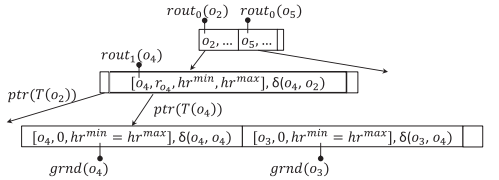


Fig. 8. Structure of PM-tree. The entry $[o_i, r_{o_i}, hr^{min}, hr^{max}]$ refers to $CR(o_i)$.

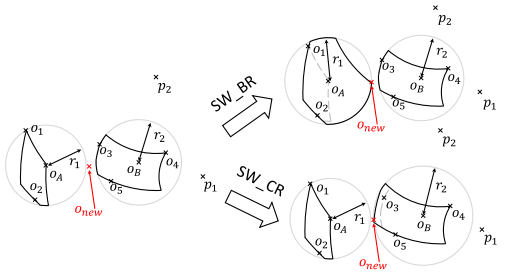


Fig. 9. Extension of a cut-region during single-way leaf selection strategy – original approach using ball-regions (SW_BR arrow) and new method using cut-regions (SW_CR arrow).

More specifically, the new technique utilizing the cut-regions changes the notion of the "good" candidate node for all variants of the leaf selection strategies (single/multi/hybrid-way). For the single-way leaf selection strategy (Fig. 10a), such node becomes the best candidate, the parent routing cut-region of which covers the newly inserted object (wrapped in the cut-region) and its routing object is as close to the new object as possible. All covering child nodes of the best candidate node are then followed down to the next PM-tree level, while only the best one is selected as the candidate node. After the pre-leaf level is reached, the candidate pre-leaves are checked for the best routing entry and the respective leaf is returned as the finally selected leaf. In the situation when no candidate node can be selected at a level (i.e., the new object is not covered by any of the node's cut-regions), the technique selects such node that guarantees the minimal extension of its cut-region (for more details see Algorithm 4). For the multi-way leaf selection strategy (Fig. 10b), more nodes can become good candidates (cut-regions of their parent routing entries cover the newly inserted object) and all covering child nodes of the candidate nodes are then followed down to the next PM-tree level. The multi-way leaf selection leads to the optimal leaf node, however, for much higher construction cost. For more details about the multi/hybrid-way leaf node selection techniques see [22].

Algorithm 4. SINGLEWAYFORCUTREGIONS(***PMTreeInner Node***, ***NewPointCR***, ***pivotCount***)↦*Leaf*.

```
1:  candidate = null

    //first, we try to find node containing new CR
2:  for each routingEntry in PMTreeInnerNode do
3:    if
    CUTREGIONINCLUSION(routingEntry.CR, NewPointCR, pivotCount)
      and
      δ(routingEntry.CR.o, NewPointCR.o) is minimal then
4:      candidate = routingEntry
5:    end if
6:  end for

6:  //if we haven't found perfect candidate yet
7:  if candidate is null then
8:    bestResultSoFar = MAX_VALUE
9:    for each routingEntry in PMTreeInnerNode do
```

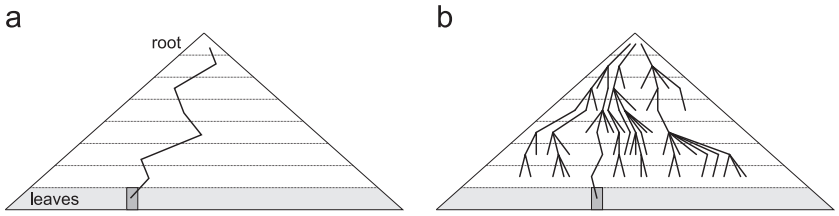


Fig. 10. (a) Single-way and (b) multi-way leaf selection strategies.

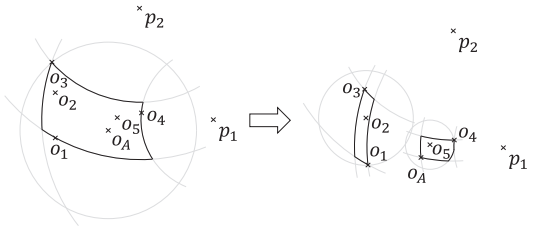


Fig. 11. Splitting overfull node using mCutReg strategy. In this case node capacity is 5 objects and minimal node utilization is 2 objects. New centers after split are o_2 and o_5.

```
10:      result = GrowthOfCutRegionExtension
         (routingEntry.CR, NewPointCR,
         pivotCount, bestResultSoFar, SUM)
11:      if result < bestResultSoFar then
12:        candidate = routingEntry
13:        bestResultSoFar = result
14:      end if
15:    end for

16:  end if
17:  CutRegionExtension(candidate.CR, NewPointCR, pivotCount)

18:  if candidate.Ptr(T(o)) is leaf then
19:    return candidate.Ptr(T(o))
20:  end if

21:  return SingleWayForCut Regions (candidate.Ptr(T(o)), NewPointCR,
     pivotCount)
```

5.2. Node splitting using cut-regions

The new node split algorithm is based on the original (P)M-tree splitting – the distance matrix is evaluated, two new routing entries are selected from the overfull node and all the remaining entries are distributed between them. Similarly as in the leaf selection strategy, the original (P)M-tree split algorithm considering only ball-regions unnecessarily "inflates" (extends, in our notation) the newly created cut-regions in the PM-tree. Hence, our new split algorithm focuses on the "volume" of candidate cut-regions. However, to create the candidate cut-regions and check their properties is an expensive task. Therefore, instead of all possible pairs of candidates only a fraction is checked.

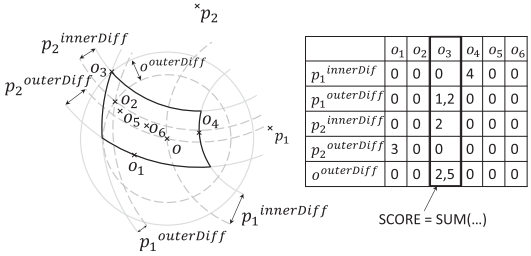
When assigning an entry e_k from the overfull node to a candidate routing entry cr_i, a growth score function $GS(cr_i, e_k) = \text{GrowthOfCutRegionExtension}(cr_i, e_k, pivotCount, MAX\_Value)$, *SUM* is used. Checking one candidate pair consists of two steps, both employing the CutRegionExtension(…) operation. First, until the minimal node utilization is reached, the entries e_k with minimal $GS(cr_i, e_k)$ and $GS(cr_j, e_k)$ are alternatively distributed between the two candidates cr_i and cr_j (cut-region extension operation is performed after each assignment). When the minimal node utilization is reached, the entry e_k is assigned to cr_i if $GS(cr_i, e_k) < GS(cr_j, e_k)$, otherwise the entry e_k is assigned to cr_j.

Finally, we select such candidate pair the greater cut-region of which is minimal among all candidate pairs (analogy to *mMaxRad* heuristic from (P)M-tree). The size of cut-region is measured as a sum of all ring widths. We denote the split heuristics as the *mCutReg* (see also example in Fig. 11).

5.3. Reinsertion using cut-regions

When redesigning forced reinsertions for cut-regions, the most important question is – which objects are optimal for forced reinserting from an overfull node? Instead of considering only the most distant objects from the parent routing object, the hr^{min} and hr^{max} distances to global pivots also have to be taken into account. Trivially, for k reinserted objects, all possible k-tuples of candidates can be checked (i.e., removed from the cut-region while the resulting cut-region is scored). To reduce an indisputable overhead of the trivial solution, we propose a simple heuristics selecting sub-optimal candidate tuple of objects for reinserting.

Let $p_i^{innerDiff}$ ($p_i^{outerDiff}$) be the absolute value of difference of two closest (farthest) objects from pivot p_i (see Fig. 12). Then, each object o_j closest (farthest) from pivot p_i can be assigned a value $p_i^{innerDiff}$ ($p_i^{outerDiff}$); see the table in Fig. 12. Such value can be utilized as a criterion for reinserting – the value determines how much the cut-region is reduced (according to the pivot p_i). Moreover, if an object o_j has assigned more $p_i^{innerDiff}$ ($p_i^{outerDiff}$) values, their sum determines the overall reduction of the cut-region. Hence, we use this sum as a score function when determining a k-tuple of objects that should be reinserted – we reinsert k objects with the highest score. When determining the final score from the set of partial scores, we also consider $o^{outerDiff}$ that represents the reduction of the cut-region radius r_o (o is the ball-region center), see



	o_1	o_2	o_3	o_4	o_5	o_6
$p_1^{innerDif}$	0	0	0	4	0	0
$p_1^{outerDiff}$	0	0	1,2	0	0	0
$p_2^{innerDiff}$	0	0	2	0	0	0
$p_2^{outerDiff}$	3	0	0	0	0	0
$o^{outerDiff}$	0	0	2,5	0	0	0

Fig. 12. Forced reinsertion of objects from a cut-region.

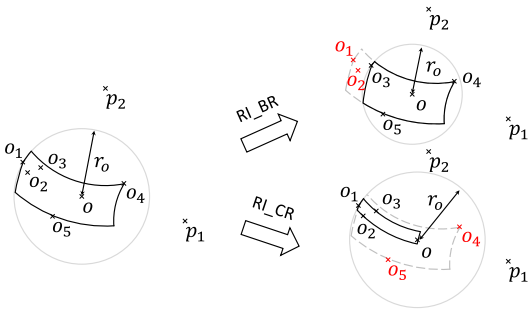


Fig. 13. Two objects from an overfull PM-tree node are reinserted according to original (RI_BR) and new (RI_CR) reinsertion strategy.

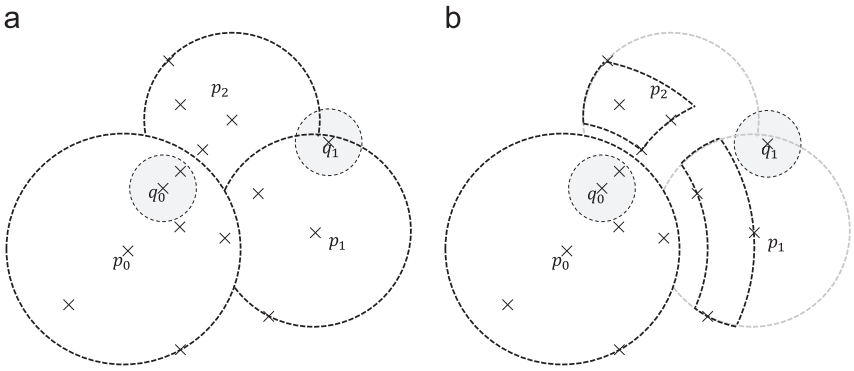


Fig. 14. (a) List of Clusters and (b) List of Cut-Regions.

the last row in the table of Fig. 12. The subsequent reinsertion processing is the same as used in [21,22] for forced reinsertions in the M-tree. In Fig. 13 see a comparison of original reinserting strategy driven just by ball-regions (see `RI_BR` arrow) and new strategy driven by cut-regions (see `RI_CR` arrow). The new strategy leads to a more compact cut-region. In both cases the operation of cut-region reduction is utilized.

6. Revisiting MAMs employing ball-regions

In this section, the cut-regions are used to improve two other popular MAMs utilizing ball-regions within their structure – the M-Index [5] and the List of Clusters (LoC) [23]. First, we describe the extension of the List of Clusters to the List of Cut-Regions where cut-regions affect the efficiency of both indexing and querying. Second, we describe very straightforward extension of the M-Index, where cut-regions reduce the internal cost of query processing. In the following paragraphs, we shortly explain the main concepts of the extended MAMs and put stress mainly on the extensible parts of the indexes. To not confuse the reader, we also remember that a cut-region in the simple form of point can be used to represent a single object mapped into the pivot space (pivot table entry), which is again useful for formal and uniform description of the structure and the algorithms.

6.1. List of Cut-Regions

In this section, we start with summary of the List of Clusters and then we describe our new metric access method – the List of Cut-Regions.

6.1.1. List of Clusters

The List of Clusters was designed as a simple memory-based MAM consisting of the ordered list of entries (representing metric ball-regions) each containing list of data objects. The entry in the List of Clusters is defined as

$$entry_{LoC}(o) = [o, r_o, L],$$

where o is the center of the metric ball, r_o is the radius $\delta(o, x_i) < r_o, \forall x_i \in L$, where L is the list containing objects x_i. The main feature of the List of Clusters is that objects from a possible intersection of two ball-regions represented by two entries are deterministically stored in just one entry. This property is achieved by a simple dynamic indexing technique, where the algorithm strictly respects the ordering of the already created entries in the list. More specifically, insertion of the new object x starts with the first entry $entry_{LoC}(o)$ and only if $\delta(o, x) > r_o$, the second region in the list is checked for placement and so on. If no ball-region in the list spatially covers the new object, a new entry is created and inserted to the end of the list. This simple technique guarantees the deterministic location of the objects belonging to possible "intersections" of all ball-regions in the list. LoC can be constructed also in a static way using range or k NN queries for creation of the entries. For details of center selection strategies and how to set their radii, see [23]. Let us also note more ball-regions result in higher indexing cost, while for the query performance there exists an optimal number of clusters.

The List of Clusters structure also determines the query processing algorithm that can benefit from the cluster ordering by a new stop condition – if the whole query ball lies inside an actually processed cluster in the list, the other clusters do not have to be processed. The stop condition is demonstrated in Fig. 14a, where after checking cluster p_0 the query q_0 processing can be stopped although query q_0 intersects also with the ball of cluster p_2. Besides the stop condition, the range and k NN query processing algorithms (recently improved in [24] for the price of higher overhead of the heap operations) employ the traditional ball-filtering rule (see Lemma 1).

6.1.2. List of Cut-Regions structure

The proposed List of Cut-Regions (LoCR) is a slight modification of LoC where the ball-regions are replaced by the cut-regions. The i-th entry in the List of Cut-Regions is defined as

$$entry^i_{LoCR}(o) = [CR(o, r_o, \mathbb{P}_i, hr, L), L],$$

where $CR(\ldots)$ is the minimal cut-region for a set L that is the set of objects stored in a data bucket associated with the entry. In addition to this straightforward extension of LoC, the List of Cut-Regions also assumes LoC-specific schema for the selection of pivots $p_j \in \mathbb{P}_i$. Instead of external selection of pivots, they are determined by the list itself. Because an i-th entry in LoC can be only accessed after the preceding $i-1$ clusters were accessed, the

respective $i-1$ cluster centers could be used as the pivots for the i-th Cut-Region. Hence, the LoCR entries do not share the same set of pivots, but $\mathbb{P}_1 \subset \mathbb{P}_2 \subset \cdots \subset \mathbb{P}$. To reduce the number of pivots in $\mathbb{P}_i$ (and thus reduce the time complexity of the cut-region operations), we can limit the maximal number of pivots for entries with large i. For an example of the List of Cut-Regions see Fig. 14b. Similarly as for point cut-regions in PM-tree ground entries, the precomputed distances between data objects and their pivots can be stored in List of Cut-Regions as well.

The proposed extension of the LoC structure is straightforward, however, what makes the extension elegant when using cut-regions is the implicit selection and usage of pivots in the indexing and querying algorithms described in the following two subsections.

6.1.3. List of Cut-Regions indexing

The dynamic indexing of List of Cut-Regions is similar to the dynamic indexing of List of Clusters, where the indexing respects the ordering of clusters, i.e., the entries from the list are always sequentially processed in the predefined order. If no entry contains new object within its ball-region, a new entry is created and appended to the end of the list. Hence, if the new object is inserted into the i-th entry, $i-1$ distances to centers of previous (unsuccessfully filtered) entries are known that can serve as $i-1$ pivots. Thus, i-th cut-region in the list can dynamically maintain $i-1$ rings centered in the centers of previously processed entries (for no additional cost).

Algorithm 5. CreateListOfCR(***k***, ***minUtil***, λ, ***maxPivot Count***, ***Data***)$\mapsto$*List of Cut−Regions*.

```
1: Let P and LoCR be two empty lists
2: pivotCount=0

3: while Data.Count() > 0 do
4:    CR=CreateNewCluster (k, minUtil, λ, pivotCount, Data, P)
5:    LoCR.Add(CR)
6:    pivotCount = MIN(LoCR.Count(), maxPivotCount)
7: end while

8: return LoCR
```

In the case a dataset is known beforehand, a bulk loading algorithm can be designed to create a more compact List of Cut-Regions. We propose a bulk-loading algorithm employing k NN queries in Algorithm 5. The key method used in this algorithm is CreateNewCluster(), see Algorithm 6, that selects a new cluster center among objects not yet indexed (see details in [23]), performs k NN(*newCenter*) query and selects a subset from the query result to form the new cut-region. Within the algorithm, we propose two heuristics for object selection:

- The *SimpleQuery* heuristics just creates new cut-region from all objects in the k NN query result.
- The *CumulativeQuery* heuristics processes objects according to their distances to the *newCenter*, where first minimal utilization of the cut-region is achieved. Then the heuristics selects consecutively such objects from the ordering so that the extension of cut-region is limited by a threshold.[7] If some object extends the actual cut-region beyond the predefined threshold, the creation of cut-region is finished and all objects in the new cut-region are excluded from the set of not indexed objects.

The purpose of the *CumulativeQuery* heuristics is to create more compact regions while, on the other hand, the *SimpleQuery* heuristics creates smaller number of cut-regions.

Algorithm 6. CreateNewCluster(***k***, ***minUtil***, λ, ***pivot Count***, ***Data***, ***P***)$\mapsto$*Cut−Region*.

```
1:  newCenter=SelectOptimalCenter (Data, P)
2:  CR=new CutRegion(newCenter, pivotCount)
3:  Data=Data.Remove(newCenter)
4:  P.Add(newCenter)

    //additive creation of the new cluster
5:  count=growth=growthSum=0
6:  candidates=κ-NN(newCenter from Data
7:  while candidates.Count() > 0 do
8:    obj=the closest object in candidates
9:    candidates=candidates - obj
10:   CR_obj=new CutRegion (obj, pivotCount)

11:   if heuristic ≠ SimpleQuery then
12:     growth=GrowthOfCutRegionExtension(CR, CR_obj, pivotCount,
      MAX_VALUE, SUM)

        //stop condition for object deteriorating
        //Cut-Region more than average case
13:     if count ≥ minUtil and count > 0 and
        growthSum/count < growth*λ and
        heuristic=CumulativeQuery then
14:       return CR
15:     end if
16:   end if

      //insert new object into the cluster
17:   CutRegionExtension(CR, CR_obj, pivotCount)
18:   Data.Remove(obj)
19:   count = count + 1
20:   growthSu = growthSum+growth
21: end while

22: return CR
```

6.1.4. List of Cut-Regions query processing

As for indexing, also the query processing respects the ordering of the clusters, furthermore, the List of Cut-Regions uses more compact cluster representations. In general, when processing a query in the i-th cluster, distances from the query object to the previously processed $i-1$ centers of the clusters are available, and so Lemma 2 could be applied for query and the cut-region overlap check. Let us remember, not all $i-1$ previously processed centers are considered as pivots; only first *maxPivotCount* centers are employed as pivots (to avoid very large pivot sets $\mathbb{P}_i$).

The cut-regions enable filtering without explicit evaluation of $\delta(q, CR_i.o)$ using the filter-and-refine schema. In the filter step the distances $\delta(q, CR_j.o)$ are computed ($j \leq maxPivotCount$) and such LoCR entries are filtered out the cut-regions of which do not overlap with the query. In

[7] The average size of the extension obtained so far can be utilized.

the refine step each remaining non-filtered i-th LoCR entry is checked by computing $\delta(q, CR_i.o)$ (if not already computed, i.e., just for $i > maxPivotCount$).

6.2. M-Index revisited

The M-Index [5] is a dynamic and persistent MAM combining almost all known metric filtering principles. The principles can be divided into two orthogonal approaches – metric space partitioning using a metric cluster tree and the locality preserving mapping (based on iDistance) of the database objects into keys in one-dimensional real domain, so the data objects can be stored in the B^+-tree index. Using these two techniques, the data objects/partitions that cannot be pruned by the cluster tree can be at least quickly located and accessed in small blocks using interval queries to the B^+-tree. Both the locality-preserving mapping and the cluster tree utilize only precomputed distances between database objects and a static set of global pivots, so the indexing cost is just linear according to the size of the database. Moreover, in the case of an expensive distance function, the precomputed distances can be attached to the database objects stored in the B^+-tree to reduce the number of explicit distance computations during query processing. From this point of view, the M-Index can be considered as an I/O-cost optimized Pivot Table implementation. Since the Pivot Table does not use any kind of the partitioning, we cannot utilize the cut-regions (except as a synonym for the Pivot Table entry) in this part of the M-Index and we have to focus only on the cluster tree.

The cluster tree utilizes the repetitive Voronoi partitioning that organizes data objects into groups according to their relation to the set of global pivots. This relation is formalized using a distance permutation [8] representing the ordering of the global pivots to the database object. The grouping is then defined simply – two objects belong to the same group if their corresponding distance permutations (or their prefixes) are equal. Objects in each group are further wrapped by the ring (two ball-regions) centered in the corresponding closest global pivot, where the min/max radii of the ring are updated during indexing. Having such data structure, the filtering based on the hyperplane and ball partitioning can be utilized during the query processing (see Fig. 15a). For such parts of the cluster tree that cannot be safely pruned, the query is transformed to a series of interval queries into B^+-tree and, finally, the returned B^+-tree leaf nodes are sequentially searched.

6.2.1. Cut-region extension of M-Index

To extend the M-Index, we just replace the ball-regions in the cluster tree by the cut-regions, so the filtering power of the cluster tree can be increased and thus the number of B^+-tree queries can be reduced. Furthermore, we can define cut-regions also for lower levels of the cluster tree, where the centers and radii of the cut-regions are the same as centers and radii at higher levels, but the cut-regions rings are more tight. The reason is that the clusters contain less objects at the lower levels of the cluster tree and thus the rings can better approximate the actual "shape" of the sub-regions (see Fig. 15b). Since all information for implementation of cut-regions is available (the distances to all global pivots are evaluated for each inserted object), the extension consists only of the maintenance of the cut-region rings in the cluster tree during indexing and query processing (using Lemma 2).

7. Experimental evaluation

In this section, we extensively test the cut-region-enhanced MAMs (PM-tree, M-Index and List of Cut-Regions) and their original variants to examine the expected performance gain in indexing and querying. As usual in papers addressing metric searching, we mainly focus on the number of distance computations spent by indexing and querying because for expensive distances the number of distance computations strongly correlates with the real time and at the same time the number of distance computations is a hardware and platform independent performance measure. However, the number of distance computations is not decisive enough in cases the utilized distance measure is cheap (e.g., L_p distances have just linear time complexity) and thus for cheap distances we prefer to present both real time and the number of distance computations.

7.1. The testbed

In order to examine cut-regions in very different conditions, we have mainly used four different datasets

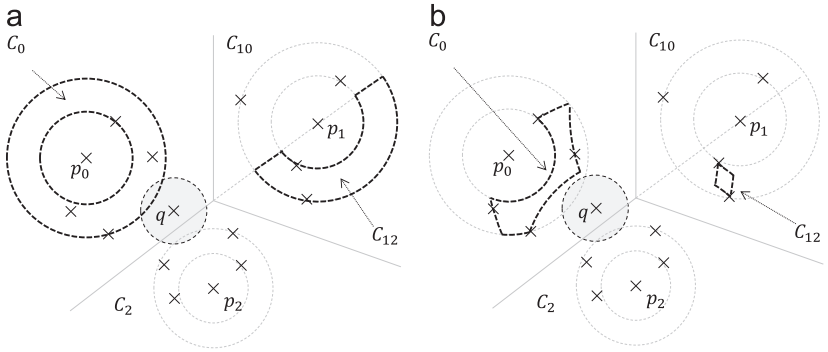


Fig. 15. M-Index cluster tree without (a) and with cut-regions (b).

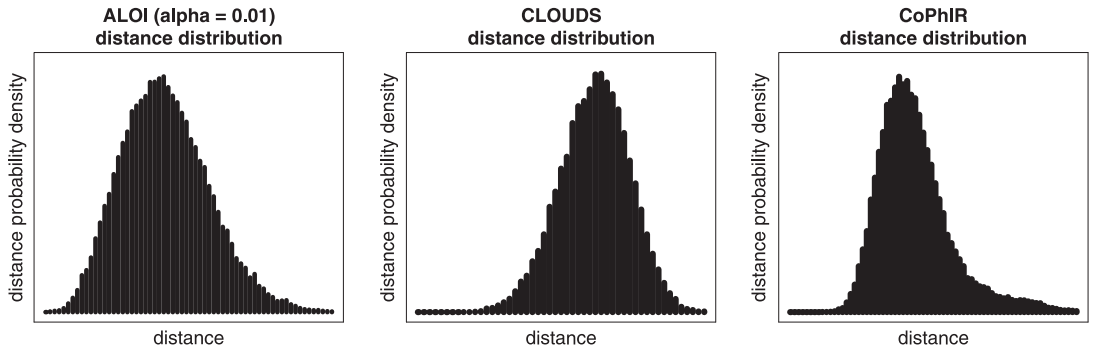


Fig. 16. Distance distribution histograms for all used datasets.

(two real-world and two synthetic) and three different distance measures. The respective distance distribution histograms indicating the intrinsic dimensionality (see Section 2.3) are depicted in Fig. 16. Specifically, we have utilized:

- The *ALOI* database [25] comprising 72,000 images extracted by the tool and settings presented in [26]. To compare two feature signatures from the ALOI dataset, the *Signature Quadratic Form Distance* (SQFD) using *Gaussian Similarity Function* was employed where alpha was fixed to 0.01. The usage of alpha = 0.01 results in metric spaces with low intrinsic dimension (≈ 4.3) but slightly less precise search [27].
- A part of the *CoPhIR* [28] database comprising 250,000 76-dimensional feature vectors (12-dimensional color layout and 64-dimensional color structure MPEG7 descriptors) extracted from selected images from Flickr.com. The *CoPhIR* dataset is vectorial and the Euclidean distance was employed to compare two object descriptors. The intrinsic dimensionality of the metric space is 6.1.
- A synthetic *CLOUDS* database comprising 100,000 clouds (sets of points), each containing up to 60 6-dimensional points from the unit hypercube. For each cloud, its center was generated at random, while the 59 remaining points were generated under normal distribution around the center. To compare two clouds of points from the CLOUDS dataset the Hausdorff distance [29] employing Euclidean distance as the internal point-to-point distance was used. The intrinsic dimensionality of the metric space is 13.6.
- In order to model a growing intrinsic dimensionality of a distance space, we have also created a synthetic *UniformVectors* dataset comprising 100,000 uniformly distributed 10-dimensional vectors (employing the Euclidean distance). We have used this dataset in the very last experiment, where we have investigated the index behavior under growing intrinsic dimensionality.

Whereas *ALOI* and *CLOUDS* datasets employ expensive distance measures with quadratic time complexity according to the number of centroids in a feature signature (or points in a cloud respectively), the *CoPhIR* and *UniformVectors* datasets with the Euclidean distance represent an example of a cheap distance space. For the latter two datasets the number of spent distance computations is not the main criterion for efficiency evaluation. Hence, in the experiments with them we always present graphs for both real time and the number of distance computations, while in the experiments with *ALOI* and *CLOUDS* datasets we present graphs with just the number of distance computations (abbreviated as DC).

Regarding the pivot sets used in cut-regions, we have used simple random selection of pivots (from the respective dataset). Although there have been many sophisticated pivot selection strategies developed [30], their benefits over random pivot selection are only significant for small numbers of pivots (say up to a few tens), where the chance to select a good pivot by random is lower. To measure the query performance, we have used mainly 10NN queries, where the query costs were always averaged for 100 randomly selected query objects from the database (query objects were not indexed). The tests ran on an Intel Core i7 920 3.4 GHz, 9 GB RAM, Win 7 $\times$ 64.

Table 1
Abbreviations of the presented methods based on PM-tree.

Abbreviations	SW, Single-Way leaf selection; MW, Multi-Way leaf selection; BR, ball-region; CR, cut-region
Method	**Description**
SW(BR, BR)	SW using BR for cand. selection and split
SW(BR, BR, BR)	SW(BR, BR) + BR reinserting
MW(BR, BR)	MW using BR for cand. selection and split
MW(BR, BR, BR)	MW(BR, BR) + BR reinserting
SW(CR, BR)	SW using CR for cand. selection, BR for split
SW(CR, BR, CR)	SW(CR, BR) + CR reinserting
MW(CR, BR)	MW using CR for cand. selection, BR for split
MW(CR, BR, CR)	MW(CR, BR) + CR reinserting
SW(CR, CR)	SW using CR for cand. selection and split
SW(CR, CR, CR)	SW(CR, CR) + CR reinserting
MW(CR, CR)	MW using CR for cand. selection and split
MW(CR, CR, CR)	MW(CR, CR) + CR reinserting

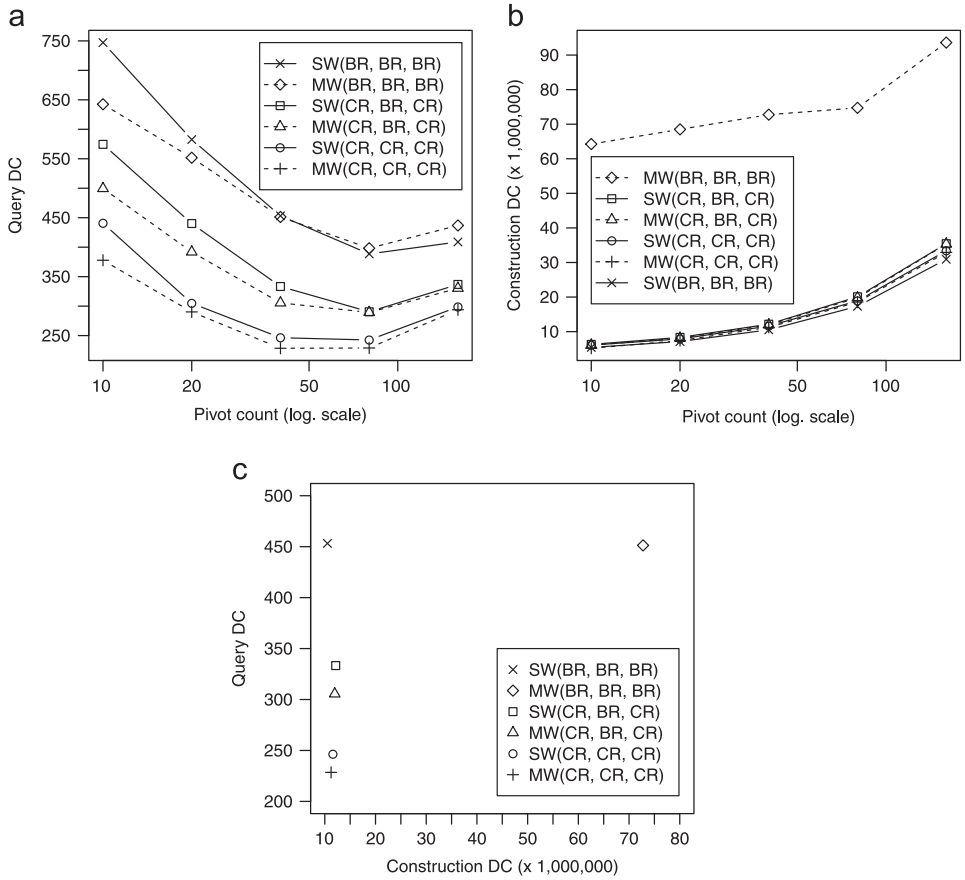


Fig. 17. PM-tree and expensive signature quadratic form distance – indexing and query costs depending on the growing number of global pivots. (a) ALOI, (b) ALOI and c) ALOI, 40 pivots.

7.2. The results

In the first set of experiments, we have investigated combinations of the new proposed construction techniques for PM-tree and compared them with the original PM-tree construction techniques. In the second set of experiments, we have investigated new inner parameters of the other extended MAMs and discussed their impact on the index performance. Finally, we have compared the cut-region-extended PM-tree, List of Cut-Regions and M-Index all together with several other state-of-the-art metric access methods, investigating also the impact of growing intrinsic dimensionality.

7.2.1. PM-tree

The first set of the experiments focuses on all the variants of the PM-tree using ball-regions or/and cut-regions. For better orientation in the plenty of PM-tree construction methods, we use abbreviations (see Table 1) in the form of *LeafSelection(CandidateSelection, SplitHeuristic, Reinserting)*, where *LeafSelection* is either single-way (SW) or multi-way (MW) leaf selection strategy, *CandidateSelection* uses either the original ball-region (BR) or new cut-region (CR) based heuristic for best candidate node selection, *SplitHeuristic* uses either the original ball-region *mMaxRad* or new cut-region *mCutReg* based node splitting strategy, and finally, *Reinserting* is optional parameter using either original ball-region (BR) or new cut-region (CR) based reinserting heuristic (see Section 4). For reinserting, we have set the recursion depth to 6 and the number of removed entries to 1 [22]. We have set the capacity of the PM-tree inner nodes to 20 entries, while the capacity of the leaf nodes was dynamically computed in order to have the same size for inner and leaf nodes. Let us note, for *CoPhIR* dataset with relatively small vectorial data the capacity of the leaf nodes ranged from 22 to 33 according to the number of global pivots used. The minimal utilization of the nodes was set to 40% and the same number of pivots both for ground and routing entries was used.

In Fig. 17, we have measured the indexing and query costs under varying number of global pivots that significantly affects the index performance when expensive signature quadratic form distance is used. Since the distance is quite expensive and the distance space has low intrinsic dimensionality, we have utilized construction methods involving reinsertions that provide slightly better query performance for some additional construction overhead. We may observe that there is an optimal number of pivots (80) in the (a) graph for all the compared methods when considering query performance, while in the (b) graph for all the techniques the

indexing costs grow linearly (let us note there is a logarithmic scale on *x*-axis). We may also observe that using more than 80 pivots negatively affects the query performance; the initial query cost of mapping the query object to the pivot space becomes significant.

In all the graphs, we may observe the standard trade-off behavior of the original ball-region based leaf selection methods – either the indexing is cheap and query costs are high or vice versa. However, on the *ALOI* dataset the cut-region based leaf selection methods break this "rule" as clearly shown in the (c) graph where the query and construction costs are shown against each other in the scatter chart (for better clarity, we have fixed the number of pivots to 40 in the scatter chart). The best possible

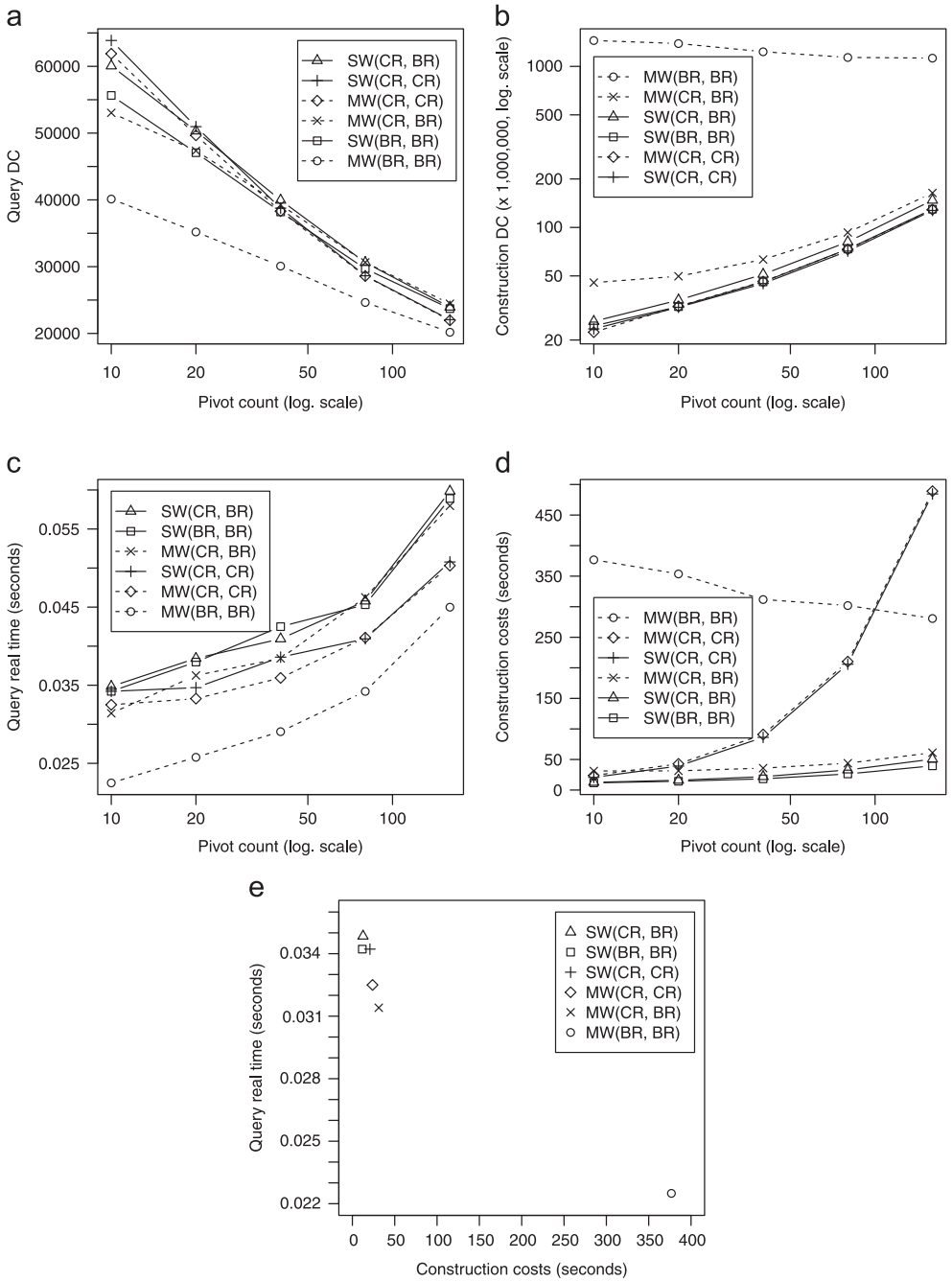


Fig. 18. PM-tree and cheap Euclidean distance measure – indexing and querying costs depending on the growing number of global pivots. (a, b) CoPhIR, (c,d) CoPhIR and (e) CoPhIR, 10 pivots.

index is located in the origin of the coordinate system. In the graph the SW(CR, CR, CR) and MW(CR, CR, CR) methods outperform all the other variants. We may observe that the (otherwise expensive) multi-way leaf selection strategies are cheap in this case, because the more compact cut-regions limit the number of visited routing entries during the leaf selection process, while the more compact hierarchy neglects the additional costs for the multi-way leaf selection method. Using the cut-region based indexing strategies for the expensive SQFD metric we obtain 2 × better query performance using the cheap single-way leaf selection and also 7 × lower indexing costs for multi-way leaf selection strategies comparing to the original ball-region based strategies.

In Fig. 18, we have again measured the indexing and query costs for *CoPhIR* under varying number of global pivots, however, now with the Euclidean distance. As the distance is cheap and the intrinsic dimensionality of the *CoPhIR* space is higher than that for *ALOI*, we do not consider reinserting strategies that just slightly reduce the number of distance computations spent by queries. If we consider again just distance computations as a performance measure, we may observe that the first two graphs (first line in Fig. 18) resemble the first two graphs in Fig. 17. Furthermore, increasing number of pivots constantly improves the query performance because the intrinsic dimensionality of the *CoPhIR* dataset is higher, queries more expensive (in terms of DC), and thus the mapping of query object into the pivot space is not as significant overhead.

However, if we consider the real time performance measure in the next two graphs (second line in Fig. 18), we may see the query processing and some cut-region based construction methods are negatively affected by the growing number of global pivots. More specifically, the query processing is deteriorated by the pivot-based lower bound estimation which, in fact, has the same time complexity ($O(n)$ where n is the number of pivots) as the utilized Euclidean distance (also complexity $O(n)$, but here n is the number of dimensions). As a result, for more pivots the query real time increases contrary to the number of distance computations. The negative effect can be observed also for the cheap metric indexing techniques using cut-region based split heuristic *mCutReg* where for growing number of pivots the SW(CR, CR) and MW(CR, CR) become the most expensive techniques. This is caused by the fact that *mCutReg* contains the CutRegionExtension method (depending linearly on the number of pivots) that has to be computed for all objects in the node for each candidate pair. The best query performance of MW

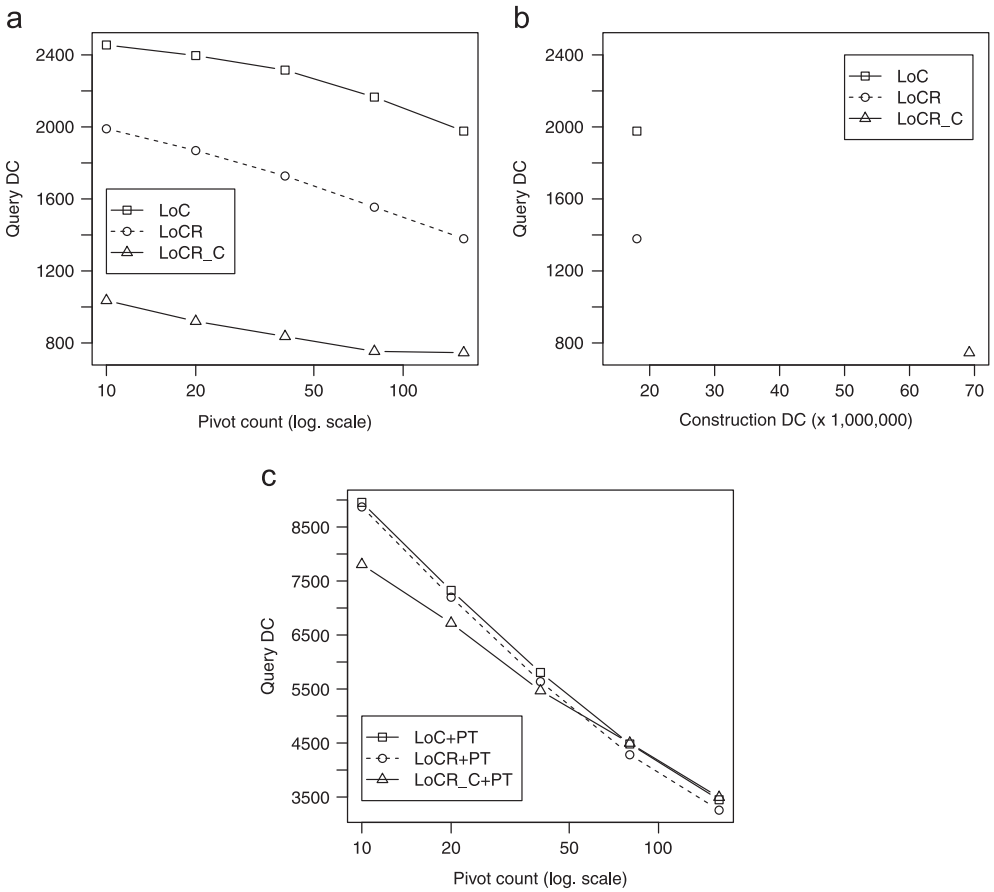


Fig. 19. List of Clusters/Cut-Regions – indexing and querying costs depending on the growing number of global pivots. (a) ALOI, (b) ALOI, 160 pivots and (c) CLOUDS.

(BR, BR) can be explained by the fact that during indexing there are many leaf nodes visited to locate the optimal one (important especially for datasets with high intrinsic dimensionality).

We may conclude that the cut-region based PM-tree indexing strategies are good choices both for cheap and expensive metrics. Furthermore, cut-regions reduce enormous complexity differences between single-way and multi-way leaf selection strategies and using cut-regions both of the strategies are able to cheaply find good sub-optimal leaf node in the PM-tree.

7.2.2. List of Cut-Regions

The second set of the experiments focuses on the List of Clusters denoted as LoC and the List of Cut-Regions denoted as LoCR (*SimpleQuery* heuristics used) or LoCR_C (*CumulativeQuery* heuristics used). To create the clusters, we have used 100NN queries and set the minimal capacity to 20% and λ to 1. The settings affected the number of clusters for all used datasets. More specifically, the number of clusters for LoC and LoCR was 600 for ALOI and 500 for CLOUDS and CoPhIR datasets, while the number of clusters for LoCR_C was 3–4 times higher. Let us note a higher number of final clusters dramatically affects the indexing costs. Also note that here we have used reduced CoPhIR and CLOUDS databases (only 50,000 objects).

In Fig. 19a the List of Cut-Regions outperforms the original List of Clusters method, especially in case of LoCR_C that is twice more efficient, although LoCR_C has 4 × more clusters than LoC. The trade-off graph is depicted in Fig. 19b, aggregating indexing and query costs together, where the best index is in the origin of the coordinate system. In the graph the LoCR method is clear winner over the LoC method, while LoCR_C lies on the skyline. In Fig. 19c, all the methods in connection with the pivot table are depicted. Storing the pivot table in the clusters reduces differences between the three methods, however, the List of Cut-Regions still outperforms the original List of Clusters method.

In Fig. 20 we may observe that using more pivots for cheap metric distances decreases the number of distance computations, however, it also results in worse query real time as in the case of the PM-tree.

7.2.3. M-Index

The third set of the experiments focuses on the M-Index where cut-regions do not reduce the number of distance computations, but reduce the number of interval queries into the B^+-tree. We have used the dynamic M-Index in two

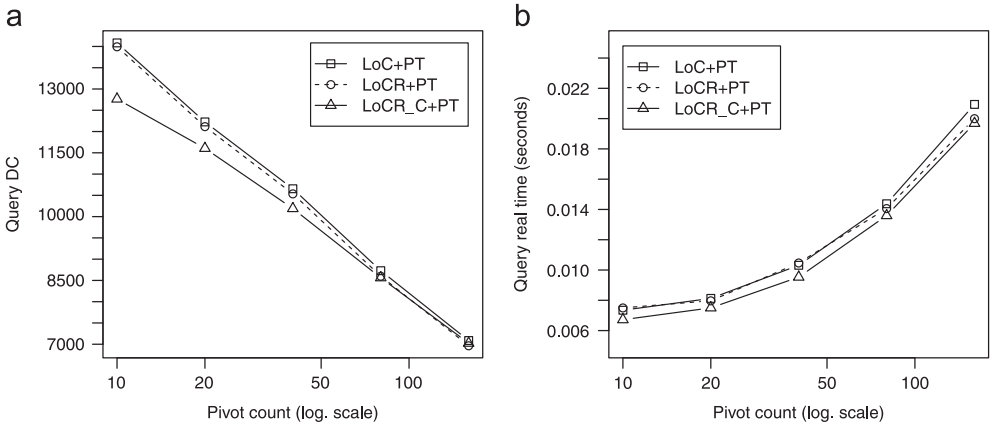


Fig. 20. List of Clusters/Cut-Regions – indexing and querying costs depending on the growing number of global pivots. (a, b) CoPhIR (reduced size 50,000).

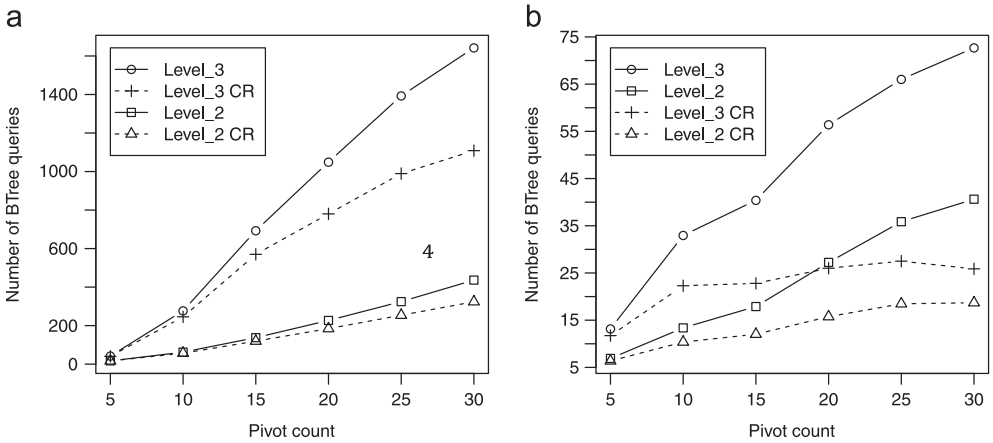


Fig. 21. M-Index – number of B^+-tree queries depending on the growing number of global pivots. (a) CoPhIR and (b) ALOI.

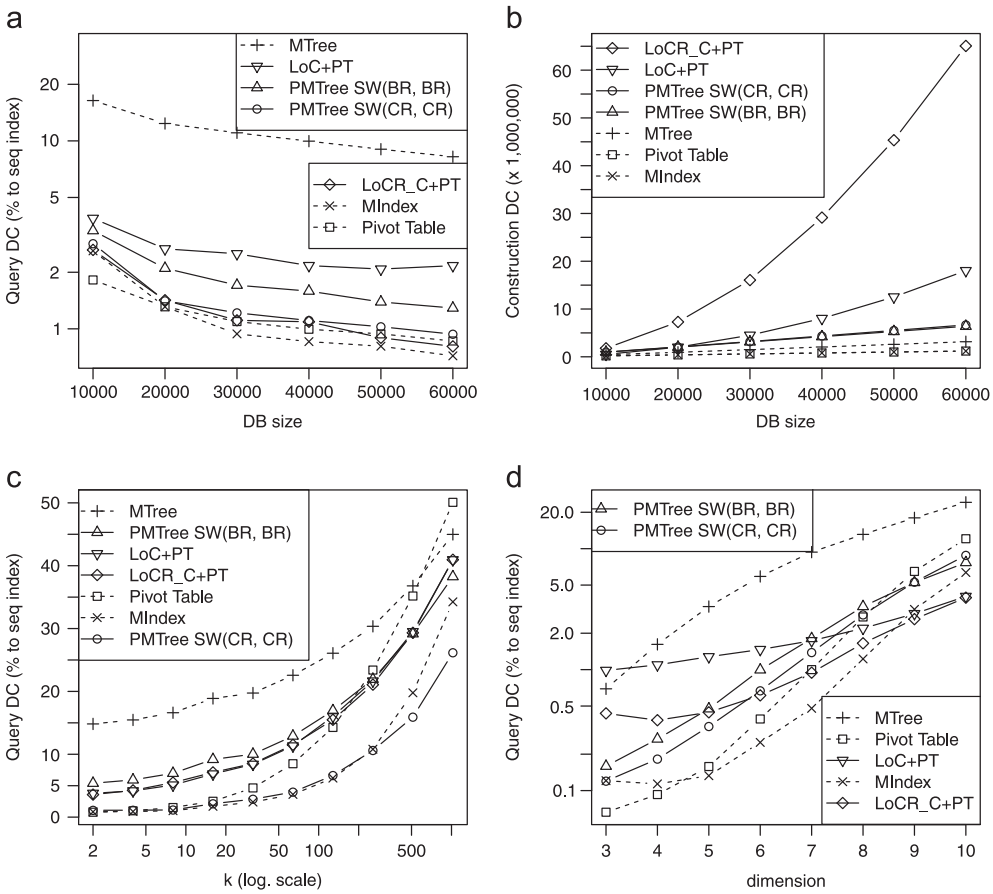


Fig. 22. Inter-MAM comparison – indexing and querying costs depending on varying database size, growing query size and varying data dimensionality. (a, b) ALOI, 20 pivots, (c) CLOUDS, 20 pivots and (d) UniformVectors, 10NN, 20 pivots.

variants differing in the level (maximal depth) of the cluster tree – Level_2 and Level_3. The maximal capacity of the leaf node of the dynamic cluster tree was set to 500.

In Fig. 21 the M-Index improved by cut-regions outperforms the original variants in the number of B^+-tree queries. When considering the general M-Index behavior, for the growing number of pivots the number of distance computations decreases, but the number of B^+-tree interval queries increases. In Fig. 21b the cut-regions reduce the increase in interval queries significantly, where the three-level M-Index using cut-regions outperforms two-level M-Index without cut-regions (from 20 pivots on). Let us note the small number of B^+-tree queries can be beneficial especially in the distributed version of the M-Index [14], where the data partitions can be located on different network nodes.

7.2.4. Overall comparison

In the last set of experiments, we have evaluated the competitiveness of all used methods plus we have added two other MAMs – M-tree [9] and Pivot table (LAESA) [7]. In the top part of Fig. 22, we may observe that MAMs enhanced by cut-regions can compete with the currently best MAM – the M-Index. We may also observe better performance of the PM-tree for less selective queries (i.e., large query radius or many nearest neighbors) that is caused by the use of many local pivots dynamically created during PM-tree construction (see Fig. 22c). In the last experiment in Fig. 22d, we have used synthetically created *UniformVectors* dataset to measure index performance under growing dimensionality of the distance space.[8] We may observe that PMTree SW(CR, CR) using cut-region construction strategy outperforms PMTree SW(BR, BR) on lower dimensions, while on higher dimensions (>9) the ball-region based strategies form more compact metric hierarchies. This is probably caused by the fact that in high-dimensional distance spaces the probability of compact cut-regions decreases (i.e., all regions resemble balls) and so simple heuristics considering only ball-regions may be more suitable for leaf selection strategies and node split operations. We may also observe that the uniformly distributed data with no natural clusters are more suitable for M-Index in lower dimensions, while the

[8] For Euclidean spaces and uniformly distributed data, the intrinsic dimensionality is roughly the same as the embedding (vectorial) dimensionality, i.e., $\rho \approx D$.

List of Clusters/Cut-Regions become the best indexing method for high-dimensional distance spaces.

8. Conclusion

In this paper we have presented the concept of cut-regions that could heavily improve the performance of metric indexes that were originally designed to employ simple ball-regions. In the experiments, we have confirmed that cut-regions improve the query performance of ball-region based metric indexes. Furthermore, cut-regions significantly cheapen the multi-way leaf selection strategy in the PM-tree which makes this strategy applicable in real problems. Although compact metric hierarchies can be outperformed by new MAM designs (e.g., M-Index), they can be still beneficial for modern retrieval modalities or in data mining applications. We have also shown that cluster tree hierarchy improved by cut-regions can reduce the internal costs of the already superior M-Index, which is beneficial for distributed environments.

Acknowledgments

This research has been supported in part by Czech Science Foundation Projects P202/11/0968, P202/12/P297 and by Grant Agency of the Charles University (GAUK) Project 910913.

References

[1] E. Chávez, G. Navarro, R. Baeza-Yates, J.L. Marroquín, Searching in metric spaces, ACM Comput. Surv. 33 (2001) 273–321.
[2] P. Zezula, G. Amato, V. Dohnal, M. Batko, Similarity Search: The Metric Space Approach, Springer, Berlin, Germany, 2005.
[3] H. Samet, Foundations of Multidimensional and Metric Data Structures, Morgan Kaufmann, San Francisco, CA, USA, 2006.
[4] M.L. Hetland, The basic principles of metric indexing, in: C.A. C. Coello, S. Dehuri, S. Ghosh (Eds.), Swarm Intelligence for Multi-objective Problems in Data Mining, Studies in Computational Intelligence, vol. 242, Springer, Berlin, Heidelberg, Germany, 2009.
[5] D. Novak, M. Batko, P. Zezula, Metric index: an efficient and scalable solution for precise and approximate similarity search, Inf. Syst. 36 (2011) 721–733.
[6] T. Skopal, J. Lokoč, B. Bustos, D-cache: universal distance cache for metric access methods, Trans. Knowl. Data Eng. 24 (2012) 868–881.
[7] M.L. Mico, J. Oncina, E. Vidal, A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements, Pattern Recognit. Lett. 15 (1994) 9–17.
[8] E. Chávez, K. Figueroa, G. Navarro, Effective proximity retrieval by ordering permutations, IEEE Trans. Pattern Anal. Mach. Intell. 30 (2008) 1647–1658.
[9] P. Ciaccia, M. Patella, P. Zezula, M-tree: an efficient access method for similarity search in metric spaces, in: VLDB'97, 1997, pp. 426–435.
[10] S. Brin, Near neighbor search in large metric spaces, in: U. Dayal, P.M. D. Gray, S. Nishio (Eds.), VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11–15, 1995, Zurich, Switzerland, Morgan Kaufmann, San Francisco, CA, USA, 1995, pp. 574–584.
[11] R. Uribe, G. Navarro, R.J. Barrientos, M. Marín, An index data structure for searching in metric space databases, in: I nternational Conference on Computational Science (1), 2006, pp. 611–617.
[12] C. Beecks, T. Skopal, K. Schoeffmann, T. Seidl, Towards large-scale multimedia exploration, in: G. Das, V. Hsristidis, I. Ilyas (Eds.), Proceedings of the 5th International Workshop on Ranking in Databases (DBRank 2011), VLDB, Seattle, WA, USA, 2011, pp. 31–33.
[13] H.V. Jagadish, B.C. Ooi, K.-L. Tan, C. Yu, R. Zhang, iDistance: an adaptive B+-tree based indexing method for nearest neighbor search, ACM Trans. Database Syst. 30 (2005) 364–397.
[14] D. Novak, M. Batko, P. Zezula, Large-scale similarity data management with distributed metric index, Inf. Process. Manag. 48 (2012) 855–872.
[15] C. Beecks, M.S. Uysal, T. Seidl, Signature quadratic form distance, in: Proceedings of ACM International Conference on Image and Video Retrieval, 2010, pp. 438–445.
[16] R. Weber, H.-J. Schek, S. Blott, A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces, in: VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998, pp. 194–205.
[17] T. Skopal, Pivoting M-tree: a metric access method for efficient similarity search, in: Proceedings of the 4th Annual Workshop on DATESO, Desná, Czech Republic, ISBN 80-248-0457-3, also available at CEUR, vol. 98, ISSN 1613-0073 ⟨http://www.ceur-ws.org/Vol-98⟩, 2004, pp. 21–31.
[18] T. Skopal, J. Pokorný, V. Snášel, Nearest neighbours search using the PM-tree, in: DASFAA '05, Beijing, China, Lecture Notes in Computer Science, vol. 3453, Springer, Berlin, Heidelberg, Germany, 2005, pp. 803–815.
[19] T. Skopal, Unified framework for fast exact and approximate search in dissimilarity spaces, ACM Trans. Database Syst. 32 (2007) 1–46.
[20] T. Skopal, J. Lokoč, Answering metric skyline queries by PM-tree, in: Proceedings of the Dateso 2010 Workshop, vol. 567, MATFYZPRESS, 2010, pp. 22–37.
[21] J. Lokoč, T. Skopal, On reinsertions in M-tree, in: SISAP '08: Proceedings of the First International Workshop on Similarity Search and Applications (sisap 2008), IEEE Computer Society, Washington, DC, USA, 2008, pp. 121–128. ⟨http://dx.doi.org/10.1109/SISAP.2008.10⟩.
[22] T. Skopal, J. Lokoč, New dynamic construction techniques for M-tree, J. Discrete Algorithms 7 (2009) 62–77.
[23] E. Chávez, G. Navarro, A compact space decomposition for effective metric indexing, Pattern Recognit. Lett. 26 (2005) 1363–1376.
[24] L. Ares, N. Brisaboa, A. Ordez Pereira, O. Pedreira, Efficient similarity search in metric spaces with cluster reduction, in: G. Navarro, V. Pestov (Eds.), Similarity Search and Applications, Lecture Notes in Computer Science, vol. 7404, Springer, Berlin, Heidelberg, 2012, pp. 70–84, http://dx.doi.org/10.1007/978-3-642-32153-5.6.
[25] J.-M. Geusebroek, G.J. Burghouts, A.W.M. Smeulders, The Amsterdam library of object images, Int. J. Comput. Vis. 61 (2005) 103–112.
[26] M. Kruliš, J. Lokoč, T. Skopal, Efficient extraction of feature signatures using multi-gpu architecture, in: S. Li, A. Saddik, M. Wang, T. Mei, N. Sebe, S. Yan, R. Hong, C. Gurrin (Eds.), Advances in Multimedia Modeling, Lecture Notes in Computer Science, vol. 7733, Springer, Berlin, Heidelberg, 2013, pp. 446–456, http://dx.doi.org/10.1007/978-3-642-35728-2.43.
[27] C. Beecks, J. Lokoč, T. Seidl, T. Skopal, Indexing the signature quadratic form distance for efficient content-based multimedia retrieval, in: Proceedings of ACM International Conference on Multimedia Retrieval, 2011.
[28] P. Bolettieri, A. Esuli, F. Falchi, C. Lucchese, R. Perego, T. Piccioli, F. Rabitti, CoPhIR: a test collection for content-based image retrieval, CoRR abs/0905.4627v2, 2009.
[29] D. Huttenlocher, G. Klanderman, W. Rucklidge, Comparing images using the Hausdorff distance, IEEE Pattern Anal. Mach. Intell. 15 (1993) 850–863.
[30] B. Bustos, G. Navarro, E. Chávez, Pivot selection techniques for proximity searching in metric spaces, Pattern Recognit. Lett. 24 (2003) 2357–2366.
[31] J. Lokoč, P. Čech, J. Novák, T. Skopal, Cut-region: a compact building block for hierarchical metric indexing, in: G. Navarro, V. Pestov (Eds.), Similarity Search and Applications, volume 7404 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2012, pp. 85–100.

Chapter 5

D-Cache: Universal Distance Cache for Metric Access Methods

Tomáš Skopal
Jakub Lokoč
Benjamin Bustos

Published in the *IEEE Transactions on Knowledge and Data Engineering* journal, volume 24, Number 5, pages 868-881. IEEE, May 2012. ISSN 1041-4347.
dx.doi.org/10.1109/TKDE.2011.19

Impact Factor in 2012: 1.892

D-Cache: Universal Distance Cache for Metric Access Methods

Tomáš Skopal, *Member*, *IEEE*, Jakub Lokoč, and Benjamin Bustos

Abstract—The caching of accessed disk pages has been successfully used for decades in database technology, resulting in effective amortization of I/O operations needed within a stream of query or update requests. However, in modern complex databases, like multimedia databases, the I/O cost becomes a minor performance factor. In particular, metric access methods (MAMs), used for similarity search in complex unstructured data, have been designed to minimize rather the number of distance computations than I/O cost (when indexing or querying). Inspired by I/O caching in traditional databases, in this paper we introduce the idea of *distance caching* for usage with MAMs—a novel approach to streamline similarity search. As a result, we present the *D-cache*, a main-memory data structure which can be easily implemented into any MAM, in order to spare the distance computations spent by queries/updates. In particular, we have modified two state-of-the-art MAMs to make use of D-cache—the M-tree and Pivot tables. Moreover, we present the D-file, an index-free MAM based on simple sequential search augmented by D-cache. The experimental evaluation shows that performance gain achieved due to D-cache is significant for all the MAMs, especially for the D-file.

Index Terms—Metric indexing, similarity search, distance caching, metric access methods, D-cache, MAM, index-free search.

1 Introduction

In database technology, the majority of problems concerns the efficiency issues, that is, the performance of a DBMS. For decades, the number of accesses to disk (required by I/O operations) was the dominant factor affecting the DBMS performance. There were developed indexing structures [1], [2], storage layouts [3], and also disk caching/buffering techniques [4]; all of these designs aimed to minimize the number of physical I/Os spent within a database transaction flow. In particular, disk caching was proven to be extremely effective in situations where access to some disk pages happens repeatedly during a single runtime session.

However, the situation is dramatically different in modern complex databases consisting of snapshots of nature (i.e., images, sounds, or other signals), like multimedia databases, bioinformatic databases, time series, etc. Here, we often adopt the similarity search within the content-based retrieval paradigm, where a similarity function $\delta(q, o)$ serves as a measure saying how much a database object $o \in \mathbf{S}$ is relevant to a query object $q \in \mathbf{U}$ (where $\mathbf{S}$ is the database and $\mathbf{U}$ is the object universe, $\mathbf{S} \subset \mathbf{U}$). To speed up similarity search in such a database, there have been many indexing techniques developed—some of them domain specific and some others more general. Also, there were distributed indexing techniques developed [5] that use parallelism to speed up similarity queries. An important fact is that the retrieval performance of such a system is more affected by CPU cost than by I/O cost. In particular, in similarity-search community the computation of a single value δ is employed as the logical unit for indexing/retrieval cost, because of its dominant impact on the overall performance [6], [7]. Thus, the I/O cost is mostly regarded as a minor component of the overall cost. The number of computations δ needed to answer a query (or to index a database) is referred to as the *computation cost*.

Among general techniques, the *metric access methods* (MAMs) are suitable in situations where the similarity measure δ is a *metric* distance (in mathematical meaning). The metric properties (1), (2), (3), (4) allow us to organize a database $\mathbf{S}$ within equivalence classes, embedded in a data structure which is stored in an *index file*

$$\begin{aligned} \delta(x,y) &= 0 \Leftrightarrow x = y && \text{identity (1)},\\ \delta(x,y) &> 0 \Leftrightarrow x \neq y && \text{nonnegativity (2)},\\ \delta(x,y) &= \delta(y,x) && \text{symmetry (3)},\\ \delta(x,y) + \delta(y,z) &\geq \delta(x,z) && \text{triangle inequal (4)}. \end{aligned}$$

The index is later used to quickly answer typical similarity queries—either a *k-nearest neighbors* (kNN) query like "return the three most similar images to my image of a horse," or a *range query* like "return all voices more similar than 80 percent to the voice of a nightingale." In particular, when issued a similarity query, the MAMs exclude many nonrelevant equivalence classes from the search (based on metric properties of δ), so only several candidate classes of objects have to be exhaustively (sequentially) searched. In consequence, searching a small number of candidate classes turns out in reduced computation cost of the query. For a comprehensive survey on MAMs, we refer to [7], [8] or monographs [6], [9]. Again, we have to emphasize the assumption on computationally *expensive* distance metric δ (i.e., $>O(n)$, where n is the size of object o_i). In other words,

- *T. Skopal and J. Lokoč are with the Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Malostranske nam. 25, Prague 118 00, Czech Republic. E-mail: {skopal, lokoc}@ksi.mff.cuni.cz.*
- *B. Bustos is with the Department of Computer Science, University of Chile, Av. Blanco Encalada 2120, Santiago 8370459, Chile. E-mail: bebustos@dcc.uchile.cl.*

Manuscript received 9 Mar. 2010; revised 17 Sept. 2010; accepted 22 Oct. 2010; online 28 Jan. 2011.
Recommended for acceptance by J. Haritsa.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2010-03-0131.
Digital Object Identifier no. 10.1109/TKDE.2011.19.

the real time spent in distance computations is assumed to dominate the real time spent in other parts of MAMs' algorithms (including I/O cost).

1.1 Motivation for Distance Caching

Importantly, after a metric index is built, the existing MAMs solve every query request *separately*, that is, every query is evaluated as it would be the only query to be answered. In general, no optimization for a *stream of queries* (query requests spread in time) has been considered for MAMs up to date. Instead, huge efforts were given to "materializing" the filtering knowledge into the index file itself.

In this paper, we change this paradigm and propose a structure for *caching distances* computed during the current runtime session. The distance cache ought to be an analogy to the classic disk cache widely used in database management to optimize I/O cost. Hence, instead of sparing I/Os, the distance cache should spare distance computations. A desired feature of distance cache should be its universal usage with all MAMs, similarly like disk caching is universal for standard I/O management. The main idea behind the distance caching resides in approximating the requested distances by providing their lower and upper bounds "for free." Since some "useful" distances could have been computed during previous querying/indexing, such distances could still "sit" in the distance cache and thus could be used to infer (more or less tight) approximations of distances we request.

1.2 Paper Contribution

We present D-cache (distance cache), a tool for general metric access methods that helps to reduce the cost of both, indexing and querying. The basic task of D-cache is to cheaply determine tight lower- and upper bound of an unknown distance between two objects, based on stored distances computed during previous querying and/or indexing. Although the D-cache was already introduced in our preliminary work [10], it was applied in a more narrowed context—as a tool for efficient index-free similarity search (resulting in a new method, the D-file). Moreover, in this paper we not only employ the D-cache in various MAMs, but we present a completely redesigned D-cache variant that is more effective (provides tighter lower/upper bounds) and also more efficient (faster bound determination) than the previous version.

2 Metric Access Methods

In the following, we consider three out of dozens of existing MAMs—the *sequential file* (a trivial MAM), the *Pivot Tables*, and the *M-tree*. Later in the paper, we will consider extensions of these MAMs by the announced D-cache structure.

2.1 Sequential File

The sequential file is simply the original database, where any query involves a sequential scan over all the database objects. For a query object q and every database object o_i, a distance $\delta(q, o_i)$ must be computed (regardless of query selectivity). Although this kind of "MAM" is not very smart, it does not require any index (and no indexing), which can be useful in many situations (as discussed in Section 4.1).

2.2 Pivot Tables

A simple but efficient solution to similarity search represent methods called *pivot tables* (or distance matrix methods). In general, a set of p objects (so-called pivots) is selected from the database, while for every database object a p-dimensional vector of distances to the pivots is created. The vectors belonging to the database objects then form a distance matrix—the pivot table. When performing a range query (q, rad), a distance vector for the query object q is determined the same way as for a database object. From the query vector and the query radius rad a p-dimensional hypercube is created, centered in the query vector (query point, actually) and with edges of length 2 rad. Then, the range query is processed on the pivot table, such that vectors of database objects that do not fall into the query cube are filtered out from further processing. The database objects that cannot be filtered have to be subsequently checked by the usual sequential search.

There have been many MAMs developed based on pivot tables. The AESA [11] treats all the database objects as pivots, so the resulting distance matrix has quadratic size with respect to the database size. Also, the search algorithms of AESA are different, otherwise the determination of a query vector would turn out in sequential scan of the entire database. The advantage of AESA is empirical average constant complexity of nearest neighbor search. The drawback is quadratic space complexity and also quadratic time complexity of indexing (creating the matrix) and of the external CPU cost (loading the matrix when querying). The LAESA [12] is a linear variant of AESA, where the number of pivots is assumed far smaller than the size of the database (so that query vector determination is not a large overhead). The concept of LAESA was implemented many times under different conditions, we name, e.g., TLAESA [13] (pivot table indexed by GH-tree-like structure), Spaghettis [14] (pivot table indexed by multiple sorted arrays), OMNI family [15] (pivot table indexed by R-tree), PM-tree [16] (hybrid approach combining M-tree and pivot tables). In the rest of the paper, we consider the simplest implementation of pivot tables—the original LAESA.

2.3 M-Tree

The *M-tree* [17] is a dynamic index structure that provides good performance in secondary memory (i.e., in database environments). The M-tree is a hierarchical index, where some of the data objects are selected as centers (local pivots) of ball-shaped regions, while the remaining objects are partitioned among the regions in order to build up a balanced and compact hierarchy of data regions, see Fig. 1. Each region (subtree) is indexed recursively in a B-tree-like (bottom-up) way of construction.

The inner nodes of M-tree store *routing entries* $rout_l(o_i) = [o_i, rad_{o_i}, \delta(o_i, Par(o_i)), ptr(T(o_i))]$, where $o_i \in \mathbf{S}$ is a data object representing the center of the respective ball region, rad_{o_i} is a *covering radius* of the ball, $\delta(o_i, Par(o_i))$ is the so-called *to-parent* distance (the distance from o_i to the object of the parent routing entry), and finally $ptr(T(o_i))$ is a pointer to the entry's subtree. The data are stored in the leaves of M-tree. Each leaf contains *ground entries* $grnd(o_i) = [o_i, \delta(o_i, Par(o_i))]$, where $o_i \in \mathbf{S}$ is an indexed database object and $\delta(o_i, Par(o_i))$ is, again, the to-parent distance.

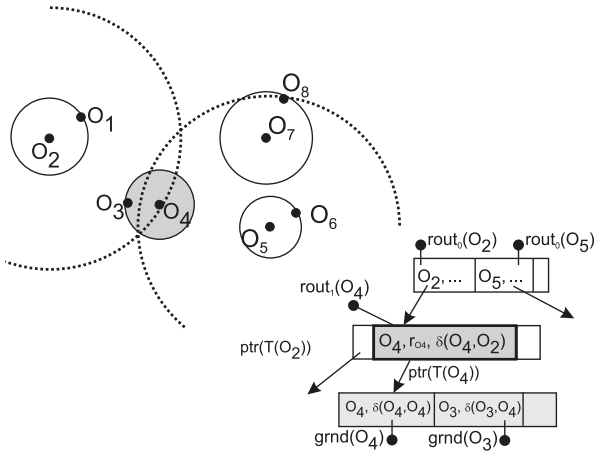


Fig. 1. M-tree (hierarchical space decomposition and the tree structure).

Range and kNN queries are implemented by traversing the tree, starting from the root. Those nodes are accessed, whose parent regions (described by the routing entries) are overlapped by the query ball (q, rad). In case of a kNN query the radius rad is not known beforehand, so we have to additionally employ a heuristics to dynamically decrease the radius during the search (initially set to ∞). The kNN algorithm performs a best-first traversal of the index, where regions are accessed in the order of increasing lower bound distance to q.

2.3.1 M-Tree Construction

In the original M-tree proposal [17], the index was constructed by multiple dynamic insertions, which consisted of two steps. First, an appropriate leaf node for the newly inserted object is found by traversing a single path in the tree (so-called *single-way* leaf selection). Second, if a leaf gets overfull after the insertion, it is split, such that two objects from the split leaf are selected as centers of the new two leafs, while the remaining objects within the split leaf are distributed among the new leafs. Simultaneously, the new centers form new routing entries that are inserted into the parent node (if the parent gets overfull as well, the splitting proceeds recursively).

In addition to the original M-tree, in this paper we consider also recent advanced techniques of dynamic M-tree construction [18]. In particular, we consider the *multiway* leaf selection. Although the multiway selection is more expensive than the single-way variant, the target leaf is more appropriate for the newly inserted object. Specifically, a point query is issued, such that from all the "touched" leaves the selected one has its center closest to the newly inserted object. Another improvement in M-tree construction is adopting the well-known technique of *forced reinsertions*. When a leaf is about to split after a new insertion, some objects are removed from the leaf and inserted again into the M-tree under the hope they will not all arrive into the same leaf again (thus avoiding the split). Both of the advanced construction techniques (multiway leaf selection and forced reinsertions) lead to more compact M-tree hierarchies, which, in turn, lead to faster query processing.

3 D-Cache

We propose a nonpersistent (main-memory) structure called *D-cache* (distance cache), that stores distances already computed by a MAM. We consider a single runtime session of a search engine, that is, a contiguous usage of a MAM for a sequence of queries, insertions, or both. The track of distance computations is stored as a set of triplets, each of form:

$$[id(o_i), id(o_j), \delta(o_i, o_j)],$$

where $id(o_i), id(o_j)$ are unique identifiers of objects o_i, o_j, and $\delta(o_i, o_j)$ is their distance.

To distinguish between the roles of "active and passive objects," we use the term *runtime object*, that denotes an object that is currently subject to an operation on MAM (either query or insertion). Once the operation is finished, the respective object becomes *past runtime object*, meaning either a regular database object (after an insertion) or a past query object. All objects are uniquely identified, regardless of their role (query, inserted object, database object). For instance, the runtime objects could be identified by the order they enter the index (forever, i.e., also for their past-runtime role), where as "entering" we mean either an insertion or a query.

Instead of considering a set of triplet entries, we can view the content of D-cache as a sparse matrix

$$D = \begin{array}{c} \\ o_1 \\ o_2 \\ o_3 \\ \dots \\ o_m \end{array} \begin{array}{c} \begin{array}{ccccc} o_1 & o_2 & o_3 & \dots & o_n \end{array} \\ \begin{pmatrix} & \delta_{12} & \delta_{13} & \dots & o_n \\ \delta_{21} & & & \dots & \delta_{2n} \\ & & & & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \delta_{m1} & & \delta_{m3} & \dots & \end{pmatrix} \end{array},$$

where the rows and columns refer to objects, and the cells store the respective object-to-object distances. Naturally, as new runtime objects appear during the session, the matrix gets larger (in number of rows and/or columns). At the beginning of the session the matrix is empty, while during the session the matrix is being extended and filled. Note that runtime objects do not have to be external, that is, a runtime object could originate from the database (e.g., a query or a reinserted object). From this point of view, an object could have different roles at different moments, however, the unique objects identification ensures the D-cache content is correct.

Because of frequent insertions of triplets into D-cache, the matrix should be efficiently updatable. Moreover, due to operations described in the next section, we should be able to quickly retrieve the value of a particular cell.

3.1 Principle of D-Cache

The desired functionality of D-cache is twofold:

First, given a pair runtime object/database object $\langle r, o \rangle$, the D-cache should quickly determine the *exact* value $\delta(r, o)$ in case the distance is stored in the D-cache. However, as the exact value could only be found when the actual distance was already computed previously in the session, this functionality is limited to rather special cases, like reindexing of data objects (or index rearrangements), repeated queries or querying by database objects.

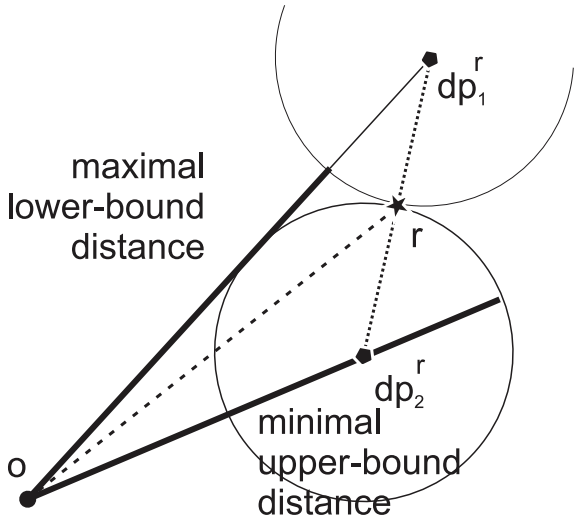


Fig. 2. Lower/upper bounds to $\delta(r,o)$.

The second functionality, which is the main D-cache contribution, is more general. Given a runtime object r and a database object o on input, the D-cache should quickly determine the tightest possible *lower* or *upper bound* of $\delta(r,o)$ without the need of an explicit distance computation. This cheap determination of lower/upper bound distances then serves a MAM in order to filter out a nonrelevant database object or even a whole part of the index. Let us denote a lower-bound distance of $\delta(r,o)$ as $\delta_{LB}(r,o) \leq \delta(r,o)$ and an upper-bound distance as $\delta_{UB}(r,o) \geq \delta(r,o)$.

In order to facilitate the second functionality, we have to feed the D-cache with relevant information about the involved objects. In particular, we would like to know distances to some past runtime objects dp_i^r which are very close to or very far from the current runtime r, that is, suppose for a while we know some $\delta(dp_1^r,r), \delta(dp_2^r,r), \ldots$. These past runtime objects will serve as *dynamic pivots* made-to-measure to r. Formally defined, $dp_i^r \in DP \subseteq PR \subset \mathbf{U}$, where PR is the set of all past runtime objects within the current session and DP is an actual set of selected dynamic pivots (see the next section). Regarding the size of DP, we could choose either $DP = PR$, or set a fixed size $|DP| = k < |PR|$. Note that dynamic pivots could originate outside the database $\mathbf{S}$ (necessary for queries and newly inserted objects).

Since the dynamic pivots are supposed either close to r or far from r, they should be effective for pruning by a MAM (they provide tight approximations of $\delta(r,o_i)$ distances). After the dynamic pivots are selected, the lower/upper bound distances are constructed using the distances $\delta(dp_i^r,o)$ still "sitting" in the D-cache matrix, where they were inserted earlier during the session. In particular, with respect to dp_i^r and available distances $\delta(dp_i^r,o)$ in the matrix, $max_{dp_i^r}\{|\delta(dp_i^r,o) - \delta(dp_i^r,r)|\}$ is the tightest lower-bound distance $\delta_{LB}(r,o)$. Similarly, $min_{dp_i^r}\{\delta(dp_i^r,o) + \delta(dp_i^r,r)\}$ is the tightest upper-bound distance $\delta_{UB}(r,o)$. See the situation in Fig. 2.

3.1.1 Selection of Dynamic Pivots

In the past decade, there were many sophisticated techniques for selection of effective pivots developed, allowing an efficient similarity search [19], [20]. This classic approach assumes the pivot selection procedure as a part of the indexing/preprocessing phase (e.g., before the distance matrix for pivot tables is established). However, in D-cache the dynamic pivots have to be selected at the moment a query or insertion starts. So, there is not much room for preprocessing, such as an expensive pivot selection, even though we select pivots from a rather small set of past runtime objects. Hence, we propose the following cheap pivot selection technique.

We need to choose some k runtime objects from all of the past runtime objects before the current runtime processing actually starts (i.e., before processing a query or insertion). Based on observations taken from the preliminary work on D-cache [10], we consider just the *recent selection policy*. That is, the k *most recent* runtime objects are selected as dynamic pivots, because it is more probable that recent runtime objects have more distances stored in the D-cache than the older ones (i.e., not replaced by other distances, see Section 3.3.2).

After the dynamic pivots are determined, their distances to r have to be computed. Note that this is the *only moment* where some extra distances are explicitly computed, that would not be computed when not using D-cache.

3.2 Distance Matrix Structure

Because the main memory is always limited and the distance matrix could expand to an enormous size, we need to choose a compact data structure that consumes a user-defined portion of main memory. In order to provide also fast retrieval, the D-cache implements the distance matrix as a linear *hash table* consisting of entries $[id1, id2, \delta(o_{id1}, o_{id2})]$. The hash key (pointing to a position in the hash table) is derived from the two ids of objects whose distance is being retrieved or stored.

In addition, there is a constant-size *collision interval* defined, that allows to move from the hashed position to a more suitable one (due to replacement policies, see below). However, in order to keep the D-cache as fast as possible, the collision interval should be very small, preferably just one position in the hash table (i.e., only the hashed position).

3.2.1 Hashing Function

To achieve uniform distribution of the hashed distances, we consider two variants of hashing function f, both taking two integer numbers $id1, id2$ as arguments (the ids of objects).

Simple. The faster variant of f multiplies the ids (modulo the size D of hash table), i.e., $f(id1, id2) = (id1 \cdot id2) \bmod D$. The motivation here is that we expect the ids entering the hashing function are random combinations, so the simple multiplication should produce distribution uniform enough.

Universal. A slightly slower variant of f is based on the Simple variant and on *universal hashing* [21]. Let p be a large prime ($p > D$), and let a, b be two random integer numbers smaller than p. All the numbers p, a, b are fixed during D-cache lifetime. Then, the hashing function is defined as $f(id1, id2) = ((a \cdot id1 \cdot id2 + b) \bmod p) \bmod D$.

3.3 Operations on D-Cache

The D-cache is initialized by a MAM when loading the index (the session begins). Besides the initialization, the D-cache is also notified by a MAM whenever a new query/insertion is to be started (the MAM calls method StartRuntimeProcessing on D-cache). At that moment, new runtime object r is announced to be processed, which also includes the computation of distances from r to the k actual dynamic pivots dp_i^r.

3.3.1 Distance Retrieval

The main D-cache functionality is operated by methods GetDistance and GetLowerBoundDistance,[1] see Algorithm 1.

Algorithm 1: (GetDistance, GetLowerBoundDistance)

```
double GetDistance(r, o_i) {
  let minId = min(id(r), id(o_i)), maxId = max(id(r), id(o_i))
  let CI = size of collision interval
  let h = GetHash(minId, maxId)                    // hashing function f, see Sec. 3.2.1
  for i = 1 to CI                                  // every + is modulo hash table size
    if hashTable[h + i].id1 = minId and hashTable[h + i].id2 = maxId then
      return hashTable[h + i].distance
  return nil }

double GetLowerBoundDistance(r, o_i) {
  let k be the number of pivots to use
  let DP be the set of k dynamic pivots and their distances to r
  if GetDistance(r, o_i) ≠ nil then
    return GetDistance(r, o_i)
  let value = 0
  for each p in DP do
    if GetDistance(p, o_i) ≠ nil then
      value = max(value, |GetDistance(p, o_i) − δ(r, p)|)
  return value }
```

The number of dynamic pivots ($k = |DP|$) used to evaluate GetLowerBoundDistance is set by the user, while this parameter is an exact analogy to the number of pivots used by Pivot tables, e.g., LAESA. There exists no general rule for the automatic determination of the number of pivots [19], [20], especially when minimizing the real-time cost rather than just the number of distance computations. In general, the effective number of pivots depends on the (expected) size of the database, its intrinsic dimensionality (see Section 6.1.1), the computational complexity of the used metric, the pivot set quality itself, etc. The same reasons apply also for D-cache.

3.3.2 Distance Insertion

Every time a distance $\delta(r, o_i)$ is computed by the MAM, the triplet $[id(r), id(o_i), \delta(r, o_i)]$ is inserted into the D-cache (the MAM calls method InsertDistance on D-cache). Since the storage capacity of D-cache is limited, at some moment the collision interval in the hash table for a newly inserted distance entry is full. Then, some older entry within the collision interval has to be replaced by the new entry. Or, alternatively, if it turns out the newly inserted distance is less useful than all the distances in the collision interval, the insertion of the new distance is canceled.

Note that we should prioritize replacing of such entries $[id1, id2, \delta(o_{id1}, o_{id2})]$ where none of the objects o_{id1}, o_{id2} belongs to the current set of k dynamic pivots anymore. Naturally, the distances of such *obsolete entries* cannot be effectively utilized to determine a lower- or upper bound distance, because for a current runtime r only the distances to the k most recent runtimes are useful. In particular, we consider two policies for replacement by a new entry:

Obsolete. The first obsolete entry (i.e., not containing id of a current dynamic pivot) in the collision interval is replaced. In case none of the entries in the collision interval is obsolete, the first entry is replaced by the new entry.

ObsoletePercentile. This policy includes two steps. First, we try to replace the first obsolete entry as in the Obsolete policy. If none of the entries is obsolete, we replace an entry with the least useful distance. As we have mentioned in Section 3.1, a good pivot is either very close to or very far from the database objects. Because the entries in D-cache consist of distances from dynamic pivots to database objects, we should preserve entries with large and small distances and get rid of those close to a "middle" distance. Hence, among all entries in the collision interval the entry that is closest to the "middle" distance is the least useful, thus it is replaced. Of course, it might turn out the least useful distance (closest to the "middle" distance) is that of the newly inserted entry. In such case, the D-cache is not updated by the new entry at all.

Ideally, the "middle" distance should be represented by the *median* distance among objects in the database, that is, a distance value d_m where 50 percent of the computed distances are greater and the other 50 percent distances are smaller than d_m. However, it might turn out that an optimal value for D-cache is not the median distance but a distance belonging to another percentile. Hence, we relax the term "middle" distance to allow not only the fixed median distance (i.e., 50 percent percentile) but also a distance belonging to a user-defined percentile.[2]

The method InsertDistance, including entry replacement, is precisely described in Algorithm 2. The method IsObsolete checks if either of the two ids appears in the actual set of k dynamic pivots' ids (if not, it is an obsolete entry).

Algorithm 2: (InsertDistance)

```
InsertDistance(r, o_i, d_new) {
  let minId = min(id(r), id(o_i)), maxId = max(id(r), id(o_i))
  let CI = size of collision interval
  let d_m be the distance belonging to a user-defined percentile
  let h = GetHash(minId, maxId)
  let finalH = h
  if entry replacement heuristics is Obsolete then {
    for i = 1 to CI                                  // every + is modulo hash table size
      if IsObsolete(hashTable[h+i]) then
        finalH = h + i
        break for
  } else if entry replacement heuristics is ObsoletePercentile then {
    let fitness = 0
    let dNew = |d_m − d_new|
    for i = 1 to CI                                  // every + is modulo hash table size
      if not IsObsolete(hashTable[h + i]) then
        dOld = |d_m − hashTable[h + i].distance|
        if dNew > dOld and fitness < dNew − dOld then
          finalH = h + i
          fitness = dNew − dOld                      // the greater fitness, the better
      else                                           // obsolete entry found
        finalH = h + i, fitness = 1
        break for
    if fitness = 0 then return }                     // do not replace (new distance is bad)
  set hashTable[finalH] = [minId, maxId, d_new] }    // finally, replace the entry
```

3.4 Filtering by D-Cache

The D-cache can be widely used with any metric access method. In particular, MAMs index data either in ball-shaped metric regions (e.g., (m)vp-tree, (P)M-tree, D-index) or in

1. GetUpperBoundDistance is similar, but the value is initialized to ∞ and updated as value $= min($value,**GetDistance**$(p, o_i) + \delta(r, p))$.

2. The calculation of percentile distances d_m is obtained for free during the indexing phase of a MAM, using the distance distribution histogram.

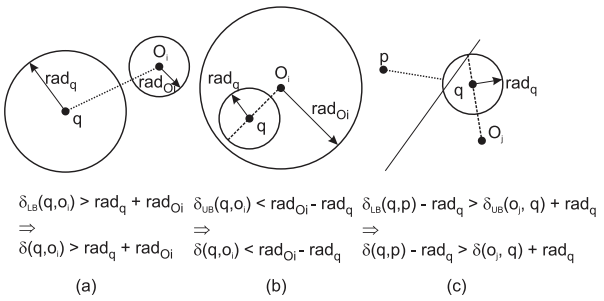


Fig. 3. (a) Ball-ball overlap. (b) Hole-ball containment. (c) Halfspace-ball overlap.

Voronoi-based regions (e.g., gh-tree, GNAT). Hence, there are basically three low-level *filtering predicates* used by MAMs to answer a query (or to insert new database object)—two predicates for ball-shaped and one for Voronoi-based regions. The most common queries (range and kNN) have also the shape of a ball.

When employing D-cache, the MAMs' filtering predicates can be weakened to be used with lower/upper bound distances inferred from D-cache, instead of computing an exact δ distance. Generally, the predicates can be weakened such that any form $\delta(\cdot,\cdot) + \cdots <$ is turned into $\delta_{UB}(\cdot,\cdot) + \cdots <$ and any $\delta(\cdot,\cdot) - \cdots >$ into $\delta_{LB}(\cdot,\cdot) - \cdots >$. This adjustment is correct, since it underestimates the filtering hits, that is, weakened form implies the original one, but not vice versa (see Fig. 3).

3.4.1 Filtering of Ball-Shaped Regions

The ball-shaped regions are generally of two kinds—a simple ball and/or a ring.

(A) Ball Data Regions

Querying. Having a query ball (q, rad_q) and a ball-shaped data region (o_i, rad_{o_i}), the data region can be excluded (filtered) from the search if the two balls do not overlap, that is, in case that predicate

$$\delta(q, o_i) > rad_q + rad_{o_i}, \tag{1}$$

is true (see Fig. 3a). Note that this simple predicate applies also on filtering database objects themselves (rather than regions), considering just DB object o_i, i.e., $rad_{o_i} = 0$.

Indexing. Let us now consider q as a new object to be inserted and $rad_q = 0$. Then the predicate (1) can be used to filter a data region which cannot cover the new object (without enlarging the radius rad_{o_i}).

D-cache usage. Prior to an application of predicate (1), a MAM could use its weakened form

$$\delta_{LB}(q, o_i) > rad_q + rad_{o_i}. \tag{2}$$

Since D-cache provides the lower bound $\delta_{LB}(q, o_i)$ for free, the index region (o_i, rad_{o_i}) could be filtered out by predicate (2) without the need of computing $\delta(q, o_i)$ otherwise required to apply predicate (1).

(B) Ring Data Regions

Some MAMs ((m)vp-tree, (P)M-tree, D-index) combine two balls to form a ring, which is a pair of two concentric balls—the smaller one is regarded as a hole in the bigger. In order to determine an overlap with query ball (or inserted object), the predicate (1) alone cannot be used to determine that a query ball is entirely inside the hole. Hence, we use predicate

$$\delta(q, o_i) < rad_{o_i} - rad_q, \tag{3}$$

to determine whether the query ball is entirely inside the hole (see Fig. 3b). A query ball is not overlapped by a ring region in case (1) is true for the bigger ball or (3) is true for the smaller ball (hole). For insertion of a new database object the predicates (1) and (3) are used in a similar way.

D-cache usage. Prior to an application of predicate (3), a MAM could use its weakened form

$$\delta_{UB}(q, o_i) < rad_{o_i} - rad_q. \tag{4}$$

3.4.2 Filtering of Voronoi-Based Regions

Several MAMs (gh-tree, GNAT, M-index) partition the metric space by use of a border composed of "Voronoi hyperplanes." Given m pivot objects, the border is formed by all such points of the universe, which are equally distant to two of the pivot objects and farther from the rest of objects.

A region assigned to pivot object p does not overlap a query region (q, rad_q) if the following predicate is true

$$\forall o_j : \delta(q, p) - rad_q > \delta(q, o_j) + rad_q, \tag{5}$$

where $\forall o_j$ are the remaining pivot objects (see Fig. 3c).

D-cache usage. Prior to an application of predicate (5), a MAM could use its weakened form

$$\forall o_j : \delta_{LB}(q, p) - rad_q > \delta_{UB}(q, o_j) + rad_q. \tag{6}$$

3.5 Use of D-Cache in Approximate Similarity Search

In addition to exact search by MAMs, the D-cache may also be used to improve the efficiency of approximate similarity search techniques [22]. In these techniques, the search algorithm saves search cost at the cost of possibly not retrieving the exact answer (i.e., all relevant objects for the given query). That is, they provide a tradeoff between the efficiency and the effectiveness of the similarity search.

Similarly to search algorithms in MAMs, approximate algorithms can take advantage of lower or upper bound distance estimations to avoid distance computations. For example, the probabilistic incremental search approach [23] fixes a number of distance computations to be performed by the search algorithm. Once a distance is computed, the algorithm decides if it must continue searching or not in a particular branch of the search hierarchy. By using D-cache, the discarding process could be done without actually computing that distance (using the returned lower bound distance), thus saving it for further searching in the hierarchy. This will improve the effectiveness of the search, as more branches of the hierarchy will be visited.

3.6 Analysis of D-Cache Performance

A fast implementation of D-cache functionality is crucial for its efficient employment by MAMs. Specifically, this requirement applies to the function GetLowerBoundDistance and method InsertDistance due to their frequent use during

querying/indexing. A D-cache-enhanced MAM would be faster in real time only in case the D-cache overhead would not be dominant. In particular, employing computationally expensive distance functions δ promises the speed up in real time will approach the reduction in distance computations (which is the theoretical speed-up maximum). Although the mentioned functions do not compute even a single distance δ, for an improper parameterization their real-time overhead might be significant. First of all, the overall cost of GetLowerBoundDistance and InsertDistance is proportional to the number of dynamic pivots k. Thus, to obtain effective usage of D-cache, k must be reasonably small. Second, the size of collision interval can heavily affect the D-cache performance, because sequential processing of the collision interval affects the real-time cost linearly. Although a large collision interval usually leads to better replacement of distances, the resulting heavy slowdown may not be a good tradeoff. Third, the hashing function is called frequently in GetLowerBoundDistance, so it should be as fast as possible but, at the same time, providing good enough distribution of keys.

In the experimental evaluation, we present different settings affecting the discussed performance issues. Basically, for smaller D-cache the collision interval of size 1 and simple hashing is the best, while for larger D-cache the interval of size 5 and universal hashing is slightly better. Regarding the dynamic pivots, their optimal number is heavily dependent on the database settings, while in our experiments it turns out that several tens to a few hundred pivots perform the best.

4 Enhancing MAMs by D-Cache

In this section, we discuss the modifications of three MAMs that take advantage of D-cache for both querying and indexing.

4.1 Enhancing Sequential Search—the D-File

Although not a proper MAM, the sequential search over the database can be enhanced by D-cache to speed the search. In Algorithms 3 and 4 see the adjusted range and kNN query.

Algorithm 3: (D-file range query)

```
set ScanRangeQuery(q, rad_q) {
  Dcache.StartRuntimeProcessing(q)
  for each o_i in database do
    if Dcache.GetLowerBoundDistance(q, o_i) ≤ rad_q then      // D-cache filt.
      compute δ(q, o_i); Dcache.InsertDistance(q, o_i, δ(q, o_i))
      if δ(q, o_i) ≤ rad_q then add o_i to the query result }   // basic filtering
```

Algorithm 4: (D-file kNN query)

```
set kNNQuery(q, k) {
  Dcache.StartRuntimeProcessing(q)
  let NN be array of k pairs [o_i, δ(q, o_i)] sorted asc. wrt δ(q, o_i),
                                  initialized to NN = [[−, ∞], ..., [−, ∞]]
  let rad_Q denotes the actual distance component in NN[k]
  for each o_i in database do
    if Dcache.GetLowerBoundDistance(q, o_i) ≤ rad_q then  // D-cache filtering
      compute δ(q, o_i); Dcache.InsertDistance(q, o_i, δ(q, o_i))
      if δ(q, o_i) ≤ rad_q then insert [o_i, δ(q, o_i)] into NN   // basic filtering
  return NN as result }
```

We have to emphasize that the D-cache together with sequential search could be used as a standalone metric access method that requires no indexing at all. We call the enhanced sequential search as the *D-file*, introduced recently in its preliminary version as a tool for index-free similarity search [10]. Hence, it could be used in situations where indexing is not possible or too expensive. Generally, any form of indexing requires at least linear time to construct an index for a database (but typically more, e.g., $O(n \log n)$ or $O(n^2)$). Thus, indexing is beneficial just in case we assume many queries, so the indexing cost will be amortized by the overall decreased query cost.

4.1.1 Motivation for Index-Free Similarity Search

In some scenarios, the indexing (and even dynamic updates) represents an obstacle. In the following, we briefly discuss three such scenarios.

"Changeable" databases. In many applications, we encounter databases intensively changing over time, like streaming databases, archives, logs, temporal databases, where new data arrive and old data are discarded frequently. Alternatively, we can view any database as "changeable" if the proportion of changes to the database exceeds the number of query requests. In highly changeable databases, the indexing efforts lose their impact, since the expensive indexing is compensated by just a few efficient queries. In the extreme case (e.g., sensory-generated data), the database could have to be massively updated in real time, so that any indexing is unpractical.

Isolated searches. In complex tasks, e.g., in data mining, a similarity query over a single-purpose database is used just as an isolated operation in the chain of all required operations to be performed. In such case, the database might be established for a single or several queries and then discarded. Hence, index-based methods cannot be used, because, in terms of the overall costs (indexing+querying), the simple sequential search would perform better.

Arbitrary similarity function. Sometimes the similarity measure is not defined a priori and/or can change over the time. This includes learning, user-defined, or query-defined similarity. In such case, any indexing would lead to many different indexes, or is not possible at all.

To address the three scenarios, the D-file, as the "founding father" of index-free MAMs, could be the solution. We also emphasize that D-file should not be viewed as an index-based MAM that just maintains its temporary index in main memory. We see the difference between *index-based* and *index-free* methods not only in the main-memory organization, but mainly in the fragmentation of "indexing." While index-based MAMs cannot search the database before the indexing is finished, the index-free MAMs are allowed to search the database instantly. Moreover, as the "indexing step" (updating the D-cache) is performed during the query, the indexing versus query efficiency tradeoff remains balanced at any time.

4.2 Enhancing Pivot Tables

When considering range queries in Pivot tables, like LAESA, we have to discuss two steps (for details see Section 2.2). First, there is filtering by pivots[3] performed, where a query vector is computed, a query box is

3. Here, we consider regular (static) pivots of Pivots tables. Please, do not confuse pivot tables' (static) pivots with D-cache's *dynamic* pivots.

established, and all the distance matrix rows are checked if they fall inside the query box. If not, these objects are filtered out from further processing, while the nonfiltered objects are processed in the second (refinement) step by usual sequential search.

In the first step (filtering by pivots), the only moment of computing δ is the construction of query vector. Then, the pivot table is checked against the query box which does not require any distance computation. Hence, the D-cache is not needed in the first step. On the other hand, it could be utilized in the second (refinement) step when sequentially searching the nonfiltered candidate objects. In fact, we can view the set of nonfiltered objects as a (small) sequential file, where the D-cache could be utilized the same way as in the D-file.

Since the distance matrix consists of exact distance values that are not repeating, the D-cache cannot be used for indexing. We implemented the D-cache-enhanced querying into Pivot tables and called the new MAM as *D-Pivot Tables* (or *D-PT*).

4.3 Enhancing M-Tree

In M-tree, the cheap filtering step based on D-cache is placed between the *parent filtering* (also cheap) and the *basic filtering* (expensive), see Algorithm 5.

Algorithm 5: (D-M-tree range query)

```
D-MtreeRangeQuery(Node N, RQuery (q, rad_q)) {
  let rout(p) be the parent routing entry of N
                                  // if N is root then let δ(o_i, p)=δ(p, q)=0
  if N is root then Dcache.StartRuntimeProcessing(q)
  if N is not a leaf then {
    for each rout(o_i) in N do
      if |δ(p, q) − δ(o_i, p)| ≤ rad_q + rad_{o_i} then        // parent filtering
                                                               // D-cache filtering
        if Dcache.GetLowerBoundDistance(q, o_i) ≤ rad_q + rad_{o_i} then
          compute δ(q, o_i); Dcache.InsertDistance(q, o_i, δ(q, o_i))
          if δ(o_i, q) ≤ rad_q + rad_{o_i} then                // basic filtering
            D-MtreeRangeQuery(ptr(T(o_i)), (q, rad_q))
  } else {
    for each grnd(o_i) in N do
      if |δ(p, q) − δ(o_i, p)| ≤ rad_q then                    // parent filtering
                                                               // D-cache filtering
        if Dcache.GetLowerBoundDistance(q, o_i) ≤ rad_q then
          compute δ(q, o_i); Dcache.InsertDistance(q, o_i, δ(q, o_i))
          if δ(o_i, q) ≤ rad_q then                            // basic filtering
            add o_i to the query result } }
```

Furthermore, the D-cache can be used also to speed up the construction of M-tree, where we use both the exact retrieval of distances (method GetDistance) and also the lower-bounding functionality. The node splitting in M-tree often uses the expensive `mM_RAD` heuristics, where a distance matrix is computed for all pairs of node entries. The values of this matrix can be stored in D-cache and some of them reused later, when node splitting is performed on the child nodes of the previously split node. Similarly, when using forced reinsertions, the distances related to the inserted objects reside in the D-cache and could be used when reinserting some of the objects in the future. Moreover, when employing the expensive multiway insertion, the D-cache could be used also in the "nonexact" way (using lower bounds similarly as by querying). We implemented the D-cache-enhanced indexing+querying into M-tree and called the new MAM as *D-M-tree*.

5 Related Work

To the best of our knowledge, there are no other approaches to distance caching for general MAMs. The most similar proposed approaches are in the line of bulk loading for batch insertion or processing multiple queries at once. However, in this case the data/queries need to be available beforehand, i.e., we cannot consider a continuous stream of queries or insertions. While in the literature it has been proposed the use of different kinds of "caching" in the context of multidimensional databases (e.g., caching in distributed systems [24], or a "L2 cache conscious" main-memory multidimensional index [25]), they do not take advantage of the computed distances at query time to speed up (future) similarity queries.

5.1 Caching Distances in M-Tree

One idea that actually uses cached distances for range queries with an M-tree was proposed by Kailing et al. [26]. For each query, the distances computed from each routing object to the query are cached. In this way, if there are duplicated routing objects at different levels of the M-tree, their distance to the query object will be computed only once. As the memory cost of saving these distances is proportional to the height of the tree, the extra space needed is "tolerable" [26]. However, the computed distances at each node are deleted from the cache once the recursive search function leaves a node. Therefore, no distances are saved for future queries, and they are only useful if the cached distance between the same two objects needs to be computed again. Moreover, as duplicate routing objects are rare in M-tree (with respect to all examined objects), the savings in distance computations are rather negligible.

5.2 Batch Indexing and Querying

The basic idea of bulk loading is to create the index from scratch but knowing beforehand the database, thus some optimizations may be performed to obtain a "good" index for that database. Usually, the proposed bulk loading techniques are designed for specific index structures, but there have been proposals for more general algorithms. For example, in [27] the authors propose two generic algorithms for bulk loading, which were tested with different index structures, like the R-tree and the Slim-tree. Note that the efficiency of the index may degrade if new objects are inserted after its construction. Recently, a parallel approach to insertion of a batch of objects was proposed for the M-tree [28].

Another approach for improving the efficiency of MAMs is the simultaneous processing of multiple queries [29]. Instead of issuing many single queries, the idea is to process a batch of similarity queries aiming at reducing I/O cost and computation cost. The proposed technique reduces the I/O cost by reading each disk page only once per batch of similarity queries, and it reduces the CPU cost by avoiding distance computations. An avoidable distance computation is detected by computing the distances between query objects and then using these distances, together with the triangle inequality, to compute lower bounds of the distances between queries and database objects. If the lower bound distance is greater than a given tolerance radius for the similarity search, then the distance calculation is

avoidable. The proposed technique is general, and it can be implemented based on a MAM or using a sequential file. However, besides the requirement to know all the queries beforehand, it also requires computing the distances between each pair of query objects to reduce the CPU cost, and it does not take advantage of distance computations between queries and database objects.

In [30] an approach to compute the k nearest neighbor graph in metric spaces was proposed, which is equivalent to compute n kNN queries, in (empirical) subquadratic time. However, the set of query objects was restricted to the database objects, which is only marginally meaningful in our framework.

5.3 Query Result Caching

In order to speed up the similarity searches, a recent approach provides a mechanism of caching query results [31], [32]. Basically, the *metric cache* stores a history of similarity queries and their answers (ids and descriptors of database objects returned by the query). When a next query is to be processed, the metric cache either returns the exact answer in case the same query was processed in the past and its result still sits in the cache. Or, in case of a new query, such old queries are determined from the metric cache, that spatially contain the new query object inside their query balls. If the new query is entirely bounded by a cached query ball, a subset of the cached query result is returned as an exact answer of the new query. If not, the metric cache is used to combine the query results of spatially close cached queries to form an approximate answer. In case the approximate answer is likely to exhibit a large retrieval error, the metric cache gives up and forwards the query processing to the underlying retrieval system/MAM (updating the metric cache by the query answer afterward). We emphasize that metric cache is a higher level concept that can be combined with any MAM employed in a search engine. Hence, metric cache is just a standalone front-end part in the whole retrieval system, while the underlying MAM alone is not aware of the metric cache at all. On the other hand, the following proposal of D-cache is a low-level concept that plays the role of integral part of a metric access method (that has to be adjusted to use D-cache functionality). Nevertheless, both approaches could be combined in the future (i.e., the metric cache in front of D-cache-enhanced MAMs).

6 Experimental Evaluation

We have extensively tested the D-cache-enhanced MAMs (D-file, D-M-tree, and D-Pivot tables) and their noncached counterparts (sequential search, M-tree, Pivot tables) to examine the expected performance gain in indexing and querying. A substantial attention in the experiments was given to the D-file, which in some cases outperformed the index-based MAMs. We have observed both the number of distance computations as well as the real time spent by indexing/querying.

6.1 The Testbed

In order to examine the D-cache in very different conditions, we used four databases (two vector spaces, one string space, and one set space) and four metric distances (three expensive and one cheap), as follows:

1. A part of the *CoPhIR* [33] database (descriptors of selected images from Flickr.com), namely, one million 282-dimensional vectors (representing five MPEG7 features), and the euclidean distance as the similarity function (i.e., time complexity $O(n)$).
2. A database of *Histograms* (descriptors of images downloaded from Flickr.com, but different to CoPhIR), namely, one million 512-dimensional histograms, and the quadratic form distance [34] as the similarity function (i.e., complexity $O(n^2)$). As the image representation we used the standard RGB histogram of dimensionality 512, where the R, G, B components were divided in 8 bins each, thus $8*8*8 = 512$ bins. Each histogram was normalized to have the sum equal to 1, while the value of each bin was stored in a float. The similarity matrix used for the quadratic form distance was computed as described in [34], using similarity of colors in the CIE Lab color space [35].
3. The *Listeria* [36] database, namely 20,000 DNA sequences of Listeria monocytogenes of lengths 200-7,000, and the edit distance [37] as the similarity (i.e., complexity $O(n^2)$).
4. A synthetic *Clouds* database [38], namely 100,000 clouds (sets) of 60 6D points (embedded in a unitary 6D cube). For each cloud, its center was generated at random, while the 59 remaining points were generated under normal distribution around the center (the mean and variance in each dimension were adjusted to not generate points outside the unitary cube). Usually, clouds of points are used for simplified representations of complex objects or objects consisting of multiple observations [39]. As an appropriate distance metric, we used the symmetric Hausdorff distance [40] for measuring similarity between sets (maximum distance between a point in one cloud to the nearest point in the other cloud). We used the euclidean distance as the internal point-to-point distance within the Hausdorff distance (hence, leading to overall complexity $O(n^2)$). The Clouds database was included into the experiments in order to examine a nonvectorial alternative to the usual synthetic database of normally distributed vectors.

In experiments where the growing database size was considered, the particular database subsets were sampled from the respective largest database at random. In the other experiments, we used the entire databases in the case of Clouds and Listeria, and random subsets of size 100,000 in the case of CoPhIR and Histograms. Unless otherwise stated, each query cost was an average over 500 queries using query objects not included in the database.

6.1.1 Database Indexability

As the fundamental assumption on metric access methods is their universal applicability on various kinds of data, the experimental databases were chosen to represent very different metric spaces. In addition to employing cheap

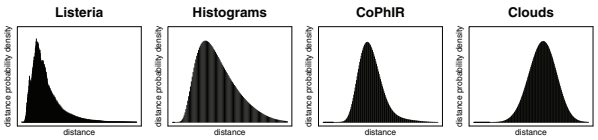


Fig. 4. Distance distribution in the databases.

(euclidean) and expensive (quadratic form, edit, Hausdorff) distance functions, the databases also exhibited different *intrinsic dimensionalities*. The intrinsic dimensionality [7] is a concept generalizing the phenomenon of the *curse of dimensionality* into metric spaces, and is defined as $\rho(\mathbf{S}, \delta) = \frac{\mu^2}{2\sigma^2}$, where μ and σ^2 are the mean and the variance of the distance distribution in the database. Informally, a database where most of the objects are far away from each other exhibits high intrinsic dimensionality and so it is hard to index by any MAM. Conversely, a database where some of the objects are close and some are distant exhibits a low intrinsic dimensionality (i.e., there exist distinct clusters). In this case, the MAMs are able to better separate the data, thus performing similarity queries in an efficient way.

The Clouds, CoPhIR, and Histograms databases exhibited high intrinsic dimensionalities (11.64, 7.5, 7.56, respectively), and the Listeria database exhibited low intrinsic dimensionality (1.19). Fig. 4 shows the distance distribution on each particular database, where a wide and left-shifted "bell" means lower intrinsic dimensionality, and vice versa. Since the intrinsically low-dimensional databases were already efficiently indexed by the MAMs not enhanced by D-cache, there was not as much room to improve the indexing/search by the D-cache as in the case of the high-dimensional databases.

6.1.2 MAM Settings

The M-tree, Pivot Tables, and sequential scan were tested against their D-cache enhanced versions on all the databases. For (D-)M-tree, the node degree was 25 in leaf nodes and 24 in inner nodes, while for its construction the `mM_RAD` node splitting [17] and various object insertion policies were employed [18]. The static pivots of (D-)Pivot Tables were selected from the respective database at random.

6.1.3 D-Cache Settings

Unless otherwise stated, the D-cache used 1,280,000 entries (i.e., 19.5 MB of main memory) and 160 dynamic pivots for the CoPhIR, Histograms, and Clouds databases, and it used 64,000 entries (i.e., 1 MB of main memory) and 50 dynamic pivots for the Listeria database. When using the Obsolete-Percentile replacement policy, the percentile was set to 36 percent for Clouds, 4 percent for Listeria, 15 percent for Histograms, and 50 percent for CoPhIR (these values were observed as optimal, as discussed later). The D-cache was reset/initialized before every query batch was started.

Table 1 describes the labels of particular MAM and D-cache configurations used in the following figures.

6.2 Indexing

Table 2 presents the index construction times for MAMs not employing D-cache. As the sequential search and the D-file are index-free methods, they were not included in the indexing experiments.

TABLE 1
Labels Used in the Figures

Label	Description
M-tree_SW	M-tree built using single-way insertion [18]
M-tree	the same as M-tree_SW
M-tree_SW_RI	M-tree built using single-way insertion + forced reinsertions [18]
M-tree_MW	M-tree built using multi-way insertion [18]
M-tree_MW_RI	M-tree built using multi-way insertion + forced reinsertions [18]
PT_x	Pivot Tables using x static pivots
D-mam	a particular MAM enhanced by D-cache (see Section 4)
Obs(cfg)	D-cache's Obsolete replacing policy (see Section 3.3.2))
ObsPct(cfg)	D-cache's ObsoletePercentile policy (see Section 3.3.2)
	cfg:
	CI=x: D-cache's collision interval (see Section 3.2), default is CI=5
	H=Simple/Universal: D-cache's hashing function (see Sec. 3.2.1), default is H=Universal
	Dc.size=x: size of D-cache in the number of distance entries
DB(x)	database containing x objects

The index construction times for M-tree and D-M-tree are presented in Fig. 5, showing the single-way leaf selection variants (left figure) and multiway leaf selection variants (right figure). The results show that larger D-cache considerably speeds the M-tree construction (up to $1.7\times$). Both of the D-M-tree variants use the GetDistance method for indexing. Since the multiway leaf selection technique issues a point query, it is reasonable to use also the GetLowerBoundDistance in the D-M-tree_MW variant. Also note that the D-M-tree_SW_RI that uses extra forced reinsertions is even faster than simple M-tree_SW.

6.3 Queries

The largest set of experiments was focused on kNN queries under different D-cache and retrieval settings. Unless otherwise stated, on databases that employed expensive distances we present just the real times for queries, because the numbers of distance computations followed exactly the same proportion. In other words, when queried by the expensive distance metrics, the real time spent outside the code computing distances was negligible. Also note that because the query objects were outside the database (i.e., unknown to D-cache), the speedup achieved by D-cache was solely based on the lower bounding functionality (see Section 3.3).

TABLE 2
Index Construction Times (D-Cache Not Used)

Database	MAM	Indexing time (seconds)
Clouds of points (100k)	PT_10	248.03
	M-tree	1509.17
Histograms (100k)	PT_10	307.23
	M-tree	1724.84
Listeria (20k)	PT_10	1277.88
	M-tree	13050.98
CoPhIR (1M)	PT_10	88.75
	M-tree	335.89

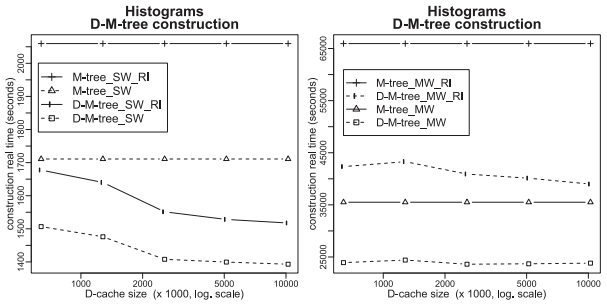


Fig. 5. M-tree and D-M-tree construction, using single-way (left figure) and multiway (right figure) leaf selection.

Table 3 shows the baseline real times when searching a database sequentially, regardless of the query selectivity. The results confirm that the euclidean distance (used on CoPhIR) is very efficient, while the edit distance used on the long Listeria sequences is very expensive.

6.3.1 Database Size

The first querying experiment was focused on the growing database size while fixing the size of the D-cache used (see Fig. 6). We observe that for small databases there is enough space in D-cache, so that distance replacements are not often needed. However, for larger databases the D-cache gets filled and the distance replacements are necessary. In such case, for D-file the ObsPct replacement policy turns out to be more effective for replacing "bad" distances, which results in a better filtering and so in a faster query processing. On the other hand, for the same D-cache size but the D-M-tree, the filling of D-cache with distances is slower (because of more aggressive filtering), so distance replacements are not often.

6.3.2 Percentile Distances

In the second experiment, we have investigated the percentile distances that optimize the replacement of the least useful distances in the D-cache, see Fig. 7. Since the Clouds database exhibits large intrinsic dimensionality, the proportion of possibly "bad" distances that cannot be used for effective lower-bound filtering is larger than in the Histogram database. Hence, for Clouds database the ObsPct replacement policy that prevents from storing the bad distances leads to faster querying than the Obs policy. This effect is even magnified for smaller D-cache sizes (up to $2\times$ query speed up).

6.3.3 D-Cache Size

Next, we performed experiments with growing D-cache size for various replacement policies, collision intervals, and hashing functions (see Fig. 8, where the D-cache size is the number of distance entries allocated). Although for smaller D-cache sizes the different settings lead to slightly different querying performance, for larger D-cache sizes the differences are negligible. Nevertheless, the ObsPct($\mathrm{CI} = 5, \mathrm{H} = \mathrm{universal}$) policy performed well under all circumstances.

TABLE 3
Real Times of Sequential Search

Database	Query time (seconds)
Histograms (100k)	29.03
Clouds (100k)	21.10
Listeria (20k)	134.53
CoPhIR (1M)	4.60

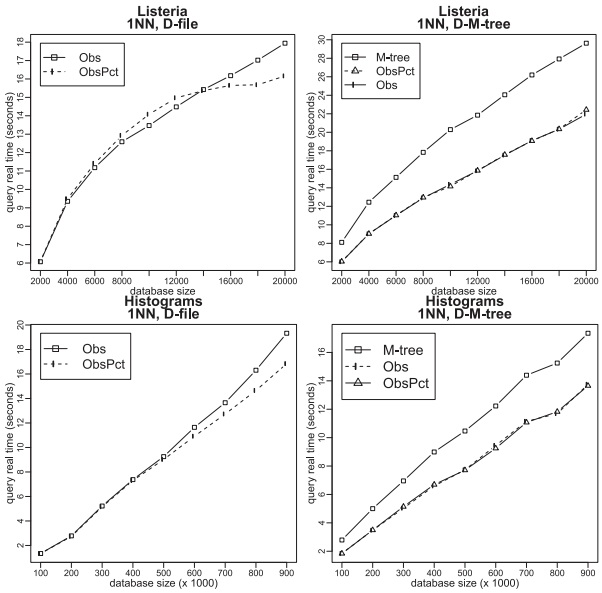


Fig. 6. 1NN queries on growing databases.

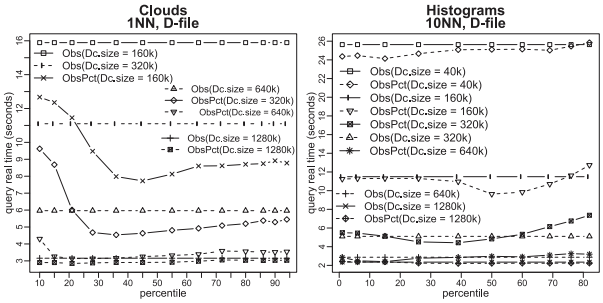


Fig. 7. Impact of various percentile distances used by ObsPct replacement heuristics.

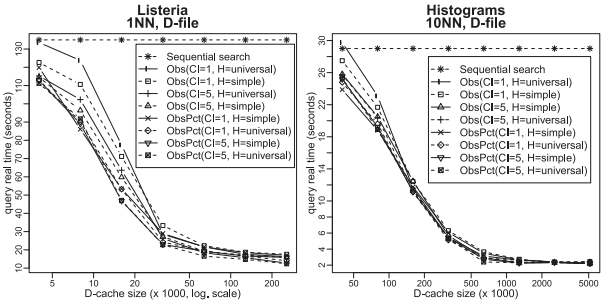


Fig. 8. Impact of D-cache size on distance replacement.

6.3.4 Number of Dynamic Pivots

Fig. 9 shows the impact of increasing number of dynamic pivots used by D-cache. Instead of the usual 500 queries, we present the averaged results over 1,000 queries for Clouds and 2,500 queries for Histograms, in order to justify the

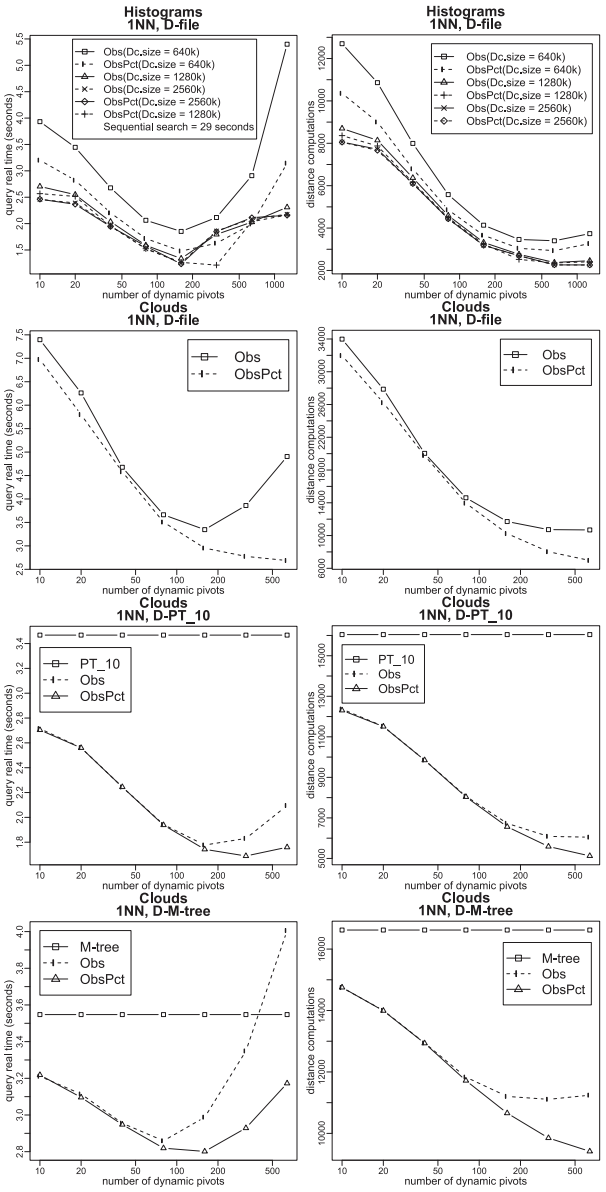


Fig. 9. Impact of the growing number of dynamic pivots.

larger numbers of dynamic pivots. Also note that in this experiment, we present both the real time and the number of distance computations.

The superiority of ObsPct replacement policy is here confirmed. For a large number of dynamic pivots and when replacing an entry in D-cache, the likelihood that the collision interval contains an obsolete entry will be low because most of the past runtime objects are still dynamic pivots. In such a case when no obsolete entry is available, the Obs policy just replaces the first entry found in the collision interval. On the other hand, the ObsPct policy replaces the least "useful" entry in the interval, based on the selected percentile distance.

To mention also the negative results, with a growing number of dynamic pivots the time complexity of the methods GetLowerBoundDistance and InsertDistance increases, resulting in increased real time spend for

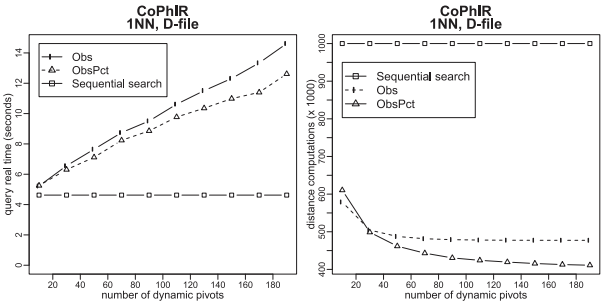


Fig. 10. Negative results on the CoPhIR database.

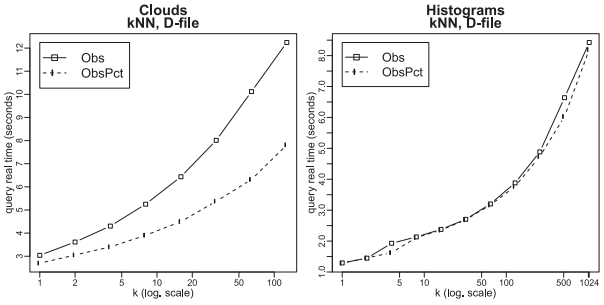


Fig. 11. kNN queries.

querying. The overhead of the mentioned methods is even more visible when using cheap metric distances. To show a clear fail of D-cache, we present results for the CoPhIR database searched under the euclidean distance. Observe in Fig. 10, the discrepancy between the real time and the number of distance computations spent for query processing.

6.3.5 kNN Queries

The next experiment investigated the performance of kNN queries on D-file, see Fig. 11. Since for larger k the number of distance computations spent by querying increases, the D-cache size becomes insufficient. This leads to less effective query processing.

6.3.6 Number of Objects in the Query Batch

In the last test we examined the "warming" of D-cache, that is, how many queries are needed to populate the D-cache to be useful enough for filtering. Fig. 12 shows the impact of the growing query batch size, that is, the average cost of a 10NN query when running queries in differently sized batches (each query batch runs as a single D-cache session).

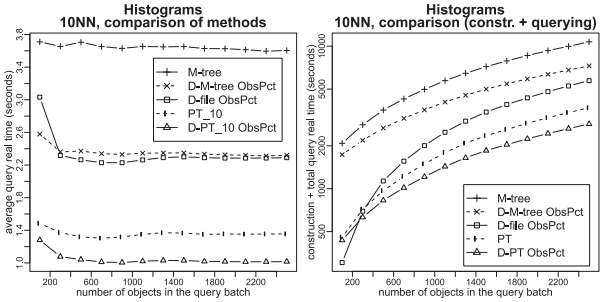


Fig. 12. Impact of the growing volume of query batch.

The trend is obvious: the more queries, the more distances get into the D-cache which the subsequent queries can benefit from. The difference is quite significant for the D-file: the average cost of a query within a 2,500 queries batch falls down to 70 percent of the average query cost within a 700 queries batch. Moreover, in the right graph of Fig. 12 see the total query cost (not the average as usual) for differently sized batches of queries, including also the indexing cost. This test aims to show the overall cost when searching a database for a limited number of queries. Obviously, when only a small number of queries are needed, say up to 300, the D-file is the clear winner because of its index-free concept. On the other hand, when reaching a sufficiently large number of queries, the index-based MAMs begin to amortize the huge initial indexing cost by the efficient query processing (but the D-file still keeps up with them). In case of MAMs employing D-cache, the amortization is quicker.

6.4 Summary

We have shown that the D-cache accelerates the indexing of M-tree significantly. When querying, the D-cached-enhanced MAMs perform up to two times faster than their noncached counterparts (up to 24 times in case of D-file).

A special attention should be devoted to the D-file, which is not only the first index-free MAM, but it can compete with efficient competitors like the Pivot tables or the M-tree.

The D-cache proved its benefits in most of the experiments. On the other hand, when a cheap metric distance is used, the overhead of D-cache is too large. Regarding the D-cache tuning, we observed that the ObsoletePercentile replacement strategy works the best in most of the cases, while the number of dynamic pivots ranging from tens to a few hundreds is sufficient. When considering the D-file, the size of employed D-cache should be proportional to the database size (e.g., 10 percent) in order to achieve an optimal performance.

7 Conclusions

In this paper we presented the D-cache, a main-memory data structure which tracks computed distances while inserting objects or performing similarity queries in the metric space model. Since distance computations stored in the D-cache may be reused in further database operations, it is not necessary to compute them again. Also, the D-cache can be used to estimate distance functions between new objects and objects stored in the database, which can also avoid expensive distance computations. The D-cache aims to amortize the number of distance computations spent by querying/updating the database, similarly like disk page buffering in traditional DBMSs aims to amortize the I/O cost.

The D-cache structure is based on a hash table, thus making efficient to retrieve stored distances for further usage. Additionally, the D-cache maintains the set of previously processed runtime objects (i.e., inserted or query objects), while the most recent of them are used as dynamic pivots. The D-cache supports three functions useful for metric access methods (MAMs)—the GetDistance (returning the exact distance between two objects, if available), the GetLowerBoundDistance (returning the greatest lower-bound distance between two objects, by means of the dynamic pivots), and the GetUpperBoundDistance (returning the lowest upper-bound distance). With these functions, the D-cache may be used to improve the construction of MAMs' index structures and the performance of similarity queries.

Our depiction of the D-cache is general, and may be used with any metric access method or even to aid a sequential scan of the database—forming a brand new concept of index-free MAM, the D-file. We have presented replacement policies for the distances stored in the cache as well as algorithms for the computation of the lower- and upper-bound distances. We have also described in detail how to enhance some of the existing metric access methods (M-tree, Pivot tables) with the D-cache.

Finally, we presented the results of an experimental evaluation with different databases using expensive and cheap metric distance functions. When considering expensive enough distance functions ($\geq O(n^2)$), the D-cache substantially improves the real times needed to query/update metric databases.

Acknowledgments

This research has been supported by Czech Science Foundation grants GAČR 201/09/0683, 202/11/0968 (first and second author), and by FONDECYT (Chile) Project 11070037 (third author).

References

[1] J.S. Vitter, "External Memory Algorithms and Data Structures: Dealing with Massive Data," *ACM Computing Surveys*, vol. 33, no. 2, pp. 209-271, citeseer.ist.psu.edu/vitter01external.html, 2001.

[2] C. Böhm, S. Berchtold, and D. Keim, "Searching in High-Dimensional Spaces—Index Structures for Improving the Performance of Multimedia Databases," *ACM Computing Surveys*, vol. 33, no. 3, pp. 322-373, 2001.

[3] S.D. Carson, "A System for Adaptive Disk Rearrangement," *Software—Practice and Experience*, vol. 20, no. 3, pp. 225-242, 1990.

[4] W. Effelsberg and T. Haerder, "Principles of Database Buffer Management," *ACM Trans. Database Systems*, vol. 9, no. 4, pp. 560-595, 1984.

[5] M. Batko, D. Novak, F. Falchi, and P. Zezula, "Scalability Comparison of Peer-to-Peer Similarity Search Structures," *Future Generation Computer Systems*, vol. 24, no. 8, pp. 834-848, 2008.

[6] P. Zezula, G. Amato, V. Dohnal, and M. Batko, *Similarity Search: The Metric Space Approach (Advances in Database Systems)*. Springer, 2005.

[7] E. Chávez, G. Navarro, R. Baeza-Yates, and J.L. Marroquín, "Searching in Metric Spaces," *ACM Computing Surveys*, vol. 33, no. 3, pp. 273-321, 2001.

[8] G.R. Hjaltason and H. Samet, "Index-Driven Similarity Search in Metric Spaces," *ACM Trans. Database Systems*, vol. 28, no. 4, pp. 517-580, 2003.

[9] H. Samet, *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[10] T. Skopal and B. Bustos, "On Index-Free Similarity Search in Metric Spaces," *Proc. 20th Int'l Conf. Database and Expert Systems Applications (DEXA '09)*, pp. 516-531, 2009.

[11] E. Vidal, "New Formulation and Improvements of the Nearest-Neighbour Approximating and Eliminating Search Algorithm (AESA)," *Pattern Recognition Letters*, vol. 15, no. 1, pp. 1-7, 1994.

[12] M.L. Micó, J. Oncina, and E. Vidal, "An Algorithm for Finding Nearest Neighbour in Constant Average Time with a Linear Space Complexity," *Proc. Int'l Conf. Pattern Recognition*, 1992.

[13] M.L. Micó, J. Oncina, and R.C. Carrasco, "A Fast Branch & Bound Nearest-Neighbour Classifier in Metric Spaces," *Pattern Recognition Letters*, vol. 17, no. 7, pp. 731-739, 1996.

[14] E. Chávez, J.L. Marroquín, and R. Baeza-Yates, "Spaghettis: An Array Based Algorithm for Similarity Queries in Metric Spaces," *Proc. String Processing and Information Retrieval Symp. & Int'l Workshop Groupware (SPIRE '99)*, p. 38, 1999.

[15] C. Traina Jr., R.F. Filho, A.J. Traina, M.R. Vieira, and C. Faloutsos, "The Omni-Family of All-Purpose Access Methods: A Simple and Effective Way to Make Similarity Search More Efficient," *The VLDB J.—The Int'l J. Very Large Data Bases,* vol. 16, no. 4, pp. 483-505, 2007.

[16] T. Skopal, "Pivoting M-Tree: A Metric Access Method for Efficient Similarity Search," *Proc. Dateso 2004 Ann. Int'l Workshop DAtabases, TExts, Specifications and Objects,* vol. 98, pp. 21-31, http://www.ceur-ws.org, 2004.

[17] P. Ciaccia, M. Patella, and P. Zezula, "M-Tree: An Efficient Access Method for Similarity Search in Metric Spaces," *Proc. 23rd Int'l Conf. Very Large Data Bases (VLDB '97),* pp. 426-435, 1997.

[18] T. Skopal and J. Lokoč, "New Dynamic Construction Techniques for M-Tree," *J. Discrete Algorithms,* vol. 7, no. 1, pp. 62-77, 2009.

[19] B. Bustos, G. Navarro, and E. Chávez, "Pivot Selection Techniques for Proximity Searching in Metric Spaces," *Pattern Recognition Letters,* vol. 24, no. 14, pp. 2357-2366, 2003.

[20] J. Venkateswaran, T. Kahveci, C. Jermaine, and D. Lachwani, "Reference-Based Indexing for Metric Spaces with Costly Distance Measures," *The VLDB J.—The Int'l J. Very Large Data Bases,* vol. 17, no. 5, pp. 1231-1251, 2008.

[21] J.L. Carter and M.N. Wegman, "Universal Classes of Hash Functions," *J. Computer and System Sciences,* vol. 18, no. 2, pp. 143-154, 1979.

[22] M. Patella and P. Ciaccia, "The Many Facets of Approximate Similarity Search," *Proc. First Int'l Workshop Similarity Search and Applications (SISAP '08),* pp. 10-21, 2008.

[23] B. Bustos and G. Navarro, "Probabilistic Proximity Search Algorithms Based on Compact Partitions," *J. Discrete Algorithms,* vol. 2, no. 1, pp. 115-134, 2004.

[24] B. Nam, H. Andrade, and A. Sussman, "Multiple Range Query Optimization with Distributed Cache Indexing," *Proc. ACM/IEEE Conf. High Performance Networking and Computing (SC '06),* p. 100, 2006.

[25] J.M. Shim, S.I. Song, Y.S. Min, and J.S. Yoo, "An Efficient Cache Conscious Multi-Dimensional Index Structure," *Proc. Int'l Conf. Computational Science and Its Applications (ICCSA '04),* vol. 4, pp. 869-876, 2004.

[26] K. Kailing, H.-P. Kriegel, and M. Pfeifle, and S. Schnauer, "Extending Metric Index Structures for Efficient Range Query Processing," *Knowledge and Information Systems,* vol. 10, no. 2, pp. 211-227, 2006.

[27] J.V. den Bercken and B. Seeger, "An Evaluation of Generic Bulk Loading Techniques," *Proc. 27th Int'l Conf. Very Large Data Bases (VLDB '01),* pp. 461-470, 2001.

[28] J. Lokoč, "Parallel Dynamic Batch Loading in the M-tree," *Proc. Second Int'l Workshop Similarity Search and Applications (SISAP '09),* pp. 117-123, 2009.

[29] B. Braunmüller, M. Ester, H.-P. Kriegel, and J. Sander, "Multiple Similarity Queries: A Basic DBMS Operation for Mining in Metric Databases," *IEEE Trans. Knowledge and Data Eng.,* vol. 13, no. 1, pp. 79-95, Jan./Feb. 2001.

[30] R. Paredes, E. Chávez, K. Figueroa, and G. Navarro, "Practical Construction of *k*-Nearest Neighbor Graphs in Metric Spaces," *Proc. Fifth Int'l Workshop Experimental Algorithms (WEA '06),* pp. 85-97, 2006.

[31] F. Falchi, C. Lucchese, S. Orlando, R. Perego, and F. Rabitti, "A Metric Cache for Similarity Search," *Proc. ACM Workshop Large-Scale Distributed Systems for Information Retrieval (LSDS-IR '08),* pp. 43-50, 2008.

[32] F. Falchi, C. Lucchese, S. Orlando, R. Perego, and F. Rabitti, "Caching Content-Based Queries for Robust and Efficient Image Retrieval," *Proc. 12th Int'l Conf. Extending Database Technology (EDBT '09),* pp. 780-790, 2009.

[33] P. Bolettieri, A. Esuli, F. Falchi, C. Lucchese, R. Perego, T. Piccioli, and F. Rabitti, "CoPhIR: A Test Collection for Content-Based Image Retrieval," *CoRR* abs/0905.4627v2, http://cophir. isti.cnr.it, 2009.

[34] J. Hafner, H.S. Sawhney, W. Equitz, M. Flickner, and W. Niblack, "Efficient Color Histogram Indexing for Quadratic Form Distance Functions," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 17, no. 7, pp. 729-736, July 1995.

[35] Y. Rubner, J. Puzicha, C. Tomasi, and J.M. Buhmann, "Empirical Evaluation of Dissimilarity Measures for Color and Texture," *Computer Vision Image Understanding,* vol. 84, no. 1, pp. 25-43, 2001.

[36] K. Figueroa, G. Navarro, and E. Chavez "Metric Spaces Library, http://www.sisap.org/library/metricSpaces/docs/manual.pdf," 2007.

[37] I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics-Doklady,* vol. 10, no. 8, pp. 707-710, 1966.

[38] J. Lokoč "Cloud of Points Generator, SIRET Research Group, http://siret.ms.mff.cuni.cz/projects/pointgenerator/," 2010.

[39] F. Mémoli and G. Sapiro, "Comparing Point Clouds," *Proc. Eurographics/ACM SIGGRAPH Symp. Geometry Processing (SGP '04),* pp. 32-40, 2004.

[40] D. Huttenlocher, G. Klanderman, and W. Rucklidge, "Comparing Images Using the Hausdorff Distance," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 15, no. 9, pp. 850-863, Sept. 1993.

Tomáš Skopal received the doctoral degree in computer science and applied mathematics from the Technical University of Ostrava, Czech Republic. He is an associate professor in the Department of Software Engineering at the Charles University in Prague, Faculty of Mathematics and Physics, Czech Republic. He is the head of the SIRET Research Group. His research interests include metric access methods, database indexing, multimedia databases, and similarity modeling. He is a member of the IEEE.

Jakub Lokoč received the doctoral degree in software systems from the Charles University in Prague, Czech Republic. He is a researcher in the Department of Software Engineering at the Charles University in Prague, Faculty of Mathematics and Physics, Czech Republic. His research interests include metric access methods, parallel database indexing, multimedia databases, and similarity modeling.

Benjamin Bustos received the doctoral degree in natural sciences from the University of Konstanz, Germany. He is an assistant professor in the Department of Computer Science, University of Chile. He is the head of the PRISMA Research Group. He leads research projects in the domains of multimedia retrieval, video copy detection, sketch-based image retrieval, and retrieval of handwritten documents. His interests include similarity search, multimedia retrieval, 3D object retrieval, and (non)metric indexing.

Chapter 6

Ptolemaic Access Methods: Challenging the Reign of the Metric Space Model

Magnus Lie Hetland
Tomáš Skopal
Jakub Lokoč
Christian Beecks

Published in the *Information Systems* journal, volume 38, Issue 7, pages 989–1006. Elsevier, October 2013. ISSN 0306-4379.
dx.doi.org/10.1016/j.is.2012.05.011

Impact Factor in 2013: 1.235
5-Year Impact Factor in 2013: 1.435

Information Systems 38 (2013) 989–1006

Contents lists available at SciVerse ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Ptolemaic access methods: Challenging the reign of the metric space model ☆

Magnus Lie Hetland [a], Tomáš Skopal [b,*], Jakub Lokoč [b], Christian Beecks [c]

[a] Department of Computer Science, Norwegian University of Science and Technology, Norway
[b] SIRET Research Group, Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic
[c] Data Management and Data Exploration Group, RWTH Aachen University, Germany

ARTICLE INFO

Available online 13 June 2012

Keywords:
Ptolemaic indexing
Metric space model
Similarity search
Database indexing
Signature quadratic form distance
Metric access methods
Content-based retrieval

ABSTRACT

Metric indexing is the state of the art in general distance-based retrieval. Relying on the triangular inequality, metric indexes achieve significant online speed-up beyond a linear scan. Recently, the idea of *Ptolemaic indexing* was introduced, which substitutes Ptolemy's inequality for the triangular one, potentially yielding higher efficiency for the distances where it applies. In this paper we have adapted several metric indexes to support Ptolemaic indexing, thus establishing a class of *Ptolemaic access methods* (PtoAM). In particular, we include Ptolemaic Pivot tables, Ptolemaic PM-Trees and the Ptolemaic M-Index. We also show that the most important and promising family of distances suitable for Ptolemaic indexing is the *signature quadratic form distance*, an adaptive similarity measure which can cope with flexible content representations of multimedia data, among other things. While this distance has shown remarkable qualities regarding the search effectiveness, its high computational complexity underscores the need for efficient search methods. We show that these distances are *Ptolemaic metrics* and present a study where we apply Ptolemaic indexing methods on real-world image databases, resolving exact queries nearly four times as fast as the state-of-the-art metric solution, and up to three orders of magnitude times as fast as sequential scan.

1. Introduction

The explosive growth of complex multimedia data including images, videos, and music challenges the effectiveness and efficiency of today's multimedia databases. Supposed to provide users access and insight into these inevitably increasing masses, multimedia databases have to manage data objects effectively and appropriately with respect to content access. When searching multimedia databases in a content-based way, users issue similarity queries by selecting multimedia objects or by sketching the intended object contents. Given an example multimedia object or sketch, the multimedia database searches for the objects that are most closely related to the query by measuring the similarity between the query and each database object, frequently by means of a distance function. As a result, the multimedia objects with the lowest distance to the query are returned to the user.

At present, the state-of-the-art query processing method for general distance-based retrieval is metric indexing [2,3]. By examining the distance relationships

☆ This paper is an extended version of a previous paper by Lokoč et al. [1]. This research has been supported by the Research Council of Norway project iAd (first author) and by the Czech Science Foundation projects GAČR 202/11/0968, P202/12/P297 (second and third authors).

* Corresponding author. Tel.: +420 22191 4227.
E-mail addresses: mlh@idi.ntnu.no (M.L. Hetland), skopal@ksi.mff.cuni.cz (T. Skopal), lokoc@ksi.mff.cuni.cz (J. Lokoč), beecks@cs.rwth-aachen.de (C. Beecks).

0306-4379/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.is.2012.05.011

between objects, metric indexing approaches use the triangle inequality to speed up query processing. While these approaches have been used for more than 20 years, another promising approach has recently been introduced, using the Ptolemaic inequality instead of the triangular one [4]. In this paper, we examine the Ptolemaic approach in depth, evaluating several data structures based on the principles of Ptolemaic indexing, and discussing which distances are most relevant for this method. The latter turns out to be a family of distances known as *signature quadratic form distance* (SQFD) [5,6], which combines high retrieval performance [7] and indexability [8].

1.1. Paper contributions

In this paper, we use *Ptolemaic Pivot table* (PtoPT), originally described by Hetland [4], as well as two other new index structures employing the principles of Ptolemaic indexing: the *Ptolemaic PM-Tree* (PtoPM-Tree) and the *Ptolemaic M-Index* (PtoM-Index). With this set of Ptolemaic indexes we establish the class of *Ptolemaic access methods*—an alternative to *metric access methods*. We apply the indexing methods to large multimedia databases to achieve efficient content-based similarity search. Ptolemaic indexing has been shown to be particularly efficient for quadratic form distances (QFDs). Unfortunately, as the Ptolemaic approach suffers from a high filtering cost, it is mainly suitable for expensive distances, where this extra complexity becomes insignificant. Hence, it may not be a feasible solution for one particular kind of QFD: the cheap (weighted) Euclidean distance. Moreover, it has recently been shown that for indexing purposes, all *static* QFDs can be mapped to the Euclidean case [9], so Ptolemaic indexing may not be viable even for them.

However, the mapping to Euclidean case does not effectively apply to the more expressive family of *signature* quadratic form distances (SQFD). In this paper we show both that these distances are Ptolemaic, and that Ptolemaic indexing is a clear improvement on the state of the art [8] for indexing them. Summarizing, the main contributions of this paper are:

- New heuristics for efficiently performing the Ptolemaic filtering which lead to an improvement in the real-time efficiency of querying.
- Ptolemaic shell filtering for region-based indexes.
- Detailed description of the Ptolemaic Pivot table.
- The Ptolemaic PM-Tree.
- The Ptolemaic M-Index.
- A proof sketch that the SQFD is a Ptolemaic metric.
- Empirical evidence that Ptolemaic access methods are efficient indexes for the SQFD, also when combined with metric principles.

The structure of this paper is as follows: In Sections 2 and 3, the basic principles of metric and Ptolemaic indexing are discussed, respectively. Section 4 deals with the Ptolemaic Pivot Table, the Ptolemaic PM-Tree, and the Ptolemaic M-Index. Section 5 describes the signature quadratic form distance in detail, as well as ways of indexing it. Section 6 lays out our experimental results, and finally Section 7 gives some conclusions.

2. Metric indexing

This section describes the basic principles of distance-based indexing using the metric axioms. The fundamental trick of these methods lies in using lower bounds that can be used to filter out irrelevant object from the search cheaply (i.e., without the need for actual distance computations). We also give a description of three metric indexes (Pivot Table, PM-Tree, M-Index) that we adapt to Ptolemaic indexes later in the paper.

A *metric space* $(\mathbb{U},\delta)$ consists of a descriptor domain $\mathbb{U}$ and a distance function δ which has to satisfy the metric postulates of *identity*, *non-negativity*, *symmetry*, and *triangle inequality* defined $\forall x,y,z \in \mathbb{U}$ as

$$\delta(x,y)=0 \Leftrightarrow x=y \quad \text{identity}$$

$$\delta(x,y)\geq 0 \quad \text{non-negativity}$$

$$\delta(x,y)=\delta(y,x) \quad \text{symmetry}$$

$$\delta(x,y)+\delta(y,z)\geq \delta(x,z) \quad \text{triangle inequality}$$

In this way, metric spaces allow domain experts to model their notion of content-based similarity by an appropriate descriptor representation and distance function serving as similarity measure. At the same time, this approach allows database experts to design index structures, so-called *metric access methods* (or metric indexes) [2,10–12], for efficient query processing of content-based similarity queries in a database $\mathbb{S}\subset\mathbb{U}$. These methods rely on the distance function δ only, i.e., they do not necessarily know the structure of the descriptor representation of the objects.

Metric access methods organize database objects (descriptors) $o_i \in \mathbb{S}$ by grouping them based on their distances, with the aim of minimizing not only traditional database costs like I/O but also the number of costly distance function evaluations. For this purpose, nearly all metric access methods apply some form of filtering based on cheap lower bounds. These bounds are constructed based on the fact that exact pivot–object distances are pre-computed (where a pivot is either a static or a dynamic reference object selected from the database).

We illustrate this fundamental principle in Fig. 1 where we depict the query object $q \in \mathbb{U}$, some pivot

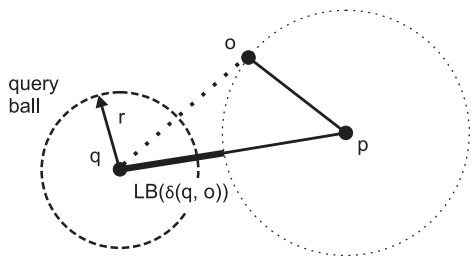


Fig. 1. The lower-bounding principle.

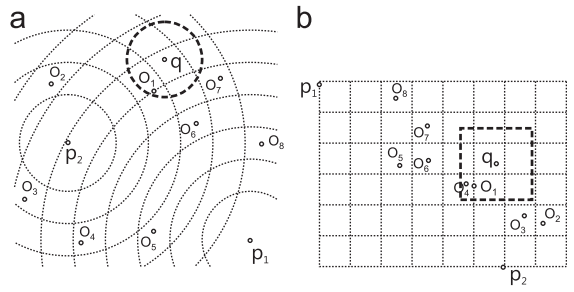


Fig. 2. (a) Original space, (b) pivot space.

object $p \in \mathbb{S}$, and a database object $o \in \mathbb{S}$ in some metric space. Given a range query (q,r), we wish to estimate the distance $\delta(q,o)$ by making use of $\delta(q,p)$ and $\delta(o,p)$, with the latter already stored in the metric index. Because of the triangle inequality, we can safely filter object o without needing to compute the (costly) distance $\delta(q,o)$ if the triangular lower bound $\delta_T(q,o) = |\delta(q,p) - \delta(o,p)|$ is greater than the query radius r. The fact that $\delta_T(q,o) \le \delta(q,o)$ is also known as the *inverse triangle inequality* [13, p. 674].

2.1. Pivot table as metric index

One of the most efficient (yet simple) metric indexes is the *Pivot Table* [14], originally introduced as LAESA [15]. Basically, the structure of a pivot table is a simple matrix of distances $\delta(o_i,p_j)$ between the database objects $o_i \in \mathbb{S}$ and a pre-selected static set of m pivots $p_j \in \mathbb{P} \subset \mathbb{S}$. For querying, pivot tables allow us to perform cheap lower-bound filtering by computing the maximum lower bound δ_T to $\delta(q,o)$ using all the pivots.

From a more intuitive perspective, pivot tables index the database objects as m-dimensional vectors in a pivot space. When querying, the range query ball (q,r) (or k NN ball with the current radius) is mapped into the pivot space, such that its center is $(\delta(q,p_1),\delta(q,p_2),\ldots,\delta(q,p_m))$. An important property of the mapping is that δ in the original space is lower-bounded by L_∞ distance[1] in the pivot space (i.e., it is a non-expansive mapping). The query ball in the pivot space (i.e., the L_∞-ball of radius r) can therefore be used to retrieve all the objects inside the query ball in the original space, possibly with some false positives that must be filtered out by δ in a refinement step. See Fig. 2 for an illustration of the pivot-based mapping from the original space into the pivot space, and the respective query balls.

2.2. PM-Tree

In addition to the pivot table where the metric lower-bounding is performed directly, we also include the PM-Tree—a metric index that conceptually merges the pivot table (flat table of object-to-pivot distances) with the M-Tree [16] (hierarchy of ball-shaped metric regions).

[1] The maximum absolute difference of two vectors' coordinate values.

The idea of PM-Tree [17,18] is to enhance the hierarchy of M-Tree by an information related to a static set of k global pivots $p_i \in \mathbb{P} \subset \mathbb{U}$. In a PM-Tree's routing entry, the original M-Tree-inherited ball region is further cut off by a set of rings (centered in the global pivots), so the region volume becomes more compact (see Fig. 3a). Similarly, the PM-Tree ground entries are enhanced by distances to the pivots, which are interpreted as rings as well. Each ring stored in a routing/ground entry represents a distance range (bounding the underlying data) with respect to a particular pivot.

A routing entry in a PM-Tree inner node is defined as

$$rout_{PM}(y) = [y, r_y, \delta(y,\mathrm{Par}(y)), ptr(T(y)), \mathrm{HR}]$$

where the first four are attributes inherited from the M-Tree routing entry, namely, the routing object y, covering radius r_y, the distance from y to the parent routing object, and a pointer to the child node. The new PM-Tree attribute HR is an array of k_{hr} intervals ($k_{hr} \le k$), where the t-th interval HR_{p_t} is the smallest interval covering distances between the pivot p_t and each of the objects stored in leaves of $T(y)$, i.e., $\mathrm{HR}_{p_t} = \langle \mathrm{HR}^{\min}_{p_t}, \mathrm{HR}^{\max}_{p_t} \rangle$, $\mathrm{HR}^{\min}_{p_t} = \min\{\delta(o_j,p_t)\}$, $\mathrm{HR}^{\max}_{p_t} = \max\{\delta(o_j,p_t)\}$, $\forall o_j \in T(y)$. The interval HR_{p_t} together with pivot p_t define a ring region (p_t,HR_{p_t}); a ball region $(p_t,\mathrm{HR}^{\max}_{p_t})$ reduced by a "hole" $(p_t,\mathrm{HR}^{\min}_{p_t})$.

A ground entry in a PM-Tree leaf is defined as

$$grnd_{PM}(z) = [z, id(z), \delta(z,\mathrm{Par}(z)), \mathrm{PD}],$$

where the new PD attribute stands for an array of p_{pd} pivot distances ($p_{pd} \le p$) where the t-th distance $\mathrm{PD}_{p_t} = \delta(y,p_t)$.

The combination of all the k entries' ranges produces a k-dimensional minimum bounding rectangle (MBR), and hence the global pivots actually map the metric regions/data into a pivot space of dimensionality k (see Fig. 3b). The number of pivots can be defined separately for routing and ground entries—we typically choose fewer pivots for ground entries to reduce storage costs (i.e., $k = k_{hr} > k_{pd}$). The pivot space mapping abstraction is much like that one used in pivot tables; however, in the PM-Tree case the pivot space also includes the hierarchy of MBRs (and so resembles R-tree partitioning to some extent).

When issuing a range or k NN query, the query object is mapped into the pivot space—this requires p extra distance computations $\delta(q,p_i), \forall p_i \in \mathbb{P}$. The mapped query ball (q,r_q) forms a hyper-cube $\langle \delta(q,p_1)-r_q, \delta(q,p_1)+r_q \rangle \times \cdots \times \langle \delta(q,p_k)-r_q, \delta(q,p_k)+r_q \rangle$ in the pivot space that is repeatedly used to check for an overlap with routing/ground entry's MBRs (see Fig. 3a, b). If they do not overlap, the entry is filtered out without any distance computations; otherwise, the M-Tree's filtering steps (parent and basic filtering) are applied. Actually, the MBRs overlap check can be also understood as L_∞ filtering, that is, if the L_∞ distance from a PM-Tree region to the query object q is greater than r_q, the region is not overlapped by the query. From the lower-bounding point of view, the MBR filtering is nothing other than applying k lower-bound tests to the closest possible object that might appear in the MBR.

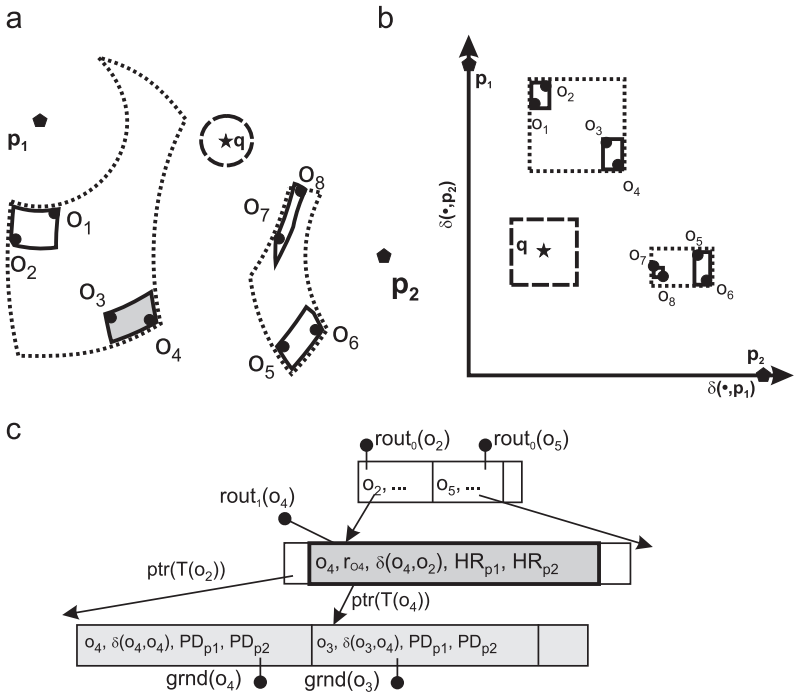


Fig. 3. (a) PM-Tree using two pivots (p_1, p_2). (b) Projection of PM-Tree into the pivot space.

Note that the MBRs overlap check does not require an explicit distance computation, so the PM-Tree usually achieves significantly lower query costs when compared with M-Tree—for more details, see previous work by Skopal et al. [17–19].

2.3. The M-Index

The M-Index [20,21] employs practically all known principles of metric space pruning and filtering. Inspired by iDistance [22] (designed for high-dimensional vector spaces), objects from the universe $\mathbb{U}$ are mapped into the real domain, which can be effectively managed by a B^+-tree. Assuming a normalized metric distance δ, the mapping function uses the set of global pivots $p_0,\ldots,p_{n-1}$ and the corresponding Voronoi partitioning. More specifically, each object is assigned a real-valued key, consisting of the object's distance to the closest pivot (the fractional part of the key) and the index of an assigned Voronoi partition (the integer part of the key). An example of such mapping is depicted in Fig. 4a. To obtain more partitions using the same number of pivots, repetitive Voronoi partitioning can be used, resulting in multi-level M-Index structure (see Fig. 4b). The key is computed as

$$key_l(o) = \delta(p_{(0)_o},o) + \sum_{i=1}^{l-1}(i)_o n^{l-1-i}$$

where $(\cdot)_o : \{0,\ldots,n-1\} \mapsto \{0,\ldots,n-1\}$ is a permutation of indexes such that $\delta(p_{(0)_o},o) \le \delta(p_{(1)_o},o) \le \cdots \le \delta(p_{(n-1)_o},o)$, l determines the l-prefix of the pivot permutation, $1 \le l \le n$ (the size of the key domain is n^l).

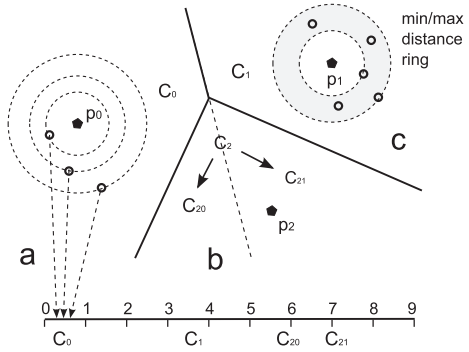


Fig. 4. (a) Mapping to the key domain (b) repetitive Voronoi partitioning (c) minimal and maximal distance to the closest pivot.

The authors also proposed a dynamic variant of the mapping where the tree of repetitively generated partitions is not balanced—see Fig. 4, where only the cluster C_2 assigned to pivot p_2 is further expanded to two clusters C_{20} and C_{21}. The dynamic variant better fits the distribution of objects in a database. The modified key function for dynamic M-Index is

$$key_l(o) = \delta(p_{(0)_o},o) + \sum_{i=1}^{l-1}(i)_o n^{l_{\max}-1-i}$$

where the size of the l-prefix of the pivot permutation is bounded by $l_{\max}$ value.

The query processing in the M-Index starts with mapping of the query object to the pivot space. Due to the repetitive

Voronoi partitioning, filtering of half-space regions can be applied up to l times. Because the ring defined by the distances $\min_{\forall o \in C_i}\{\delta(p,o)\}$ and $\max_{\forall o \in C_i}\{\delta(p,o)\}$ is stored in the structure for each leaf cluster C_i (see Fig. 4c), filtering of ball-shaped regions can also be employed. If none of the previously mentioned rules filters out a processed cluster, an interval of the searched key domain has to be determined and inspected. Finally, because the distances from each object to all global pivots are stored (determined during indexing), the efficient pivot-based filtering can be used.

3. The principles of Ptolemaic indexing

In metric indexes, the triangle inequality is used to construct lower bounds for the distance. Analogously, in *Ptolemaic indexing* [4], *Ptolemy's inequality* is used to construct such lower bounds as well. A distance function is called a *Ptolemaic distance* if it has the properties of *identity*, *non-negativity*, and *symmetry*, and satisfies *Ptolemy's inequality*. If a Ptolemaic distance also satisfies the *triangle inequality*, it is a *Ptolemaic metric*. Note that there is no implication in either direction between metricity and Ptolemaicity. A distance can be Ptolemaic without being metric, and vice versa [4].

Ptolemy's inequality states that for any quadrilateral, the pairwise products of opposing sides sum to more than the product of the diagonals. In other words, for any four points $x, y, u, v \in \mathbb{U}$, we have the following:

$$\delta(x,v)\cdot\delta(y,u) \le \delta(x,y)\cdot\delta(u,v)+\delta(x,u)\cdot\delta(y,v) \tag{1}$$

One of the ways the inequality can be used for indexing is in constructing a pivot-based lower bound. For a query q, object o, and pivots p and s, we get the *candidate bound*:

$$\delta_C(q,o,p,s) = \frac{|\delta(q,p)\cdot\delta(o,s)-\delta(q,s)\cdot\delta(o,p)|}{\delta(p,s)} \tag{2}$$

For simplicity, we let $\delta_C(q,o,p,s)=0$ if $\delta(p,s)=0$.[2] As for triangular lower-bounding, one would normally have a set of pivots $\mathbb{P}$, and the bound can then be maximized over all (ordered)[3] pairs of distinct pivots drawn from this set, giving us the final Ptolemaic bound [4]:

$$\delta(q,o) \ge \delta_P(q,o) = \max_{p,s\in\mathbb{P}} \delta_C(q,o,p,s) \tag{3}$$

Just as for the triangular case, the Ptolemaic lower bound δ_P could be used to filter objects not contained in the query ball, i.e., exclude those $o_i \in \mathbb{S}$ from search for which $\delta_P(q,o_i) > r$. In order to demonstrate the expected benefits of Ptolemaic filtering over the triangular, see an example in Fig. 5 (Euclidean space). Given just two pivots p, s, the Ptolemaic bound gives much tighter value than any of the two triangular bounds.

[2] This is simply so we do not have to treat $p=s$ as a special case later on.

[3] Computing the absolute value is redundant when examining all ordered pairs. It is, however, useful when only *some* pairs are examined, as explained later.

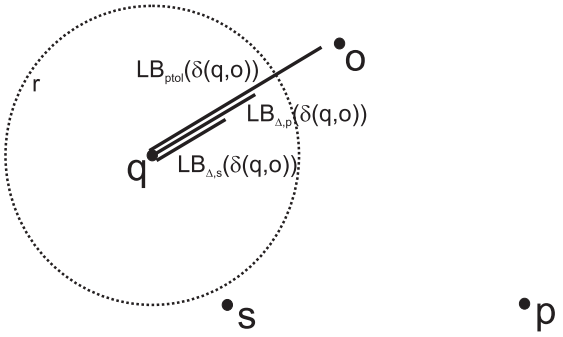


Fig. 5. Tightness of Ptolemaic bound LB_{ptol} vs. triangular bounds $LB_{\Delta,s}$,$LB_{\Delta,p}$.

3.1. Ptolemaic shell filtering

Beyond the basic pivot filtering, Hetland [4] describes how to derive bounds also when the exact pivot–object distances are not known, such as for objects in shell regions.[4] The more general bound given in his Theorem 3 can be instantiated with one lower bound r_p^- on $\delta(o,p)$ and an upper bound r_s^+ for $\delta(o,s)$, for two pivots p and s, and an object o, to yield the following:

$$\delta(q,o) \ge (\delta(q,s)\cdot r_p^- - \delta(q,p)\cdot r_s^+)/\delta(p,s)$$

In other words, the bound uses the inner shell radius around p and the covering radius for s. This means that for any structure that maintains regions consisting of intersecting shells, the outer radius of one shell can be combined with the inner radius of another to form a query–region bound usable for filtering.

3.2. What is special about Ptolemaic distances?

Why focus on the Ptolemaic inequality and not some other, arbitrary inequality? First, the inequality holds for the well-known reference for all distance intuitions: Euclidean distance. It also holds for the larger, important family of quadratic form distances, and, as we show in this paper, the recent (and more flexible) signature quadratic form distances. While the metric axioms have rather intuitive interpretations [24], the properties of Ptolemy's inequality are perhaps not quite as obvious. Even showing that the inequality holds for the Euclidean plane can be challenging—certainly more so than demonstrating triangularity. The bound defined by Eq. (2) does, however, seem to correspond to certain intuitions of distance or dissimilarity. We can think of p and s as representatives of, or *proxies* for, o and q, respectively. If o and q are close to their respective proxies (indicating accuracy and applicability of the bound) while far away from the other proxy (indicating actual distance), the bound is high. Contrast this with metric (triangle) lower-bounding, where there is only *one* pivot, and the bound is high when one of o and q is close, and the other is far away.

[4] This is tentatively explored by Reksten in his Master's thesis [23].

If we could place the pivots closer together without changing the other factors, we would reduce the divisor in Eq. (2), thereby increasing the bound. This observation seems to fly in the face of the proxy intuition; how can placing the proxies for q and o closer together *increase* the bound? This may not be as paradoxical as it seems, though. The strength of the bound is based on the *differences* in the distances from q and o to the two pivots. The closer together the pivots are, the smaller we would expect these differences to be.

Importantly, when using pivot-based lower-bounding with a set of pivots $\mathbb{P}$, the metric approach is limited to just $|\mathbb{P}|$ triangle inequality tests, while the Ptolemaic approach yields $\binom{|\mathbb{P}|}{2}$ candidate bounds (i.e., pairs of pivots to apply in Ptolemy's inequality).

The problem with Ptolemaic indexing is the small set of practical distances known to obey the Ptolemy's inequality. From the few known ones used in multimedia retrieval, we can name the quadratic form distance (and therefore also Euclidean distance). Fortunately, the SQFD is also a Ptolemaic metric, as we shown in Section 5.

3.3. Computing the bound efficiently

One of the most important sources of filtering power for Ptolemaic lower-bounding is the increased number of candidate bounds—quadratic in the number of pivots. Although this is certainly an advantage for filtering, it can significantly increase the computation time of the final bound. Unless the distance calculation itself is very slow, not many pivots are needed before the computation of the bound becomes prohibitively expensive. One of the contributions of this paper is a specific procedure for computing the bound, which takes advantage of the Ptolemaic filtering power and that is efficient (fast) in practice.

The idea is to perform *online pivot selection*, as used in the original pivot table methods, AESA [25] and LAESA [15], as well as in some more recent methods [26,27]. It is a fair assumption that the candidate bounds will differ significantly—this is the motivation for using multiple pivots, after all. Our task is to find a good pivot pair, a pair that will let us exclude an irrelevant object. Rather than trying out every pivot pair in an arbitrary order (which we refer to as the *naïve approach*), we can perform a heuristic search for the best ones. As long as the search radius (or the *current* search radius, in a k NN search) is available to us when computing the bound, we can terminate as soon as the radius has been exceeded. With a good heuristic ordering of the pairs, this will usually allow us to terminate early.

A high-quality heuristic will not only allow us to terminate the bound computation early when we are *able* to discard an object; it will give us the confidence needed to end our computation early *by fiat*. That is, if we know the best candidate bounds are probably computed first, and we are unable to eliminate an object early on, we should probably give up. This "giving up point" can either be set to a fixed number of candidate bounds or it can be based on lack of improvement over a series of candidate bounds.

3.3.1. Pivot permutations

Given the structure of the Ptolemaic lower bound (2), it would seem reasonable to expect a good pivot pair to consist of one object-like pivot p and one query-like pivot s.[5] This would minimize the subexpression $\delta(q,s)\cdot\delta(o,p)$, which would, all else being equal, maximize the bound. The problem is, of course, that minimizing these two distances will most likely change the rest of the bound as well. For example, we would like p and s to be *far away* from the query and the object, respectively, while staying close to each other. Even so, we have decided to use low values for $\delta(q,s)$ and $\delta(o,p)$ as the heuristic guidelines in our search. In the heuristics, we require an ordering of the pivots based on distance to a particular object o (either a database or a query object). For an object o we define a *pivot permutation* [28,29], as follows.

Having a fixed set of m pivots $\mathbb{P}=\{p_1,p_2,\ldots,p_m\}$ and an object $o\in\mathbb{U}$, let $()_o:\{1,2,\ldots,m\}\mapsto\{1,2,\ldots,m\}$ be a permutation such that $\forall i,j\in 1,\ldots,m$: $(i)_o\le(j)_o\leftrightarrow\delta(p_{(i)_o},o)\le\delta(p_{(j)_o},o)\vee(\delta(p_{(i)_o},o)=\delta(p_{(j)_o},o)\wedge i<j)$. Hence, the sequence $p_{(1)_o},p_{(2)_o},\ldots,p_{(m)_o}$ is ordered with respect to distances between the pivots and object o.

In order to efficiently look for pivots that are close to the query or to a given object, we need to precompute and store the pivot permutations $(\cdot)_o$ for every object o. We also compute $(\cdot)_q$ at the beginning of the search. Using these permutations, we generate a sequence of pivot pairs (p,s), and these pivot pairs are used to create candidate bounds.

3.3.2. An unbalanced heuristic

The simplest way of using these permutations is to have two nested loops, each iterating over one of the permutations (see Algorithm 3.1). With this *unbalanced* heuristic, either the q-like pivots or the o-like pivots are preferred.[6] In each iteration, the algorithm checks whether it should terminate early, that is, before all pivot pairs have been examined. This happens either if the bound is large enough ($\delta_P>r$) or if we have already tried κ pivot pairs, where κ is a cut-off parameter.

Algorithm 3.1. UNBALANCEDFILTER$(q,o,r,\kappa)\mapsto\delta_P$

```
1:  δP = c = 0
2:  for i=1 to m
3:    for j=1 to m
4:      δP ← max{δP, δC(q,o,p_(i)_o, p_(j)_q)}
5:      c ← c+1
6:      if δP > r or c = κ
7:        return
```

One seeming shortcoming of this procedure is that every pair of pivots will be used twice (once with the roles of p and s reversed). If both loops used the same permutation, we could easily have avoided this by constraining the range of the inner loop; we could then have taken the absolute value of δ_C, and saved approximately

[5] That is, p and s are close to object and query, respectively.

[6] Which permutation is assigned to the inner and outer loops does not seem to matter much.

half the bound calculations. It is, in fact, possible to avoid such duplicate computation efficiently even when using two different permutations, if the inverse permutations are available. This complicates the code quite a bit, though (and it is not at all clear that it improves performance), so we have decided to use the simpler version presented here.

3.3.3. A balanced heuristic

Giving preference to one of the permutations, as the unbalanced heuristic does, will probably postpone unduly pivot pairs that represent favorable tradeoffs between $\delta(q,s)$ and $\delta(o,p)$. In fact, even with *random* pivot permutations, tentative experiments indicate that the naïve heuristic is worse than one that examines the *pairs* in random order.

To avoid focusing on one of the permutations, we propose the *balanced* traversal heuristic (see Algorithm 3.2). This explores Cartesian product of $(\cdot)_o$ and $(\cdot)_q$ in a breadth-first fashion, starting at $((1)_o,(1)_q)$. Every o-like pivot (in order of distance from o) is combined with every q-like pivot that is at least as good, i.e., with at most the same rank in distance ordering from q, and vice versa. The checks for early termination work like in the unbalanced heuristic.

Algorithm 3.2. BALANCEDFILTER$(q,o,r,\kappa)\mapsto\delta_P$

```
1:   δ_P = c = 0
2:   for i=1 to m
3:     for j=1 to i
4:       δ_P ← max{δ_P, δ_C(q,o,p_(i)_o, p_(j)_q)}
5:       c ← c+1
6:       if i≠j
7:         δ_P ← max{δ_P, δ_C(q,o,p_(j)_o, p_(i)_q)}
8:         c ← c+1
9:       if δ_P > r or c ≥ κ
10:        return
```

Fig. 6 shows a comparison of the naïve approach and both heuristics in a Euclidean space (using 20 pivots drawn from a 10D uniform unit cube distribution). As expected, the balanced heuristic achieves a tight bound faster than the unbalanced one: After examining 15 pivot pairs the bound gets to 90% of the value obtained by examining all 190 unique pivot pairs.

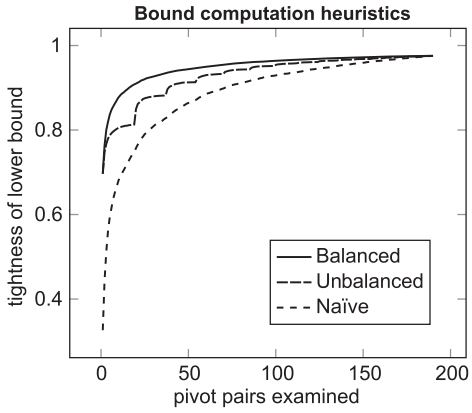


Fig. 6. Pivot pair selection heuristics.

4. Ptolemaic access methods

We have shown and argued that the Ptolemy's inequality could be used, instead of the triangle inequality, to construct distance lower bounds. In this section we present three Ptolemaic access methods (PtoAMs)—the Ptolemaic Pivot table, the Ptolemaic PM-Tree, and the Ptolemaic M-Index—we have adopted from the metric originals (Pivot Table, PM-Tree, M-Index). The Ptolemaic pivot table could be used solely with Ptolemy's inequality, so it might be used with metric or non-metric Ptolemaic distances. The other two methods assume also the triangle inequality to be satisfied due to metric partitioning of the data space, i.e., they support Ptolemaic metrics only.

4.1. Ptolemaic pivot table

As mentioned in Section 2.1, the pivot table is a simple, yet efficient and extensible metric access method. Moreover, it was recently shown [4] that pivot table could also be used as a Ptolemaic access method. In this paper we call this method *Ptolemaic Pivot Table*, or PtoPT. Note that in addition to the Ptolemaic lower bounds, the PtoPT can also employ the lower bound provided by the triangle inequality (thus becoming a Ptolemaic *and*/*or* metric access method). The data structure needed in order to accommodate the new heuristics presented in Section 3.3 is described in the following.

4.1.1. The PtoPT index structure

The index structure of PtoPT consists of two components:

- The pivot file, storing the set $\mathbb{P}$ of m pivots, and a pivot distance matrix, storing the distances between all distinct pairs of pivots from $\mathbb{P}$.
- The index file, storing the distances between each database object $o\in\mathbb{S}$ and all the pivots, the pivot permutation for each object o, and the object o itself. Formally, the index file consists of $|\mathbb{S}|$ entries, each belonging to a database object o, as $[o,(\cdot)_o,\delta(o,p_{(1)_o}),\delta(o,p_{(2)_o}),\ldots,\delta(o,p_{(m)_o})]$.

The difference between the original pivot table and the PtoPT is thus the extra information stored: the pivot distance matrix and the pivot permutation for each object o (used by the proposed heuristics).

Algorithm 4.1. PTOPT RANGEQUERY$(q,r,mode,\kappa)\mapsto Result$

```
1:   Result = ∅
2:   for each o in 𝕊
3:     if mode=triangle or mode=triangle+pto
4:       if TriangleFilter(q,o,r) > r
5:         continue for
6:     if mode=ptolemaic or mode=triangle+pto
7:       if PtolemaicFilter(q,o,r,κ) > r
8:         continue for
9:     compute δ(q,o)
10:    if δ(q,o) ≤ r
11:      add o to Result
```

4.1.2. Querying the PtoPT

A similarity query is processed by traversing the index file sequentially. However, for each database object o, we

try to avoid computing $\delta(q,o)$ by applying either triangle or Ptolemaic lower-bounding (or both). For the implementation of the PtoPT range queries, see Algorithm 4.1.

Note that TriangleFilter and PtolemaicFilter compute the lower bound to $\delta(q,o)$, including possible early termination—either due to the current bound exceeding the query radius r or the number of examined pivot pairs exceeding the limit κ. While the PtolemaicFilter function refers to either UnbalancedFilter or BalancedFilter (see Algorithms 3.1, 3.2), the TriangleFilter function just applies the triangle inequality test for each pivot $p \in \mathbb{P}$, as usual in the original pivot table.

As mentioned, the parameter κ refers to the maximum number of pivot pairs to be examined before giving up on computing the bound. Although this parameter could be left to the user, by setting $\kappa = |\mathbb{P}|$ we obtain Ptolemaic lower-bounding of the same time complexity as the triangular one (i.e., $\mathcal{O}(|\mathbb{P}|)$).

4.2. The Ptolemaic PM-Tree

As mentioned in Section 2.2, the PM-Tree is a dynamic, persistent, paged and extensible metric access method that synergically combines the basic principles of the M-tree and the pivot table. In this paper we show that the PM-Tree could be also used as a Ptolemaic metric access method. We call this method *Ptolemaic PM-Tree* or PtoPM-Tree. Note that in addition to the Ptolemaic lower bounds, the PtoPM-Tree can also employ the lower bound provided by the triangle inequality, i.e., we can employ all filtering conditions used in the original PM-Tree. The following sections describe how some simple heuristics can be used to compute the lower bound more efficiently. The data structure needed in order to accommodate these new heuristics is described in Section 4.2.1.

4.2.1. The PtoPM-Tree index structure

The index structure of PtoPM-Tree consists of two components:

- The pivot file, storing the set $\mathbb{P}$ of m pivots, and a pivot distance matrix, storing the distances between all distinct pairs of pivots from $\mathbb{P}$.
- The index file, storing the PM-Tree index structure, where only the ground entry is slightly changed—again the pivot permutation is included. A ground entry in PtoPM-Tree leaf is defined as $grnd_{PM}(z) = [z, id(z), (\cdot)_z, \delta(z,\mathrm{Par}(z)), \mathrm{PD}]$.

Algorithm 4.2. PtoPM-Tree RangeQuery(q,r,*TreeRout*, *Result*)$\mapsto$*Result*

```
1:  if ptr(T(TreeRout)) is InnerNode
2:    for each routingEntry in ptr(T(TreeRout))
3:      if TriangleParentFilter(q,routingEntry.O,r) > r
4:        continue for
5:      if PtolemaicShellFilter(q,routingEntry.O,r) > r
6:        continue for
7:      compute δ(q,routingEntry.O)
8:      if TriangleFilter(q,routingEntry.O,r) > r
9:        continue for
10:     RangeQuery(q,r,routingEntry,Result)
11:   else
12:     for each groundEntry in ptr(T(TreeRout))
13:       // triangle LB for parent object, hyperrings
14:       if TriangleParentFilter(q,groundEntry.O,r) > r
15:         continue for
16:       // Ptolemaic LB for parent object and a global pivot
17:       if ExtraPivotPtolemaicFilter(q,groundEntry.O,r) > r
18:         continue for
19:       // Ptolemaic LB for all tuples of global pivots
20:       if PtolemaicFilter(q,groundEntry.O,r) > r
21:         continue for
22:       compute δ(q,groundEntry.O)
23:       if δ(q,groundEntry.O) ≤ r
24:         Result.Add(groundEntry.O)
```

4.2.2. Querying the PtoPM-Tree

Because the PM-Tree's PD attribute stands for an array of p_{pd} pivot distances ($p_{pd} \le p$) where the t-th distance $\mathrm{PD}_{p_t} = \delta(y,p_t)$, we can use this information for basic Ptolemaic filtering. The original PM-Tree structure (without permutations) could be used in this way; however, then only the naïve heuristic could be employed. Besides the basic Ptolemaic filtering applicable in the leaf level of the PtoPM-Tree, there are two additional filtering methods applicable in the PtoPM-Tree, as discussed in the following paragraphs.

- *Extra pivot for Ptolemaic filtering in PM-Tree leaves*: The PtoPM-Tree leaf node represents a metric region with the center object $Par(z)$ stored in the parent routing entry, while the ground entries in the leaf node store the distance $\delta(z,\mathrm{Par}(z))$. $Par(z)$ can be considered as a dynamically selected local pivot for objects in the leaf node and thus could be used for Ptolemaic filtering. In the case the leaf node contains object $\mathrm{Par}(z)$, the distances between $\mathrm{Par}(z)$ and all global pivots can be determined from PD stored in $grnd_{PM}(z)$. Thus, the Ptolemaic lower bound can be computed for all objects in the leaf node using the additional pivot pairs $(\mathrm{Par}(z), p_i)$, where $p_i \in \mathbb{P}$.
- *Ptolemaic shell filtering in PM-Tree*: Because the routing entry $rout_{PM}(y)$ contains hyper-ring (shell) information stored in HR, the shell filtering described in Section 3.1 can be used for subtree elimination. Then the lower bound between query object q and region formed by the intersection of the hyper-rings HR is determined as

  $$(\delta(q,s) \cdot \mathrm{HR}_p^{\min} - \delta(q,p) \cdot \mathrm{HR}_s^{\max}) / \delta(p,s)$$

  To get the greatest possible lower bound, all pivot pairs $p,s \in \mathbb{P}$ have to be checked.

For the range query details see Algorithm 4.2.

4.3. The Ptolemaic M-Index

As mentioned in Section 2.3, M-Index combines all commonly used metric filtering principles; in particular, a sort of pivot Table is stored in the M-Index buckets and thus can be simply extended to the Ptolemaic version. In this paper we call this method *Ptolemaic M-Index*, or PtoM-Index.

4.3.1. The PtoM-Index index structure

The data structure needed in order to accommodate the Ptolemaic filtering is very similar to the original M-Index structure. Again the pivot file, storing the set $\mathbb{P}$ of m pivots, and a pivot distance matrix, storing the distances between all distinct pairs of pivots from $\mathbb{P}$, have to be included. The second difference from the original M-Index structure is the use of the Ptolemaic Pivot Table in the buckets, i.e., with distances to pivots and the pivot permutations stored for each object.

4.3.2. Querying the PtoM-Index

The query processing in the PtoM-Index consists of two phases. First, the metric filtering in the cluster tree is applied to determine buckets containing relevant objects and, second, the parts of the Ptolemaic Pivot Table stored in the buckets have to be sequentially processed. Hence, the same principles as in Section 4.1.2 are employed. For the range query details see thus again Algorithm 4.1.

5. Indexing the signature quadratic form distance

In this section, we review the utilized similarity model, the SQFD on feature signatures, outline existing query processing techniques, and explain the relationships between Ptolemaic indexing and quadratic form distances more in detail.

5.1. Ptolemaic norm metrics = QFDs

Before dealing with SQFDs, we first recall a result about quadratic form distances (QFDs), due to Hetland [4], a result that motivates the proof sketch in Section 5.4.

A distance can be metric without being Ptolemaic, and vice versa—neither property implies the other [30].

However, it turns out that there is one class of metrics that is a very natural example of Ptolemaicity, namely QFDs.

A QFD can be expressed as follows:

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

In matrix notation, this becomes $\sqrt{z'Az}$, where x and y are vectors, and $z=x-y$. The (symmetric) weight matrix $A=[a_{ij}]$ defines how related (or rather, unrelated) the dimensions are [31]. If (and only if) A is positive-definite, the distance is a metric.

There are, of course, an infinite number of Ptolemaic metrics. For example, for any metric $\delta(\cdot,\cdot)$, the distance $\sqrt{\delta(\cdot,\cdot)}$ will be a Ptolemaic metric (with an identical distance ordering to that of the original), although in general with a greatly increased intrinsic dimensionality [19], making indexing harder. There are also other, specific examples, such as the discrete metric ($\delta(x,y)=1\Leftrightarrow x\neq y$) or the chordal metric on the unit Riemann sphere [32]. However, none of these examples are actually in use in similarity retrieval.

On the other hand, if we restrict our attention to the relatively general case of normed vector spaces $(X,\|\cdot\|)$, and norm metrics $\delta(x,y)=\|x-y\|$ (that is, metrics induced by the vector norm), we see that subset of these that are Ptolemaic corresponds exactly to the set of quadratic form distances:

Theorem 1 (*Hetland [4]*). *A distance function is a quadratic form metric if and only if it is a Ptolemaic norm metric.*

In other words, not only are QFDs Ptolemaic; when dealing with vector spaces and norm-induced distances—an important case, for sure—they are *the* Ptolemaic distances.

5.2. Measuring the similarity between feature signatures

The SQFD is an adaptive similarity measure defined for the comparison of feature signatures. This type of feature representation gathers object properties by individually aggregating them in a compact way. Unlike conventional feature histograms, feature signatures are frequently obtained by clustering the objects' properties, such as color, texture, or other more complex features [33,34], within a feature space and storing the cluster representatives and weights. Thus, given a feature space $\mathbb{F}$, the *feature signature* S^o of a multimedia object o is defined as a set of tuples from $\mathbb{F}\times\mathbb{R}^+$ consisting of representatives $r^o\in\mathbb{F}$ and weights $w^o\in\mathbb{R}^+$.

We depict an example of image feature signatures according to a feature space comprising position and color information, i.e. $\mathbb{F}\subseteq\mathbb{R}^5$, in Fig. 7. For this purpose we applied the k-means clustering algorithm [35] where each representative $r_i^o\in\mathbb{F}$ corresponds to the centroid of the cluster $\mathcal{C}_i^o\subseteq\mathbb{F}$, i.e., $r_i^o=\sum_{f\in\mathcal{C}_i^o}f/|\mathcal{C}_i^o|$, with relative frequency $w_i^o=|\mathcal{C}_i^o|/\sum_i|\mathcal{C}_i^o|$. We depict the feature signatures' representatives by circles in the corresponding color. The weights are reflected by the diameter of the circles. As can be seen in this example, feature signatures adjust to individual image contents by aggregating the features according to their appearance in the underlying feature space. Beyond images, feature signatures can also be applied to other kinds of multimedia data which fit into this flexible feature representation form. In general, there are no obstacles to using a non-vectorial feature space $\mathbb{F}$, so the SQFD has prospects of becoming a universal distance for measuring locality-sensitive similarity between any complex signatures consisting of local features.

In order to compare feature signatures, we make use of the SQFD which is a generalization of the conventional *quadratic form distance* (QFD) [36]. In contrast to the well-known *earth mover's distance* (EMD) [37], the SQFD makes it possible to balance the tradeoff between indexability and retrieval quality [8]. Moreover, it has been recently shown how to compare continuous probability distributions, such as *mixtures of Gaussian densities*, with the SQFD [38,39]. Thus, the SQFD is able to successfully cope with a broad range of feature representations of different structure and size. In this paper, however, we define the SQFD for the comparison of feature signatures as follows.

Definition 1 (*SQFD*). Given two feature signatures $S^q=\{\langle r_i^q,w_i^q\rangle\}_{i=1}^n$ and $S^p=\{\langle r_i^p,w_i^p\rangle\}_{i=1}^m$ and a similarity function $f_s:\mathbb{F}\times\mathbb{F}\to\mathbb{R}$ over a feature space $\mathbb{F}$, the *signature*

Fig. 7. Three example images with their corresponding feature signature visualizations.

quadratic form distance SQFD_{f_s} between S^q and S^p is defined as

$$\mathrm{SQFD}_{f_s}(S^q,S^p)=\sqrt{(w_q|-w_p)\cdot A_{f_s}\cdot(w_q|-w_p)^T},$$

where $A_{f_s}\in\mathbb{R}^{(n+m)\times(n+m)}$ is the similarity matrix arising from applying the similarity function f_s to the corresponding representatives, i.e., $a_{ij}=f_s(r_i,r_j)$. Furthermore, $w_q=(w_1^q,\ldots,w_n^q)$ and $w_p=(w_1^p,\ldots,w_m^p)$ form weight vectors, and $(w_q|-w_p)=(w_1^q,\ldots,w_n^q,-w_1^p,\ldots,-w_m^p)$ denotes the concatenation of weights w_q and $-w_p$.

As can be seen in Definition 1, the SQFD is defined by means of a similarity function f_s determining the similarity relationship between any two representatives of the feature signatures. Gathering all possible similarity relationships between and also within the feature signatures defines the similarity matrix A_{f_s}. In fact, including *self-similarities*, i.e. similarity values between representatives from the same feature signature, is necessary to obtain good retrieval results [40]. The similarity matrix has to be determined for each distance computation individually. Thus, the complexity of a single distance computation is in $\mathcal{O}((n+m)^2\cdot\phi)$ where n and m denote the size of feature signatures S^q and S^p, respectively, and ϕ denotes the complexity of the similarity function f_s over a feature space $\mathbb{F}$.

The choice of an appropriate similarity function depends on the domain the SQFD is applied to. In particular, the *Gaussian similarity function* $f_g(r_i,r_j)=e^{-\alpha\cdot L_2(r_i,r_j)^2}$ with the Euclidean distance function L_2 shows high retrieval performance in the image domain when adapting the parameter $\alpha\in\mathbb{R}^+$ to the current multimedia database [7]. Although more efficient similarity functions exist, such as $f_-(r_i,r_j)=-L_2(r_i,r_j)$, the Gaussian similarity function enables the SQFD to outperform the other adaptive similarity measure as shown in [7] and recently in [41]. Therefore, we use the Gaussian similarity function throughout the experimental evaluation. We continue with a description of SQFD-specific approaches to efficient query processing with a particular focus on metric indexing.

5.3. Metric indexing of the SQFD

Unlike previous approaches [42–44] focusing on improvements in efficiency through speeding up the sequential scan in an exact or approximate way, Beecks et al. [8] exploit the indexability of the SQFD by investigating the parameters of the similarity function and indexing the data through simple pivot tables. They have shown that the similarity functions' parameters affect the indexability of the underlying data space thus allowing to balance the tradeoff between indexability and retrieval quality. In fact, by adapting the parameter α of the Gaussian similarity function $f_g(r_i,r_j)=e^{-\alpha\cdot L_2(r_i,r_j)^2}$, their pivot table approach reached a speed-up factor of up to 170 when compared to the sequential scan. In addition, the combination of the SQFD and Ptolemaic pivot tables has shown a speed-up factor of up to 300 (see the previous version [1] of this paper).

In the meantime, Krulis et al. [45,46] came up with the idea of processing the SQFD on many-core GPU architectures. By implementing the query evaluation process on many-core GPUs and also multi-core CPUs, they have shown a significant improvement in efficiency compared to the non-parallelized approaches. Their approach is complementary to the techniques proposed in this paper.

5.4. Signature QFDs are QFDs

This section outlines a proof sketch justifying the use of Ptolemaic and metric indexing for the SQFD. The idea is

simple: the SQFD is basically an efficient way of calculating the QFD between extremely sparse weight vectors. Consider an ordinary QFD with vectors $\mathbb{R}^N$. Assume that for any dimension, at most one vector in the space has a nonzero value. By assuming a representative for each dimension, the similarity matrix for this space can be computed exactly as in the SQFD. It should now be clear that for the QFD over these vectors, most dimensions will be irrelevant. Any dimension where both of the vectors have a value of 0 will be eliminated from the inner product, and we end up actually using a tiny portion of the similarity matrix. This is exactly what is done in the SQFD.

We can look at this the other way around, starting with the SQFD signatures. We first embed the weight vectors of feature signatures into $\mathbb{R}^N$, where each dimension of each weight vector in the original feature space is assigned to a separate dimension in $\mathbb{R}^N$. For simplicity, we initially assume that we are working with a finite distance space, and that all representatives are distinct. (These assumptions will be relaxed later.)

The number of dimensions, N, is thus the sum of all feature signature sizes in our original distance space. Embedding a weight vector in $\mathbb{R}^N$ simply involves padding it with zeros on either end, so that its weights end up in the correct dimensions (assuming that the dimensions, which we are free to choose, are adjacent). Let $w' \in \mathbb{R}^N$ be the embedded version of any weight vector w. We then have

$$(w_a|-w_b)' = w'_a - w'_b$$

Here, the embedding $(w_a|-w_b)' \in \mathbb{R}^N$ is taken to mean the embedding that assigns the values from w_a and w_b to the same dimensions as the individual embeddings w'_a and w'_b.

We can now define a global similarity matrix A, using the same similarity function as for the local matrices A_{f_s}. We can then calculate the QFD between w'_a and w'_b as follows:

$$\begin{aligned}\mathrm{QFD}(w'_a,w'_b) &= \sqrt{(w'_a - w'_b)\cdot A\cdot (w'_a - w'_b)^T}\\ &= \sqrt{(w_a|-w_b)'\cdot A\cdot (w_a|-w_b)'^T}\end{aligned}$$

In the matrix product $x'Ax'^T$, the extra dimensions involved in the embedding are all zero, and do not contribute at all (see Fig. 8). Only the original dimensions, and the corresponding entries in A (which together form A_{f_s}), are used in computing the distance. Thus we have $\mathrm{QFD}(w_a',w_b') = \mathrm{SQFD}(a,b)$. If A is well-behaved (symmetric positive definite), QFD is a Ptolemaicmetric [4]. Since the mapping $x \mapsto x'$ is a distance-preserving isomorphism, the same holds for SQFD.

Now, if two input signatures a and b both contain a given representative, then that representative would be represented by (at least) two different dimensions in $\mathbb{R}^N$. These two dimensions would then have identical rows and columns in A, and the similarity between them would be 1, which makes QFD a pseudometric.[7] Note, however, that because of the way the vectors in $\mathbb{R}^N$ are constructed,

[7] We could get zero distances between distinct objects.

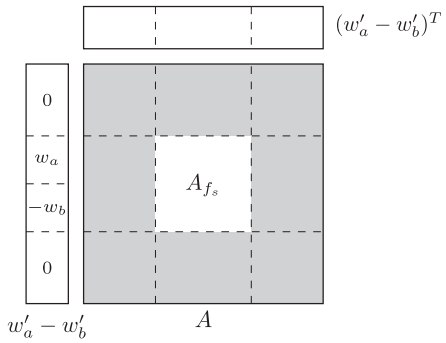


Fig. 8. Embedding SQFD vectors in a QFD space. Because of the zero components of $w'_a - w'_b$, the gray areas of A will not affect the inner product.

w'_a and w'_b will have zero components for one of these dimensions each. In other words, a zero distance will only occur between w'_a and w'_b if $a=b$.

The only unresolved issue is what happens if the original distance space is infinite. We will then end up with infinite-dimensional vector space. We can still define an equivalent inner product $\langle \cdot, \cdot \rangle$, extending the original quadratic form, and the resulting (infinite-dimensional) distance $\delta(a',b') = \sqrt{\langle w'_a - w'_b, w'_a - w'_b \rangle}$ would still be a Ptolemaicmetric [47]. Whether the infinite-dimensional case fits under the common definition of a QFD is perhaps debatable, but the main point here is that it is still Ptolemaic, and can therefore be used with Ptolemaicindexing techniques.

6. Experimental evaluation

In this section, we evaluate the efficiency of search under SQFD. We compare the PtoPT with the original pivot tables, so far the state-of-the-art metric index applied to SQFD. We also evaluate all the introduced variants of PtoPM-Tree and compare them to the original PM-Tree. Finally, we propose a first performance comparison of the state-of-the-art metric and Ptolemaic access methods.

6.1. The Testbed

We made use of the two real databases—the *MIR Flickr* database [48] including 25 000 web-images with textual annotations, and the *ALOI* database [49] consisting of 72 000 images. The selected databases are not very large but they provided a ground truth for evaluating the search effectiveness. We also employed a larger synthetic dataset *CLOUDS* representing 251 000 clouds of points.

We extracted feature signatures from the real databases, based on seven-dimensional features $(L,a,b,x,y,\chi,\eta) \in \mathbb{F}$ including color (L,a,b), position (x,y), contrast χ, and coarseness η information. These features were extracted for a randomly selected subset of pixels for each image and then aggregated by applying an adaptive variant of the k-means clustering algorithm described by Leow and Li [50]. Thus, we obtained one feature signature for each single image. The signatures varied in size between 5 and 115 feature representatives. On average, a feature

signature consisted of 54 representatives (i.e., 432 numbers per signature).

The synthetic *CLOUDS* database was generated [51], namely I251 000 clouds (sets) of 20–40 10-dimensional points (embedded in a unitary 10D cube). This database was chosen as a set analogy to synthetic vector datasets when evaluating vectorial similarity search. Moreover, the cloud of points is common for simplified representations of complex objects or objects consisting of multiple observations [52]. Each point has assigned a weight where the sum of all weights in the cloud was 10 000. For each cloud, its center was generated at random, while other 10 000 points were generated under normal distribution around the center (the mean and variance in each dimension were adjusted to not generate points outside the unitary cube). Then an adaptive variant of the *k*-means clustering [50] was used to create 20–40 centroids representing the original data. The weight of each centroid corresponded to the number of points assigned to the centroid in the last iteration of the *k*-means clustering. On average, a feature signature consisted of 30 representatives (centroids), i.e., 330 numbers per signature.

Table 1
Labels used in the figures.

Label	Description
PT	Pivot Table
PtoPT	Ptolemaic Pivot Table
PtoPM-Tree	Ptolemaic PM-Tree
PtoPM-Tree + E	Ptolemaic PM-Tree using extra pivot
PtoPM-Tree + E + SH	Ptolemaic PM-Tree using extra pivot and shell filtering
PtoM-Index	Ptolemaic M-Index
Tri(n)	PtoPT using triangle mode
Pto(n, U—B—N, κ)	PtoPT using Ptolemaic mode
TriPto(n, U—B—N, κ)	PtoPT using triangle+Ptolemaic mode n is the number of pivots used U—B—N stands for Unbalanced, Balanced or Naïve heuristics κ is the max. number of pivot pairs used
% of Tri(n)	Performance related to query realtime achieved by Tri(n)
% of PM-Tree	Performance related to query realtime achieved by PM-Tree

The remaining settings in our experiments were the same as those used by Beecks et al. [8]. The tests ran on a workstation 2x Intel Xeon X5660 2.8 GHz, 24 GB RAM, Windows Server 2008 R2 64bit (non-virtualized). We have used our own C++ implementation of PT, PtoPT, PM-Tree and PtoPM-Tree index structures. For the M-Index we have downloaded the original Java implementation [53]. In order to correctly compare the Java-based M-Index and PtoM-Index with the C++-based indexes we have used the number of evaluated distance computation in Section 6.2.4 instead of real times. However, as SQFD is an expensive distance, the real times strongly correlate with the number of distance computations.

In Table 1, we summarize the description of labels used within the following figures. We investigate all heuristics proposed in Section 4.1.2 only using PtoPT so we utilize special labels in Section 6.2.2. Note that Tri(···) denotes the original pivot tables used as a reference metric access method, while the Pto(···) and TriPto(···) labels refer to specific filtering modes for efficient Ptolemaic lower-bound estimation (see Section 4.1.2).

6.2. The results

6.2.1. SQFD properties

We first present the SQFD properties in terms of *intrinsic dimensionality* (iDim) [2] and *mean average precision* (MAP) values [54], as they indicate whether the SQFD allows for efficient and effective indexing (see Fig. 9). The crucial parameter alpha (denoted α in Section 5.2) is applied inside the Gaussian similarity function f_s when computing the SQFD matrix, and it has significant impact on iDim and MAP. For more about tuning iDim and MAP using alpha, see the paper by Beecks et al. [8].

Next, we study the increase in efficiency in terms of k NN query response times. As the numbers of distance computations perfectly correlate with the real response times (because of SQFD's computational cost), we present mainly the real times in the figures. However, since we have two different platforms for various PtoAMs/MAMs, we present also distance computations as a platform independent performance measure in Section 6.2.4. In general, a single SQFD computation takes on average

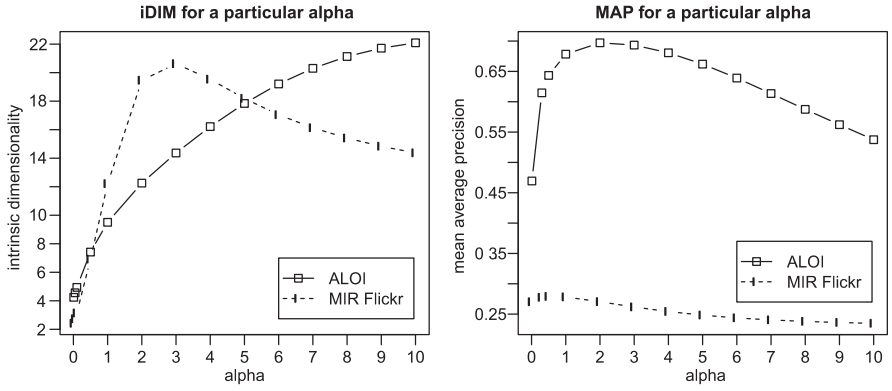


Fig. 9. Intrinsic dimensionality vs. mean average precision.

0.65 ms for the ALOI database, 0.79 ms for the MIR Flickr database, and 0.25 ms for the CLOUDS database. The number of distance computations could be reconstructed using these values, as well as the real response times of the sequential scan. The query costs were averaged for 100 different queries, while the query signatures were not indexed.

6.2.2. Ptolemaic pivot tables

See Fig. 10 for the performance of k NN queries on the ALOI database, 50 pivots, and alpha$=0.32$. The PtoPT is a clear winner in all configurations, while the Balanced heuristic always works best. Also note that the maximum relative speed-up with respect to Tri(50) is achieved for $k=10$–50.

The impact of the maximum number of pivot pairs used in the Ptolemaic bound computation is presented in Fig. 11. It shows that the Ptolemaic filtering is effective even for a small number of pivot pairs (Pto), while it is further improved when combined with the triangle filtering (TriPto).

Fig. 12 shows the relative performance w.r.t. Tri(10) for varying alpha. Note that for very small alpha also the iDim is very small (≈ 2), so that the triangle filtering using 50 pivots is efficient enough and leaves a relatively smaller room for improvement. On the other hand, large alpha leads to high iDim (≈ 20), as well as smaller differences between triangular and Ptolemaic filtering—the space is hard to index, whichever filtering is used. Considering 50 pivots, the best speed-up of PtoPT is achieved for alpha$=0.4$ and ALOI (iDim ≈ 4), alpha$=0.1$ and MIR Flickr (iDim ≈ 3). The results for k NN queries on the MIR Flickr (see Fig. 13) are similar to those for ALOI (see Fig. 10); however, the largest speed-up of PtoPT was achieved for 1NN queries.

The results showing the varying number of pivot pairs on the MIR Flickr (see Fig. 14) are also similar to those for ALOI (see Fig. 11). Note that for a large enough number of pivot pairs (>60) the Ptolemaic filtering works equally well using only 10 pivots as does triangle filtering with 50 pivots.

6.2.3. Ptolemaic PM-Tree

In the following set of experiments, we have compared the original PM-Tree to the proposed Ptolemaic variants. To create PM-Tree, we have used 10 pivots both for ground and routing entries, the capacity of leaf and inner nodes was set to 21. To estimate the Ptolemaic lower bound, all possible pivot pairs were used (naïve heuristics). The experiments

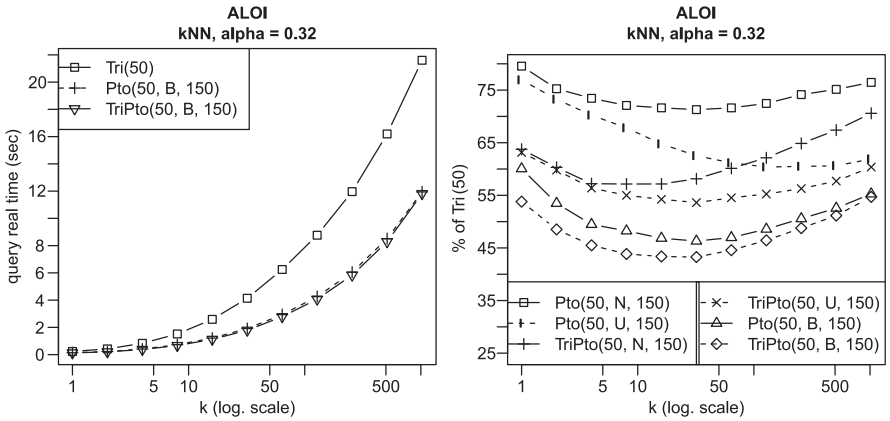


Fig. 10. k NN queries on the ALOI database.

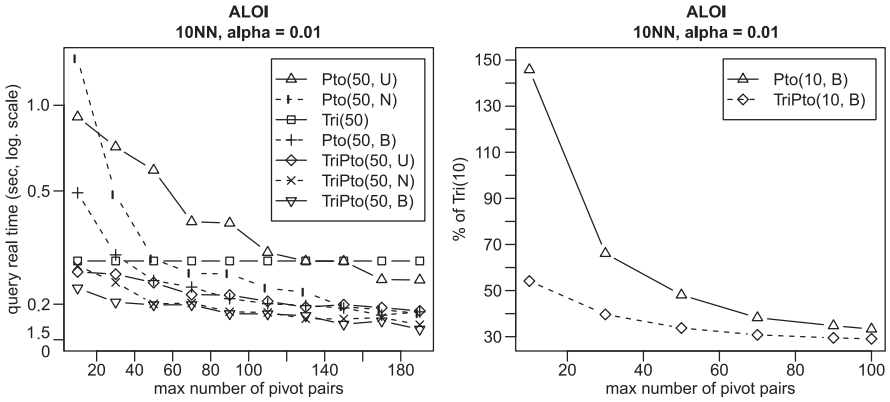


Fig. 11. Number of pivot pairs in PtoPT (ALOI).

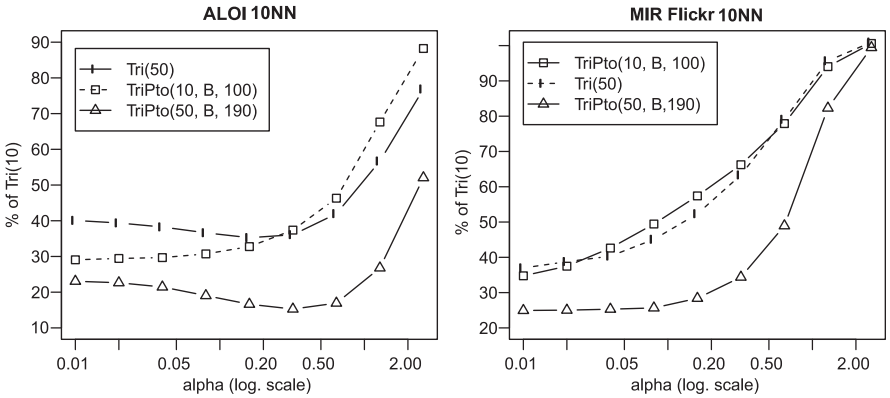


Fig. 12. Impact of varying alpha on the relative performance.

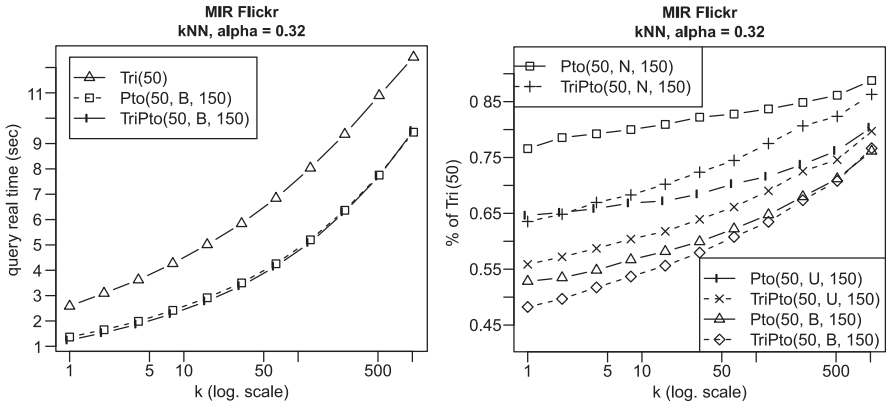


Fig. 13. *k* NN queries on the MIR Flickr database.

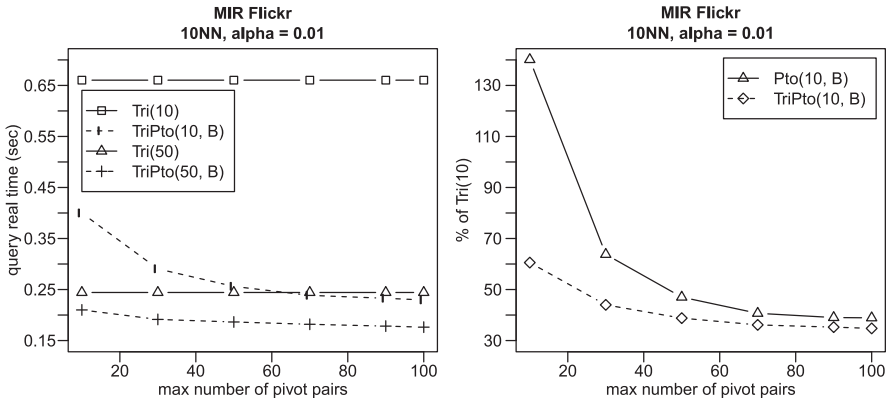


Fig. 14. Number of pivot pairs in PtoPT (MIR Flickr).

show that shell filtering significantly improves the PtoPM-Tree performance for easy to index datasets (low alpha), while the unique extra pivot employed in each leaf node improves the index quality for less indexable datasets (see Fig. 15). The PtoPM-Tree + E + SH variant reduces the costs up to 50% of the original PM-Tree on the CLOUDS dataset and up to 66% of the original PM-Tree on the ALOI dataset.

Similar behavior can be observed also for the growing radius of the *k*NN query (Fig. 16), which might be caused by better resistance of the Ptolemaic lower-bounding to the growing radius of the query. Moreover, for each dataset and alpha, we observe that there is some optimal radius for Ptolemaic lower-bounding.

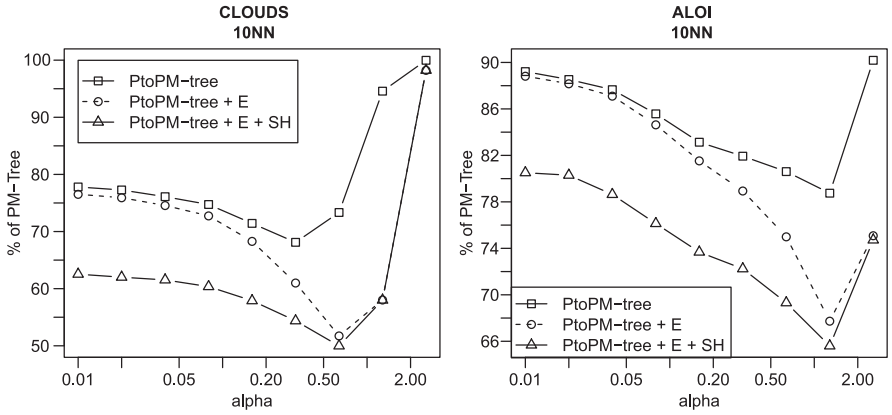


Fig. 15. Performance of PtoPM-Tree for varying alpha (ALOI and CLOUDS).

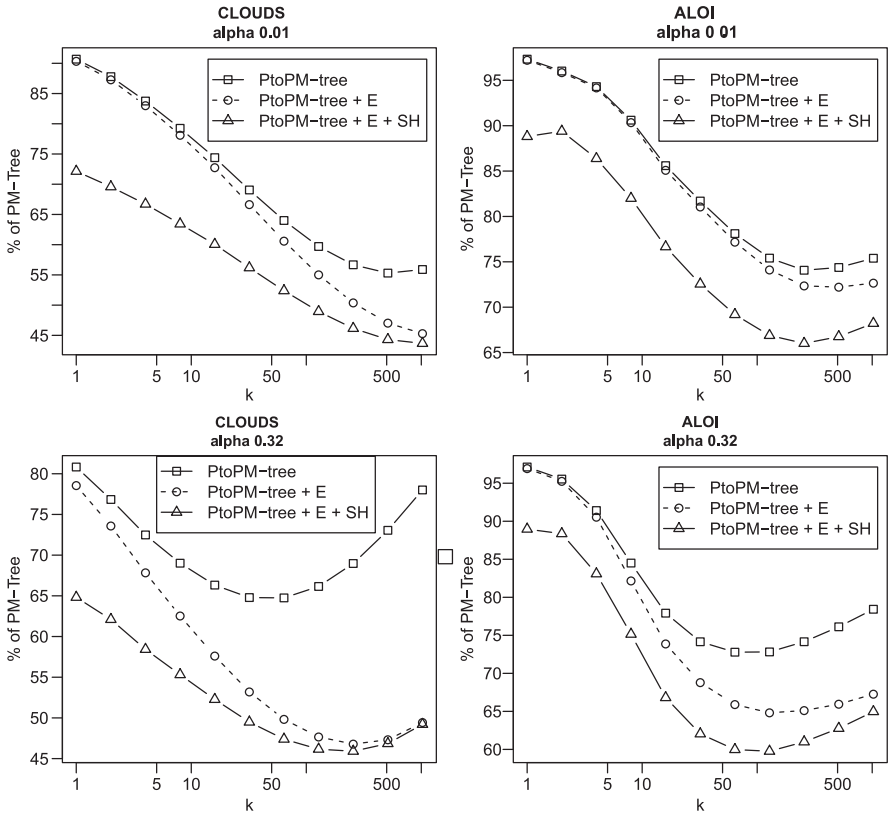


Fig. 16. Performance of PtoPM-Tree for varying kNN (ALOI and CLOUDS).

6.2.4. MAMs and PtoAMs comparison

In the last set of experiments (see Fig. 17), we have compared the state-of-the-art metric access methods with their corresponding Ptolemaic variants. We have also added the M-Index and PtoM-Index (dynamic variant, bucket size set to 500, max height of the cluster tree set to 6, and 10 pivots used). In all cases, the Ptolemaic variant results in a performance boost. We can also observe that for small radii and low alpha, the PtoPT and PtoM-Index are very efficient, while for large radii and high alpha the PtoPM-Tree becomes the best choice. Comparing to the sequential scan, the PtoM-Index is even 1100 times faster. However, the three orders of magnitude performance boost can be achieved only for 1NN queries and alpha = 0.01. For growing radius or for growing alpha, metric index performance decreases faster than for Ptolemaic indexes.

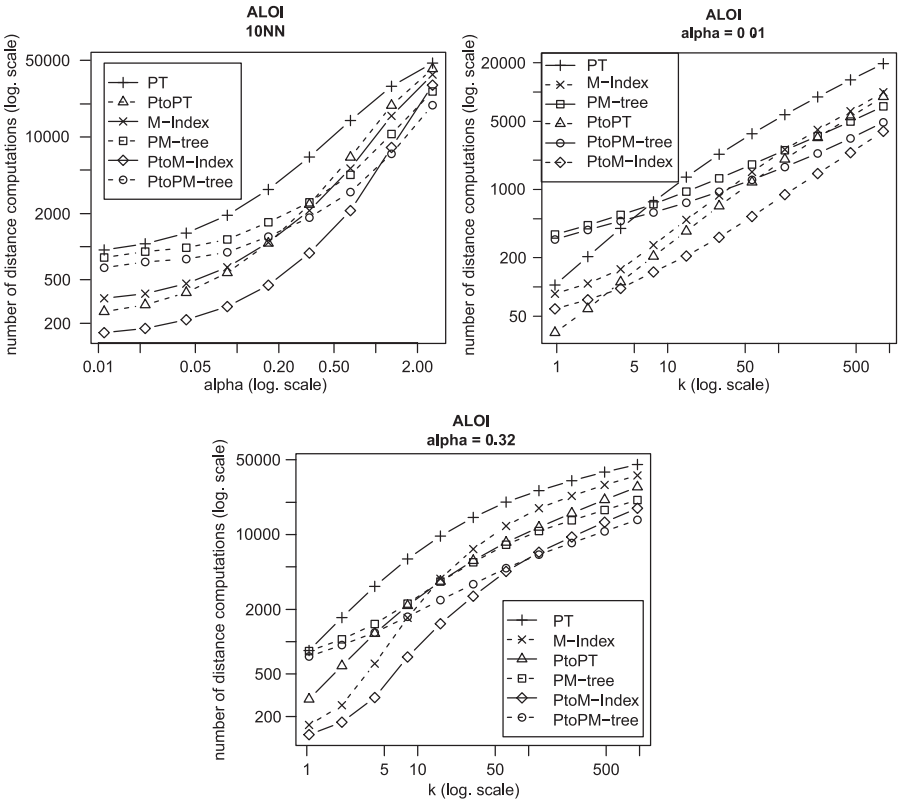


Fig. 17. Performance of all indexes (ALOI).

6.3. Discussion

In summary, we have experimentally proven that Ptolemaic (metric) access methods can challenge any metric access method when using SQFD. Even the very simple PtoPT can be a superior solution for indexing SQFD with low values of alpha, beating the state-of-the-art metric indexes by responding 1NN queries almost four times as fast compared to PT, three times as fast compared to M-Index and 10 times as fast compared to PM-Tree, respectively. When compared to the sequential scan, the speed-up factor of three orders of magnitude is even more remarkable. All Ptolemaic metric access methods show better resistance on the growing query radius and can be used for efficient similarity search in SQFD distance spaces with higher intrinsic dimensionality.

From another point of view, we have shown that the Ptolemaic filtering in PtoPT achieves the same filtering power as the triangle filtering in regular pivot tables using only 20% of the pivots. Let us remember that PtoPT is used also in PtoM-Index and is de facto part of PtoPM-Tree.[8] This suggests that PtoAMs can be used as an economical solution that needs only small time and space to construct an index, while achieving the same query efficiency as a metric index that is five times as large and five times as slow to construct.

Moreover, our method uses a two-pivot online pivot selection technique where the best pair of pivots is heuristically chosen for every single query and database object pair. In the metric indexing area this is the "holy grail" of pivot selection techniques, as a good pivot should be either close to the query or close to the database object. Instead of selecting pivots from the database like the offline techniques do, the pivot pairs are picked from a pre-selected set of pivots. It seems that the effort usually spent on the offline pivot selection process does not play such an important role here, as the main pivot pair selection procedure is performed online. The number of candidate pairs grows quadratically with the number of pivots, so even a pivot set that is bad from the metric indexing point of view could provide good pivot pairs for Ptolemaic filtering. However, additional analysis of this hypothesis has to be done in the future.

7. Conclusions

In this paper we have applied the principles of Ptolemaicindexing to existing metric indexing methods, yielding Ptolemaic Pivot Tables, the Ptolemaic PM-Tree, and the Ptolemaic M-Index. We have used these to index the signature quadratic form distance (SQFD), and found significant speed-up compared to the state-of-the-art

[8] The PD attribute in PtoPM-Tree ground entries stores distances to all global pivots.

metric indexing approach. Overall, our results have the immediate benefit of making similarity search more efficient for applications using the SQFD, but they are also important for Ptolemaicindexing in general. At present, the main family of Ptolemaicdistances of practical importance is quadratic form distances (QFDs) [4]. Because QFDs are expensive to compute, the internal overhead of Ptolemaicindexing is not an obstacle. However, as has been recently shown [9], static QFDs can be mapped to Euclidean distance for the purpose of indexing, making Ptolemaicindexing seemingly unfeasible. In this paper we show that the more expressive SQFDs are, in fact, equivalent to QFDs, except that their dynamic nature precludes this form of preprocessing. This, together with the increased performance over metric indexing, means that the matching of Ptolemaicindexing and SQFDs is a very natural and fruitful one.

Acknowledgments

We would like to thank Dr. David Novak and Dr. Michal Batko for the M-Index sources and their valuable consultations on them.

References

[1] J. Lokoč, M.L. Hetland, T. Skopal, C. Beecks, Ptolemaic indexing of the signature quadratic form distance, in: Proceedings of the Fourth International Conference on Similarity Search and Applications (SISAP), Springer, 2011.
[2] E. Chávez, G. Navarro, R. Baeza-Yates, J.L. Marroquín, Searching in metric spaces, ACM Computing Surveys 33 (3) (2001) 273–321. 〈doi:http://doi.acm.org/10.1145/502807.502808〉.
[3] P. Zezula, G. Amato, V. Dohnal, M. Batko, Similarity Search: The Metric Space Approach, Springer, 2006.
[4] M.L. Hetland, Ptolemaic indexing, arXiv:0911.4384 [cs.DS], 2009.
[5] C. Beecks, M.S. Uysal, T. Seidl, Signature quadratic form distances for content-based similarity, in: Proceedings of the 17th ACM International Conference on Multimedia, 2009.
[6] C. Beecks, M.S. Uysal, T. Seidl, Signature quadratic form distance, in: Proceedings ACM International Conference on Image and Video Retrieval, 2010.
[7] C. Beecks, M.S. Uysal, T. Seidl, A comparative study of similarity measures for content-based multimedia retrieval, in: Proceedings of the IEEE International Conference on Multimedia & Expo, 2010.
[8] C. Beecks, J. Lokoč, T. Seidl, T. Skopal, Indexing the signature quadratic form distance for efficient content-based multimedia retrieval, in: Proceedings of the ACM International Conference on Multimedia Retrieval, 2011.
[9] T. Skopal, T. Bartoš, J. Lokoč, On (not) indexing quadratic form distance by metric access methods, in: Proceedings of the Extending Database Technology (EDBT), ACM, 2011.
[10] P. Zezula, G. Amato, V. Dohnal, M. Batko, Similarity Search: The Metric Space Approach, Springer, 2005.
[11] H. Samet, Foundations of Multidimensional and Metric Data Structures, Morgan Kaufmann, 2006.
[12] M.L. Hetland, The basic principles of metric indexing, in: C.A.C. Coello, S. Dehuri, S. Ghosh (Eds.), Swarm Intelligence for Multi-objective Problems in Data Mining, Studies in Computational Intelligence, vol. 242, Springer, 2009.
[13] T.F. Havel, I.D. Kuntz, G.M. Crippen, The theory and practice of distance geometry, Bulletin of Mathematical Biology 45 (5) (1983) 665–720.
[14] G. Navarro, Analyzing metric space indexes: what for?, in: IEEE SISAP 2009, 2009, pp. 3–10.
[15] M.L. Mico, J. Oncina, E. Vidal, A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements, Pattern Recognition Letters 15 (1) (1994) 9–17. 〈http://doi:http://dx.doi.org/10.1016/0167-8655(94)90095-7〉.
[16] P. Ciaccia, M. Patella, P. Zezula, M-tree: an efficient access method for similarity search in metric spaces, in: VLDB'97, 1997, pp. 426–435.
[17] T. Skopal, Pivoting M-tree: a metric access method for efficient similarity search, in: Proceedings of the 4th Annual Workshop on DATESO, Desná, Czech Republic, ISBN 80-248-0457-3, also available at CEUR, vol. 98, ISSN 1613-0073 〈http://www.ceur-ws.org/Vol-98〉, 2004, pp. 21–31.
[18] T. Skopal, J. Pokorný, V. Snášel, Nearest neighbours search using the PM-tree, in: DASFAA '05, Beijing, China, Lecture Notes in Computer Science, vol. 3453, Springer, 2005, pp. 803–815.
[19] T. Skopal, Unified framework for fast exact and approximate search in dissimilarity spaces, ACM Transactions on Database Systems 32 (4) (2007) 1–46.
[20] D. Novak, M. Batko, Metric index: an efficient and scalable solution for similarity search, in: SISAP '09: Proceedings of the 2009 Second International Workshop on Similarity Search and Applications, IEEE Computer Society, Washington, DC, USA, 2009, pp. 65–73. 〈http://doi:http://dx.doi.org/10.1109/SISAP.2009.26〉.
[21] D. Novak, M. Batko, P. Zezula, Metric index: an efficient and scalable solution for precise and approximate similarity search, Information Systems 36 (4) (2011) 721–733.
[22] H.V. Jagadish, B.C. Ooi, K.-L. Tan, C. Yu, R. Zhang, iDistance: an adaptive B+-tree based indexing method for nearest neighbor search, ACM Transactions on Database Systems 30 (2) (2005) 364–397. 〈doi:http://doi.acm.org/10.1145/1071610.1071612〉.
[23] E.B. Reksten, Ptolemaic Indexing: An Evaluation, Master's Thesis, Norwegian University of Science and Technology, 2010.
[24] T. Skopal, On fast non-metric similarity search by metric access methods, in: Proceedings of the 10th International Conference on Extending Database Technology (EDBT'06), Lecture Notes in Compute Science, vol. 3896, Springer, 2006, pp. 718–736.
[25] E. Vidal, New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (AESA), Pattern Recognition Letters 15 (1) (1994) 1–7.
[26] K. Figueroa, E. Chávez, G. Navarro, R. Paredes, On the least cost for proximity searching in metric spaces, in: Proceedings of WEA, Lecture Notes in Computer Science, vol. 4007, Springer, 2006.
[27] S.E. Bratsberg, M.L. Hetland, Dynamic optimization of queries in pivot-based indexing, Multimedia Tools and Applications 60 (2) (2012) 261–275, http://dx.doi.org/10.1007/s11042-010-0614-z.
[28] M. Skala, Counting distance permutations, Journal of Discrete Algorithms 7 (2009) 49–61, http://dx.doi.org/10.1016/j.jda.2008.09.011. URL: 〈http://portal.acm.org/citation.cfm?id=1501025.1501131〉.
[29] E. Chávez, K. Figueroa, G. Navarro, Effective proximity retrieval by ordering permutations, IEEE Transactions on Pattern Analysis and Machine Intelligence 30 (9) (2008) 1647–1658.
[30] I.J. Schoenberg, On metric arcs of vanishing Menger curvature, Annals of Mathematics 41 (4) (1940) 715–726.
[31] J. Hafner, H.S. Sawhney, W. Equitz, M. Flickner, W. Niblack, Efficient color histogram indexing for quadratic form distance functions, IEEE Transactions on Pattern Analysis and Machine Intelligence 17 (7) (1995) 729–736.
[32] B.J. Malešević, The Möbius–Pompeïu metric property, Journal of Inequalities and Applications 2006 (83206) (2006) 1–9.
[33] T. Deselaers, D. Keysers, H. Ney, Features for image retrieval: an experimental comparison, Information Retrieval 11 (2) (2008) 77–107 . 〈http://doi:http://dx.doi.org/10.1007/s10791-007-9039-3〉.
[34] K. Mikolajczyk, C. Schmid, A performance evaluation of local descriptors, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (10) (2005) 1615–1630. 〈http://doi:http://dx.doi.org/10.1109/TPAMI.2005.188〉.
[35] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, 1967.
[36] J. Hafner, H.S. Sawhney, W. Equitz, M. Flickner, W. Niblack, Efficient color histogram indexing for quadratic form distance functions, IEEE Transactions on Pattern Analysis and Machine Intelligence 17 (1995) 729–736. 〈doi:http://doi.ieeecomputersociety.org/10.1109/34.391417〉.
[37] Y. Rubner, C. Tomasi, L.J. Guibas, The Earth Mover's distance as a metric for image retrieval, International Journal of Computer Vision 40 (2) (2000) 99–121.
[38] C. Beecks, A.M. Ivanescu, S. Kirchhoff, T. Seidl, Modeling image similarity by gaussian mixture models and the signature quadratic form distance, in: Proceedings of the 13th IEEE International Conference on Computer Vision, 2011, pp. 1754–1761.
[39] C. Beecks, A.M. Ivanescu, S. Kirchhoff, T. Seidl, Modeling multimedia contents through probabilistic feature signatures, in: Proceedings

of the ACM International Conference on Multimedia, 2011, pp. 1433 –1436.
[40] C. Beecks, T. Seidl, Analyzing the inner workings of the signature quadratic form distance, in: Proceedings of the IEEE International Conference on Multimedia and Expo, 2011.
[41] C. Beecks, T. Seidl, On stability of adaptive similarity measures for content-based image retrieval, in: Proceedings of the International Conference on MultiMedia Modeling, 2012.
[42] C. Beecks, M.S. Uysal, T. Seidl, Efficient *k*-nearest neighbor queries with the signature quadratic form distance, in: Proceedings of the IEEE International Conference on Data Engineering Workshops, 2010.
[43] C. Beecks, M.S. Uysal, T. Seidl, Similarity matrix compression for efficient signature quadratic form distance computation, in: Proceedings of the International Conference on Similarity Search and Applications, 2010.
[44] C. Beecks, M.S. Uysal, T. Seidl, L2-signature quadratic form distance for efficient query processing in very large multimedia databases, in: Proceedings of the International Conference on Multimedia Modeling, 2011, pp. 381–391.
[45] M. Krulis, J. Lokoc, C. Beecks, T. Skopal, T. Seidl, Processing the signature quadratic form distance on many-core gpu architectures, in: CIKM, 2011, pp. 2373–2376.
[46] M. Kruliš, T. Skopal, J. Lokoč, C. Beecks, Combining cpu and gpu architectures for fast similarity search, Distributed and Parallel Databases 30 (3–4) (2012) 179–207, http://dx.doi.org/10.1007/s10619-012-7092-4.
[47] I.J. Schoenberg, A remark on M.M. Day's characterization of inner-product spaces and a conjecture of L.M. Blumenthal, Proceedings of the American Mathematical Society 3 (6) (1952) 961–964.
[48] M.J. Huiskes, M.S. Lew, The mir flickr retrieval evaluation, in: Proceedings of the 1st ACM International Conference on Multimedia Information Retrieval, 2008, pp. 39–43.
[49] J.-M. Geusebroek, G.J. Burghouts, A.W.M. Smeulders, The Amsterdam library of object images, International Journal of Computer Vision 61 (1) (2005) 103–112. 〈http://doi:http://dx.doi.org/10.1023/B:VISI.0000042993.50813.60〉.
[50] W.K. Leow, R. Li, The analysis and applications of adaptive-binning color histograms, Computer Vision and Image Understanding 94 (1–3) (2004) 67–91. 〈http://doi:http://dx.doi.org/10.1016/j.cviu.2003.10.010〉.
[51] J. Lokoč, Cloud of Points Generator, SIRET Research Group 〈http://siret.ms.mff.cuni.cz/projects/pointgenerator/〉, 2010.
[52] F. Mémoli, G. Sapiro, Comparing point clouds, in: SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing, ACM, New York, NY, USA, 2004, pp. 32–40. 〈doi:http://doi.acm.org/10.1145/1057432.1057436〉.
[53] D. Novák, M. Batko, Java implementation of the M-Index similarity indexing structure, similarity group from Masaryk University, Brno, Czech Republic 〈http://mufin.fi.muni.cz/trac/m-index/〉, 2009.
[54] C.D. Manning, P. Raghavan, H. Schütze, Introduction to Information Retrieval, Cambridge University Press, New York, NY, USA, 2008.

Chapter 7

Combining CPU and GPU architectures for Fast Similarity Search

Martin Kruliš
Tomáš Skopal
Jakub Lokoč
Christian Beecks

Published in the *Distributed and Parallel Databases* journal, volume 30, Issue 3-4, pages 179–207. Springer, August 2012. ISSN 0926-8782.
dx.doi.org/10.1007/s10619-012-7092-4

Impact Factor in 2012: 0.806

Distrib Parallel Databases (2012) 30:179–207
DOI 10.1007/s10619-012-7092-4

Combining CPU and GPU architectures for fast similarity search

Martin Kruliš · Tomáš Skopal · Jakub Lokoč · Christian Beecks

Published online: 23 May 2012

Abstract The Signature Quadratic Form Distance on feature signatures represents a flexible distance-based similarity model for effective content-based multimedia retrieval. Although metric indexing approaches are able to speed up query processing by two orders of magnitude, their applicability to large-scale multimedia databases containing billions of images is still a challenging issue. In this paper, we propose a parallel approach that balances the utilization of CPU and many-core GPUs for efficient similarity search with the Signature Quadratic Form Distance. In particular, we show how to process multiple distance computations and other parts of the search procedure in parallel, achieving maximal performance of the combined CPU/GPU system. The experimental evaluation demonstrates that our approach implemented on a common workstation with 2 GPU cards outperforms traditional parallel implementation on a high-end 48-core NUMA server in terms of efficiency almost by an order of magnitude. If we consider also the price of the high-end server that is ten times higher than that of the GPU workstation then, based on price/performance ratio, the GPU-based similarity search beats the CPU-based solution by almost two orders of magnitude. Although proposed for the SQFD, our approach of fast GPU-based simi-

Communicated by: Kaushik Chakrabarti.

M. Kruliš · T. Skopal (✉) · J. Lokoč
SIRET Research Group, Faculty of Mathematics and Physics, Charles University in Prague, Prague, Czech Republic
e-mail: skopal@ksi.mff.cuni.cz

M. Kruliš
e-mail: krulis@ksi.mff.cuni.cz

J. Lokoč
e-mail: lokoc@ksi.mff.cuni.cz

C. Beecks
Data Management and Data Exploration Group, RWTH Aachen University, Aachen, Germany
e-mail: beecks@cs.rwth-aachen.de

larity search is applicable for any distance function that is efficiently parallelizable in the SIMT execution model.

Keywords Similarity search · Database indexing · Parallel computing · GPU · Pivot table · Metric · Ptolemaic · Multimedia databases

1 Introduction

Multimedia retrieval systems frequently store billions of images and provide users with different ways of searching and browsing (e.g., catalog-based or keyword-based search). However, effective yet efficient techniques for content-based similarity search are still a hot research topic. To this end, multimedia retrieval systems are designed based on advanced similarity models consisting of image representations and similarity/distance measures.

A flexible way to represent the content of an image is by means of *feature signatures* [28]. In general, a feature signature of an image is a set consisting of multiple local image features, where the length of a feature signature is not fixed (to distinguish images of different complexities). However, the comparison of feature signatures requires more sophisticated and computationally expensive adaptive distance measures [4], such as the Earth Mover's Distance (EMD) [28] or the Signature Quadratic Form Distance (SQFD) [3, 5]. In this paper, we focus on the latter, as the SQFD shows higher retrieval quality [4], higher stability [2], and lower time complexity compared to the EMD ($\mathcal{O}(n^2)$ vs. $\mathcal{O}(n^4)$). Nevertheless, the quadratic complexity is still too high to use the SQFD for a sequential search of a large database. In order to reduce the computational effort, indexing [31] approaches have been applied to the SQFD. It has been shown that *metric indexing* [1] and *ptolemaic indexing* [19] reach a speed-up of more than two orders of magnitude with respect to the sequential scan. However, even when using indexing approaches, the speed-up is generally limited due to the high intrinsic dimensionality [31]. Thus, in order to use the SQFD for large-scale image retrieval, we propose to parallelize the SQFD query processing.

Parallelization of data retrieval problems on many-core architectures has already been addressed from many perspectives. For instance the kNN query algorithm which is used in almost every data retrieval system has been successfully parallelized on GPUs by Bustos et al. [7] and later by Garcia et al. [10]. Pan et al. [26] showed that the solution can be improved even further using a hashing approach to compute the approximate kNN on GPUs. Other similarity-based operations can benefit from parallelization aswell. Lieberman et al. [18] suggested using GPUs for similarity joining operations. All these solutions exploited the parallel nature of GPUs to achieve significant speedup over CPU. However, the potential of the GPU lies especially in numeric computations, thus we can utilize its power even more efficiently to compute expensive distance functions that offer higher precision of the similarity search.

In this paper, we consider the combination of many-core GPU devices and multi-core CPU processors for parallel SQFD query processing. While parallel CPU processing is straightforward and supported by many development tools, designing efficient algorithms for GPUs is a challenging task for content-based retrieval purposes.

Although GPUs generally contain more cores than CPUs, they suffer from slow data transfer rates and code execution restrictions. We discuss GPU processing limitations and introduce two new schemes for efficient similarity search utilizing the combination of indexing approaches and the computational power of CPUs + GPUs.

The paper is organized as follows. Section 2 introduces the task of similarity search, the motivation and definition of the SQFD, and also the indexing techniques used for fast similarity search by the SQFD. Section 3 discusses the most important aspects of GPU architectures. The contribution of the paper are two algorithms addressing the implementation of similarity search on CPU and GPUs, described in Sects. 4 and 5. The first algorithm (SQFD-only) utilizes the GPUs only to compute the SQFD, leaving the other processing on CPU, while the second algorithm (SQFD + LB) utilizes the GPUs also to compute lower bound distances used in index pre-filtering. Section 6 presents the experimental results, and Sect. 7 concludes this paper.

2 Similarity search in multimedia databases

When searching multimedia databases by content, users issue similarity queries by selecting multimedia objects or by sketching the intended object contents. Given an example multimedia object or sketch q, the multimedia database $\mathbb{S} \subset \mathbb{U}$ (where $\mathbb{U}$ is the object universe) is searched for the most related objects with respect to the query by measuring the similarity between the query and each database object by means of a distance function δ. As a result, the multimedia objects with the lowest distance to the query are returned to the user. In particular, a *range query* (q, r), $q \in \mathbb{U}$, $r \in \mathbb{R}^+$, reports all objects in $\mathbb{S}$ that are within a distance r to q, that is, $(q, r) = \{x \in \mathbb{S} \mid \delta(x, q) \leq r\}$. The subspace defined by q and r is called the *query ball*. Another popular similarity query is the k *nearest neighbors query (kNN)*. It reports the k objects from $\mathbb{S}$ closest to q. That is, it returns the set $\mathbb{C} \subseteq \mathbb{S}$ such that $|\mathbb{C}| = k$ and $\forall x \in \mathbb{C}, y \in \mathbb{S} - \mathbb{C}, \delta(x, q) \leq \delta(y, q)$. The kNN query also defines a query ball (q, r), but the distance r to the kth NN is not known in advance.

2.1 Model representation

When determining content-based similarity between two multimedia objects, the *distance* is evaluated on *feature descriptors* which aggregate the inherent properties of the multimedia objects. The conventional feature descriptors aggregate and store these properties in *feature histograms*, which can be compared by vectorial distances [15, 27].

2.1.1 Feature signatures

Unlike conventional feature histograms, feature signatures are frequently obtained by clustering the objects' properties, such as color, texture, or other more complex features [9, 23], within a feature space and storing the cluster representatives and weights. Thus, given a feature space $\mathbb{F}$, the *feature signature* S^o of a multimedia

Fig. 1 Three example images with their corresponding feature signature visualizations

object o is defined as a set of tuples from $\mathbb{F} \times \mathbb{R}^+$ consisting of representatives $r^o \in \mathbb{F}$ and weights $w^o \in \mathbb{R}^+$.

We depict an example of image feature signatures according to a feature space comprising position, color and texture information, i.e. $\mathbb{F} \subseteq \mathbb{R}^7$, in Fig. 1. For this purpose we applied the k-means clustering algorithm where each representative $r_i^o \in \mathbb{F}$ corresponds to the centroid of the cluster $\mathcal{C}_i^o \subseteq \mathbb{F}$, i.e., $r_i^o = \frac{\sum_{f \in \mathcal{C}_i^o} f}{|\mathcal{C}_i^o|}$, with relative frequency $w_i^o = \frac{|\mathcal{C}_i^o|}{\sum_i |\mathcal{C}_i^o|}$. We depict the feature signatures' representatives by circles in the corresponding color. The weights are reflected by the diameter of the circles. As can be seen in this example, feature signatures adjust to individual image contents by aggregating the features according to their appearance in the underlying feature space.

2.1.2 Signature quadratic form distance

The Signature Quadratic Form Distance (SQFD) [3, 5] is an adaptive distance-based similarity measure for the comparison of feature signatures, generalizing the classic vectorial Quadratic Form Distance (QFD) [12]. It is defined as follows.

Definition 1 (SQFD) Given two feature signatures $S^q = \{\langle r_i^q, w_i^q \rangle\}_{i=1}^n$ and $S^o = \{\langle r_i^o, w_i^o \rangle\}_{i=1}^m$ and a similarity function $f_s : \mathbb{F} \times \mathbb{F} \to \mathbb{R}$ over a feature space $\mathbb{F}$, the *signature quadratic form distance* SQFD_{f_s} between S^q and S^o is defined as:

$$\mathrm{SQFD}_{f_s}\left(S^q, S^o\right) = \sqrt{(w_q \mid -w_o) \cdot A_{f_s} \cdot (w_q \mid -w_o)^T},$$

where $A_{f_s} \in \mathbb{R}^{(n+m)\times(n+m)}$ is the similarity matrix arising from applying the similarity function f_s to the corresponding feature representatives, i.e., $a_{ij} = f_s(r_i, r_j)$. Furthermore, $w_q = (w_1^q, \ldots, w_n^q)$ and $w_o = (w_1^o, \ldots, w_m^o)$ form weight vectors, and

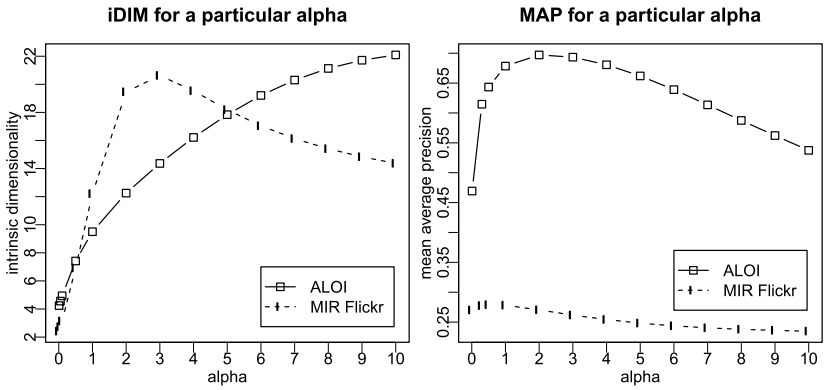


Fig. 2 The impact of α on the intrinsic dimensionality and mean average precision

$(w_q \mid -w_o) = (w_1^q, \ldots, w_n^q, -w_1^o, \ldots, -w_m^o)$ denotes the concatenation of weights w_q and $-w_o$.

The similarity function f_s is used to determine similarity values between all pairs of representatives from the feature signatures. In our implementation we use the similarity function $f_s(r_i, r_j) = e^{-\alpha L_2(r_i, r_j)^2}$, where α is a constant for controlling the precision-indexability tradeoff, as investigated in our previous works [1, 19], and L_2 denotes the Euclidean distance. In particular, lower values of the parameter α lead to better indexability, that is, to a smaller *intrinsic dimensionality* (iDIM) [8]. However, lower values of the parameter α also decrease the retrieval effectiveness (frequently measured in terms of *mean average precision* values), as can be seen in Fig. 2 for the ALOI [11] and MIR Flickr [16] databases as examples. On the contrary, the best mean average precision values can be reached using a large value of the parameter α making the SQFD space no longer indexable. In such cases a parallel query processing approach could be one feasible solution to significantly speedup the search process. Nevertheless, before we proceed to the parallel implementation of the SQFD query processing, we briefly summarize available indexing methods.

2.2 Indexing

When processing content-based similarity queries by the naïve sequential scan, the computation of the SQFD has to be carried out for each database object individually. Unlike the cheap L_p distances, the SQFD is of more than quadratic time complexity (w.r.t. dimension), so the sequential scan, sometimes acceptable for L_p distances, is impractical for the SQFD even on a moderately sized database. Although it has been shown that the SQFD is a generalization [5] of the well-known Quadratic Form Distance [12], recent approaches indexing the data by a homeomorphic mapping into the Euclidean space [30] can not be applied to the SQFD, as the similarity matrix changes from computation to computation.

Nevertheless, recent papers showed that SQFD can be indexed by metric access methods [1] and ptolemaic indexing [19], achieving a speed-up of up to two orders

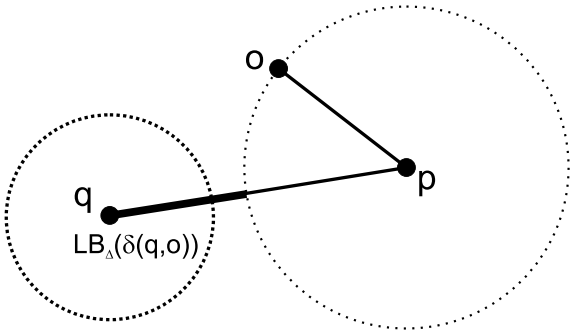


Fig. 3 Lower-bound distance computed using triangle inequality and a single pivot

of magnitude with respect to the sequential scan. In this section we review both approaches and detail the simplest and most intuitive metric/ptolemaic index: the *pivot tables*.

2.2.1 Metric indexing

A *metric space* $(\mathbb{U}, \delta)$ consists of a feature descriptor domain $\mathbb{U}$ (in this paper, the set of all possible signatures) and a distance function δ which has to satisfy the metric postulates: *identity*, *non-negativity*, *symmetry*, and *triangle inequality*. In this way, metric spaces allow domain experts to model their notion of content-based similarity by an appropriate feature representation and distance function serving as similarity measure. At the same time, this approach allows database experts to design index structures, so-called *metric access methods* (or metric indexes) [8, 13, 29, 31], for efficient query processing of content-based similarity queries in a database $\mathbb{S} \subset \mathbb{U}$. These methods rely on the distance function δ only, i.e., they do not necessarily know the structure of the feature representation of the objects.

Metric access methods (or metric indexes) organize database objects $o_i \in \mathbb{S}$ by grouping them based on their distances, with the aim of minimizing not only traditional database costs like I/O but also the number of costly distance function δ evaluations—in our case the number of SQFD evaluations. For this purpose, nearly all metric access methods apply some form of filtering based on cheap computation of *lower bounds* $\mathrm{LB}_{\Delta}(\delta(q, o))$. These bounds are based on the fact that exact pivot–object distances are precomputed, where *pivot* is a suitable reference object selected from the database $\mathbb{S}$.

We illustrate this fundamental principle in Fig. 3 where we depict the query object $q \in \mathbb{U}$, some pivot object $p \in \mathbb{S}$, and a database object $o \in \mathbb{S}$ in some metric space. Given a range query (q, r), we wish to estimate the distance $\delta(q, o)$ by making use of $\delta(q, p)$ and $\delta(o, p)$, with the latter already stored in the metric index. Because of the triangle inequality, we can safely filter object o without needing to compute the (costly) distance $\delta(q, o)$ if the triangular lower bound

$$\mathrm{LB}_{\Delta}\big(\delta(q, o)\big) = \big|\delta(q, p) - \delta(o, p)\big|, \tag{1}$$

also known as the *inverse triangle inequality*, is greater than the query radius r. The SQFD has been proved [19] to be a metric distance, so metric indexing can be applied for efficient similarity search using SQFD.

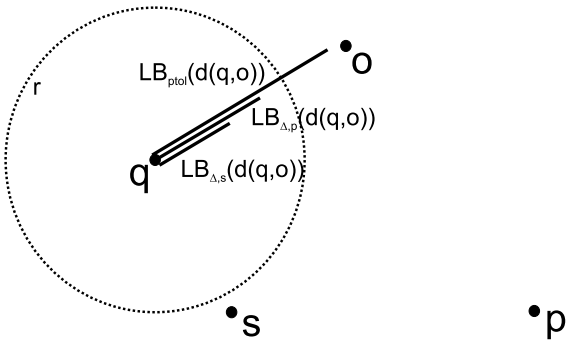


Fig. 4 Comparison of triangle/Ptolemy's lower-bound distances computed for two pivots

2.2.2 *Ptolemaic indexing*

In metric indexes, the triangle inequality is used to construct lower bounds for the distance. Analogously, in *Ptolemaic indexing* [14, 19], *Ptolemy's inequality* is used to construct such lower bounds as well. A distance function is called a *Ptolemaic distance* if it has the properties of *identity*, *non-negativity*, and *symmetry*, and satisfies *Ptolemy's inequality*. If a Ptolemaic distance also satisfies the *triangle inequality*, it is a *Ptolemaic metric*.

Ptolemy's inequality states that for any quadrilateral, the pairwise products of opposing sides sum to more than the product of the diagonals. In other words, for any four points $x, y, u, v \in \mathbb{U}$, we have the following:

$$\delta(x, v) \cdot \delta(y, u) \leq \delta(x, y) \cdot \delta(u, v) + \delta(x, u) \cdot \delta(y, v) \tag{2}$$

One of the ways the inequality can be used for indexing is in constructing a pivot-based lower bound. For a query q, object o, and pivots p and s, we get the *candidate bound*:

$$\delta_{\mathrm{C}}(q, o, p, s) = \frac{|\delta(q, p) \cdot \delta(o, s) - \delta(q, s) \cdot \delta(o, p)|}{\delta(p, s)} \tag{3}$$

For simplicity, we let $\delta_{\mathrm{C}}(q, o, p, s) = 0$ if $\delta(p, s) = 0$. As for triangular lower-bounding, one would normally have a set of pivots $\mathbb{P}$, and the bound can then be maximized over all (ordered) pairs of distinct pivots drawn from this set, giving us the final Ptolemaic bound [14, 19]:

$$\delta(q, o) \geq \mathrm{LB}_{\mathrm{ptol}}\big(\delta(q, o)\big) = \max_{p,s \in \mathbb{P}} \delta_{\mathrm{C}}(q, o, p, s) \tag{4}$$

As for the triangular case, the Ptolemaic lower bound $\mathrm{LB}_{\mathrm{ptol}}$ could be used to filter objects not contained in the query ball, i.e., exclude those $o_i \in \mathbb{S}$ from search for which $\mathrm{LB}_{\mathrm{ptol}}(\delta(q, o_i)) > r$.

Figure 4 illustrates a comparison (in two-dimensional Euclidean space) showing that ptolemaic indexing could provide much tighter lower bounds. Having two pivots s, p, both lower bounds constructed using triangle inequality would not filter the object o from search, as the value is lower than a radius of the range query r. On the other hand, the lower bound obtained using the ptolemaic approach leads to very tight distance approximation, and so filtering the object o from search.

Luckily, the SQFD has been proved [19] to be both a metric and a ptolemaic distance, so ptolemaic indexing can be applied for efficient similarity search using SQFD.

2.3 Pivot tables

One of the most efficient (yet simple) indexes for similarity search is the *pivot table* [24], originally introduced as LAESA [22]. Basically, the structure of a pivot table is a simple matrix of distances $\delta(o_i, p_j)$ between the database objects $o_i \in \mathbb{S}$ and a pre-selected static set of m pivots $p_j \in \mathbb{P} \subset \mathbb{S}$. For querying, pivot tables allow us to perform cheap lower-bound filtering by computing the maximum lower bound to $\delta(q, o)$ using all the pivots. Moreover, the lowerbounding could be coupled with the querying more tightly because of the monotonous increase of the lower bound during its computation (i.e., usage of an additional pivot leads to possibly tighter/greater value). In particular, if the actual value of the lower bound being computed exceeds the radius of a query, the computation of the lower bound can be safely terminated and the object filtered out from further processing (so-called *early termination* optimization).

Although pivot tables have been introduced as a metric index, they could be used beyond the context of the metric space model. In fact, the data structure is just a distance matrix, so there is no metric-specific aggregate information stored (unlike in hierarchical metric indexes) that would prevent from usage elsewhere. In consequence, the original filtering based on triangular lower bounds (1) can be easily extended to the ptolemaic case using (4), or even combined. This extension was already presented as *ptolemaic pivot tables* [14, 19]. Because in the ptolemaic case there are pairs of pivots used in the lowerbounding, the quadratic size could lead to a large internal overhead when filtering. Therefore, there were also heuristics proposed for reduction of the set of pivot pairs yet preserving their filtering power [19]. It was experimentally confirmed that ptolemaic indexing could speedup the SQFD similarity search up to 4 times with respect to the metric case and up to 300 times with respect to the sequential scan [19].

2.4 Motivation for parallel indexing

The feature signatures and SQFD have been proved as an elegant and effective model for similarity search allowing to compare multimedia descriptors based on local features. There was also substantial effort spent on speeding up the SQFD search using the metric and ptolemaic indexing. However, despite these advances the SQFD similarity search is still not prepared for large-scale applications. Let us now analyze the empirical evidence. Depending on the parameter α of the internal SQFD's similarity function f_s, where higher α lead to more precise but slower search, the single-core query times on Intel Xeon X5660 using a 25,000 database range from 150 ms to 1 s per query (see [19]). Obviously, even when the search complexity was heavily reduced by the ptolemaic indexing (two orders of magnitude), the practical performance is still not sufficient. In order to achieve competitive performance, it seems necessary to parallelize the approach and reduce the real times by another two orders

of magnitude, yet keeping the hardware platform cheap (using common GPU cards). Accomplishing this goal would enable searching databases comprising millions of multimedia objects in real time.

In the rest of the paper we propose two algorithms. The *SQFD-only algorithm* parallelizes only the SQFD computation on GPU, leaving the other processing on CPU (work dispatching, pivot table filtering, results aggregation). This approach is efficient in case the workloads of SQFD computations and index filtering are balanced, so that GPU need not to wait for CPU. However, advanced filtering techniques (e.g., ptolemaic indexing) reduce the workload of SQFD computations by pruning a number of candidates, thus shifting the workload from GPU to CPU. For such cases we propose the SQFD + LB *algorithm* that precomputes on GPU the lower-bound values used by candidate pre-filtering, reducing thus the workload of CPU.

3 GPU fundamentals

GPU architectures [25] differ from CPU architectures in multiple ways. In the remainder of this section, we describe the GPU device architecture and its two major aspects, the *thread execution* and the *memory organization*, which have direct impact on the design of our framework and the SQFD implementation. The following description may be incomplete or simplified as we focus mainly on details important for GPU programming.

3.1 GPU architecture

A GPU card is a peripheral device connected to the host system via the PCI-Express (PCIe) bus. The device consist of a GPU processor and on-board memory modules. The device also consists of other parts related to image processing, but they are out of scope of our method.

The GPU processor (Fig. 5) consists of several symmetric multiprocessing units (SMPs), while the SMPs share only the main memory bus and the L2 cache, otherwise they are completely independent. Each SMP consists of multiple GPU cores, single instruction decoder, L1 cache, and local memory. The GPU cores are tightly coupled since they share SMP resources, even the instruction decoder. As a result, all cores execute the same instruction at the same time. Each core has its own arithmetical units for integer and float operations and a private set of registers.

The most significant differences from the classic CPU architecture is the specific instruction execution by multiple cores in SMP and also multiple types of memory. Therefore, we address these issues in more detail in the following.

3.2 Thread execution

When it comes to parallel execution, we usually distinguish between two types of parallelism—*task parallelism* and *data parallelism*. The task parallelism is usually employed by CPUs as each core executes different code. In case of data parallelism, all cores execute the same code but on different portions of data. The GPUs are tailored to data parallelism since their original graphic-acceleration design is aimed at

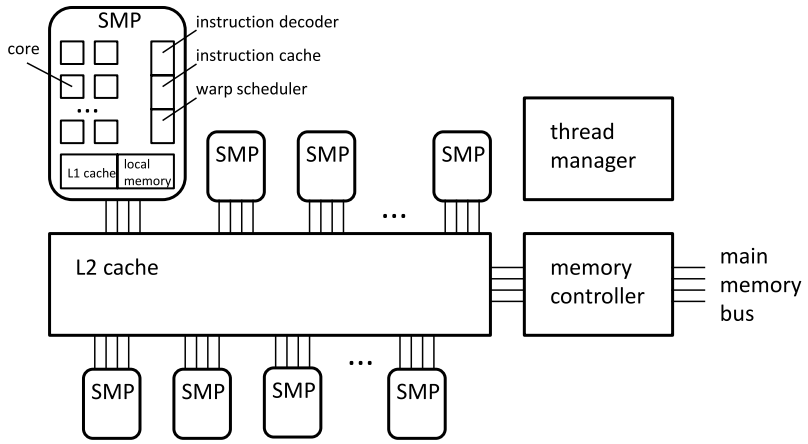


Fig. 5 GPU processor architecture

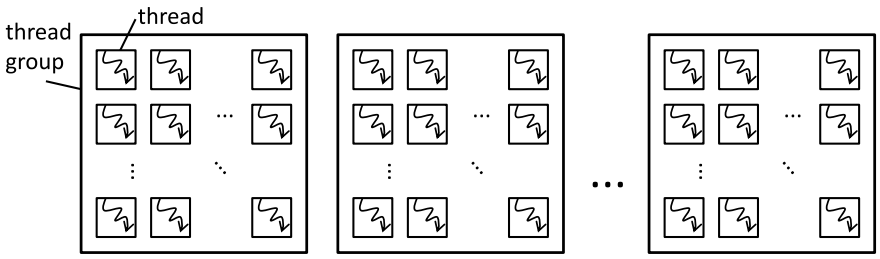


Fig. 6 Example of thread allocation and grouping

processing large number of geometric vertices or image fragments using the same algorithm.

The portions of code that are executed on the GPU are called *kernels*. A kernel is a procedure that is invoked multiple times simultaneously, thus spawning multiple threads that execute the same code. Each spawned thread gets the same set of calling arguments and a unique identifier which is used to select the proper parts of the parallel work. The threads are organized into one-, two-, or three-dimensional array and the thread identifier is an index into this array. The thread managing and context switching capabilities of the GPU are very advanced. Thus, it is usually better to create a multitude of threads, even if they execute only a few instructions each, in order to optimize the load balancing. In addition, fast context switching capabilities of the GPU are used to inhibit the latency of global memory transactions.

Threads are aggregated into small bundles called *groups* (Fig. 6). A group usually contains tens to hundreds of threads which are mapped to one SMP unit, thus executing the kernel code in SIMT (Single Instruction Multiple Threads) or virtual SIMT fashion. Usually, there are many more thread groups than SMPs, where the groups are planned sequentially and non-preemptively on available multiprocessors. When a group is assigned to an SMP, it must finish its execution before another group can

be assigned to that SMP. Therefore, threads in one group must not wait for results of another group, because such behavior could easily lead to a deadlock.

Threads in a group are divided into subgroups called *warps* (NVIDIA) or *wavefronts* (ATI/AMD). The number of threads in these subgroups is equal to the number of GPU cores in SMPs, so threads in a subgroup run in real SIMT mode. Exactly one subgroup is actually running while others are waiting. When a subgroup is forced to wait (e.g., when transferring data from memory), SMP performs a fast context switch so that another subgroup may compute meanwhile.

The SIMT execution suffers from branching problems. When different threads in the group choose different branches—for instance when executing 'if' statements—all branches must be executed by all threads. Each thread masks instruction execution according to local result of the condition to ensure correct results. Therefore, heavily branched code or 'while' loops with highly different number of iterations will not perform well on GPUs. On the other hand, the SIMT approach simplifies synchronization within the group and allows threads to communicate and collaborate through SMP's shared local memory.

3.3 Memory organization

The second difference is the memory organization which is depicted in Fig. 7. As we can observe, there are four types of memory:

- host memory (RAM),
- global memory (VRAM),
- local memory,
- and private memory (GPU core registers).

The *host memory* is the operational memory of the computer. It is directly accessible by the CPU, but it cannot be accessed by any peripheral devices such as the GPU. Input data needs to be transferred from the host memory (RAM) to the graphic device

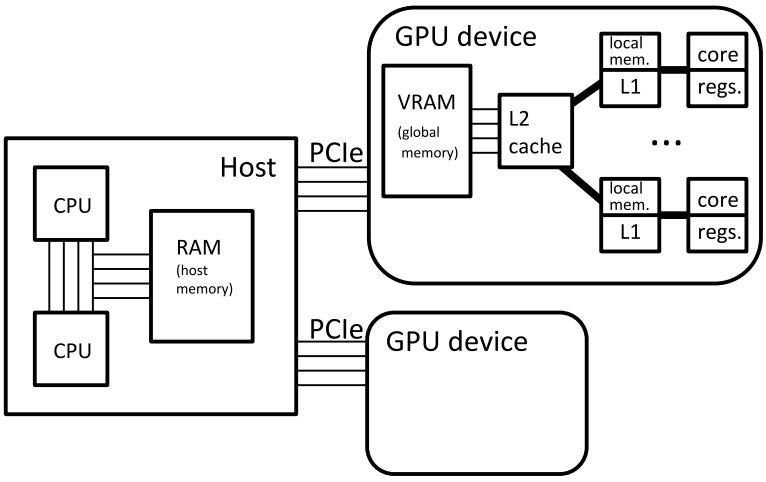


Fig. 7 Memory organization scheme of host and GPU device

global memory (VRAM), and the results need to be transferred back when the kernel execution finishes. For the transfer the PCI-Express bus is used, which is rather slow (8 GB/s) when compared to the internal memory buses.

The *global memory* is directly accessible from GPU cores, while input data and the results computed by a kernel are stored here. The global memory bus shows both high latency and high bandwidth. In order to access the global memory optimally, threads in one group are encouraged to use *coalesced loads*. A coalesced load is performed when all threads of a group load or store a contiguous memory area, so that each thread transfers a single 4-byte word of this block.

The *local memory* is shared among threads within one group. It is very small (tens of kB) but almost as fast as the GPU registers. The local memory can play the role of a program-managed cache for global memory, or the threads may share intermediate results in here while they cooperate on a task. The memory is divided into several (usually 16 or 32) banks. Two subsequent 4-byte words are stored in two subsequent banks (modulo number of banks). When two threads access the same bank (except if they read the same address), the memory operations are serialized which creates undesirable delay for all threads due to the SIMT execution model.

Finally, the *private memory* belongs exclusively to a single thread and corresponds to the GPU core registers. Private memory size is very limited (tens to hundreds of words), therefore it is suitable just for a few local variables.

3.4 Summary

Finally, we would like to summarize the implications for our implementation.

- The latency of data transfers between the host system and the GPU devices needs to be inhibited. The best way is to form a pipeline so that one block of data is being transferred to GPU, one block of data is being processed and one block of results is being transferred from GPU at the same time. Furthermore, the processing of a data block should take at least as much time as its transfer.
- Each algorithm being adapted for GPU must be carefully analyzed and its data transfers must be planned according to memory limitations of the GPU. The utilized data structures need to be designed with respect to the memory architecture, so that data can be fetched by coalesced loads from global memory and bank conflicts do not occur when accessing data in local memory by individual threads.
- Furthermore, the algorithm must embrace the SIMT execution model, at least for the parts of the work processed by one thread group. Usually, it is not feasible to parallelize an algorithm by simply assigning its inner loop to every spawned thread as the resources of the threads are limited. In such cases the algorithm must be redesigned so that threads of one group collaborate more closely and share their resources.
- Multitude of threads (at least thousands) needs to be spawned in order to utilize all available cores and balance the load efficiently.

4 Similarity search using GPU

The most time consuming operation in a search engine employing SQFD for similarity search is the computation of a distance between two signatures. This operation takes $\mathcal{O}((m+n)^2)$ time, where m, n are the sizes of signatures being compared. Even when using indexing techniques that massively reduce the number of SQFD computations needed to compute, such as the pivot tables, there still remains a set of candidate database signatures that has to be filtered using direct SQFD computations.

Therefore, our primary objective is to utilize the computational power of GPU to calculate distances between query and database signatures in parallel. In our approach, we consider both the parallel execution of multiple SQFD computations during the query evaluation as well as the parallel computation of a single SQFD between two feature signatures.

4.1 Computing multiple distances in parallel

Since the SQFD is computed between the query signature and many database signatures, it would be inefficient to execute each computation separately on the GPU due to high latencies caused by data transfer and kernel executions. Therefore, we perform a block-wise computation of multiple SQFDs in parallel. Each block contains $N+1$ feature signatures. The first feature signature is the query signature and remaining N feature signatures are the database signatures, thus each block yields a vector of N distances as a result. The choice of N is essential for good performance. In general, a large number of N performs better.

The query processor treats the GPU implementation of the SQFD as an asynchronous operation that does not block the CPU when started, so the system can wait for its termination. The system may start as many operations as required, while the operations are queued and distributed over available GPU devices equally.[1] Since the architecture is flexible and leaves the CPU relatively low-utilized, it could be easily used with a distance-based index implemented in the CPU part of the system.

4.2 Computing each distance in parallel

In case of multi-core CPUs, computing multiple distances in parallel would be sufficient to achieve optimal speedup, since the number of distances computed vastly exceeds the number of available cores. Unfortunately, the same approach is not feasible on GPUs. The signatures need to be cached in local memory of the SMP, which is very limited, so they are able to accommodate just a few signatures. Furthermore, it would produce very imbalanced tasks for the threads in one group, which are running

[1] In theory, two subsequent blocks dispatched to the same GPU device may overlap in some operations. Modern GPUs have independent units for host-device memory transfers, therefore it should be possible to overlap data transfer and SQFD computation of two subsequent blocks. In order to do so, the size of the block needs to be restricted so that at least two data blocks would fit the GPU device memory. Unfortunately, we have encountered many technical problems when attempting to pipeline execution and data transfers. It is our belief that these problems are caused by flaws in hardware drivers and/or OpenCL implementation.

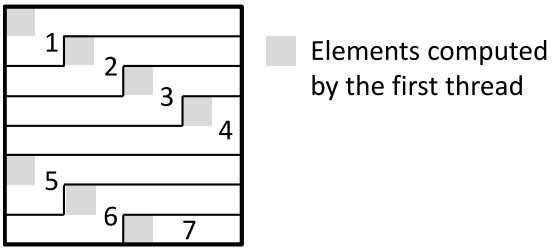


Fig. 8 Work decomposition when computing the similarity matrix A_{f_s} and $\overline{A}$

in SIMT fashion on one SMP. Hence, to efficiently utilize all the cores on the SMP unit we need to compute each distance in parallel as well.

Each SQFD is computed by a group of 256 threads, thus $256 \times N$ threads are spawned for one block. The constant 256 was selected based on current hardware capabilities. We have assigned one thread group to compute a single SQFD in a block, because these threads benefit from shared local memory, as the group does cache the input data from global memory and keeps intermediate results. Using multiple groups to compute one SQFD would be problematic as the groups do not have any effective means of communication. The opposite approach (using one group to compute multiple SQFDs) is feasible. However, in case of sufficient signature lengths, the parallelism would not be exploited any further and many technical complications would arise due to the limited size of local memory.

The SQFD between two feature signatures has been defined in Definition 1. For the sake of parallelism, we compute the elements of the similarity matrix A_{f_s} concurrently by available threads in the group. Each element of the matrix is multiplied with the corresponding weights of $\overline{w} = (w_q \mid -w_o)$, so that new matrix $\overline{A}$ is created, where $\overline{A}_{(i,j)} = \overline{w}_{(j)} A_{f_s(i,j)} \overline{w}_{(i)}$. Finally, we compute a sum of every element in the matrix $\overline{A}$ and we find its square root. These modifications are direct applications of distributivity and associativity laws, thus the result will not be affected in any way. The SQFD GPU implementation has the following phases:

1. Load feature signatures into local memory.
2. Compute the similarity matrix A_{f_s} and multiply its elements by corresponding elements in the weight vectors (creating $\overline{A}$).
3. Sum up elements in the matrix $\overline{A}$ and yield the square root.

In the first phase, data are loaded into local memory as they are required multiple times and it would be ineffective to load them from global memory each time. Furthermore, the loading is more efficient when all threads cooperate in coalesced loads. The similarity matrix has $(m+n) \times (m+n)$ entries, where m and n are the numbers of feature representatives in S^q and S^o, respectively. Since $m+n$ is usually smaller than 256 and varies for each pair of feature signatures, we use an irregular mapping of similarity matrix elements to threads. Figure 8 depicts the mapping scheme, where each area represents elements being computed in parallel. The numbers indicate consecutive (serial) steps in which the element areas are processed. In the last step the remaining area of the similarity matrix could be smaller than the total number of threads. In such case some threads remain idle.

In the second phase the matrices are not stored in memory but rather computed on-the-fly since only a sum of elements in $\overline{A}$ is required. When a thread computes a

new element in the similarity matrix, its value is added to a partial sum and the element itself is discarded. Even though this method requires significantly less memory, it creates a synchronization problem as multiple elements are being computed and added to the partial sum concurrently. To avoid explicit synchronization, every thread is provided with its own instance of the partial sum.

When the second phase terminates, the total sum of the partial sums of each thread is computed as the third phase of the algorithm. The total sum is only computed by the first thread in the group, which is also responsible for determining the square root and for writing the computed distance into the global memory. The total sum can also be computed cooperatively by all threads using reduction tree of logarithmic depth. However, such improvement has no measurable impact on the performance as the time required by the second phase dominates significantly the time required by the final summation.

4.3 The SQFD-only algorithm

The above described parallel computation of (multiple) SQFDs could be utilized in query processing, either directly in sequential scan of the entire database, or with the pivot table index. The *SQFD-only algorithm* utilizing the pivot table is depicted in Fig. 9.[2] When a query is started, the algorithm computes the SQFD distances between query and pivot objects (signatures). These distances are used by the pivot table for construction of lower bounds. Then, the pre-filtering based on the lower bounds takes place, resulting in a set of remaining candidate objects that have to be filtered using the expensively computed SQFDs. As depicted in the figure, only the SQFD computations take place on the GPU, while the lower bound construction, pre-filtering and filtering steps are performed on CPU. Since the computation of SQFDs is assumed as the most expensive operation, the rest of the functionality is left to the CPU. Moreover, because both the construction of lower bounds and the pre-filtering steps are implemented together on CPU, the lower bound computation can benefit from the early termination optimization (see Sect. 2.3).

In summary, the CPU iterates over the entire database, pre-filters the all the objects using the pivot table, and asynchronously dispatches blocks of candidate objects (signatures) to the GPU. The GPU computes the distances for each block and sends them back to CPU. Finally, the CPU compares the distances against the query range and forms the results set of objects.

4.4 Integration to indexing and query processor

We have described how to compute distances between a query signature and a block of database signatures on the GPU and also how to integrate such parallel computation of SQFD into a query algorithm using the pivot table index. In the remainder of this section we detail how to integrate the SQFD-only algorithm into a database indexer and query processor that evaluates range and kNN queries.

[2]We use a kind of schema together with a conceptual explanation of the algorithm, because a code listing in parallel framework would be not as concise and easy to read.

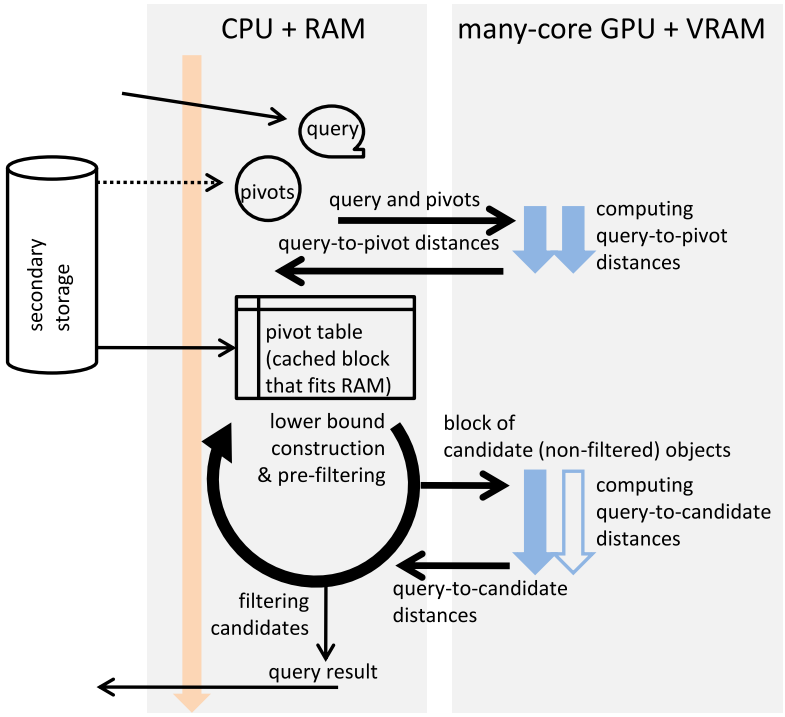


Fig. 9 Workflow of the SQFD-only algorithm

4.4.1 Computing pivot table

When a database of signatures is being indexed, a pivot table needs to be computed. The pivot table consists of distances from selected pivots to all objects in the database. Even though these distances are computed only when new objects are inserted, we can easily modify the SQFD-only algorithm to construct the pivot table in parallel as well. In order to do so, we disable the lower bound construction and pre-filtering steps and execute a query for each pivot object. Moreover, no result set is formed in the filtering step but the distances are saved into the pivot table instead.

4.4.2 Range query

The sequential range query algorithm (i.e., without an index) is easy to implement by the SQFD-only algorithm. The database is divided into blocks of appropriate size[3] and all blocks are enqueued for GPU processing. The system waits for all SQFD computations to complete, and the computed distances are filtered on the CPU to exclude objects outside of the query range. Hence, the lower bound construction and pre-filtering steps are just omitted (all database objects are candidates).

When using the pivot table index (either metric or ptolemaic variant), the SQFD-only algorithm is used as described. In the pre-filtering and filtering steps the actual

[3] As mentioned before, the larger the better.

radius of the range query is used. Concerning the blocks of signatures that are dispatched to GPU, block size of 128–256 for $\alpha = 0.01$ and 1024–2048 for $\alpha = 0.2$ were observed as empirically optimal (see Sect. 6.5.1).

4.4.3 kNN query

The kNN query evaluation is slightly more complicated. When no indexing is used, it works very much like sequential range query. When the pivot table indexing is employed, some additional modifications are required. The problem is that the kNN query has no fixed query range for the pivot table pre-filtering, as this range is dynamically refined during the kNN query processing using heuristics. In order to adapt to the heuristics, we limit the block size to a value between 64 and 512 (depending on index type and α value). Also, there are at most as many blocks pending as there are the GPU devices available. These constants have been chosen empirically[4] (see Sect. 6.5.2). When the limit of pending blocks is reached, the system waits for the first enqueued block to finish, its results are taken, and the query range is refined. This way a pipeline effect is achieved, so that the CPU pre-filters the database objects and refines the resulting kNN set while the GPU computes the SQFD.

5 Moving the index to GPU

The design of the basic SQFD-only algorithm assumes that implementing SQFD computation as an asynchronous operation performed on GPU leaves the CPU rather low-utilized and so capable of performing other tasks like lower bound construction, pre-filtering, and filtering. Although this holds true for the metric version of pivot table, the lower bound construction step becomes quite expensive when using the ptolemaic version (or combined ptolemaic and metric version). Instead of taking the maximum value over the p lower bounds, in the ptolemaic case we need to maximize over up to $\mathcal{O}(p^2)$ bounds (see Sect. 2.3 for details). The SQFD-only algorithm, when applied on the ptolemaic pivot table, cannot fully utilize the GPUs due to the CPU, which is overloaded by the lower bound construction. In consequence, the CPU cannot timely dispatch the blocks of signatures to GPUs and these must wait (see the experiments for empirical evidence). To overcome this bottleneck, in this section we propose the SQFD + LB algorithm that moves the lower bound construction to GPU, thus reducing the computational load of CPU.

5.1 The SQFD + LB algorithm

The SQFD + LB *algorithm* is depicted in Fig. 10. The main difference is that the query evaluation is divided into two stages. In the first (new) stage, the lower bound construction step is moved to GPU. The second stage works as the original SQFD-only algorithm, except that the CPU has much less work due to the lower bounds constructed in the first stage.

[4] Actually, these constants are suitable only for $\alpha = 0.2$ and $\alpha = 0.01$. The value $\alpha = 3$ requires the largest possible blocks since it does not benefit much from indexing.

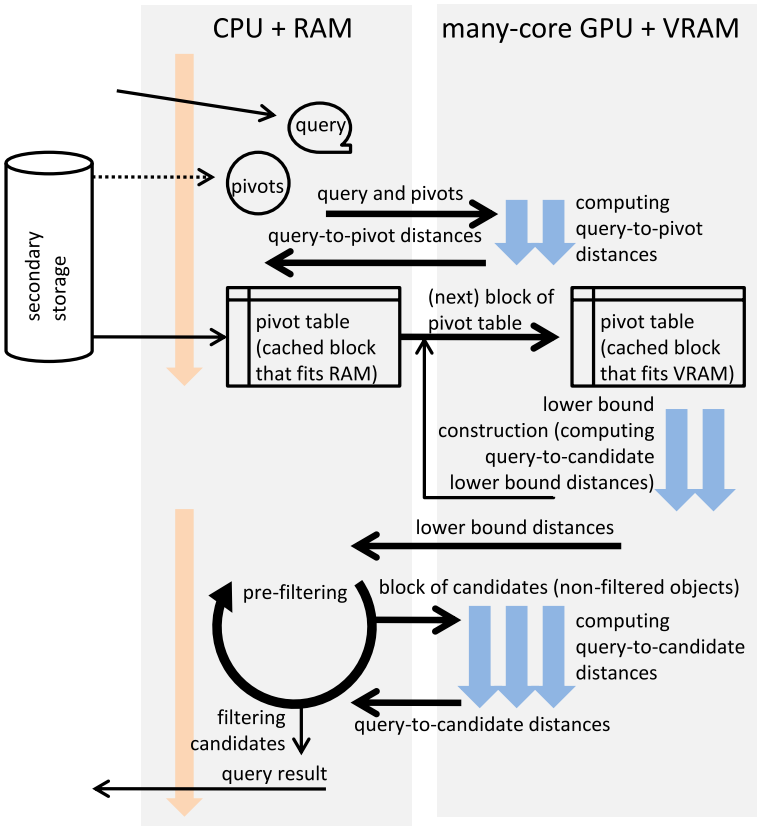


Fig. 10 Workflow of the SQFD + LB algorithm

Despite the improvement in the GPUs/CPU load balance, moving the lower bound construction to GPU brings also an unpleasant side effect. Because now the lower bound construction and pre-filtering steps run separately and asynchronously (the former on GPUs, the latter on CPU), the lower bound construction cannot benefit from the early termination optimization anymore (Sect. 2.3), which makes the whole computation less efficient. However, the sacrifice is worth the overall gain in better utilized GPUs (as shown in the experiments). We must note that moving both of the steps to GPU (also the pre-filtering) cannot help, because the CPU still has to dispatch the blocks of candidate signatures to GPU (which is done together with pre-filtering).

Computing lower bounds for all the database objects on GPUs means the pivot table as well as the query-to-pivot distances must be transferred to global memory of the GPU (VRAM). In case the pivot table cannot fit the memory or in case we have multiple GPU devices available, the table is divided into blocks which are as large as possible.[5] The lower bound construction is then performed in block-wise fashion the same way as SQFD computation is performed on the blocks of signatures. In the

[5]It is safe to say that modern GPUs have sufficient memory capacity to accommodate pivot tables for databases that fit the host memory of an ordinary server.

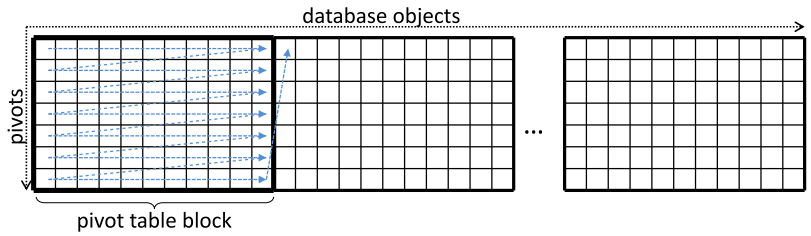


Fig. 11 Pivot table memory representation

following we take more detailed look at the parallel lower bounds construction on GPU.

5.2 Pivot table representation

The most delicate issue of the lower bound construction is the memory representation of the pivot table. A pivot table is two dimensional array that holds distances between a small number of pivots to every object in the database. There are many ways how to represent two-dimensional array in linear memory. However, both direct approaches (row-wise or column-wise concatenation) are not suitable in our case. We need to consider the following requirements:

- The pivot table must be divisible (with acceptable granularity) to blocks in case it does not fit the VRAM or there are multiple GPU devices available.
- Pivot table fragment being processed by one thread group must be organized so that the data transfers are performed in coalesced loads.
- Data required by threads of one group at the same time should be distributed into the local memory banks as evenly as possible.

In order to meet these requirements, we have chosen a memory representation as depicted in Fig. 11. The pivot table is divided into blocks of equal size. Each block is assigned to one thread group so its size is determined accordingly. In our case the block spans over 256 columns of the pivot table as we use 256 threads per group.

Each pivot table block is stored in a contiguous part of the memory, where distances to each particular pivot are stored consecutively. This representation is suitable for a model where one thread computes lower bound value for one database object. When a thread iterates over pivots, all threads in a group process distances to one pivot at the same time. Therefore, the data loaded by the threads lie in aligned continuous range of memory. Furthermore, distances are evenly distributed over memory banks as each distance is represented by one float value.

5.3 Computing lower bounds on the GPU

Given the pivot table memory representation described above, the GPU-based lower bound construction is much simpler than the GPU-based SQFD computation. Each thread computes the lower bound of one database object and each thread group operates on one pivot table block. To compute a lower bound for a database object, a

vector of query-to-pivot distances and the matrix of pivot-to-pivot distances is additionally required to be transferred and stored in the memory of GPUs.

In particular, the query-to-pivot distances are computed and stored into a buffer on every GPU device available. It is cached in the local memory when the computation starts. The pivot-to-pivot matrix could be extracted from pivot table, but for the sake of simplicity and faster loading the data are duplicated so that all pivot-to-pivot distances are in one compact block. Also this matrix is cached in the local memory. The corresponding pivot table block may be cached in the local memory too; however, on the state-of-the-art GPUs we need not to cache it implicitly as the data is accessed in such manner that they are cached in L1 and L2 automatically.

6 Experiments

In this section we evaluate the efficiency of parallel similarity search using the SQFD. We have compared the performance of high-end multi-core CPU server with a common workstation that used one or two GPU cards. In the experiments we have observed the behavior of the two proposed query algorithms under various parameters, like the α used in SQFD computation, the type of lowerbounding used by pivot table indexes, and the size of the blocks dispatched for parallel processing. The last one in the list was especially important for the evaluation, as the block size heavily determined the throughput of the system and the load balancing between CPU and GPU. In all the experiments we measured just the real times, because other types of cost, like the number of distance computations, were not affected by the parallel processing.

6.1 Methodology and hardware setup

Each test was performed using 100 query signatures with different numbers of centroids, while each query was measured five times and then the mean value was determined by computing arithmetic average of the measured values. If any of the time values deviated from the average more than 15 %, the value was discarded and the test was repeated. In the results we show the mean value of the average times of all 100 queries.

Tests conducted on the GPU platform are denoted GPU1 and GPU2 in the figures, where the number refers to either on one or two GPU used. The workstation was based on Intel Core i7 870 CPU clocked at 2.93 GHz, and was equipped with 16 GB of RAM and two NVIDIA GTX 580 GPU cards with 512 CUDA cores and 1.5 GB of RAM each. Tests conducted on the multi-core CPU server platform are denoted CPU48 in the figures. We used Dell M910 server with four six-core Intel Xeon E7540 processors with hyper-threading (i.e., 48 logical cores) clocked at 2.0 GHz. The server was equipped with 128 GB of RAM organized as 4-node cache coherent NUMA. A RedHat Enterprise Linux 6 was used as operating system on both machines.

In order to compare the proposed algorithms to the multi-core CPU platform (CPU48), we have also modified the SQFD-only algorithm for pure CPU system

by utilizing all available cores. Its architecture mimics the original SQFD-only algorithm, where one CPU core performs the pre-filtering, block dispatching, and final filtering, while the remaining cores compute SQFD distances in parallel instead of the GPU.

6.2 Datasets

The experiments were conducted on one synthetic dataset representing clouds of points and one real dataset consisting of feature signatures extracted from images.

A synthetic Clouds database was generated [20], namely 2,097,152 clouds (sets) of 100–140 5-dimensional points (embedded in a unitary 5D cube). This database was chosen as a set analogy to synthetic vector datasets when evaluating vectorial similarity search. Moreover, the cloud of points is a common representation for simplified representations of complex objects or objects consisting of multiple observations [21]. Each point has assigned a weight where the sum of all weights in the cloud was 10,000. For each cloud, its center was generated at random, while another 10,000 points were generated under normal distribution around the center (the mean and variance in each dimension were adjusted to not generate points outside the unitary cube). Then an adaptive variant of the k-means clustering [17] was used to create 100–139 centroids representing the original data. The weight of each centroid corresponded to the number of points assigned to the centroid in the last iteration of the k-means clustering. On average, a feature signature consisted of 120 representatives (centroids), i.e., 720 numbers per signature. The distribution of the number of representatives for Clouds is depicted on Fig. 12a.

As a dataset from the real world, we have extracted feature signatures from 950,000 images from the CoPhIR database [6]. The extraction was based on seven-dimensional features (L; a; b; x; y; c; e)—color (L; a; b), position (x; y), contrast c, and entropy e. These features were extracted for a randomly selected subset of pixels for each image and then again aggregated by applying the adaptive variant of the k-means clustering algorithm. Thus, we have obtained one feature signature for each

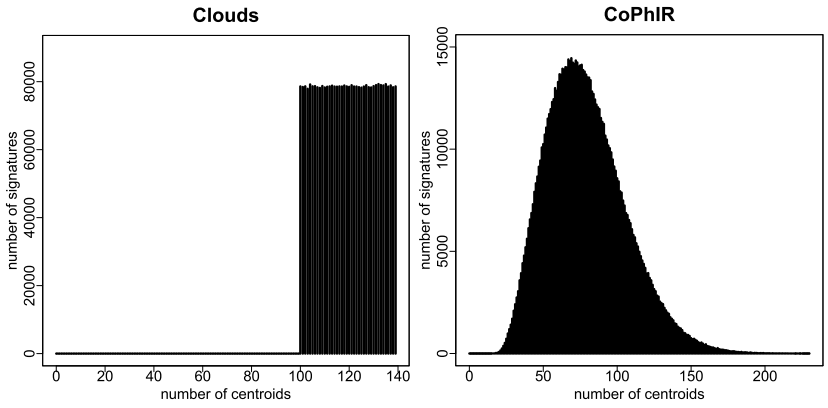


Fig. 12 Distribution of the number of signature centroids for (**a**) Clouds and (**b**) CoPhIR databases

single image. These signatures vary in size between 15 and 215 feature representatives (for more details about the size distribution see Fig. 12b). On average, a feature signature consists of 75 representatives (i.e., 600 numbers per signature).

6.3 Index setup

For both the hardware platforms we have used a parallel implementation of the pivot table index (see Sect. 2.3). In order to observe the difference between SQFD-Only and SQFD + LB algorithm, we have used three types of lowerbounding in the pivot table, the metric type using triangle inequality (denoted Tri in the figures), ptolemaic type using Ptolemy's inequality (denoted Pto), and both metric + ptolemaic type (denoted TriPto). In all experiments we used 32 pivots. Actually, we used as many pivots as possible with respect to memory and cache sizes available on present hardware. The limited number of pivots, however, is not crucial when using the ptolemaic pivot tables, because ptolemaic filtering exploits every distinct pair of pivots (e.g., the number of pivots squared).

6.4 Sequential search and indexing

In the first set of tests we performed similarity search without the aid of an index, that is, sequential search over the database, however, parallelized for both CPU48 and GPU platforms. The overhead of particular query result construction is negligible, thus we do not distinguish between range queries or kNN queries in sequential search. Furthermore, all tests were conducted only for $\alpha = 0.2$ as different alpha values affect only the efficiency of pivot table pre-filtering, but they have no measurable impact on the speed of SQFD evaluation. Besides query processing, these tests can also be interpreted as parallel construction of the pivot table, since the sequential search/pivot table construction procedures are similar.

First, we will examine how the performance is affected by different block size (Fig. 13). As there is no pre-filtering, this graph helps us determine the overhead of block dispatching. The experimental results show that dispatching distance computations in blocks of at least 1024 signatures is sufficient for optimal performance.

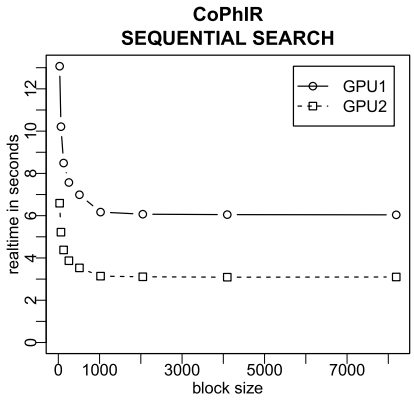


Fig. 13 The impact of the varying block size

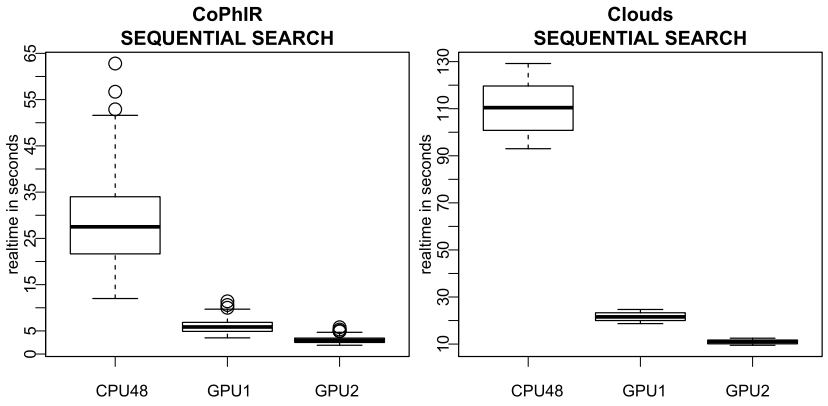


Fig. 14 Comparison of total GPU speedup over multi-core CPU server

However, this predicament holds only in case the CPU is capable of supplying GPU steadily with data blocks.

Next, we compare the best possible result on GPU (using block size of 8,192 signatures) against our CPU implementation running on 48 logical cores (Fig. 14). The best speedup was achieved for Clouds database on 2 GPUs (1024 cores total)—10.08× w.r.t. to CPU 48 version. The Clouds dataset with signatures formed on average by 120 centroids shows better speedup than CoPhIR containing signatures formed on average by 75 centroids. Furthermore, we have observed that the CPU version has rather higher variance of measured times since the SQFD computation depends heavily on signature length which differs amongst the test queries. On the other hand, this effect is considerably reduced on GPU, where better speedup on larger signatures and stronger resistance to length variance were observed, because the GPU utilized the parallelism better on large signatures.

6.5 Index search

In the second set of tests, we executed queries on pivot table indexes. Three types of pre-filtering were used in pivot table: the metric filter with triangular inequality (*Tri*), Ptolemaic filter (*Pto*) and combination of both (*TriPto*). We were testing both SQFD-only and SQFD + LB algorithms. The results are shown for $\alpha = 0.01$, which gave us the best indexability, and for $\alpha = 0.2$, which gave us the best tradeoff between performance and retrieval precision. Larger α values (such as $\alpha = 3$ which gave us the best precision but worst indexability) did not benefit much from indexing, so the results were similar to sequential search.

Furthermore, the SQFD + LB algorithm had preloaded the pivot table into the GPU memory and the table was kept in the memory during the whole test so all queries could use it. In our case the pivot table was small enough to not affect SQFD computations in any way. The upload of the pivot table to GPU memory took 54 ms for CoPhIR database and 118 ms for Clouds database.

6.5.1 Range queries

In order to normalize sizes of query results, the range queries were designed to have always the same selectivity (0.1 % of the database size). The results of tests performed to determine the optimal block size for each method are shown in Fig. 15.

All results exhibit the same behavior. Unlike the sequential search, all methods were parameterized by an optimal block size where the CPU workload, GPU workload, and overhead were in balance. Increasing the block size beyond the optimal value did not help the performance, since it increased time periods when GPU waits for CPU or vice versa.

On single GPU the SQFD + LB algorithm is slower than ($\alpha = 0.01$) or approximately as fast as ($\alpha = 0.2$) the SQFD-only algorithm. This result is caused by fact that in case of single GPU, the CPU-GPU workload is almost in balance and the parallel lower bound construction does not completely make up for sacrifice of the early termination optimization. However, as we can see from 2 GPU tests, the SQFD + LB scales much better than SQFD-only algorithm and gives better results. In case of $\alpha = 0.2$, the SQFD + LB using TriPto index is by 21 % faster than SQFD-only algorithm with the same parameters. We believe that on more GPUs the difference between these two algorithms would be even greater.

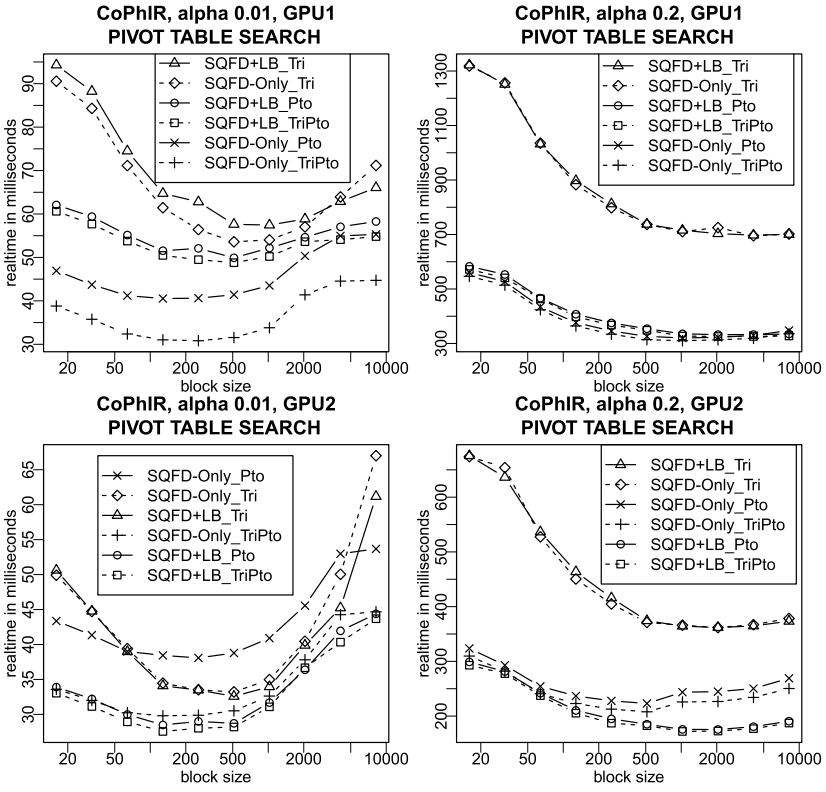


Fig. 15 The impact of the varying block size for range queries

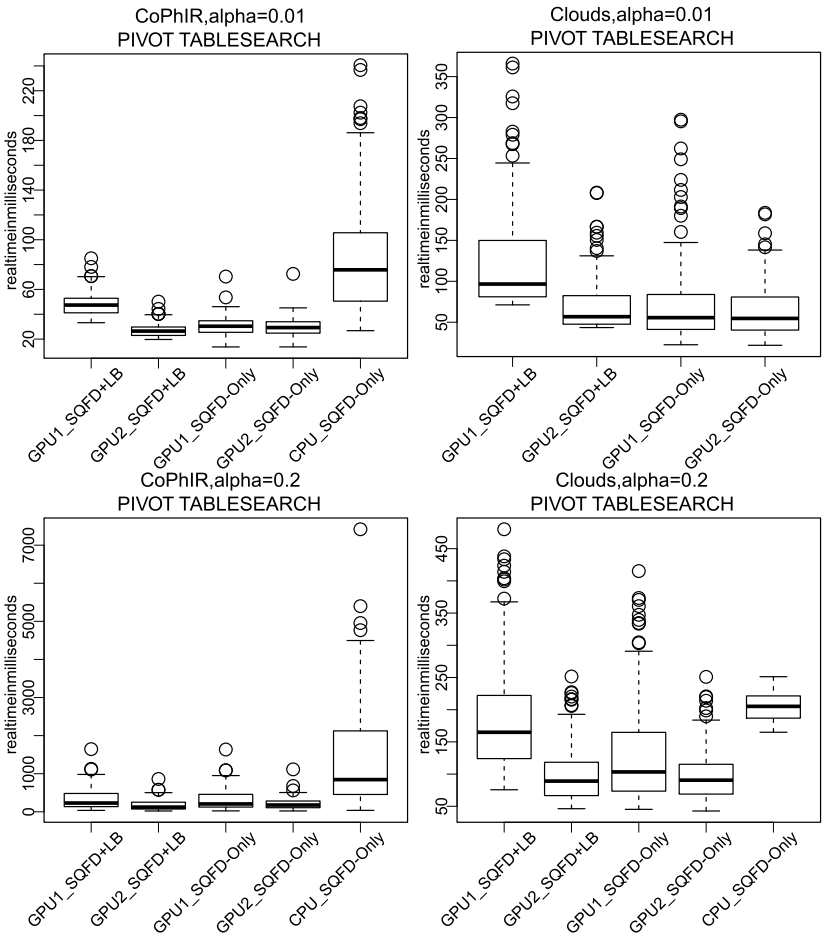


Fig. 16 Comparison of the best results on different architectures

Finally, we present comparison of best results for both algorithms employing TriPto index and choosing optimal block size on GPU 1, GPU 2 and CPU 48 (Fig. 16). The CPU version ran solely the SQFD-only algorithm as the SQFD + LB algorithm is not suitable for CPUs.

6.5.2 kNN queries

For the kNN queries we used $k = 100$, so that 100 nearest neighbors of the query object were selected. The kNN query differs from range query in fact that the query radius, which was also used for the pre-filtering step, was refined during the computation. Therefore, selecting appropriate block size was even more delicate than in sequential search or range queries (Fig. 17).

The results indicate that even smaller blocks are required in order to achieve optimal performance, especially for $\alpha = 0.01$. For most algorithms, the optimal block size is 64–128 for $\alpha = 0.01$ and about 256 for $\alpha = 0.2$.

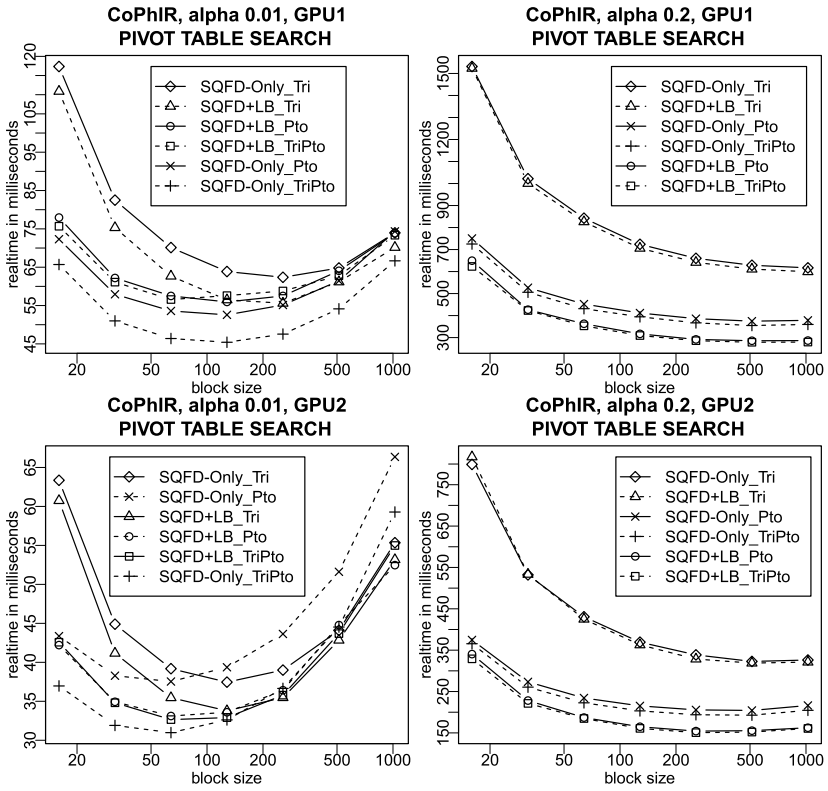


Fig. 17 The impact of the varying block size for kNN queries

As shown in range queries tests, the SQFD + LB algorithm was slightly slower in case of $\alpha = 0.01$ on single GPU and slightly faster for $\alpha = 0.2$. But most importantly, it exhibits better speedup when comparing GPU1 and GPU2 results, thus provides much better opportunities for scalability.

The overall comparison of kNN results is reviewed in the remaining set of graphs (Fig. 18).

6.6 Summary

We have experimentally proved that our GPU-based algorithms are significantly faster than multi-core CPU implementation in every type of query processing and also in indexing. Furthermore, the SQFD + LB algorithm demonstrates great scalability potential and offers better performance in case there is more GPU computational power available.

7 Conclusion

We have proposed a parallel approach to fast similarity search using the Signature Quadratic Form Distance (SQFD) on combined CPU and GPU architectures. In particular, we proposed two algorithms that adopt metric/ptolemaic indexing within a

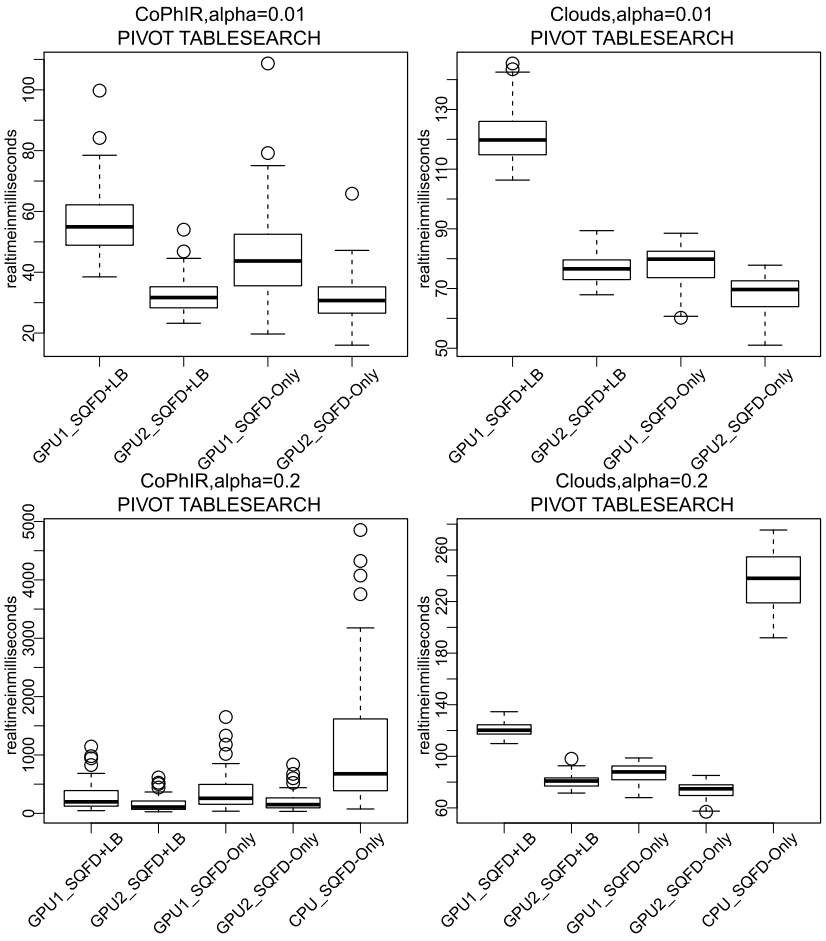


Fig. 18 Comparison of the best results on different architectures

parallel architecture, such that the query processing workload is split between the CPU and multiple GPUs. The first algorithm utilizes the GPUs just by computation of SQFDs batches, leaving the other processing on CPU. The second algorithm utilizes the GPUs additionally by construction of lower-bound distances used in the index pre-filtering, leading to better balance of workload between the CPU and GPUs when expensive lower bound construction is used (such as the ptolemaic lowerbounding). In experimental evaluation we have shown that our implementation on a common workstation with just 2 GPU cards outperforms the traditional parallel implementation on a high-end 48-core server by up to an order of magnitude. If we consider also the price of the high-end server which is ten times higher than the GPU workstation, then based on price/performance ratio, the GPU-based similarity search beats the CPU-based solution by almost two orders of magnitude.

Acknowledgements This research has been supported by Czech Science Foundation (GAČR) project 202/11/0968, by the grant agency of Charles University (GAUK) project no. 277911, and by the Deutsche Forschungsgemeinschaft within the Collaborative Research Center SFB 686.

References

1. Beecks, C., Lokoč, J., Seidl, T., Skopal, T.: Indexing the signature quadratic form distance for efficient content-based multimedia retrieval. In: Proc. ACM Int. Conf. on Multimedia Retrieval, pp. 24:1–24:8 (2011)
2. Beecks, C., Seidl, T.: On stability of adaptive similarity measures for content-based image retrieval. In: MMM, pp. 346–357 (2012)
3. Beecks, C., Uysal, M.S., Seidl, T.: Signature quadratic form distances for content-based similarity. In: Proc. ACM Multimedia, pp. 697–700 (2009)
4. Beecks, C., Uysal, M.S., Seidl, T.: A comparative study of similarity measures for content-based multimedia retrieval. In: Proc. IEEE ICME, pp. 1552–1557 (2010)
5. Beecks, C., Uysal, M.S., Seidl, T.: Signature quadratic form distance. In: Proc. ACM CIVR, pp. 438–445 (2010)
6. Bolettieri, P., Esuli, A., Falchi, F., Lucchese, C., Perego, R., Piccioli, T., Rabitti, F.: CoPhIR: a test collection for content-based image retrieval. 0905.4627v2 (2009). http://cophir.isti.cnr.it
7. Bustos, B., Deussen, O., Hiller, S., Keim, D.: A graphics hardware accelerated algorithm for nearest neighbor search. In: Computational Science—ICCS 2006, pp. 196–199 (2006)
8. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. **33**(3), 273–321 (2001). doi:10.1145/502807.502808
9. Deselaers, T., Keysers, D., Ney, H.: Features for image retrieval: an experimental comparison. Inf. Retr. **11**(2), 77–107 (2008). doi:10.1007/s10791-007-9039-3
10. Garcia, V., Debreuve, E., Barlaud, M.: Fast k nearest neighbor search using gpu. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPRW'08, pp. 1–6. IEEE, New York (2008)
11. Geusebroek, J.M., Burghouts, G.J., Smeulders, A.W.M.: The Amsterdam library of object images. Int. J. Comput. Vis. **61**(1), 103–112 (2005)
12. Hafner, J., Sawhney, H.S., Equitz, W., Flickner, M., Niblack, W.: Efficient color histogram indexing for quadratic form distance functions. IEEE Trans. Pattern Anal. Mach. Intell. **17**, 729–736 (1995). doi:10.1109/34.391417
13. Hetland, M.L.: The basic principles of metric indexing. In: Coello, C.A.C., Dehuri, S., Ghosh, S. (eds.) Swarm Intelligence for Multi-objective Problems in Data Mining. Studies in Computational Intelligence, vol. 242. Springer, Berlin (2009)
14. Hetland, M.L.: Ptolemaic indexing. arXiv:0911.4384 [cs.DS] (2009)
15. Hu, R., Rüger, S., Song, D., Liu, H., Huang, Z.: Dissimilarity measures for content-based image retrieval. In: Proc. IEEE International Conference on Multimedia & Expo, pp. 1365–1368 (2008). doi:10.1109/ICME.2008.4607697
16. Huiskes, M.J., Lew, M.S.: The mir flickr retrieval evaluation. In: Proc. ACM MIR, pp. 39–43 (2008)
17. Leow, W.K., Li, R.: The analysis and applications of adaptive-binning color histograms. Comput. Vis. Image Underst. **94**(1–3), 67–91 (2004). doi:10.1016/j.cviu.2003.10.010
18. Lieberman, M., Sankaranarayanan, J., Samet, H.: A fast similarity join algorithm using graphics processing units. In: IEEE 24th International Conference on Data Engineering, ICDE 2008, pp. 1111–1120. IEEE, New York (2008)
19. Lokoč, J., Hetland, M., Skopal, T., Beecks, C.: Ptolemaic indexing of the signature quadratic form distance. In: Proceedings of the Fourth International Conference on Similarity Search and Applications, pp. 9–16. ACM, New York (2011)
20. Lokoč, J.: Cloud of points generator. SIRET Research Group (2010). http://siret.ms.mff.cuni.cz/projects/pointgenerator/
21. Mémoli, F., Sapiro, G.: Comparing point clouds. In: SGP'04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing, pp. 32–40. ACM, New York (2004). doi:10.1145/1057432.1057436
22. Mico, M.L., Oncina, J., Vidal, E.: A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. Pattern Recognit. Lett. **15**(1), 9–17 (1994). doi:10.1016/0167-8655(94)90095-7
23. Mikolajczyk, K., Schmid, C.: A performance evaluation of local descriptors. IEEE Trans. Pattern Anal. Mach. Intell. **27**(10), 1615–1630 (2005). doi:10.1109/TPAMI.2005.188
24. Navarro, G.: Analyzing metric space indexes: what for? In: IEEE SISAP 2009, pp. 3–10 (2009)
25. NVIDIA: Fermi GPU Architecture. http://www.nvidia.com/object/fermi_architecture.html

26. Pan, J., Manocha, D.: Fast gpu-based locality sensitive hashing for k-nearest neighbor computation. In: Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 211–220. ACM, New York (2011)
27. Puzicha, J., Buhmann, J., Rubner, Y., Tomasi, C.: Empirical evaluation of dissimilarity measures for color and texture. In: Proc. IEEE International Conference on Computer Vision, vol. 2, pp. 1165–1172 (1999)
28. Rubner, Y., Tomasi, C., Guibas, L.J.: The earth mover's distance as a metric for image retrieval. Int. J. Comput. Vis. **40**(2), 99–121 (2000). doi:10.1023/A:1026543900054
29. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann, San Mateo (2006)
30. Skopal, T., Bartoš, T., Lokoč, J.: On (not) indexing quadratic form distance by metric access methods. In: Proc. Extending Database Technology (EDBT). ACM, New York (2011)
31. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. Advances in Database Systems. Springer, New York (2005)

Chapter 8

Conclusions

During last five years, we have focused on efficient content-based similarity search using models based on feature signatures. The feature signatures enable flexible representation and effective retrieval of multimedia objects. However, the efficiency of the retrieval using feature signatures still represents a challenge for large multimedia collections. Therefore, we have investigated parallel feature extraction techniques to speed up multimedia indexing phase and several approaches for efficient retrieval. More specifically, we have demonstrated that metric/ptolemaic access methods can be employed for efficient retrieval using metric/ptolemaic adaptive distance measures and also that the distances represent a suitable task for parallel processing. Furthermore, many of the presented techniques have been designed as general approaches not restricted just to feature signatures and adaptive distance measures. Generally, any similarity model with expensive distance measure satisfying metric/ptolemaic properties could benefit from our new techniques.

In our work, we have focused also on applications of models based on feature signatures. For example, we have shown that position-color-texture feature signatures can be utilized for intuitive multimedia exploration, where users can just browse an unknown database using actually visible multimedia objects. Another example is the video retrieval area, where position-color-texture feature signatures can be used to represent contents of video keyframes. Such representation enables the design of a simple and intuitive query formulation interface, where users can draw colored circles into time ordered sketches. The effectiveness of such simple approach has been demonstrated at two international video browsing competitions, where our signature-based video browser has won twice in a row (years 2014 and 2015).

8.1 Future research

The multimedia retrieval still represents a dynamically evolving area of research, where new successful paradigms are emerging every five to ten years. For example, in recent years we have observed the rise of deep learning approaches that demonstrate impressive effectiveness in various classification tasks [16, 45, 89]. Nevertheless, we believe that there is not a universal similarity model for all possible retrieval tasks and that each model can find its applications. In the following, we present several research areas/questions which we plan to investigate in the future.

- We would like to combine different orthogonal similarity models to design effective multimedia exploration systems. For example, a cheap model can be used to select a candidate set of images (e.g., text-based search, caffe descriptors), while a model based on feature signatures can be used for an interactive navigation in the candidate set.

- Actually, we work on new applications and domains for models based on feature signatures. For example, the 3D object based retrieval where feature signatures could be used to model surface parts of objects, or, HTTPS traffic modeling where feature signatures could be used to model communication properties between users and servers, now employing statistical fingerprints [43]. Recently, feature signatures have been employed as a spatio-temporal video representation approach [93] that could be used in a future version of our signature-based video browser.

- The success of deep learning and feature selection techniques logically raises questions, whether feature signatures could be significantly improved by selecting only specific sets of tuples for each multimedia object. Another question is, whether feature signatures and/or perceptual distances could improve the effectiveness of the actual state-of-the-art deep learning architectures. Furthermore, the research in this area will be probably again tightly coupled with the design of new algorithms for GPU architectures.

Bibliography

[1] G. Amato and P. Savino. Approximate similarity search in metric spaces using inverted files. In *Proceedings of the 3rd International Conference on Scalable Information Systems*, InfoScale '08, pages 28:1–28:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[2] L. Ares, N. Brisaboa, A. Ordóñez Pereira, and O. Pedreira. Efficient similarity search in metric spaces with cluster reduction. In G. Navarro and V. Pestov, editors, *Similarity Search and Applications*, volume 7404 of *Lecture Notes in Computer Science*, pages 70–84. Springer Berlin Heidelberg, 2012.

[3] I. Assent, A. Wenning, and T. Seidl. Approximation techniques for indexing the earth mover's distance in multimedia databases. In *Data Engineering, 2006. ICDE '06. Proceedings of the 22nd International Conference on*, pages 11–22. IEEE, 2006.

[4] I. Assent, M. Wichterich, T. Meisen, and T. Seidl. Efficient similarity search using the earth mover's distance for large multimedia databases. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 307–316. IEEE, 2008.

[5] M. Batko, D. Novak, F. Falchi, and P. Zezula. On scalability of the similarity search in the world of peers. In *Proceedings of the 1st International Conference on Scalable Information Systems, Infoscale 2006, Hong Kong, May 30-June 1, 2006*, New York, NY, USA, 2006. ACM.

[6] H. Bay, A. Ess, T. Tuytelaars, and L. J. V. Gool. Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[7] C. Beecks. Distance-based similarity models for content-based multimedia retrieval. In *Dissertation, Fakultat fur Mathematik, Informatik und Naturwissenschaften, RWTH Aachen University.*, 2013.

[8] C. Beecks, A. M. Ivanescu, T. Seidl, D. Martin, P. Pischke, and R. Kneer. Applying similarity search for the investigation of the fuel injection process. In *Proceedings of the Fourth International Conference on SImilarity Search and APplications*, SISAP '11, pages 117–118, New York, NY, USA, 2011. ACM.

[9] C. Beecks, S. Kirchhoff, and T. Seidl. Signature matching distance for content-based image retrieval. In *Proc. ACM International Conference on Multimedia Retrieval (ICMR 2013), Dallas, Texas, USA*, pages 41–48, New York, NY, USA, 2013. ACM.

[10] C. Beecks, J. Lokoč, T. Seidl, and T. Skopal. Indexing the signature quadratic form distance for efficient content-based multimedia retrieval. In *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, ICMR '11, pages 24:1–24:8, New York, NY, USA, 2011. ACM.

[11] C. Beecks, T. Skopal, K. Schoeffmann, and T. Seidl. Towards large-scale multimedia exploration. In *Proc. 5th International Workshop on Ranking in Databases (DBRank 2011), Seattle, WA, USA*, pages 31–33, 2011.

[12] C. Beecks, M. Uysal, and T. Seidl. A comparative study of similarity measures for content-based multimedia retrieval. In *Multimedia and Expo (ICME), 2010 IEEE International Conference on*, pages 1552–1557. IEEE, July 2010.

[13] C. Beecks, M. S. Uysal, and T. Seidl. Efficient k-nearest neighbor queries with the signature quadratic form distance. In *Proc. 4th International Workshop on Ranking in Databases (DBRank 2010) in conjunction with IEEE 26th International Conference on Data Engineering*

(ICDE 2010), Long Beach, California, USA, pages 10 – 15, Washington, USA, 2010. IEEE.

[14] C. Beecks, M. S. Uysal, and T. Seidl. Signature quadratic form distance. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, CIVR '10, pages 438–445, New York, NY, USA, 2010. ACM.

[15] C. Beecks, M. S. Uysal, and T. Seidl. L2-signature quadratic form distance for efficient query processing in very large multimedia databases. In K.-T. Lee, W.-H. Tsai, H.-Y. Liao, T. Chen, J.-W. Hsieh, and C.-C. Tseng, editors, *Advances in Multimedia Modeling*, volume 6523 of *Lecture Notes in Computer Science*, pages 381–391. Springer Berlin Heidelberg, 2011.

[16] Y. Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, Jan. 2009.

[17] A. Blažek, J. Lokoč, F. Matzner, and T. Skopal. Enhanced signature-based video browser. In *MultiMedia Modeling - 21st International Conference, MMM 2015, Sydney, NSW, Australia, January 5-7, 2015, Proceedings, Part II*, volume 8936 of *Lecture Notes in Computer Science*, pages 243–248. Springer International Publishing, 2015.

[18] A. Blažek, J. Lokoč, and T. Skopal. Video retrieval with feature signature sketches. In *Similarity Search and Applications - 7th International Conference, SISAP 2014, Los Cabos, Mexico, October 29-31, 2014. Proceedings*, volume 8821 of *Lecture Notes in Computer Science*, pages 25–36. Springer International Publishing, 2014.

[19] B. Braunmüller, M. Ester, H. Kriegel, and J. Sander. Multiple similarity queries: A basic DBMS operation for mining in metric databases. *IEEE Trans. Knowl. Data Eng.*, 13(1):79–95, 2001.

[20] S.-F. Chang, T. Sikora, and A. Purl. Overview of the mpeg-7 standard. *Circuits and Systems for Video Technology, IEEE Transactions on*, 11(6):688–695, Jun 2001.

[21] S. A. Chatzichristofis and Y. S. Boutalis. Cedd: Color and edge directivity descriptor: A compact descriptor for image indexing and re-

trieval. In *Proceedings of the 6th International Conference on Computer Vision Systems*, ICVS'08, pages 312–322, Berlin, Heidelberg, 2008. Springer-Verlag.

[22] E. Chávez, K. Figueroa, and G. Navarro. Effective proximity retrieval by ordering permutations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(9):1647–1658, Sept. 2008.

[23] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, Sept. 2001.

[24] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, pages 426–435, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[25] R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Comput. Surv.*, 40(2):5:1–5:60, May 2008.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, June 2009.

[27] T. Deselaers, D. Keysers, and H. Ney. Features for image retrieval: an experimental comparison. *Information Retrieval*, 11(2):77–107, 2008.

[28] C. Eickhoff. Crowd-powered experts: Helping surgeons interpret breast cancer images. In *Proceedings of the First International Workshop on Gamification for Information Retrieval*, GamifIR '14, pages 53–56, New York, NY, USA, 2014. ACM.

[29] B. Everitt, S. Landau, M. Leese, and D. Stahl. *Cluster analysis*. Wiley, 5th edition, 2011.

[30] P. Galuščáková, M. Kruliš, J. Lokoč, and P. Pecina. CUNI at mediaeval 2014 search and hyperlinking task: Visual and prosodic features in hyperlinking. In *Working Notes Proceedings of the MediaEval 2014*

Workshop, Barcelona, Catalunya, Spain, October 16-17, 2014. CEUR-WS.org, 2014.

[31] J. Gantz and D. Reinsel. THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East, 2012.

[32] V. Gil-Costa and M. Marin. Approximate distributed metric-space search. In *Proceedings of the 9th Workshop on Large-scale and Distributed Informational Retrieval*, LSDS-IR '11, pages 15–20, New York, NY, USA, 2011. ACM.

[33] A. D. Gordon. A review of hierarchical classification. *Journal of the Royal Statistical Society. Series A (General)*, 150(2):pp. 119–137, 1987.

[34] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3:1157–1182, Mar. 2003.

[35] J. Hafner, H. S. Sawhney, W. Equitz, M. Flickner, and W. Niblack. Efficient color histogram indexing for quadratic form distance functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:729–736, 1995.

[36] M. L. Hetland. Ptolemaic indexing. `arXiv:0911.4384 [cs.DS]`, 2009.

[37] M. L. Hetland, T. Skopal, J. Lokoč, and C. Beecks. Ptolemaic access methods: Challenging the reign of the metric space model. *Inf. Syst.*, 38(7):989–1006, 2013.

[38] D. P. Huttenlocher, G. A. Klanderman, G. A. Kl, and W. J. Rucklidge. Comparing Images Using the Hausdorff Distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:850–863, 1993.

[39] Y. Ishikawa, R. Subramanya, and C. Faloutsos. Mindreader: Querying databases through multiple examples. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 218–227. Morgan Kaufmann, 1998.

[40] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *Proceedings of*

the 10th European Conference on Computer Vision: Part I, ECCV '08, pages 304–317, Berlin, Heidelberg, 2008. Springer-Verlag.

[41] H. Jegou, F. Perronnin, M. Douze, J. Sánchez, P. Perez, and C. Schmid. Aggregating local image descriptors into compact codes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(9):1704–1716, Sept. 2012.

[42] K. Kavukcuoglu, P. Sermanet, Y. lan Boureau, K. Gregor, M. Mathieu, and Y. L. Cun. Learning convolutional feature hierarchies for visual recognition. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1090–1098. Curran Associates, Inc., 2010.

[43] J. Kohout and T. Pevný. Unsupervised detection of malware in persistent web traffic. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 1757–1761. IEEE, 2015.

[44] S. Kozak and P. Zezula. Efficiency and security in similarity cloud services. *Proc. VLDB Endow.*, 6(12):1450–1455, Aug. 2013.

[45] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1097–1105. Curran Associates, Inc., 2012.

[46] M. Kruliš, J. Lokoč, C. Beecks, T. Skopal, and T. Seidl. Processing the signature quadratic form distance on many-core gpu architectures. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, pages 2373–2376, New York, NY, USA, 2011. ACM.

[47] M. Kruliš, J. Lokoč, and T. Skopal. Efficient extraction of feature signatures using multi-gpu architecture. In S. Li, A. El Saddik, M. Wang, T. Mei, N. Sebe, S. Yan, R. Hong, and C. Gurrin, editors, *Advances in Multimedia Modeling*, volume 7733 of *Lecture Notes in Computer Science*, pages 446–456. Springer Berlin Heidelberg, 2013.

[48] M. Kruliš, J. Lokoč, and T. Skopal. Efficient extraction of clustering-based feature signatures using gpu architectures. *Multimedia Tools and Applications*, pages 1–33, 2015.

[49] M. Kruliš, T. Skopal, J. Lokoč, and C. Beecks. Combining CPU and GPU architectures for fast similarity search. *Distributed and Parallel Databases*, 30(3-4):179–207, 2012.

[50] A. Kumar, J. Kim, W. Cai, M. Fulham, and D. Feng. Content-based medical image retrieval: A survey of applications to multidimensional and multimodality data. *Journal of Digital Imaging*, 26(6):1025–1039, 2013.

[51] L. Liu, L. Wang, and X. Liu. In defense of soft-assignment coding. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2486–2493. IEEE, Nov 2011.

[52] J. Lokoč. Approximating adaptive distance measures using scalable feature signatures. *Multimedia Tools and Applications*, pages 1–26, 2014.

[53] J. Lokoč. Approximating the signature quadratic form distance using scalable feature signatures. In C. Gurrin, F. Hopfgartner, W. Hurst, H. Johansen, H. Lee, and N. O'Connor, editors, *MultiMedia Modeling*, volume 8325 of *Lecture Notes in Computer Science*, pages 86–97. Springer International Publishing, 2014.

[54] J. Lokoč, A. Blažek, and T. Skopal. Signature-based video browser. In C. Gurrin, F. Hopfgartner, W. Hurst, H. Johansen, H. Lee, and N. O'Connor, editors, *MultiMedia Modeling*, volume 8326 of *Lecture Notes in Computer Science*, pages 415–418. Springer International Publishing, 2014.

[55] J. Lokoč, T. Grošup, and T. Skopal. Image exploration using online feature extraction and reranking. In *Proceedings of the 2nd ACM International Conference on Multimedia Retrieval*, ICMR '12, pages 66:1–66:2, New York, NY, USA, 2012. ACM.

[56] J. Lokoč, T. Grošup, and T. Skopal. On scalable approximate search with the signature quadratic form distance. In N. Brisaboa, O. Pedreira, and P. Zezula, editors, *Similarity Search and Applications*,

volume 8199 of *Lecture Notes in Computer Science*, pages 312–318. Springer Berlin Heidelberg, 2013.

[57] J. Lokoč, T. Grošup, P. Čech, and T. Skopal. Towards efficient multimedia exploration using the metric space approach. In *Content-Based Multimedia Indexing (CBMI), 2014 12th International Workshop on*, pages 1–4. IEEE, June 2014.

[58] J. Lokoč, M. L. Hetland, T. Skopal, and C. Beecks. Ptolemaic indexing of the signature quadratic form distance. In *Proceedings of the Fourth International Conference on SImilarity Search and APplications*, SISAP '11, pages 9–16, New York, NY, USA, 2011. ACM.

[59] J. Lokoč, J. Moško, P. Čech, and T. Skopal. On indexing metric spaces using cut-regions. *Information Systems*, 43(0):1 – 19, 2014.

[60] J. Lokoč, D. Novák, M. Batko, and T. Skopal. Visual image search: feature signatures or/and global descriptors. In *Proceedings of the 5th international conference on Similarity Search and Applications*, volume 7404 of *Lecture Notes in Computer Science*, pages 177–191. Springer Berlin Heidelberg, 2012.

[61] J. Lokoč, P. Čech, J. Novák, and T. Skopal. Cut-region: A compact building block for hierarchical metric indexing. In G. Navarro and V. Pestov, editors, *Similarity Search and Applications*, volume 7404 of *Lecture Notes in Computer Science*, pages 85–100. Springer Berlin Heidelberg, 2012.

[62] D. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157 vol.2. IEEE, 1999.

[63] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov. Scalable distributed algorithm for approximate nearest neighbor search problem in high dimensional general metric spaces. In G. Navarro and V. Pestov, editors, *Similarity Search and Applications*, volume 7404 of *Lecture Notes in Computer Science*, pages 132–147. Springer Berlin Heidelberg, 2012.

[64] M. L. Mico, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with

linear preprocessing time and memory requirements. *Pattern Recogn. Lett.*, 15(1):9–17, 1994.

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

[66] MPEG-7. Multimedia content description interfaces. Part 3: Visual. ISO/IEC 15938-3:2002, 2002.

[67] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 2161–2168, Washington, DC, USA, 2006. IEEE Computer Society.

[68] D. Novák, M. Batko, and P. Zezula. Metric index: An efficient and scalable solution for precise and approximate similarity search. *Information Systems*, 36(4):721–733, 2011.

[69] D. Novák, M. Batko, and P. Zezula. Large-scale similarity data management with distributed metric index. *Inf. Process. Manage.*, 48(5):855–872, Sept. 2012.

[70] D. Novák and P. Zezula. Rank aggregation of candidate sets for efficient similarity search. In *Database and Expert Systems Applications*, volume 8645 of *Lecture Notes in Computer Science*, pages 42–58. Springer International Publishing, 2014.

[71] NVIDIA. Kepler GPU Architecture `http://www.nvidia.com/object/nvidia-kepler.html`.

[72] NVIDIA. *Maxwell GPU Architecture* `http://developer.nvidia.com/maxwell-compute-architecture`.

[73] B. Park, K. Lee, and S. Lee. A new similarity measure for random signatures: Perceptually modified hausdorff distance. In J. Blanc-Talon, W. Philips, D. Popescu, and P. Scheunders, editors, *Advanced Concepts for Intelligent Vision Systems*, volume 4179 of *Lecture Notes in Computer Science*, pages 990–1001. Springer Berlin Heidelberg, 2006.

[74] M. Patella and P. Ciaccia. Approximate similarity search: A multi-faceted problem. *J. Discrete Algorithms*, 7(1):36–48, 2009.

[75] F. Perronnin, Y. Liu, J. Sánchez, and H. Poirier. Large-scale image retrieval with compressed fisher vectors. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3384–3391. IEEE, 2010.

[76] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), 18-23 June 2007, Minneapolis, Minnesota, USA*, pages 1–8. IEEE Computer Society, 2007.

[77] J. Ross, L. Irani, M. S. Silberman, A. Zaldivar, and B. Tomlinson. Who are the crowdworkers?: Shifting demographics in mechanical turk. In *CHI '10 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '10, pages 2863–2872, New York, NY, USA, 2010. ACM.

[78] Y. Rubner and C. Tomasi. *Perceptual Metrics for Image Database Navigation*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[79] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *Int. J. Comput. Vision*, 40(2):99–121, Nov. 2000.

[80] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[81] K. Schoeffmann, M. A. Hudelist, and J. Huber. Video interaction tools: A survey of recent work. *ACM Comput. Surv.*, 48(1):14:1–14:34, Sept. 2015.

[82] S. Shirdhonkar and D. Jacobs. Approximate earth movers distance in linear time. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1 –8. IEEE, june 2008.

[83] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pages 1470–, Washington, DC, USA, 2003. IEEE Computer Society.

[84] T. Skopal. Unified framework for fast exact and approximate search in dissimilarity spaces. *ACM Trans. Database Syst.*, 32(4), 2007.

[85] T. Skopal, T. Bartoš, and J. Lokoč. On (not) indexing quadratic form distance by metric access methods. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 249–258, New York, NY, USA, 2011. ACM.

[86] T. Skopal, J. Lokoč, and B. Bustos. D-cache: Universal distance cache for metric access methods. *Knowledge and Data Engineering, IEEE Transactions on*, 24(5):868–881, May 2012.

[87] T. Skopal, J. Pokorný, and V. Snášel. Nearest Neighbours Search Using the PM-Tree. In L. Zhou, B. Ooi, and X. Meng, editors, *Database Systems for Advanced Applications*, volume 3453 of *Lecture Notes in Computer Science*, pages 803–815. Springer Berlin Heidelberg, 2005.

[88] M. Swan. Emerging patient-driven health care models: an examination of health social networks, consumer personalized medicine and quantified self-tracking. *International journal of environmental research and public health*, 6(2):492–525, 2009.

[89] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 1–9. IEEE, June 2015.

[90] H. Tamura, S. Mori, and T. Yamawaki. Textural features corresponding to visual perception. *Systems, Man and Cybernetics, IEEE Transactions on*, 8(6):460–473, June 1978.

[91] R. Uribe, G. Navarro, R. Barrientos, and M. Marín. An index data structure for searching in metric space databases. In V. Alexandrov, G. van Albada, P. Sloot, and J. Dongarra, editors, *Computational Science ICCS 2006*, volume 3991 of *Lecture Notes in Computer Science*, pages 611–617. Springer Berlin Heidelberg, 2006.

[92] M. Uysal, C. Beecks, and T. Seidl. On efficient content-based near-duplicate video detection. In *Content-Based Multimedia Indexing (CBMI), 2015 13th International Workshop on*, pages 1–6. IEEE, June 2015.

[93] M. S. Uysal, C. Beecks, D. Sabinasz, and T. Seidl. Felicity: A flexible video similarity search framework using the earth mover's distance. In G. Amato, R. Connor, F. Falchi, and C. Gennaro, editors, *Similarity Search and Applications*, volume 9371 of *Lecture Notes in Computer Science*, pages 347–350. Springer International Publishing, 2015.

[94] M. S. Uysal, C. Beecks, J. Schmücking, and T. Seidl. Efficient filter approximation using the earth mover's distance in very large multimedia databases with feature signatures. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, pages 979–988, New York, NY, USA, 2014. ACM.

[95] M. S. Uysal, C. Beecks, and T. Seidl. On efficient query processing with the earth mover's distance. In *Proceedings of the 7th Workshop on Ph.D Students*, PIKM '14, pages 25–32, New York, NY, USA, 2014. ACM.

[96] M. Wichterich, I. Assent, P. Kranen, and T. Seidl. Efficient emd-based similarity search in multimedia databases via flexible dimensionality reduction. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 199–212, New York, NY, USA, 2008. ACM.

[97] L. Wu, S. C. H. Hoi, and N. Yu. Semantics-preserving bag-of-words models and applications. *Trans. Img. Proc.*, 19(7):1908–1920, July 2010.

[98] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Springer US, 2006.

[99] P. Zezula, V. Dohnal, and D. Novák. *Towards Scalability of Similarity Searching*, pages 277–300. IOS Press, Amsterdam, The Netherlands, 2006.

[100] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2), July 2006.