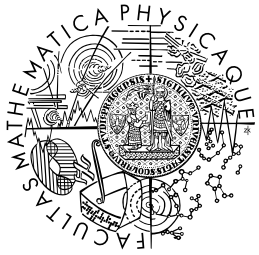


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

DOCTORAL THESIS

RNDr. Přemysl Čech

Content-based exploration of unstructured data

Department of Software Engineering

Supervisor of the doctoral thesis: RNDr. Jakub Lokoč, PhD.
Study programme: Computer Science
Study branch: Software systems

Prague 2019

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

Dedication.

Mainly, I would like to thank immensely my supervisor, RNDr. Jakub Lokoč, Ph.D., for his time dedicated to me and our research, his great ideas, guidance, commitment, proofreading, numerous corrections, and patience. He helped me a lot when I started with research as well as motivated me all the time to finish this thesis.

Next, I thank all coauthors of all the papers we have written together. Especially to Prof. Tomáš Skopal, Dr. Tomáš Grošup, Dr. Juraj Moško, Gregor Kovalčík, BSc., and Tomáš Souček, BSc. My thanks also go to the researchers at the Cisco company (especially Jan Kohout, MSc. and Tomáš Komárek, MSc.) who cooperated with us greatly during our joint GAČR project regarding network security.

I would like to thank enormously my family and girlfriend as well. They supported me greatly and motivated me to finish my doctoral degree.

Furthermore, I would like to acknowledge all the projects and grants that supported our research. Particularly, the Charles University grant GAUK 201515, the Czech Science Foundation (GAČR) projects Nr. 19-22071Y, Nr. 17-22224S and 15-08916S and Charles University grant SVV-260451.

Last but not least, I would like to thank many times Assoc. Prof. Yasin Silva, Ph.D. who arranged my internship at Arizona State University and was my supervisor there and Shenk family who accommodated me in Arizona.

Title: Content-based exploration of unstructured data

Author: RNDr. Přemysl Čech

Department: Department of Software Engineering

Supervisor: RNDr. Jakub Lokoč, PhD., Department of Software Engineering

Abstract: Effective analysis, searching and browsing throughout arbitrary multimedia collections is still a challenging task. To perform a search among multimedia objects, first, a similarity model has to be defined. Such a model establishes methods describing how the content of individual objects is processed and how key features and descriptors, that are used for modeling similarity between objects, are formed. This task is not trivial since there can be many ways of determining how to comprehend the content of multimedia data. Furthermore, with the growing size of contemporary database collections, multimedia retrieval and exploration are extremely computationally intensive. Hence, researchers investigate support indexing structures that can evaluate similarity queries and can respond to user's queries in almost real-time even on datasets counting billions of objects. Another very important aspect of a retrieval system is the user interface for defining queries as well as presenting retrieved results. A multimedia system should offer various inputs for formulating user's queries, especially for situations in which a user cannot provide an ideal query example. Finally, a well-arranged and easy to read interface for visualization of retrieved results is essential for the success of a multimedia exploration and retrieval framework.

In this thesis, we showcase many aspects of content-based retrieval and multimedia exploration in specific scenarios in multiple domains (e.g., images, video, network traffic data). On top of that, we investigate state-of-the-art retrieval prototypes and applications and discuss their advantages and limitations identified by automatic and user experimental evaluations.

To deal with scalability issues, we profoundly study similarity joins for evaluating queries in metric spaces implemented in a distributed MapReduce environment adopting Hadoop and Spark platforms. We propose several variants of similarity joins offering a wide range of algorithms with different speed/precision (accuracy) trade-offs. Specifically, we study exact, approximate, and *epsilon*-approximate joins based on different approaches to data processing parallelization. Moreover, we have published java source codes of presented similarity joins for the Spark platform on the GitHub.com server.

Keywords: Content-based multimedia retrieval, Similarity search, Multimedia exploration, HTTPS classification, MapReduce, Similarity joins

Contents

Introduction

During past years a multimedia world has surrounded us. All of us consume media in a smaller or greater amount, either actively or passively. We are exposed to music and radio broadcast in public places and we watch movies, shows, and various Internet channels on daily bases. The Internet combined with smartphones, laptops, and tablets has become such an important part of our daily activities that living without it is almost unthinkable. A big factor in the multimedia world is also the influence of advertisement across all the media. Internet banners, commercial spots, and even billboards follow us on every step and silently pressure our desires to buy and consume even more multimedia products and content.

From many different sources, including among other things photo and video collections, medical records, voice and sound snapshots, mankind observes enormous volumes of multimedia data (Reinsel et al. [2017]). Moreover, most of the data are unstructured in a raw format and they are difficult to automatically process by machines. Nevertheless, if some descriptive information such as a label or a timestamp is provided with each present object, data can be processed and searched more easily by taking advantage of such information. Another way, which is the main focus of this thesis, is to consider the content of each multimedia object to solve various retrieval tasks (Smeulders et al. [2000], Joshi and Wang [2005]). Multimedia retrieval systems usually employ similarity models that require an object representation/descriptor universe $\mathcal{U}$ and relevance scoring function (usually a distance function δ).

The essential challenge is to find an algorithm for detection and extraction of key features f_i (usually $f_i \in \mathbb{R}^n$) that are used to form descriptors of multimedia objects $o \in \mathcal{U}$. The features should satisfy several conditions. They should be, above all, representative and repeatedly detectable for the modeled objects, and invariant to various deformations. Last but not least, the features and derived descriptors should be as compact as possible, meaning that huge data collections can be represented by significantly smaller representations. Another challenge is to combine the descriptors with effective relevance scoring functions to rank database objects in order to satisfy user needs.

The next aspect of the content-based similarity search are effective and convenient search modes for a given task (e.g., query by sketch/example approaches) and efficient query processing (e.g., Hamming embedding (Jégou et al. [2008]), or vector/metric space filtering described in Chapter 1) in multimedia databases. A straightforward option is to consider established (open source) data management frameworks that natively support querying and filtering of data. An example of such a framework can be the Lucene Image Retrieval (LIRE) project[1], Vitrivr project[2], or applications developed by the DISA team[3] (e.g., image retrieval framework Mufin[4] or the motion retrieval demo[5]). However, established multimedia database frameworks do not have to fit intended retrieval models, data access pipelines and implementation needs. In addition, it can be sometimes undesired

[1] http://www.lire-project.net
[2] https://vitrivr.org
[3] http://disa.fi.muni.cz/demos
[4] http://mufin.fi.muni.cz/imgsearch
[5] http://disa.fi.muni.cz/mocap-demo

from the system design perspective to include a huge framework for only one minor subtask. Hence, a popular option (especially for research purposes) is to design and implement a custom data access prototype tailored for specific retrieval tasks, considering just a more general framework (e.g., Spark framework).

There are two common scenarios for manipulating a multimedia database. The first approach is to directly search and filter data by a suitable query object. In such case, the most relevant objects are returned. Another approach is called multimedia exploration (Heesch [2008], Beecks et al. [2011b]). The goal of the exploration is to browse a collection, explore its content and refine user's interests during the process. The multimedia exploration is especially useful in cases when a user cannot provide a perfect input query object. For example, cases when a user saw a picture of a building, but they do not know what building that is, and they cannot describe it in any specific way. In this thesis, we tackle both scenarios with an emphasis on efficient query processing of large multimedia collections.

The key idea behind a fast query evaluation is the construction of some support data access structures that consider distribution and additional properties of database multimedia objects. These structures are called indexes. Indexes can be static or dynamic determining whether an index is built only once before performing any similarity queries or if it is updated with new data. Moreover, when an index is built using a similarity function satisfying metric space postulates (see Chapter 1) it is called a metric index or a metric access method. Our research focuses mainly on the metric space approach and we further develop indexing in a distributed environment that satisfies the needs of fast query evaluation on big databases.

The thesis draws from our research published in reviewed conference papers Lokoc et al. [2012, 2014a], Mosko et al. [2015b], Grosup et al. [2015a,b], Cech and Grosup [2015], Mosko et al. [2015a], Cech et al. [2016], Lokoc et al. [2016, 2017], Cech et al. [2017], Lokoc et al. [2019a,b,c], Lokoč et al. [2019] and in journal papers Lokoc et al. [2014b], Kohout et al. [2018]. Chapter 5 is based on the work of Cech et al. [2020] published in the Information Systems journal written mostly by the author of this thesis.

Contributions

In the thesis, different domains suitable for similarity search and multimedia exploration are studied. In each domain, we investigate retrieval techniques including feature extraction, effective similarity modeling and fast query evaluation typically using some sort of support indexing structures. In order to process huge volumes of data (i.e., beyond a single machine), we propose distributed similarity join algorithms running on a cluster of computers. The areas of our contributions are summarized in the following points:

- Multimedia exploration approaches for big image databases. We present several prototypes demonstrating various similarity models and browsing techniques. The applications were tested in multiple scenarios performed by both expert and novice (voluntary) users.
- Video retrieval models and applications for interactive known-item and ad-hoc search. We summarize approaches for effective exploration of large

video collections which were (successfully) tested during the Video browser showdown[6] and Lifelog search challenge competitions.

- Network security retrieval and modeling. We study also a different domain tackling HTTPS communication and discuss how to capture key features in network communications, aggregate them into communication snapshot descriptors, and how to detect malicious activities. Moreover, we present demonstration applications for network data retrieval that can help domain experts to interactively search for potentially infected nodes in a network.

- Centralized and distributed similarity queries. The main contribution of the thesis covers distributed similarity joins based on the metric space approach. We propose several algorithms for fast exact and approximate (with or without an approximation guarantee) similarity joins, we discuss the advantages and disadvantages of presented approaches and all joins are compared in thorough experimental evaluations. On top of that, source codes for MapReduce similarity joins are publicly available on Github[7].

My contributions vary for specific topics. For image and video domain, I mainly worked on interface design and implemented prototypes. I also studied and developed responsive index/exploration structures that support fast evaluation of similarity queries. In the network security project, I analyzed solutions for distributed computation, focused on descriptor extraction methods in MapReduce and evaluation of large scale k-NN joins in Hadoop. Finally, my main contributions concern thorough work on distributed joins and analyses of various exact and approximate solutions. I worked on three MapReduce-based pivot k-NN joins for different levels of guarantee of an approximation error implemented in two MapReduce platforms Hadoop and Spark.

Thesis overview

The thesis is organized in the following order. In Chapter 1, key definitions and terms are presented, focusing on the basics of similarity search, metric space approach, metric indexing structures, and distributed computing. In Chapter 2, we summarize the essentials of multimedia browsing and exploration including motivation, challenges, and difficulties that researchers encountered in this area. Furthermore, we cite some state-of-the-art applications presented in this field of research and we study visualization methods and interfaces as well. Beginning with Chapter 3, we present our contributions to multiple domains. Specifically, in Chapter 3 we recapitulate our interactive retrieval applications in image and video domains starting with first pioneering prototypes to frameworks that have been systematically developed by our research team for a couple of last years. Another domain for searching and multimedia exploration is presented in Chapter 4, where we debate similarity models, classification of HTTPS requests, and applications in (secured) network traffic. We focus mainly on the detection of malicious communications and also on user-assisted applications that can expose suspicious

[6] http://www.videobrowsershowdown.org/

[7] https://github.com/PremyslCech/kNN-joins-spark

traffic. Chapter 5 presents (with minor modifications) our journal paper by Cech et al. [2020] on distributed similarity joins with big data. Moreover, the investigated approaches can be conveniently used to create similarity graphs (e.g., k nearest neighbors networks) that are beneficial in many exploration scenarios. Finally, we conclude the thesis and discuss new challenges.

1. Preliminaries

Fundamental definitions and concepts for a proper understanding of the presented methods and algorithms are summarized in this chapter. Notations come mainly from works of Zezula et al. [2006], Song et al. [2016], Čech [2014], or are cited directly in specific sections.

The chapter includes four sections. Similarity search, similarity queries and characteristics of metric spaces are described first. We also show examples of methods for modeling multimedia objects, the feature extraction process, and examples of similarity measures. Then we explain how metric indexing structures help with processing of similarity queries and we detail some selected structures. Finally, we define distributed computations sticking to the MapReduce paradigm including frameworks Hadoop and Spark. Hadoop and Spark are introduced also with their main components such as the Hadoop file system, the YARN resource manager and we discuss the advantages and disadvantages of both MapReduce platforms.

1.1 Similarity search

Nowadays, research trends regarding big data analysis, analytics, transformation, and processing focus heavily on artificial intelligence and machine learning algorithms (Pedregosa et al. [2011], Abadi et al. [2016], Jordan and Mitchell [2015]). However, machine learning techniques usually rely on a lot of training data to correctly setup models (learning process). In case of a lack of training data, unsupervised learning can be viable to cluster or organize data. To achieve that, some sort of similarity between studied data objects has to be defined. First, for multimedia objects, descriptors are extracted. The extraction process describes the transformation of objects from a domain $\mathcal{D}$ to a descriptor universe $\mathcal{U}$. Furthermore, a similarity function δ is defined between two arbitrary objects $x_1, x_2 \in \mathcal{U}$ as: $\delta : \mathcal{U} \times \mathcal{U} \mapsto \mathbb{R}$. Finally, a set of mechanisms for querying, filtering, browsing and manipulating the data $\mathcal{D}$ concerning the feature space $\mathcal{U}$ and the function δ is specified for a *similarity search* system.

Similarity search can be useful not only for querying a database but also for data browsing and exploration. Using similarity search we can, for example, collect more training samples for machine learning algorithms. Or users can explore a collection and iteratively narrow their interests and filter data according to retrieved multimedia objects. Such process is called *multimedia exploration*. In other words, multimedia exploration can be characterized as an iterative user-assisted process of browsing, navigating and querying steps in a potentially unknown database (Beecks et al. [2013]). The exploration is described in more detail in Chapter 2.

1.2 Metric Space

A dataset $\mathcal{S} \subseteq \mathcal{U}$ containing many objects is usually hard to process sequentially. Furthermore, an arbitrary function δ can be complex and expensive to evaluate.

Restricting δ by a set of metric postulates enables to construct support structures (see Section 1.3.5) which can significantly speed up query processing. An important postulate is the triangle inequality to estimate an unknown distance between two objects from other pre-computed distances (for more details see Section 1.3.1). Formally, a metric space (Zezula et al. [2006]) is formalized in Definition 1:

Definition 1 (Metric space)**.** *Let $\mathcal{U}$ be a descriptor (feature) space and δ a distance function measured on $\mathcal{U}$. A tuple $\mathcal{M} = (\mathcal{U}, \delta)$ is called the metric space, if the distance function δ: $\mathcal{U} \times \mathcal{U} \mapsto \mathbb{R}$ fulfill following postulates:*

(p1) $\forall x, y \in \mathcal{U},\ \delta(x, y) \geq 0$ *non-negativity*

(p2) $\forall x, y \in \mathcal{U},\ \delta(x, y) = \delta(y, x)$ *symmetry*

(p3) $\forall x \in \mathcal{U},\ \delta(x, x) = 0$ *reflexivity*

(p4) $\forall x, y \in \mathcal{U},\ x \neq y \Rightarrow \delta(x, y) > 0$ *positiveness*

(p5) $\forall x, y, z \in \mathcal{U},\ \delta(x, z) \leq \delta(x, y) + \delta(y, z)$ *triangle inequality*

One advantage of the metric space approach is that a dataset $\mathcal{S} \subseteq \mathcal{U}$ is not restricted just to n-dimensional vectors $\mathbb{R}^n$, but to any kind of multimedia data (for example, text strings, feature signatures, DNA sequences, etc.).

In the following, we discuss in more detail selected examples of descriptors, distances, query types, and evaluation metrics. We primarily inspect topics related to our research, developed prototypes, and published papers.

1.2.1 Descriptor universe

In this subsection, examples of algorithms for obtaining objects' key features and descriptors (mapping of objects $obj \in \mathcal{D}$ to descriptors $x \in \mathcal{U}$) are presented. Extraction process can produce a set of features $\{f_i\}$ or one descriptor x for each object obj. A set of features $\{f_i\}$ can be converted to one descriptor x by various methods, for example, employing joint-histogram or the bag of features (BoF) approaches (Sivic and Zisserman [2003], Delaitre et al. [2010]) inspired by the bag of words methods from the text retrieval domain. In this subsection, we cover some of the popular image modeling techniques that were also investigated/tested in our research.

In recent years, many algorithms related to content-based image retrieval (CBIR) (Zhou et al. [2017a]) as well as many techniques for extracting key features from images (Karuppusamy and Marappan [2016], Schettini et al. [2018]) were proposed. For images, the similarity is typically modeled by visual or conceptual features recognizable in pictures. One image typically contains more features such as color, texture or an interesting (key) point in a picture. A specific similarity measure is usually connected with each type of descriptor.

Some decades ago, the SIFT (Lowe [1999, 2004]) and SURF (Bay et al. [2006]) key features became very popular (Figure 1.1). SIFTs represent significant points or regions in images using scale and rotation invariant vectors. For an image, a set of so-called octaves is produced. Each octave represents one image scale and contains a set of blurred images formed by employing Gaussian filters convolved with the original image. Significant points are identified as local maxima or

Source: https://upload.wikimedia.org/wikipedia/commons/4/44/Sift_keypoints_filtering.jpg

Figure 1.1: Example of SIFT key points.

minima of the difference of Gaussians that occur at multiple scales (octaves). Furthermore, the SIFT extraction algorithm removes noisy points (e.g., spots with low contrast or points located on edges without unique location specifics) and outputs oriented key points. Finally, having an oriented SIFT key point, its feature descriptor is computed as a joint vector of orientation histograms around the key point (typically, 4x4 histograms from surrounding area 16x16 regions in the neighborhood of the point). For each image, multiple key points are computed. A whole image descriptor can be formed utilizing the BoF approach.

The SURF descriptors find relevant points by utilizing the Hessian detector which is composed of convolutions of Gaussian second-order derivatives. SURF features are not based on orientation histograms but describe a distribution of Haar wavelet responses within the interest point neighborhood.

Furthermore, for local feature points utilizing BoF methods, several approaches increasing the precision and/or speeding up evaluation time of such representations were proposed. For example, Jégou et al. [2008] introduced the Hamming embedding (HE) algorithm that encodes the geometric location of a descriptor in a feature space using binary codes. Employing the BoF algorithm, features are typically organized in an inverted file of visual features (words) b_i. Hence, a query feature f_q matches another feature f_i if they belong to the same visual feature b_i from the bag. Nevertheless, HE adds an additional comparison condition specifying that f_q matches f_i if also the distance between hamming codes of features is below the given threshold. It helps especially in cases in which bag features b_i represent wide clusters and even more distant features would belong to the same bag word b_i. On top of that, evaluating the hamming distance is very fast since it

can be computed exploiting bit operations such as XOR.

Another problem connected with matching local features is called geometric verification (Stewénius et al. [2012]). Images are considered similar if they share some similar local features. However, it has to be verified that the matched features are located in similar positions in images (considering also, for example scale and rotation transformations) to confirm that the pictures are truly similar.

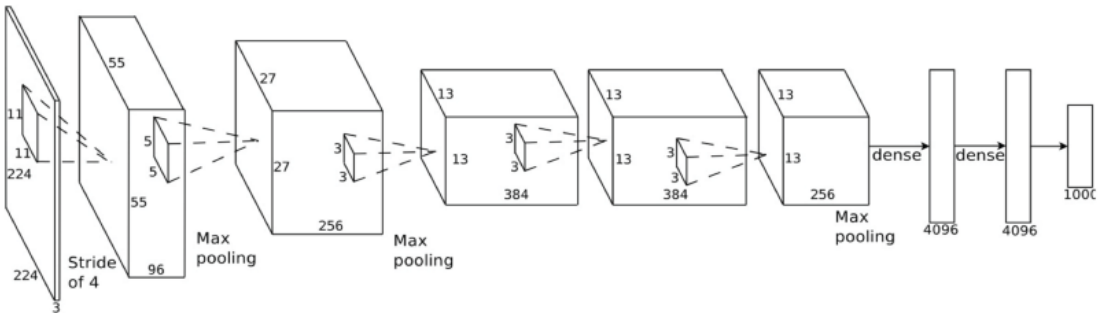
The biggest advantage of local feature approaches is good precision and performance for searching near duplicates[1]. Vector representations of the features are not long (e.g., 128 dimensions for SIFT or 64 dimensions for SURF features) and can be compared with the cosine similarity or other fast similarity measures. However, the disadvantage is the extraction speed since computing multiple derivatives and selecting significant and relevant points can be computationally intensive (Grabner et al. [2006]).

Figure 1.2: Feature signature descriptors based on color distribution in the images. The illustration comes from the paper by Beecks et al. [2011a].

Another example of descriptors are position-color-texture (PCT) Feature signatures (Rubner and Tomasi [2013], Beecks et al. [2011a], Krulis et al. [2016]). An example is depicted in Figure 1.2. From a predefined set of selected (pixel) features (position, color, and texture), a set of centroids can be formed by the k-means algorithm. Final image descriptors consist of the centroids, each with an assigned weight. Unlike BoF relying on a shared feature space dictionary, feature signatures flexibly model the contents of each multimedia object. Feature signatures are typically compared by the Signature Quadratic Form Distance (SQFD) that considers similarity relations between centroids in the dynamically computed matrix M.

Since 2012, state-of-the-art Deep Convolutional Neural Networks (DCNN Szegedy et al. [2015], He et al. [2016], Krizhevsky et al. [2017]) started to outperform traditional approaches in many multimedia retrieval and classification tasks. Networks are usually composed of convolutional, pooling and fully connected layers. An example of a DCNN architecture is shown in Figure 1.3. Using stacked convolutional layers enables to learn an effective hierarchy of features

[1]Nonetheless, today machine learning approaches such as deep convolutional networks usually overcome the precision of the mentioned methods.



Source: https://ailephant.com/glossary/convolutional-neural-network

Figure 1.3: An example of a deep convolutional neural network.

for classification tasks. The outputs from the last pooling, convolutional, or fully connected layer can be used to represent images (Donahue et al. [2014]). These representations showcase impressive effectiveness in many similarity search tasks because they can capture semantic information in images. The DCNN descriptors usually form long vectors which can be easily compared by, e.g., the Euclidean or cosine distances. The problem of the DCNN approach is the training phase. For training a neural network, a lot of annotated data have to be prepared (ideally millions of images) and also the training itself can be very computationally intensive. Nevertheless, there are DCNNs available for public use (e.g., DCNN trained on the ImageNet: Russakovsky et al. [2015], Krizhevsky et al. [2017]) which are pre-trained and can be used directly or adjusted for specific needs by just a smaller training sample. Lately, DCNNs were used to determine local feature points as well (for example, LIFT by Yi et al. [2016]).

Moreover, deep convolutional networks can be also used to obtain annotations (e.g., class labels assigned to each picture) that enable keyword search and so the initial query image is not necessary (see Chapter 2). This approach is especially handy in cases in which users cannot provide a suitable initial query image.

1.2.2 Distance measures

In this subsection, we list some commonly used distance functions. Distance functions can be divided into two categories. The first category is composed of continuous distances returning a wide range of values (typically infinity different values). Examples are Minkowski or Quadratic form distances. The second class contains discrete distances that return only a limited number of values. An example is the edit distance on data represented as text strings.

Minkowski distances

The Minkowski distances represent a whole set of functions marked as L_p defined in vector spaces $\mathbb{R}^n$, where $p \geq 1$:

$$\forall x, y \in \mathbb{R}^n : L_p(x, y) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \tag{1.1}$$

One of the most popular Minkowski distance function is the Euclidean L_2 distance.

Cosine distance

The cosine distance is a similarity measure between two vectors in $\mathbb{R}^n$. It measures the cosine angle between them and because $\cos(0°)$ is equal to 1 and vectors with the angle equal to 0° are the most similar ones, the result is subtracted from the number 1 constant. Formally:

$$\forall x, y \in \mathbb{R}^n : \delta_{cos}(x, y) = 1 - \frac{x \cdot y}{|x| \cdot |y|} \tag{1.2}$$

Quadratic form distance

The quadratic form distance (Zezula et al. [2006]) is useful in scenarios where some parts of vectors should be compared in a specific way. Formally,

$$\forall x, y \in \mathbb{R}^n : \delta_{QF}(x, y) = \sqrt{(x - y)^T \cdot M \cdot (x - y)}, \tag{1.3}$$

where M is a positive semidefinite matrix modeling similarity relations between dimensions. This distance can be, for example, employed for vectors in which dimensions have various meanings (e.g., a histogram of colors). For such vectors, it can be desirable to compare different values from two vectors employing different correlations. For example, the red color dimension should have the strongest connection with the red counterpart, weaker relation with pink and orange dimensions and zero (no) connection with the blue color part. Note that the L_2 distance is a special case of the quadratic form distance when M is equal to a diagonal 1-matrix.

The signature quadratic form distance (SQFD by Beecks et al. [2009, 2011a]) inspired by δ_{QF} applies similar principles to sets of feature signatures presented earlier and proved to be useful in similarity search and exploration.

Edit distance

The edit distance also called the Levenshtein distance (Levenshtein [1965]) is defined as the smallest number of operations needed for a transformation of one string to another one. In total, three operations are defined. *Insert* for inserting a character somewhere in a string, *delete* for removing a character from a string and *replace* for replacing one character with another one.

Similarly, we can define the tree edit distance measure. It defines the smallest number of operations required for a transformation of a tree structure to another one. There can be defined more operations for tree transformations (e.g., relabel) which can be also weighted by a tree depth (Bille [2005]). XML documents can be modeled also as trees so this distance can also measure the similarity between the documents (Guha et al. [2002]).

Jaccard's distance

The Jaccard's distance is defined on sets. Simply, it represents the shifted ratio between sizes of intersection and union of two sets A, B:

$$\delta_J(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{1.4}$$

This measure is useful for comparing, for example, sets of words in documents or bag of entities.

The time complexity of similarity measures has a huge influence on the performance of the similarity search. To evaluate a query, a large number of similarity computations have to be performed. A faster similarity function speeds up the whole content-based querying framework substantially. In general, it is usually more efficient to produce key features from multimedia data in a way that they can be compared by a fast similarity function (e.g., an L_p distance, having linear asymptotic complexity) even though such extraction could take longer pre-processing time.

1.2.3 Similarity queries

Similarity queries formally express user's desires and demands put on a similarity querying system. Queries are usually specified by a given example (a query object) and a threshold that limits the number of the most similar retrieved objects from the database. The two most frequent queries are called the range and the k nearest neighbors query. In the similarity search area, there are other defined similarity queries (e.g., skyline queries) but they are not much related to the research presented in this thesis. Similarity joins, defined later in Chapter 5, generalize query types for cases when provided input is composed of a whole set of objects (queries) instead of just one query example.

Range query

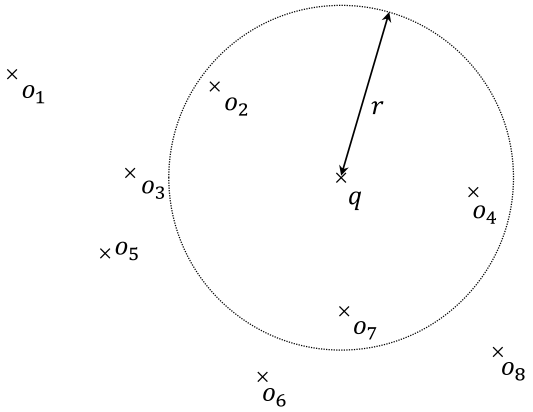


Figure 1.4: Range query example from an object q with a distance r. The image comes from the work of Čech [2014].

Definition 2 (Range query)**.** *Let* $\mathrm{q} \in \mathcal{U}$ *be a query object,* $\mathcal{S} \subseteq \mathcal{U}$ *be a dataset, and* $\mathrm{r} \geq 0$ *be a distance. The range query is defined as* $R(\mathrm{q}, \mathrm{r}, \mathcal{S}) = \{o \in \mathcal{S}, \delta(o, \mathrm{q}) \leq \mathrm{r}\}$*, where* δ *is a distance function defined on* $\mathcal{U}$*.*

The range query returns objects $o \in \mathcal{S}$ within the given distance r from q (Figure 1.4). In the real world, this query can be used, for example, to find all gas stations within 50 kilometers from the current location.

K nearest neighbors query

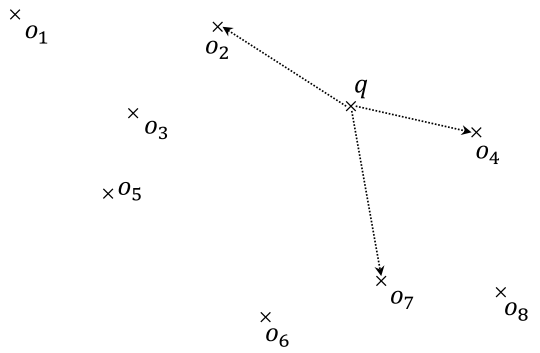


Figure 1.5: Example of k-nearest neighbor query from an object q with $k = 3$. The figure was originally published in the work of Čech [2014].

Definition 3 (k-NN query)**.** *For a dataset* $\mathcal{S} \subseteq \mathcal{U}$*, an input query* $\mathrm{q} \in \mathcal{U}$*, and distance function* δ *defined on* $\mathcal{U}$*, the* k *nearest neighbors query is defined as:* $kNN(q, \mathcal{S}) = \{X \subset \mathcal{S}; |X| = k \wedge \forall x \in X, \forall y \in \mathcal{S} - X : \delta(q, x) \leq \delta(q, y)\}$

The k nearest neighbors query returns the k most similar objects $o \in \mathcal{S}$ to the given input example q (Figure 1.5). In the real world, this query can be used, for example, to find the three closest restaurants or to find the closest one to your location.

1.2.4 Precision, recall

The *precision* and *recall* represent popular measures to assess effectiveness of an information retrieval system. Precision defines the ratio between the number of retrieved relevant and all retrieved objects.

$$Precision = \frac{|retrieved\ relevant\ objects|}{|retrieved\ objects|} \tag{1.5}$$

Recall defines the ratio between the number of retrieved relevant and all relevant objects in a database $\mathcal{S}$.

$$Recall = \frac{|retrieved\ relevant\ objects|}{|relevant\ objects|} \tag{1.6}$$

In similarity search frameworks, it is almost impossible to maximize both criteria. Typically, with growing precision, the recall is dropping and, vice versa, with growing recall the precision is lower. This behavior is explained by the fact

that for getting more relevant results, the result set must be larger and, usually, more irrelevant objects are returned leading to lower precision.

Other than that, precision can be also used for measuring an approximation error. In that case, objects obtained by the exact search are considered relevant and precision indicates how many objects from the exact search are missing when some approximate rules are employed.

1.3 Metric access methods - indexes

The naive approach for evaluating similarity queries is to sequentially browse a collection, evaluate distances and return the most similar objects. However, for large multimedia collections, this approach is not viable or is just too slow. Constructing support structures storing pre-computed distances helps filter out and prune irrelevant database objects and only distance computations from the query q to suitable database candidates are evaluated. These structures are called metric indexes or metric access methods. There are many known indexing principles (Chen et al. [2017]). First, we start with the basics of pruning and filtering methods and then we present some related methods in the following subsections.

1.3.1 Metric filtering principles

In this section (taken from the paper by Cech et al. [2020]), we briefly recapitulate the fundamental principles of distance based metric indexing for exact similarity search (Zezula et al. [2006]) since filtering strategies are also important for similarity joins studied in Chapter 5. Note that a range or k-NN query corresponds to a metric space ball-region $B = Ball(q, r_q)$ (Definition 4) selecting objects from $\mathcal{S}$ based on the dynamically estimated radius $r_q \in \mathbb{R}_0^+$.

Definition 4 (Ball-region)**.** *Let* $\mathrm{p} \in \mathcal{U}$ *be an object in a universe* $\mathcal{U}$*,* δ *a similarity measure, and* $\mathrm{r} \geq 0$ *be a distance (radius). Then* $B(\mathrm{p}, \mathrm{r}) = \{o \in \mathcal{U} \mid \delta(\mathrm{p}, o) \leq \mathrm{r}\}$ *is called the ball-region.*

Given a metric space $M = (\mathcal{U}, \delta)$, distance based approaches rely on pre-computed distances to a set of reference points, so called pivots $\mathbb{P} \subset \mathcal{S}$. In conjunction with the triangle inequality property of δ, these distances can be used to efficiently estimate the lower-bound δ_{LB} and upper-bound δ_{UB} distances between a query object q and a database object $o \in \mathcal{S}$. Formally, given a pivot $p \in \mathbb{P}$ and pre-computed distances $\delta(q, p)$ and $\delta(o, p)$:

$$\delta_{LB}(q, o) = |\delta(q, p) - \delta(o, p)| \leq \delta(q, o) \leq \delta(q, p) + \delta(o, p) = \delta_{UB}(q, o). \tag{1.7}$$

The k-NN query processing is usually designed as an algorithm that maintains and greedily updates the actual set of k closest candidate objects from $\mathcal{S}$, using also the actual query ball radius $r'_q \in \mathbb{R}_0^+, r'_q \geq r_q$. Hence, given the actual query ball $\mathrm{Q} = Ball(q, r'_q)$, an object $o \in \mathcal{S}$ can be filtered if

$$r'_q < \max_{\forall p \in \mathbb{P}} |\delta(q, p) - \delta(o, p)|, \tag{1.8}$$

without the evaluation of $\delta(q, o)$ which gets costly with high-dimensional data.

The metric space approach provides also two basic data partitioning options for filtering entire groups of objects. The first option is by making use of the already mentioned ball-regions. Given a ball-region based data chunk $X \subset Ball(p, r_p)$ containing selected dataset objects and the actual query ball $\mathrm{Q} = Ball(q, r'_q)$, all objects in X can be filtered if

$$r_p + r'_q < \delta(p, q). \tag{1.9}$$

Formally:

Lemma 1 (Overlap of two ball-regions)**.** *Let $B_o = B(\mathrm{o}, \mathrm{r})$ be a covering ball-region and $B_q = B(\mathrm{q}, \mathrm{r_q})$ be a range query ball-region. If $\delta(\mathrm{o}, \mathrm{q}) > \mathrm{r} + \mathrm{r_q}$ then B_o and B_q do not share any object (Figure 1.6).*

Proof 1 (Overlap of two ball-regions)**.** *A possible intersection of two balls have to be investigated. There are three assumptions:*

(1) $\forall x, y \in \mathcal{S},\ \delta(x, y) \geq 0$ — *from the metric space definition*

(2) $\delta(o, q) > r + r_q$ — *from the lemma assumptions*

(3) $\forall o_i \in B,\ \delta(o, o_i) \leq r$ — *from the ball-region definition*

Now:

$\delta(o, q) \leq \delta(o, o_i) + \delta(o_i, q)$ — *from the triangle inequality*

$\delta(o, q) - \delta(o, o_i) \leq \delta(o_i, q)$ — *subtracting $\delta(o, o_i)$ from both sides*

$\delta(o, q) - r \leq \delta(o_i, q)$ — *using (1) and (3)*

$r + r_q - r < \delta(o_i, q)$ — *composition with (2)*

$r_q < \delta(o_i, q)$

If $\delta(o_i, q) > r_q$ then it stands that for any $o_i \in B_o$ the query B_q does not include any object from B. □

The (generalized) hyperplane partitioning represents the second option, which incorporates two pivots. Using pivots $p_1, p_2 \in \mathbb{P}$, the metric space is divided into two sets $S_1 = \{o \mid o \in \mathcal{S}, \delta(p_1, o) \leq \delta(p_2, o)\}$ and $S_2 = \{o \mid o \in \mathcal{S}, \delta(p_2, o) < \delta(p_1, o)\}$. Given a hyperplane based data chunk $X \subset S_1$ containing selected dataset objects and the actual query ball $\mathrm{Q} = Ball(q, r'_q)$, all objects in X can be filtered if

$$\delta(p_1, q) - r'_q > \delta(p_2, q) + r'_q. \tag{1.10}$$

The presented principles are frequently used by various metric access methods for an efficient exact k-NN search (Zezula et al. [2006]). The principles are directly implied from metric axioms. However, in high-dimensional spaces, the distances between dataset objects are often high and similar (the curse of dimensionality effect mentioned in Section 1.3.5). Hence, the conditions to safely prune some

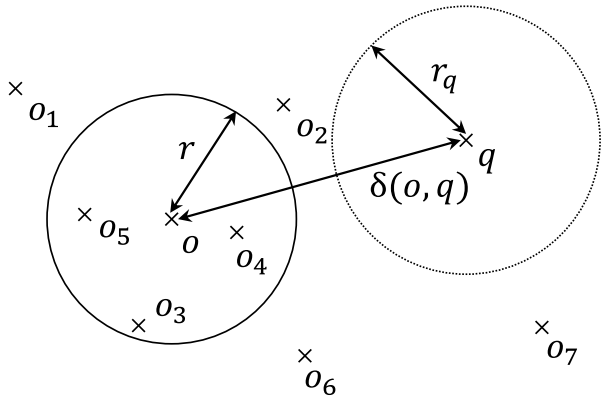


Figure 1.6: Schema of a covering ball and query range ball-region overlap. The figure was originally published in the paper by Lokoc et al. [2014b].

objects or partitions are mostly not satisfied and the approximate and/or distributed search becomes necessary for more efficient retrieval (more details are presented in Chapter 5).

In the following sections, selected metric access methods are presented to give an idea how pivots and pruning strategies are used to filter out irrelevant database objects without computing distances δ.

1.3.2 Pivot table

One of the most straight forward indexes is the Pivot table (Figure 1.7). It basically stores only pre-computed distances from a set of global pivots $p_i \in \mathbb{P} \subset \mathcal{S}$ to all database objects $o \in \mathcal{S}$. In the pre-processing phase, all distances to pivots p_i have to be evaluated. The distance matrix is usually stored in memory if possible.

To evaluate a query q, it has to be mapped in the same pivot space $\mathbb{P}$ first, meaning distances from q to all pivots $p_i \in \mathbb{P}$ are computed. Then the lower bounds of distances between all objects $o \in \mathcal{S}$ and q are evaluated as

$$LB(\delta(q,o)) = \max_{p_i \in \mathbb{P}} \{|\delta(q,p_i) - \delta(o,p_i)|\} \leq \delta(q,o). \tag{1.11}$$

The range query $R(q, r, \mathcal{S})$ algorithm focuses only on such objects for which the lower bound is smaller than the radius r. For those objects, the similarity measure δ has to be computed and the distance is compared with the radius r. A kNN$(q, \mathcal{S})$ query processing algorithm can employ the Pivot table in the following way. All objects $o \in \mathcal{S}$ are ordered in the ascending order by the lower bound values and until the lower bound of the currently processed object o is lower or equal to the actual k-NN radius, $\delta(q, o)$ is evaluated and compared with the current k-NN result set (o is included in the current k-NN result set if $\delta(q, o)$ is lower than the distance to the k^{th} neighbor, and then the k^{th} neighbor is removed from the set). When the actual k-NN query radius is smaller than the lower bound of an object, the k-NN search is stopped and the actual k-NN result set is returned.

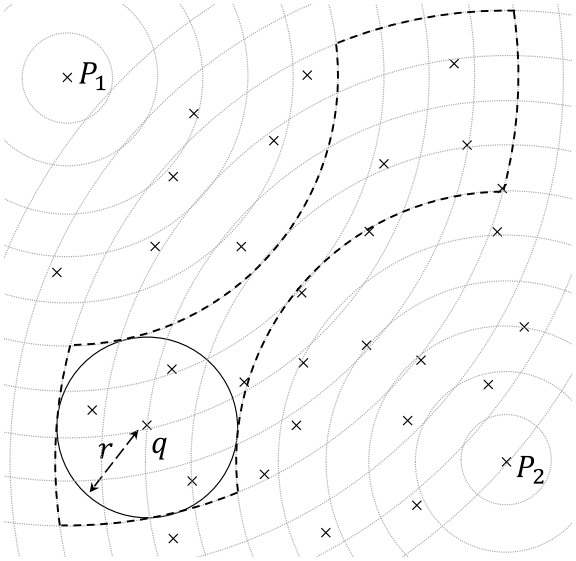


Figure 1.7: Figure illustrating the mapping of objects into the Pivot table with respect to pivots P_1, P_2. The query q with the radius r is also mapped to the pivot space. Using triangle inequality, all objects outside of the black dashed area are irrelevant to the query q and can be pruned from the search. The image comes from the work of Čech [2014].

1.3.3 Metric tree

Another notable metric access method is called the Metric tree or shortly M-Tree (Ciaccia et al. [1997]). It is inspired by the R-tree indexing structure (Guttman [1984]) and applies a similar idea in the metric space. The tree is composed of a hierarchy of covering ball-regions (Figure 1.8) that are formalized in Definition 4.

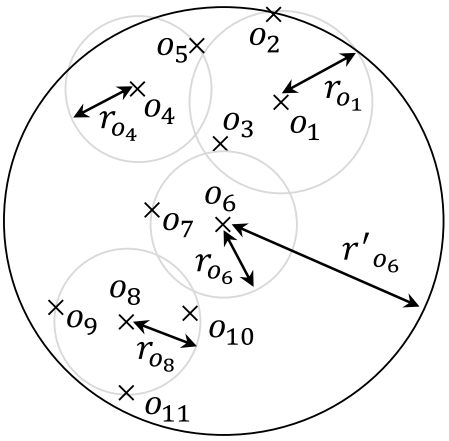


Figure 1.8: Example of a part of the metric tree hierarchy. Objects in the gray ball-regions are stored in leaf nodes that are associated with entries stored in the corresponding inner node. The figure comes from the work of Čech [2014].

The main idea of the indexing algorithm is that each inner node entry represents a ball-region with a center o_i and a covering radius r_{o_i} for covering all entries in the corresponding subtree. Furthermore, each database object has to be assigned to some leaf node. The tree can be constructed and maintained either dynamically

or it can be built once statically if all data are known upfront. In the case of a static structure, in the pre-processing phase, all database objects $o \in \mathcal{S}$ are clustered to a set of ball-regions of given capacity by the similarity function δ (a region contains the most similar objects) and then the regions are recursively organized and again clustered to form up higher levels in the tree hierarchy. In the second case, objects are, usually, inserted to the tree one by one and ball-regions are dynamically adjusted and split according to inserted entities. A new tree layer is formed when the root is overfull (like the R-Tree construction techniques). Dynamically maintained M-Tree supports the delete operation as well. More specifics are presented in the paper by Ciaccia et al. [1997].

A similarity query can be formalized also as a ball-region $B_q = B(q, r)$. In the case of the range query, r is given. For the k-NN search, r is dynamically changing according to the distance $\delta(q, k^{th}\ neighbor)$ starting with $r = \infty$. For evaluating queries, overlap of two ball regions (Lemma 1) is employed.

Given a range query, the M-Tree is traversed starting in the root node and on each level l all ball-regions B_l that do not overlap query ball B_q are pruned. Then the search continues on the lower level $l + 1$ and only descendants of not pruned balls from the level l are considered for further search. On the leaf level, distances $\delta(q, o)$ for objects $o \in \mathcal{S}$ from visited (not pruned) balls are computed if the parent filtering does not prune objects o and the final result set is formed. A k-NN query is processed similarly. Since starting query radius $r = \infty$, no ball-regions can be pruned when the search begins, hence, the k-NN evaluation processes the "closest" ball-regions $B_i = (o_i, r_i)$ based on $\delta_i = \max\{\delta(q, o_i) - r_i, 0\}$. Every time the query radius r is updated (the k^{th} neighbor has changed), all ball-regions B_i, for which $\delta_i > r$, are pruned.

1.3.4 Pivoting metric tree

A combination of the Pivot table with the M-Tree is called the Pivoting metric tree or shortly PM-Tree (Skopal [2004]). The PM-Tree utilizes a set of global pivots $p_i \in \mathbb{P} \subset \mathcal{S}$ to further cut off empty areas in ball-regions. Formally, the cut-regions (Lokoc et al. [2012, 2014b]) are defined as:

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

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

For an example of minimal cut-region for a set $X = \{o_1, o_2, o_3, o_4, o_5\}$ see Figure 1.9 (b). Cut-regions were also used for defining new similarity queries in my diploma thesis Čech [2014].

The PM-Tree is composed of a hierarchy of cut-regions which form tighter and more compact regions compare to ball-regions. Similarity queries in the PM-Tree

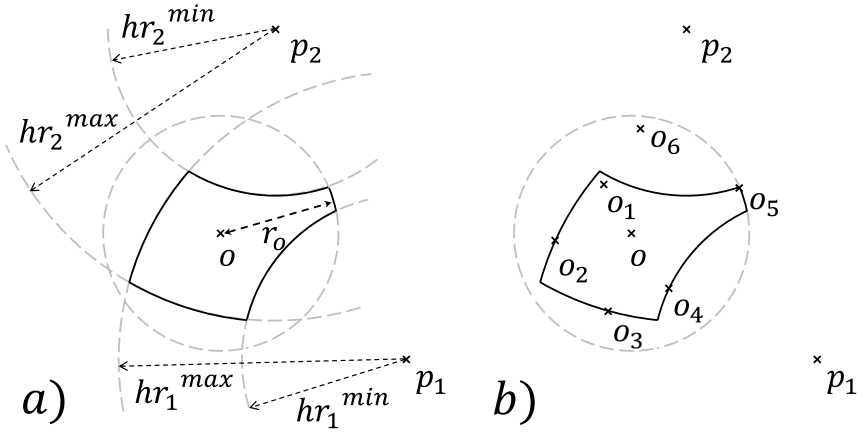


Figure 1.9: (a) Cut-region (CR) (b) Minimal CR for a set $X = \{o_1, o_2, o_3, o_4, o_5\}$. The image comes from the paper by Lokoc et al. [2014b].

are processed likewise queries in the M-Tree. The following lemma summarizes a query ball and a cut-region overlap.

Lemma 2 (Overlap of a cut-region and a query ball-region)**.** *Let $CR(o, r_o, hr)$ be a cut-region and $B_q(q, r_q)$ be a range query ball-region, if $\delta(o, q) > r_o + r_q$ or for some interval hr_i it holds $hr_i \cap \langle \delta(p_i, q) - r_q, \delta(p_i, q) + r_q \rangle = \emptyset$ then CR and B_q do not share any object.*

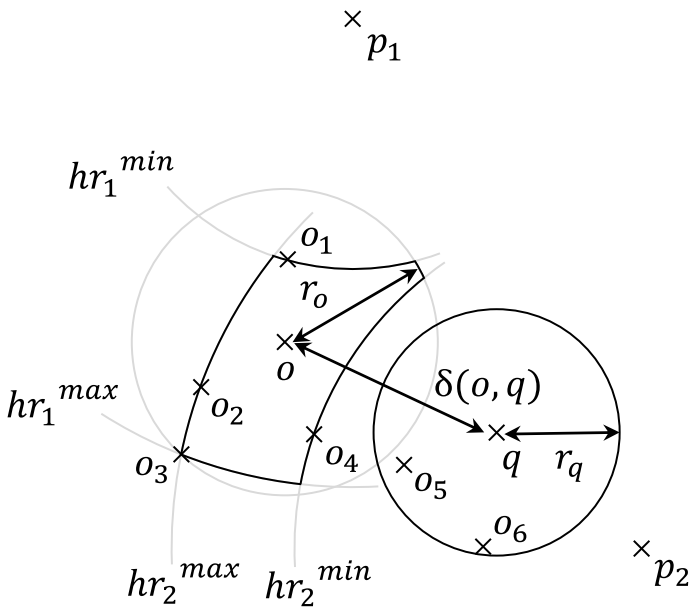


Figure 1.10: Overlap of a cut and ball-region. The radius of cut and ball-region overlap but structures don't share any objects because the cut-region is more compact thanks to the hr_i boundaries. The illustration comes from the paper by Lokoc et al. [2012].

Proof 2 (Overlap of a cut-region and a query ball-region)**.** *In case of $\delta(o, q) > r_o + r_q$, the proof is exactly the same as the proof of Lemma 1.*

Otherwise, if balls overlap, the intersection with pivot rings have to be tested. Because all objects in CR lie inside hr_i boundaries, the test for overlap of B_q with all hr_i is required. We know that $\exists i$, for which $hr_i \cap \langle \delta(p_i, q) - r_q, \delta(p_i, q) + r_q \rangle = \emptyset$ and, for contradiction, assume that $\exists o_m \in CR, \delta(o_m, q) \leq r_q$. We know that o_m must be in hr_i boundaries and there is no intersection with B_q and hr_i, hence, $\delta(o_m, q) > r_q$. □

On each level l of the PM-Tree, all cut-regions are tested for an overlap of the query ball-region using Lemma 2, and, like in the M-Tree, only not pruned cut-regions are further investigated in the lower level $l + 1$. The PM-Tree offers faster query evaluation thanks to more compact cut-regions. However, the tree has to store more information and dynamic operations on the PM-Tree are more complex (more details are described in the paper by Lokoc et al. [2014b]).

1.3.5 Metric index

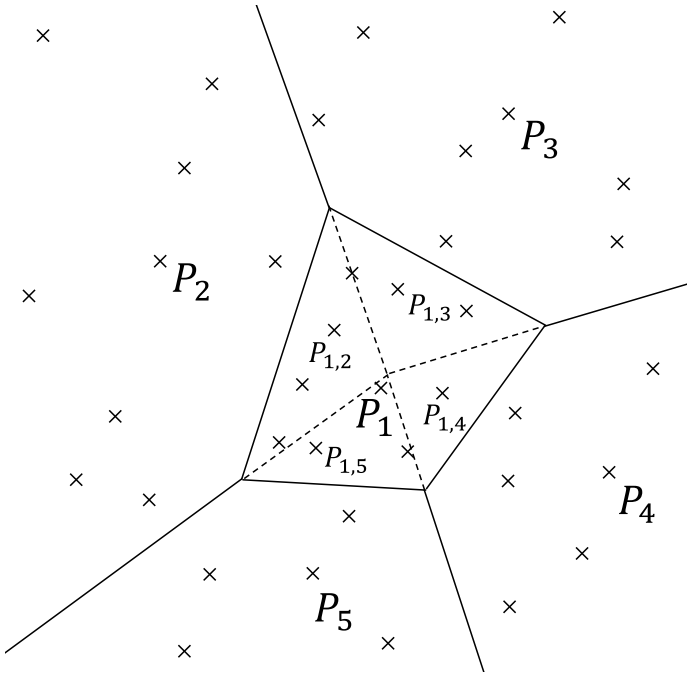


Figure 1.11: Example of the metric space partitioning in the M-Index. The region close to the pivot P_1 displays the Voronoi partitioning of objects on the second level. The image was originally published by Čech [2014].

Another metric access method combining more indexing principles is called the Metric index, or shortly M-Index (Novak et al. [2011]). Objects $o \in \mathcal{S}$ are distributed to Voronoi cells (or clusters) C_i according to distances to a set of pre-selected global pivots $P_i \in \mathbb{P} \subset \mathcal{S}$. This algorithm is called the Voronoi partitioning (Aurenhammer [1991]). Each cell C_i (with the corresponding pivot P_i) contains objects o which are closer to the pivot P_i than to other pivots $(\forall P_j \in \mathbb{P} - \{P_i\} : \delta(o, P_i) \leq \delta(o, P_j))$. Each cell C_i also stores a covering radius $r_i = \max_{o \in C_i} \delta(o, P_i)$. This partitioning principle is applied recursively (Figure 1.11, inner cell P_1) until cells on the lowest levels contain only predefined maximal number of objects or a cell reached the maximal depth. Eventually, cells $C_{i_1,\ldots,i_n}$

form a cluster hierarchy tree. Besides, objects localized in every lowest bucket C_{i_n} can be further organized in another index structure. Moreover, cut-regions were implemented in M-Index as well (Lokoc et al. [2014b]) increasing the compactness of cells C_i even more. The biggest advantages of the M-Index are relatively fast construction time and efficient query processing.

Briefly, a query defined by the ball $B_q(q, r)$ is evaluated in the M-Index by traversing the cluster tree and checking overlap of B_q with cells C_{i_l} on each level l employing all metric space filtering principles (Section 1.3.1). On the next level $l + 1$, only not pruned cells $C_{i_{l+1}}$ are considered for further evaluation. On the lowest level, objects are processed concerning leaf indexing structures (e.g., objects in leafs can store distances to all pivots) and distance measures are evaluated only for unfiltered query candidates. The persistent variant of M-index uses a mapping of objects to one-dimensional domain and a B-Tree structure for accessing candidates on disk (for more details see the work of Novak et al. [2011]).

The M-Index can be also efficiently implemented in a distributed environment (Novak et al. [2012]). The key idea is to share a cluster tree containing C_i across a worker cluster and store data in chunks including pre-computed object IDs locally on different nodes (hence, data searching is independent of data storage). To evaluate a query B_q, candidate cells C_{can} from a cluster tree are retrieved employing standard M-Tree querying principles and data from leaf cells C_{can} are retrieved by IDs from local nodes. On each node, data objects can be organized in a different index such as B+Tree to effectively return objects for a specific interval of IDs. We were inspired by these techniques in our research intending to develop scalable frameworks for evaluating distributed similarity joins. More details are presented in Chapter 5.

The curse of dimensionality

An important thing to mention in connection with metric access methods is the phenomenon called the *intrinsic dimensionality* (or the *curse of dimensionality*, Zezula et al. [2006], Chávez et al. [1999], Chávez et al. [2001]). Fundamentally, data in a database $\mathcal{S}$ can be effectively organized in a metric index if distances between objects in $\mathcal{S}$ are distinctive enough. However, in higher dimensions, which are frequently seen in vector data representations, distances between objects can be relatively similar. Frequencies of unique distance occurrences can be displayed in a distance histogram that describes a distribution of distance values in a simple plot. When more diverse distances are observed more frequently (Figure 1.12) the dataset is easier to index and metric access methods have better performance. On the contrary, when this histogram is narrow and many distance values are almost identical we say that the dataset $\mathcal{S}$ is hard to index (Figure 1.13).

The curse of dimensionality is also observable in the machine learning area. With the growing number of dimensions in key features, the number of training data has to be significantly larger to reliably distinguish objects with such features. On the other hand, more complex multimedia objects may not be described with enough expressiveness in lower-dimensional spaces.

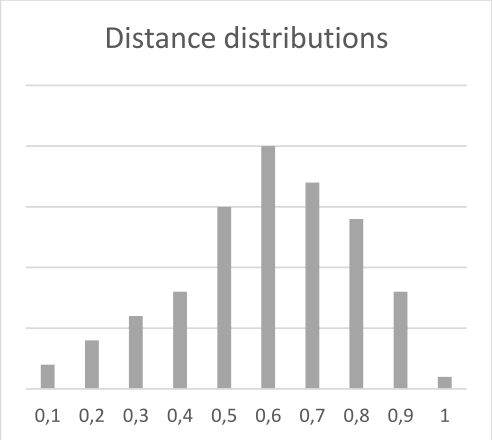


Figure 1.12: An example of a dataset with a wider distance distribution. An index built on such a dataset has better performance.

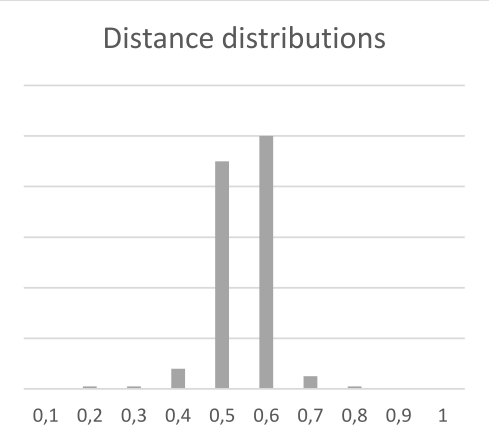


Figure 1.13: An example of a dataset with a narrower distance distribution. Indexing such a dataset can be challenging.

1.4 Distributed computing - MapReduce

Nowadays, centralized solutions for similarity search are becoming insufficient or even not viable since multimedia collections can count billions or even more objects. Therefore, the need for distributed data processing is emerging and is requested. In this thesis, we focus on the MapReduce (Dean and Ghemawat [2008]) paradigm that is often used for parallel processing of big datasets in the divide and conquer like approach. The algorithms described later in Chapter 5 are implemented in the Hadoop[2] and Spark[3] MapReduce environments which were designed to run not only on designated server clusters but also on commodity hardware (personal computers) connected in a network.

1.4.1 Hadoop

The Hadoop system is one of the pioneering projects build on the MapReduce paradigm. It consists of several components that provide full support for storing data, managing and organizing cluster resources, and it also exposes a code skeleton for developers to design their MapReduce applications.

Datasets are typically stored in the Hadoop distributed file system (HDFS), which is designed to form a big virtual file space to contain data in one place. From the user's perspective, the whole file space acts like one big storage. Physically, real data files are stored on different data nodes across the cluster and are replicated in multiple copies (protection against hardware failure or a data node disconnection). Name nodes manage access to data according to the distance from a request source to a data node (they find the closest data node to a request).

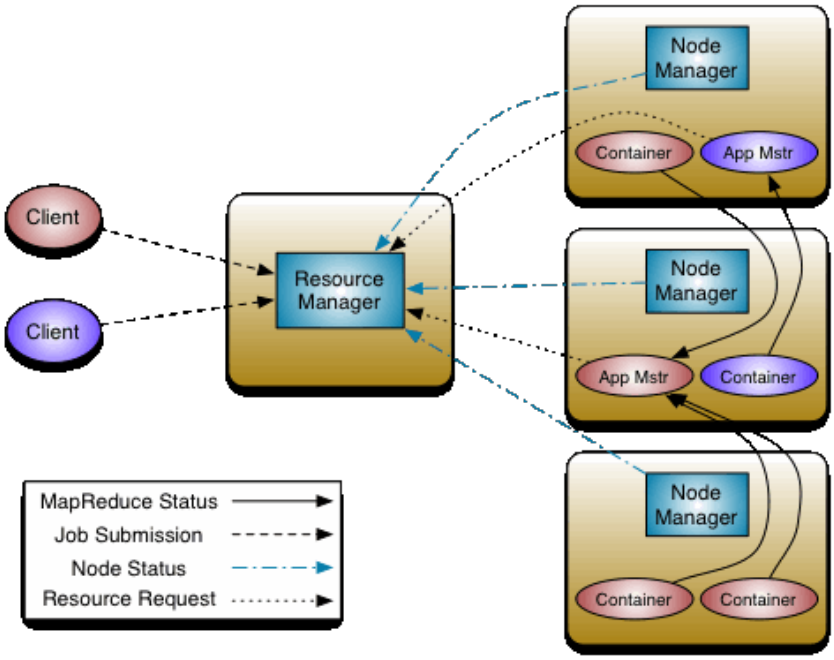
Another part of the Hadoop environment is YARN. Hadoop Yarn is a model for resource management and job scheduling and monitoring across the cluster. The main components are a global resource manager and application masters for running jobs. The resource manager has two main parts: the scheduler and the

[2]https://hadoop.apache.org
[3]https://spark.apache.org

applications master. Hadoop Yarn also defines node managers which serve as agents of each machine (worker) and report to the resource manager (scheduler). More information can be found on the official site referenced in the footnote.

In Hadoop, every program submitted to the cluster is composed of one or more MapReduce jobs. Each job has three main phases: a map phase, a shuffle phase, and a reduce phase. In the map phase, data are loaded from the HDFS file system, split into fractions, and sent to mappers where a fraction of data is parsed, transformed, and prepared for further processing. The output of the map phase is $(key, value)$ pairs. In the shuffle phase, all $(key, value)$ pairs are grouped and sorted by the key attribute and all values for a specific key are sent to a target reducer. Ideally, each reducer receives the same (or similar) number of groups to equally balance the workload of the job. In the reduce phase, all reducers process their assigned groups and usually perform the main execution part of the whole job. Finally, all computed results from the reduce phase are written back to the HDFS. When an application contains more MapReduce jobs, intermediate results are always written back to the file system. This is one of the biggest disadvantages of Hadoop because iterative algorithms need a lot of I/O operations for each iteration.

When a program is submitted to the system, the applications manager has to find a first application master which accepts the application. Then the application master negotiates resources from the scheduler to get enough computation power (worker containers from node managers) needed for the application execution. In case of failure or an application error, the application manager tries to restart and rerun the application. The system also provides monitoring statistics of running applications, consumed and available resources for each application and status of the cluster. The whole scheme is depicted in Figure 1.14.



Source: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

Figure 1.14: The Hadoop YARN architecture.

It is worth mentioning that for big clusters the name node and the global resource manager may be a bottleneck of the Hadoop system.

1.4.2 Spark

In contrary to Hadoop, Spark is an in-memory distributed processing engine utilizing the MapReduce design. Thanks to memory utilization, some iterative algorithms (e.g., machine learning models) run substantially faster on Spark compared to Hadoop. Spark uses resilient distributed datasets (RDDs) as its data storage foundation. An RDD is a read-only multiset that is distributed over a computer cluster and can be kept in memory or persisted to a file system if needed. RDDs can be loaded directly from the file system, from a database or can be generated directly in memory by an application.

Spark has several components that enable different types of data manipulation, for example, Spark Core (supports core data processing), Spark SQL (provides support for structured and semi-structured data and a domain-specific query language), and Spark Streaming (supports analytics of streaming data).

Spark Core is one of Spark's main data processing components. It defines a wide array of distributed data manipulation operations including transformations such as *map*, *filter*, *reduceByKey*, *groupByKey*, *join* and actions such as *reduce*, *collect* and *count*. Most of these operations take RDDs as input and produce RDDs as their output. All Spark transformation functions are evaluated lazily, meaning operations are executed in one optimized data stream after an action function is called. If an RDD is used multiple times, caching techniques can be employed. Spark supports multiple levels of caching, e.g., MEMORY_ONLY persistence (keep all objects or RDDs in main memory) or MEMORY_AND_DISK persistence (prefer main memory but if RDDs don't fit there the rest is saved to hard drives). Spark can run in different setups, for example on Hadoop or stand-alone. For our testing purposes, we used Spark running on Hadoop utilizing the Hadoop Yarn resource manager and HDFS services.

In the next Chapter 2, we focus on multimedia searching, browsing, and exploration in more detail. We study benefits, limitations, and challenges for methods in the area of research and showcase some demonstrations of multimedia retrieval frameworks presented in recent years.

2. Basics of multimedia browsing and exploration approaches

One of the first encounters with the concept of multimedia exploration can be found in the paper written by Santini and Jain [1998]. The paper tackles a scenario in which users cannot provide or formulate a satisfactory query example for a retrieval system. Another difficulty can appear in a case in which a user provides a query but a retrieval system returns results relevant for different concepts in the query. For example, the input text query "jaguar" is relevant to pictures of animals as well as cars. In both cases, the exploration concept is useful to refine and narrow the user's interests.

In the work by Smeulders et al. [2000] published at the beginning of this century is defined a categorization that characterizes different approaches to the exploration and exploitation of multimedia collections. From the formal point of view there is a difference between the terms *searching*, *browsing* (or *surfing*) and *exploration*. Users who are searching are typically just submitting well-defined queries to a system that outputs relevant results to each query. In the browsing scenario, users usually know what they want to find but are not able to provide or define a query example. On the contrary, an exploration process typically involves exploring a dataset and familiarizing with its content without a clear target objective. Hence, searching and browsing is usually categorized as a direct search while exploration is not.

This categorization is further augmented in the work by Datta et al. [2008]. The authors define three types of users.

Browser. This term defines users who work with a database without clear intent. Browsers typically switch between different topics and their work with a retrieval framework consist of many different queries possibly covering a whole database.

Surfer. Surfers are users who have a moderate idea of what they are looking for. Surfers usually start with less specific queries but then their focus gets narrower and more and more specific. Finally, queries just refine their interests while users are closing to their goals.

Searcher. A searcher is a user who has a clear search objective which can also be well formulated in a retrieval system. Searchers typically spend the least time with a system and they just provide (perfect) queries (without any further refinement) that, in general, give them information about what they are looking for.

Various types of users have different requirements and expectations from a multimedia retrieval framework. Whereas browsers (and surfers) appreciate a more complex user interface with richer options for query formulation, searchers welcome more detail results presentation with additional details about multimedia objects because it is expected that retrieved results are relevant.

Query initialization

To relate with the concept of a browser or a surfer user, the term *mental queries* is proposed in the paper written by Heesch [2008]. Basically, it characterizes users who are limited by a retrieval system and queries are formulated iteratively in their heads. Since a perfect query usually cannot be provided immediately for different reasons (e.g., a multimedia object is too complex) the author proposes a couple of alternatives for the query formulation. The first option is to browse a collection first and find some suitable query candidates (for example a browsing system by Sclaroff et al. [1997]). Another possibility is to draw or otherwise formulate sketches that can roughly characterize multimedia objects (Müller et al. [1999]). Nevertheless, sketches can have limited expressive power and may be hard to formulate for some users. One of the most promising alternatives is to (automatically) obtain some kind of annotations for multimedia objects (e.g., images annotations described by Zhang et al. [2006], Yavlinsky et al. [2005]). Assigning annotations to objects can be, however, an exhausting and challenging task. Nowadays, machine learning techniques can help with it, but they require training data that, typically, have to be created manually. Nonetheless, no approach is almighty and can fail for a certain domain or type of multimedia data.

However, one important thing to mention is that in cases when a perfect query example cannot be formulated it still can be possible to filter or prune some parts of a database. For example, if a user is looking for a picture of a building, it is safe to prune images of nature and beaches. Hence, query initialization can be used to shrink a set of objects that needs to be explored or browsed.

Since multimedia browsing and exploration is the main topic of this thesis, we would like to go into more detail of two subjects that are studied and proved to be challenging for this field of research (Beecks et al. [2013, 2011b], Heesch [2008]): structures for exploration and approaches for vivid and organized results presentation. Specifically, Section 2.1 is tackling exploration structures and Section 2.2 characterizes visualization techniques.

2.1 Exploration structures

Arguably, one of the most challenging parts of an exploration framework is designing exploration structures. Intuitively, exploration structures should form some kind of hierarchy of multimedia objects in which higher levels contain only selected representatives from layers bellow. The hierarchy can be useful for exploration purposes since browsing in an excessive detail is, usually, too confusing for users. Besides, to get a general notion of a multimedia database it is necessary to overview only selected representatives of a dataset and digging into specific details generally follow after users are more familiar with the dataset and they can specify their search interests.

Moreover, similar objects should be organized close to each other. However, this requirement can be hard to achieve since content-based similarity can have multiple semantic interpretations and can utilize composite models (more modalities). For such cases, multiple similarity models employing different key features have to be respected. This situation can lead to constructing multiple exploration structures that need to be maintained which results in an even more complex problem.

In the work by Heesch [2008] three types of exploration structures are defined.

Static hierarchical structures. The general idea of this type is to form a static hierarchy of clusters respecting a similarity model composed of key features and a similarity measure. Such structure is, usually, not updated frequently and retrieves objects only from clusters on the same level or one level up or down from the level in which a query object is located. The browsing of hierarchical structures is intuitive and easy. However, finding specific objects can be challenging because users have to navigate throughout the structure from the top to the lowest level following a correct path which may not be clear to users. Imagine a user wants to find pictures of beaches but on the top layer only images of people, buildings, computers, and trees are displayed. In that case, it is very difficult to decide whether a beach is more similar to a building or a tree, especially when a user is not familiar with the utilized similarity model. An example of this method can be the (P)M-Tree (Section 1.3.4), MLES (Section 3.1.2), or the ImageMap pyramid (Figure 2.5).

Static networks. Static networks are inspired by human brain activity. Similar objects should be linked together based on semantic relations. Retrieving objects is done by activating initial node(s) in a network and then spreading the information further until a sufficient number of objects is fetched. Typically, data are organized in a network based on a similarity model. Examples of static networks can be a nearest neighbors network (graph) (Figure 2.1), thresholded graphs or navigable hypercubes. The advantage of networks is that they provide less constrained navigation compared to hierarchical structures. On the other hand, in networks, it is more difficult to get a general overview of a dataset. Another disadvantage of networks is that they can be hardly indexed compared to hierarchies.

Dynamic structures. The previously mentioned approaches typically require one or more query objects in each exploration step. Nevertheless, in cases in which queries are hard to provide, other methods for browsing are presented in the paper by Heesch [2008]. The authors discuss ostensive browsing proposed by Campbell and van Rijsbergen [1996], Campbell [2000] in which a tree structure is dynamically unfolding during navigation by updating relevance feedback (users iteratively mark their interests). Another dynamic method describes how to project high-dimensional object descriptors to, for example, only 2D space and how to organize them in a map-like environment. This approach is typically applied to only a subset of database defined by retrieved results, hence it is not pre-computed. There are many known techniques for dimensionality reduction such as principal component analysis (PCA by Hotelling [1933], Pearson [1901]), multidimensional scaling or MDS (by Rubner et al. [1997], Young [2013]) that uses optimization solvers for 2D mapping or self-organizing maps (by Kohonen [1990] or newly proposed by Kratochvil et al. [2019]) for 2D embedding. Searching and browsing by sketches or annotations can be also considered as an exploration of dynamic structures.

Another dilemma connected with exploration structures is the initial view

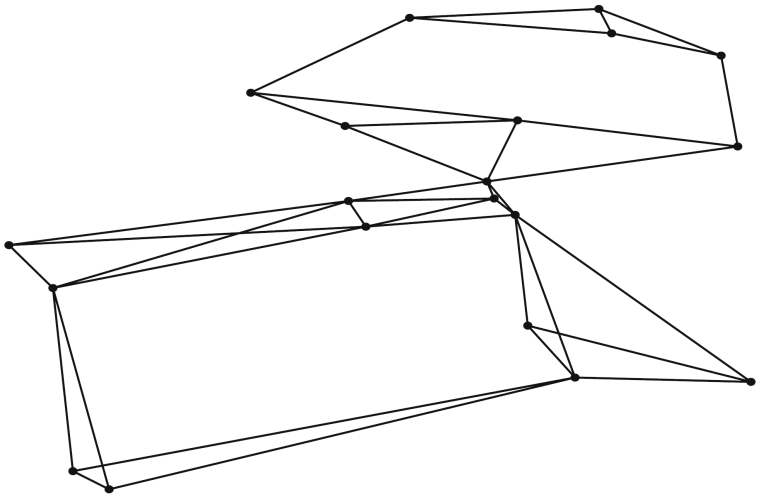


Figure 2.1: Example of a static network, specifically a nearest neighbors network (graph) by Heesch [2008].

problem. It deals with a situation at the beginning of an exploration process in which a user comes to a retrieval system and they have not provided any initial input yet. Nevertheless, the system needs to show some objects to a user so they can start browsing a collection. The problem can be easier for hierarchical structures because it is possible to display objects from the top level as the initial view. However, this solution might not be ideal as well since the top level can be too general and may not reflect the nature of a dataset. For other approaches, it can be harder to determine representatives to display. For example, self-organizing maps can show center nodes but for networks, it can be unclear how to determine representatives for the initial display (nevertheless, for example, randomly sampled objects can work decently well in some cases).

In our works by Cech and Grosup [2015], Grosup et al. [2015b], Lokoc et al. [2014a], Mosko et al. [2015a], we have also designed and proposed several exploration structures generally based on static hierarchical structures. More details are described in the following Chapter 3.

Scalability challenges

In the work by Beecks et al. [2011b] another arduous topic for multimedia exploration is discussed. Since an exploration framework should retrieve results almost immediately, it often has to deal with scalability issues. The bottleneck of each system can be fast query evaluation requirements due to huge data volumes counting millions or billions of objects each (typically) represented by high-dimensional vectors. The authors debate the need for an index responding almost in real-time that provides support for the evaluation process. However, because of the curse of dimensionality effect (see Section 1.3.5), constructing such index may be difficult or even not feasible.

There can be several solutions for solving this problem but none of them is universal or self-saving. One possibility is to apply some dimensionality reducing algorithm (as mentioned in dynamic structures) such as PCA by Hotelling [1933] or employ lower-dimensional embedding algorithms to make data easier to index. A different strategy is to introduce an approximate search that can speed up

query evaluation time substantially while keeping obtained results still relevant to users if an approximation error is small. Nonetheless, if the approximation is not controlled in any way it can produce poor results in some situations.

The need for research on scalable multimedia analytics (Worring et al. [2016]) is discussed in the paper by Jónsson et al. [2016] as well. The authors formalized several objectives that should be accomplished to achieve fully scalable multimedia analysis, visual analytics, and database management. Furthermore, Jónsson et al. [2016] established the following major axes of scalability required for multimedia analytics: volume, variety, velocity, and visual interaction.

Nowadays, trends are leaning towards distributed computing. Lately, single machine computational power is not increasing very much but parallelism can be easily achieved by connecting worker machines in a cluster or utilizing graphic cards that each contains thousands of cores. A query evaluating system built utilizing a distributed environment (e.g., MapReduce presented in Section 1.4) can satisfy scalability issues at the cost of more complex algorithms that have to work in parallel. Another complication connected with a distributed environment is data storage. Usually, it is not possible to easily access a whole database on each worker machine, therefore distributed systems have to reckon with the fact that only a fraction of data can be available for each worker. In our work, we have focused on distributed similarity joins that are presented in more detail in Chapter 5.

2.2 Visualization techniques

Another very important part of a multimedia exploration system is the way how results are presented to users. It can make a huge difference in overall system popularity and acceptance in the community. Visualization techniques can be directly connected with exploration structures (e.g., hierarchical exploration structures can directly correspond to presented results) or the retrieved results can be further rearranged to present multimedia objects to users in a more flexible manner. Therefore, there is sometimes a very thin line dividing which component of a retrieval system is part of an exploration structure and which is the member of a more complex and sophisticated visualization component.

In the paper written by Datta et al. [2008] there are five categories of results presentation (with the main focus on the image domain).

Relevance-ordered. This is one of the most popular methods of how to present results adopted by many successful companies such as Google or Yahoo. Results are just ordered by the relevance scoring to a query.

Time-ordered. Using this method, results are sorted in chronological ordering. Such an ordering can be found, for example, in the Google photos service[1].

Clustered. Clustering is frequently used for either better results organization (similar objects are presented on a display closer to each other whereas dissimilar ones are further from each other) or to further filter obtained multimedia objects (e.g., to prune almost identical objects).

[1] https://photos.google.com

Hierarchical. Similar to hierarchical structures, results can be presented also in tree order. Hierarchical results visualization can be especially useful in scenarios in which objects can be organized in a hierarchy that is easy to understand for users (e.g., text data with hypernyms and hyponyms).

Composite. The composite form combines more than one previously mentioned approaches. For example, hierarchical clustering with the relevance ordering can be a potent way of how to arrange multimedia objects for users.

Keep in mind that clear and easy to understand results visualization is one of the key elements of a multimedia retrieval and exploration framework. Having a great similarity model and fast and efficient indexing structure may not be sufficient if users are unable to find and notice relevant objects. Furthermore, a well-arranged visualization display can overcome shortcomings of similarity models.

In the next subsection, we demonstrate some specific visualization techniques that were used in recently published multimedia browsing and exploration systems.

2.2.1 Visualization examples

In this section, we present several layouts for result visualization that were successfully used for state-of-the-art retrieval systems. Even though the well-arranged visualization is crucial in many domains, we showcase applications working mainly with images because pictures are easy to perceive by humans. Moreover, for images, users can easily tell if ordering is understandable and if objects are correctly organized by the similarity between them.

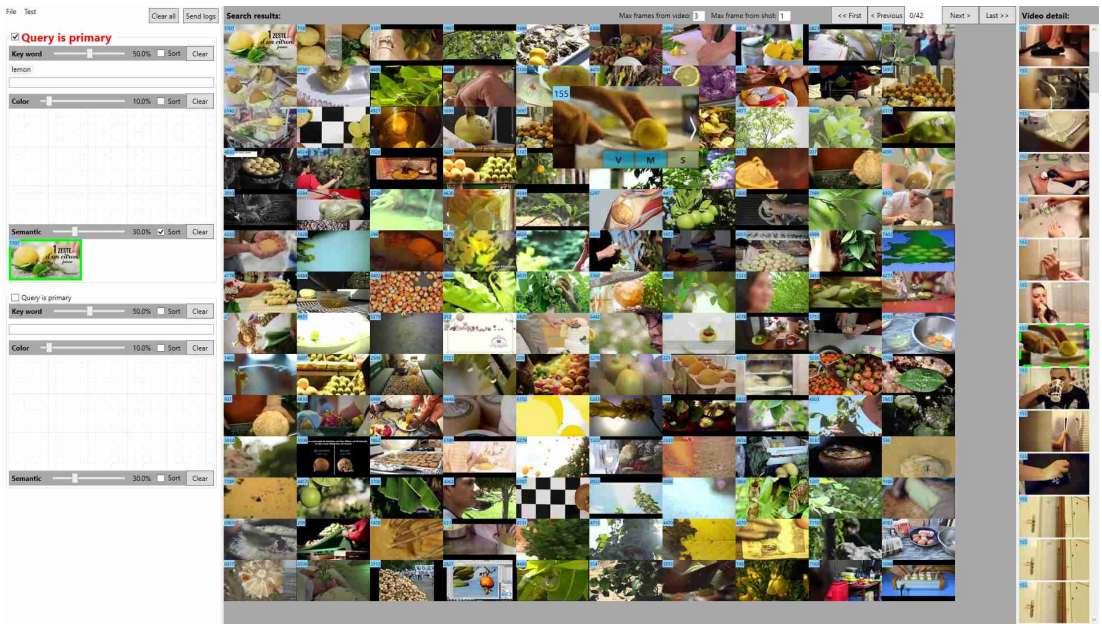


Figure 2.2: VIRET tool developed by Lokoc et al. [2019a].

The first example in Figure 2.2 shows basic results grid. It is a common approach for presenting results used in industry as well (e.g., the Google search engine). Results are typically ordered by relevance measure from the top left to

the bottom right corner. It is a great method for displaying a larger number of objects (many objects fit on a display). Nevertheless, users can miss (overlook) some relevant objects if the display is too large.

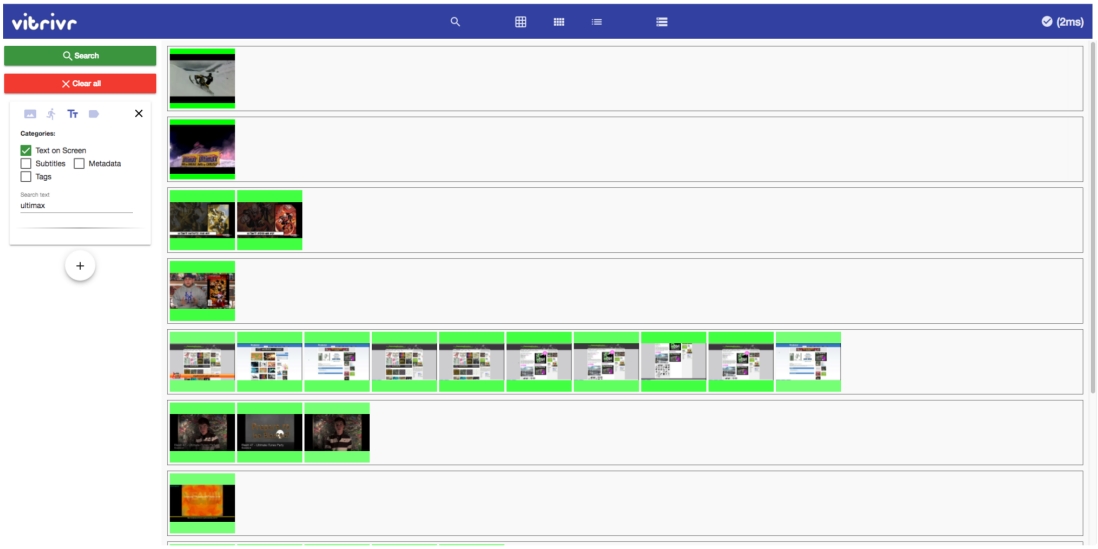


Figure 2.3: Vitrivr tool proposed by Rossetto et al. [2019a].

Figure 2.4: The hierarchical exploration system implemented by Grosup et al. [2015b].

The image and video retrieval tool called Vitrivr developed by Rossetto et al. [2019a] portrayed in Figure 2.3 demonstrates a more complex type of results grid. In the example, each line represents results for one video and pictures on each line are ordered in chronological order. Moreover, the color of background symbolizes how relevant an image is to a query (darker green indicates more relevant results while lighter green declares less relevant images). This visualization can be easier to read by users because they are not overwhelmed by many images and results are organized also by video content and time chronology. However, the result display is not utilized effectively and contains a lot of unused white space.

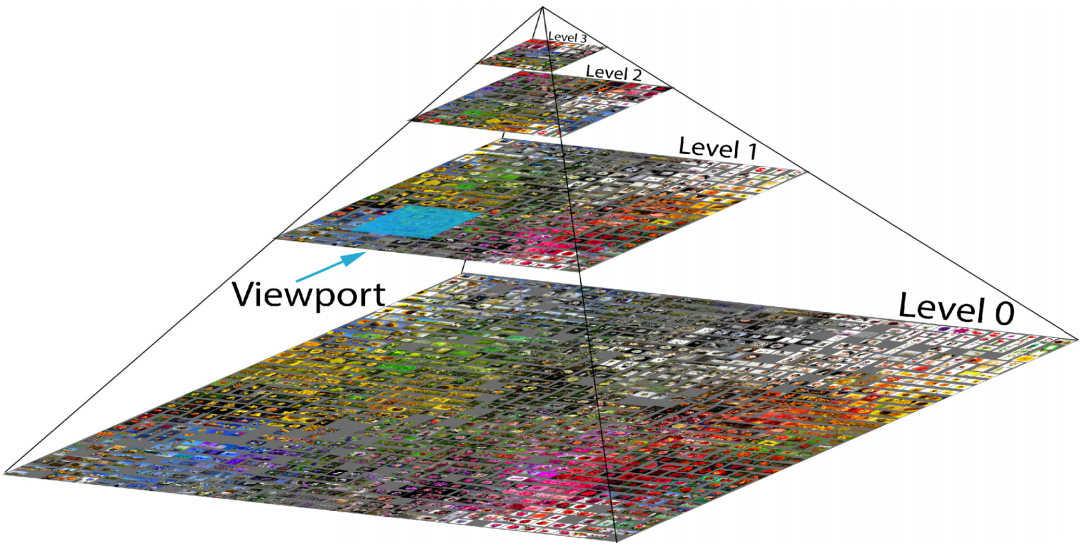


Figure 2.5: Example of a hierarchical structure called the ImageMap pyramid by Barthel et al. [2015b].

Figure 2.6: Display mode called the Fractal tree map proposed by Barthel et al. [2015a].

Another visualization approach is depicted in Figure 2.4. This system presents results on a canvas on which images are organized by an algorithm based on the behavior of particles in physics. This algorithm spreads pictures over canvas according to the similarity between images. In other words, similar images are attracted to each other while dissimilar ones are repulsed. Furthermore, only some result pictures are connected by edges based on their resemblance. The advantages of this method are that close objects (according to utilized similarity model) are

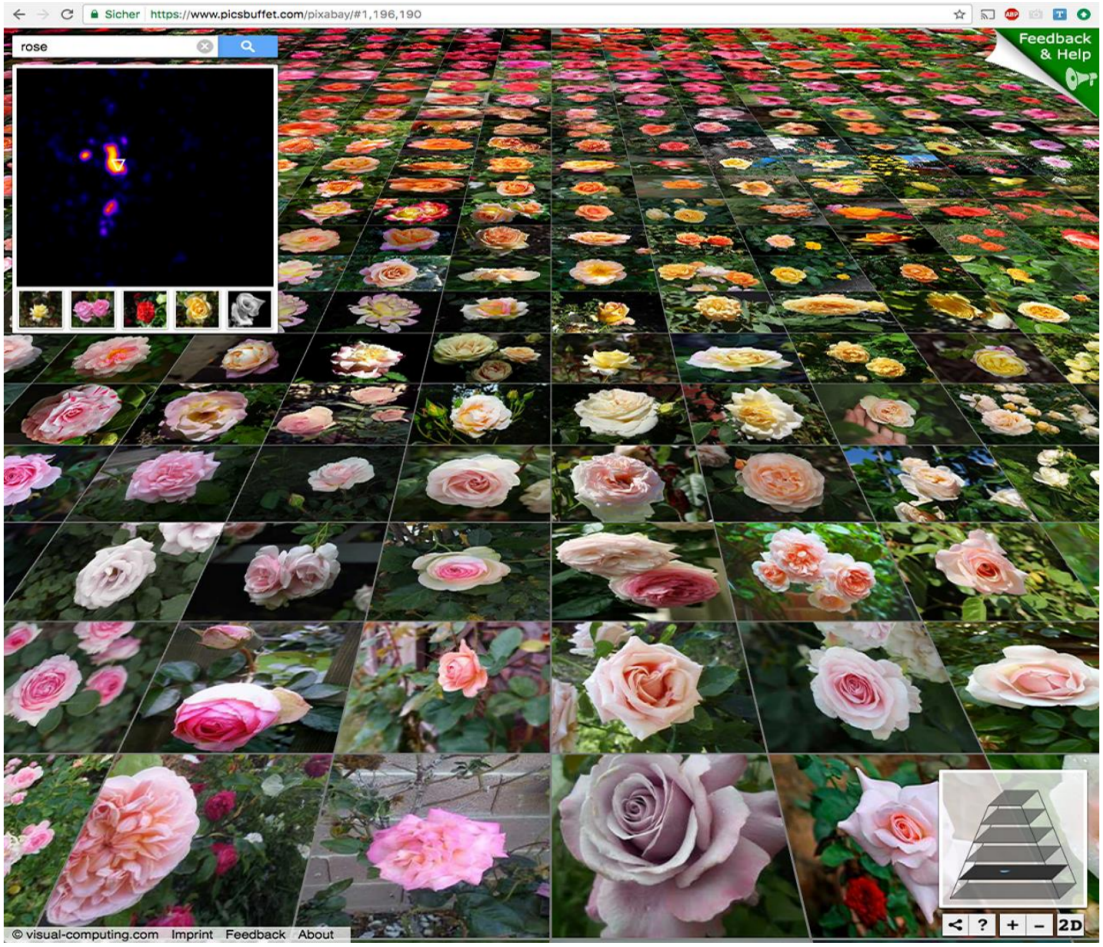


Figure 2.7: Demo application for navigating 2D image maps Picsbuffet by Barthel and Hezel [2019].

situated near to each other and results can be more easily comprehended by users. On the other hand, establishing such layout can be computationally intensive since distances between all retrieved objects have to be evaluated and then the layout is typically generated in many iterative steps. Besides, such the layout does not fully utilize display space and also contains white space.

In Figure 2.5, Barthel et al. [2015b] presented a hierarchical visualization structure called ImageMap. It organizes images in a pyramid by its visual and semantic features employing sorting and clustering techniques. The pyramid is built from the bottom to the top level. On each level, images are positioned by self-organizing maps and representatives for higher levels are selected utilizing principles of the Quad tree proposed by Finkel and Bentley [1974]. Browsing a dataset corresponds to the exploration of the tree structure. Retrieved results are composed of images in a different level of detail in the tree depending on the user's interests. The advantages and limitations of this approach have been discussed in the previous section. Shortly, this approach is good for getting an idea of what type of objects a dataset contains (due to properly selected representatives) but it may be difficult to locate specific objects because a path from the top to the lowest level may be unclear.

Furthermore, the authors of the ImageMap application also presented a graph structure called the Fractal tree map (Barthel et al. [2015a], Figure 2.5). Specifi-

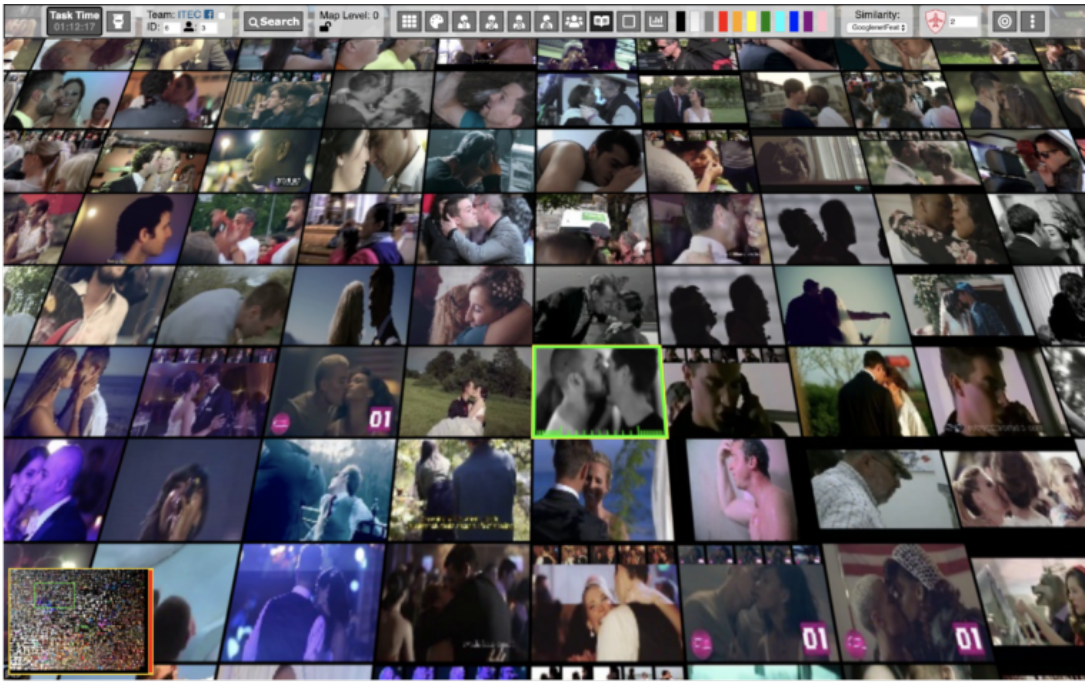


Figure 2.8: ITEC (diveXplore) application developed by Schoeffmann et al. [2019].

cally, they proposed a method that creates a network in which images are located in vertexes, and edges represent the similarity between them. Each vertex is connected to three successors and users can easily choose which path they want to follow. This structure enables quick navigation through pictures, nevertheless, it can be more challenging to find specific images if the collection is large. Besides, it may be difficult to notice relevant images since pictures in the top part of a screen are rather small.

In the following work by Barthel and Hezel [2019], the authors proposed a demo application that combines hierarchy and map browsing strategies (Figure 2.7). Images are organized in several layers and on each layer are sorted employing the self-sorting map (SSM). Representatives for higher levels are selected from an area of 2x2 pictures located on the lower layer. Users can navigate through the structure either directly by processing queries defined by a picture example or keywords or can pan, zoom in and out in the tree in a similar way to viewing a map like Google maps.

Another map-like visualization method displayed in Figure 2.8 showcases a browsing strategy of a large pre-computed 2D map on which images are organized according to a similarity model. The map allows users to navigate (pan) throughout the view and, eventually, jump to further parts faster via minimap available in the left bottom corner.

Both approaches employing a pre-computed map for results presentation can be easy to use by users and well-arranged for datasets with a lower number of objects. However, browsing larger databases can be challenging because absorbing that much information may be confusing for users. Or it may be slow to get an idea where relevant objects are located on the map. It may be advised to employ some pruning techniques to filter a result set first.

The following two examples illustrate visualization in 3D space. In Figure 2.9,

Figure 2.9: 3D cylinder proposed by Schoeffmann et al. [2011].

the authors study visualization on 3D cylinders and 3D globes in a map-like manner. The main motivation for these techniques is to display a larger number of images on one display. In addition, the authors discuss visualization methods for devices with a touch screen (e.g., tablets and smartphones) because controlling such devices require support for other inputs like swipes or touch gestures.

3D visualization canvas is studied in Figure 2.10 as well. The authors propose methods for generating 3D layout based on a particle physics model that can effectively organize up to thousands of multimedia objects (e.g., images). The efficiency is also discussed in the paper since determining such layout can be computationally demanding and any multimedia retrieval system should be as responsive as possible to be attractive to users.

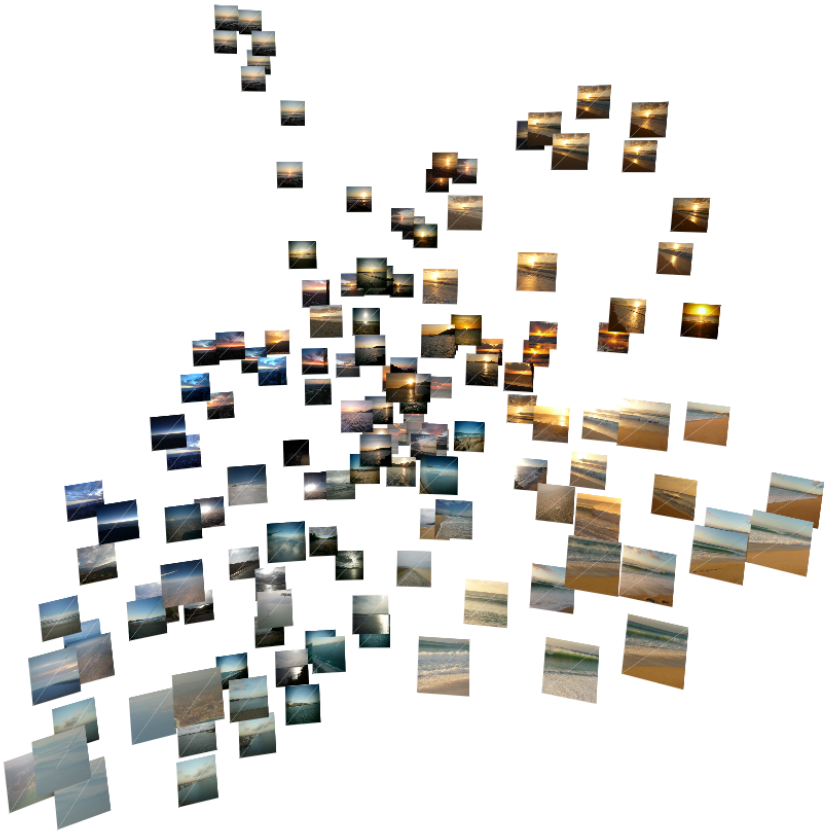


Figure 2.10: 3D image visualization introduced by Macik et al. [2016].

2.3 Evaluation strategies

It is also important to somehow measure the success rate and get feedback to researchers of newly developed algorithms and techniques for multimedia browsing and exploration. Although many experiments are performed in research papers, they vary significantly, and it can be problematic for readers to get an idea which methods are better under given circumstances because experimental methods are usually not very unified. Among other methods, integration tests of a whole multimedia retrieval system can be evaluated either manually or automatically.

Manual methods can consist of user studies (e.g., Hastings [1999]). User studies are composed of a set of tests that are conducted by users with or without prior knowledge of the retrieval system. Users can fill questionnaires or data are collected automatically based on their executed operations during the time they worked with the system. The advantages of user studies are a wide array of tested scenarios and also the ability of users to express their experience with the system in a much richer way. The main disadvantage of such experiments is that they are very difficult to reproduce and repeat.

Automatic experiments of a whole multimedia framework can be performed by, for example, simulations (e.g., Pietquin and Hastie [2013]). Simulations are replicating user's interactions. They are easy to reproduce but it is hard to properly

mimic actions of different user types (e.g., ideal user, expert user, novice user, a user with or without prior knowledge of the multimedia retrieval framework, etc.). Simulations are also important in the current era of machine learning applications to generate more training data applicable, for example, to reinforced learning techniques.

Moreover, it is difficult to define test databases that could researchers use for unified benchmarks because such datasets would require a lot of multimedia objects (small databases would not test the scalability of retrieval systems). On top of that, dealing with copyrights of many multimedia objects and sharing and processing huge datasets can be troublesome as well. For an idea, examples of some benchmark collections are ImageNet by Deng et al. [2009] or the V3C dataset by Rossetto et al. [2019b]. Other challenges regarding not only evaluation strategies are debated in the survey by Lew et al. [2006].

In the next Chapter 3, we present our proposed retrieval and exploration applications concerning image and video multimedia collections and discuss experimental evaluations performed automatically or with real users. We also present results from international competitions.

3. Interactive multimedia retrieval applications

In this chapter, we present our contributions in the image and video domain that is, mostly, focused on interactive research demonstration applications/prototypes and effective and efficient indexing and exploration structures. Searching in video collections is usually translated to a search over a collection of keyframes detected in video shots, so both domains are usually employing techniques for comparing images by visual or conceptual features. We started with desktop and web applications and our goal was to apply (and adjust/improve if necessary) models and algorithms from Chapter 1 and principles from Chapter 2 in various multimedia search and exploration scenarios. Of course, video search engines can take advantage of other information such as audio tracks or temporal context (time-ordered sequences of images) which are further discussed in Section 3.2.

3.1 Image retrieval systems

For validating scientific assumptions and theorems, we implemented some demonstration applications which conveniently and comfortably visualize algorithms for the multimedia exploration and similarity search. In such applications, we observed precision and benefits of different key features and latency and performance of indexing structures. The goal of our research team was also to design structures for effective multimedia exploration that fulfill needs for traversing through different layers of an objects hierarchy (zoom in/out in a multimedia collection), panning (moving) on one layer or performing fast queries or multi-queries in the system (an overview is described in the doctoral thesis by Moško [2016]). Last, we also studied different forms of presentation and visualization of retrieved results. Eventually, we developed an image exploration portal[1] that can work with multiple datasets, many descriptors $\mathcal{U}$ (SIFTs, Feature signatures, DCNN descriptors presented in Section 1.2.1) and various indexing structures (e.g., PM-Tree, M-Index, Pivot table, Multi-layer exploration structure).

3.1.1 Find-the-image

Our pioneering attempts started with the application called Find-the-image (FTI) by Lokoc et al. [2014a]. We have implemented several indexing structures (M-Tree, PM-Tree, M-Index) for exploration purposes and designed algorithms for browsing the Profimedia dataset (Budíková et al. [2011]). In this application, we have employed Feature signatures descriptors $\mathcal{U}$ for the image comparison. The goal of the tool is to find as many images from a selected image class (e.g., find all climbers) in just ten user actions. One user action is typically one (approximate) k-NN query from a selected image. This is, basically, a variation of the ad-hoc search task explained in the following Section 3.2 concerning video retrieval.

In the application interface, there are three main components. On the left side, users can see an example of the wanted image class. There is also the history panel

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

which allows users to go back and browse the dataset in a different direction. In the center-right part, there is a canvas for results visualization. For visualization purposes, we have utilized the physical particle model that pulls similar images close to each other and repulse different ones. Hence, all retrieved results are organized over the canvas by the similarity measure. In the bottom part, there users can observe a list of found pictures relevant to the searched class.

The main purpose of the FTI tool was to understand the strengths and weaknesses of different image similarity models and also the suitability of various metric access methods for exploration purposes. We measured the success of particular approaches in different aspects, such as recall, query evaluation time, and the number of exploration operations needed to find the first relevant image. Moreover, we performed even simpler user studies to determine which algorithms worked better under given circumstances. Results were published in works by Cech and Grosup [2015], Čech [2014].

Briefly, we concluded that our proposed PM-Tree browsing algorithms that used directly the index hierarchy outperformed other tested approaches in the distance computations free scenario. However, to retrieve most objects from the given class, it is better to use standard (approximate) k-NN queries in the M-Index. Nevertheless, for larger datasets, it might be hard to find a first representative object from the given image class using only k-NN queries if no representative is shown in the initial display. In such cases, it is more efficient to define multiple layers of detail over a dataset and perform queries only on the next layer so the exploration does not immediately dive too deep. This idea led us to design a structure for effective multimedia exploration presented in the following subsection.

3.1.2 Multilayer Exploration Structure

The Multilayer Exploration Structure (MLES) by Mosko et al. [2015a,b] provides support for multimedia exploration and querying a database from different points of view and different levels of detail. The key idea behind MLES is to define m layers L_i of a set $\mathcal{S} \subseteq \mathcal{D}$ which satisfy: $\forall i \in \{0, ..., m-1\} : L_i \subset L_{i+1}$ and

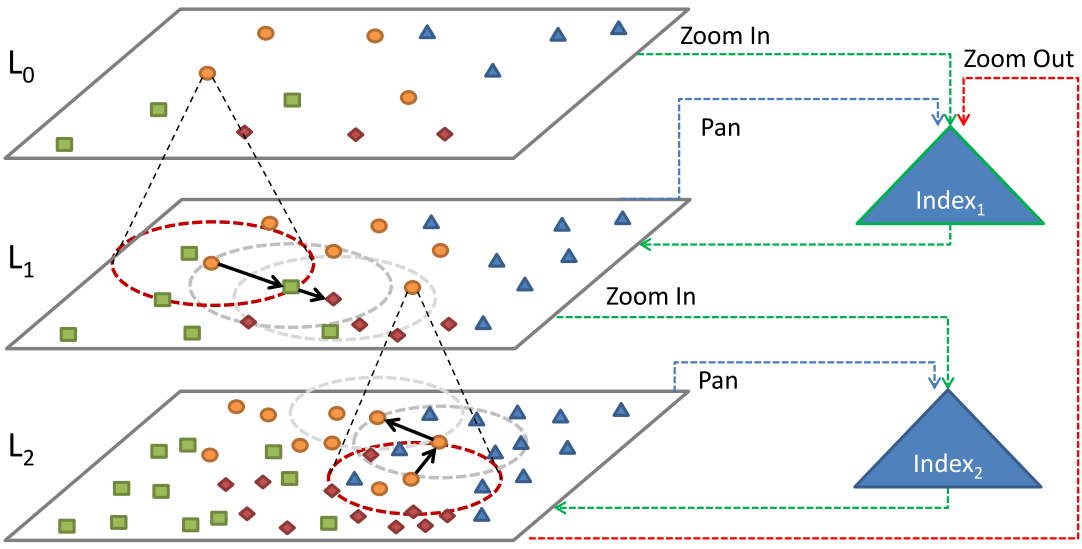


Figure 3.1: Structure of MLES by Mosko et al. [2015a] with described operations.

$L_m = \mathcal{S}$. The MLES schema is presented in Figure 3.1.

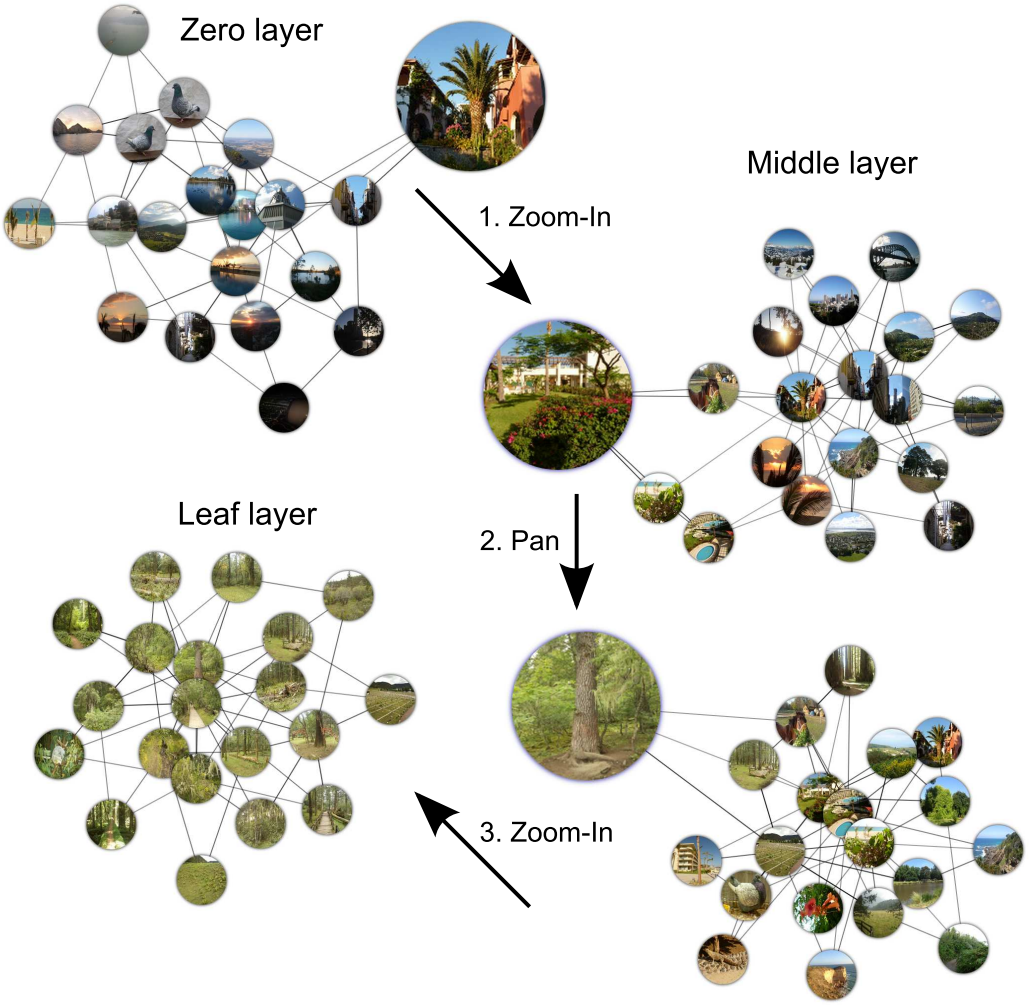


Figure 3.2: Example of MLES exploration operations (Mosko et al. [2015a], Moško [2016]).

MLES also defines more exploration operations: pan and zoom in and out. The panning queries allow us to move on the same level of MLES. Zooming enables us to go higher or deeper in the hierarchy and explore objects in a different level of detail. For quick querying responses, objects on each level are indexed in an independent metric access method such as the PM-Tree or M-Index.

The exploration process always starts from the zero level ($i = 0$ – the initial view). From that point, a user can pan, zoom in and zoom out throughout the collection and refine their exploration interests. For this application, we utilized a similar graphic interface to the Find-the-image system. An example is depicted in Figure 3.2. The thorough description can be found in the papers by Mosko et al. [2015a,b].

Furthermore, we evaluated experiments for MLES structures composed of 2 and 3 layers. We employed position-color-texture feature signature descriptors $\mathcal{U}$ of images compared by the SQFD measure δ. The study was performed by users without prior knowledge of our system. Their objective was to find as many images from a given class as possible in 15 exploration steps. No image from a

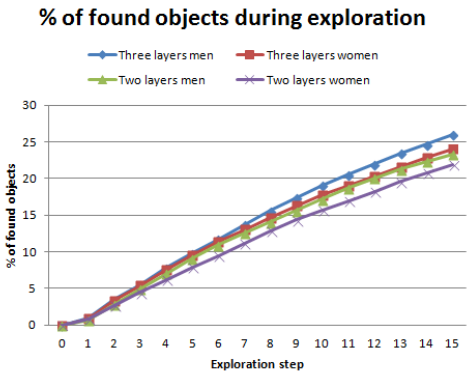


Figure 3.3: Percentage of found objects during exploration (Mosko et al. [2015b]).

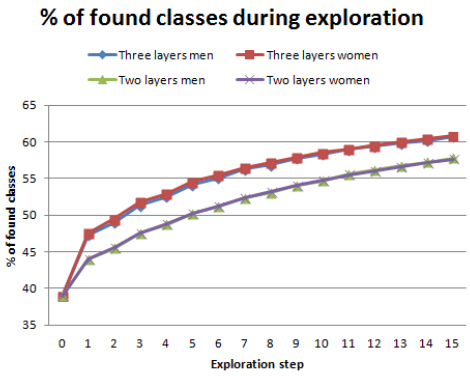


Figure 3.4: Percentage of found classes during exploration (Mosko et al. [2015b]).

searched class was presented in the initial view. We measured several statistics. Among others, in Figures 3.3 and 3.4, the performance of MLES for the percentage of found objects and classes is presented for both men and women participants. We concluded, that MLES containing 3 layers outperformed the other variant in terms of recall and success rate of finding the first relevant image from the desired class. More graphs and detailed discussions are presented in the paper by Mosko et al. [2015b].

3.2 VIRET framework

Searching in video data proved to be a challenging task especially in today's collections with HD videos counting a large number of broadcasts (Hu et al. [2011]). The video retrieval research tackles many difficulties. To mention some, we investigate the *known item search* (KIS), which studies the search for known (previously observed) shots or video scenes, while in *ad-hoc video search* (AVS) tasks users try to find as many images from a given topic as possible. KIS tasks can be further divided by the quality of knowledge of the known item. If a user has seen a whole video scene including a soundtrack the searching problem is typically easier compared to a case when an investigated scene is described, for example, only by a text paragraph. First, let's take a look at methods for processing, annotating and detecting keyframes and shot boundaries in videos based on their content.

3.2.1 Video pre-processing

If additional information is stored within a video collection they might help with the similarity search and exploration. Such information can contain for example timestamps (date, time) of video files or audio tracks. Time information can be used to filter only relevant data or to define queries concerning the time information. The audio track can be translated to text using automatic speech recognition (ASR). However, this information might not be completely reliable and ASR can fail for different languages and can be sensitive to the sound quality. Hence, the

video search is usually transferred to the search over an image collection based on the picture's content employing techniques mentioned in previous Section 1.2.1.

Video data are usually composed of sequences counting 25 frames per second (FPS). The first challenge of video retrieval systems is to automatically detect shots and key frames in videos. Searching over all frames from videos (25 FPS) would be very exhausting and also close frames usually carry almost the same information, hence, there would be too much redundancy. There are more approaches for keyframes and shots detection. Our team has designed a convolutional neural network for detecting basic shot transitions, which is inspired by the work of Gygli [2018]. The whole keyframe selection scheme is depicted in Figure 3.5. Details are described in our paper by Lokoč et al. [2019] accepted to the ACM Multimedia 2019 conference.

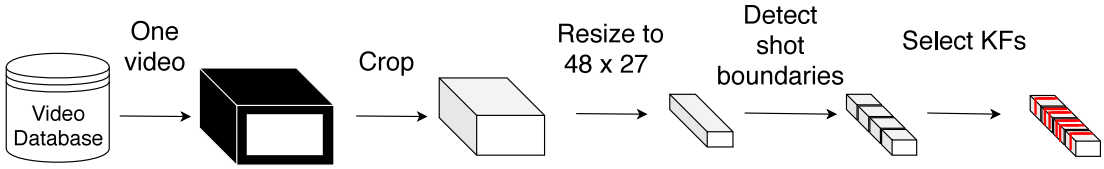


Figure 3.5: Keyframe selection scheme. The image comes from the work of Lokoč et al. [2019].

Once shot boundaries are detected using the neural network, keyframes are extracted utilizing a hierarchical clustering algorithm considering temporal ordering of frames.

Selected keyframes from a video collection are further processed to extract annotations and descriptors for retrieval purposes. Nowadays, machine learning techniques are frequently used to obtain DCNN descriptors $\mathcal{U}$, detect faces and texts and/or classify basic entities visible on the keyframes (e.g., person, car, beach, tree, dog). Model combinations help in situations in which single model filtering/pruning algorithms do not work sufficiently. Fusion strategies for aggregating results from different models are used as using multiple models simultaneously can improve retrieval results significantly (Atrey et al. [2010], Budíková et al. [2017]). In our work, we usually evaluate every model independently, then we fuse results and send the final sorted set to the visualization component.

3.2.2 VIRET tool description

Recently, a survey about video retrieval applications has been published by Schoeffmann et al. [2015]). The goal of each application is to present several similarity models, a friendly user interface and a well-arranged retrieved results visualization. Some tools also tackled possibilities to control the interface using touch gestures or discuss adjustments to online or mobile environments. In our research team, we have been systematically developing a video retrieval application called VIRET which we briefly describe in the following subsections.

Ranking models

In VIRET tool, we have implemented a bunch of similarity models for comparing different features and descriptors extracted from keyframes (Lokoč et al. [2019]).

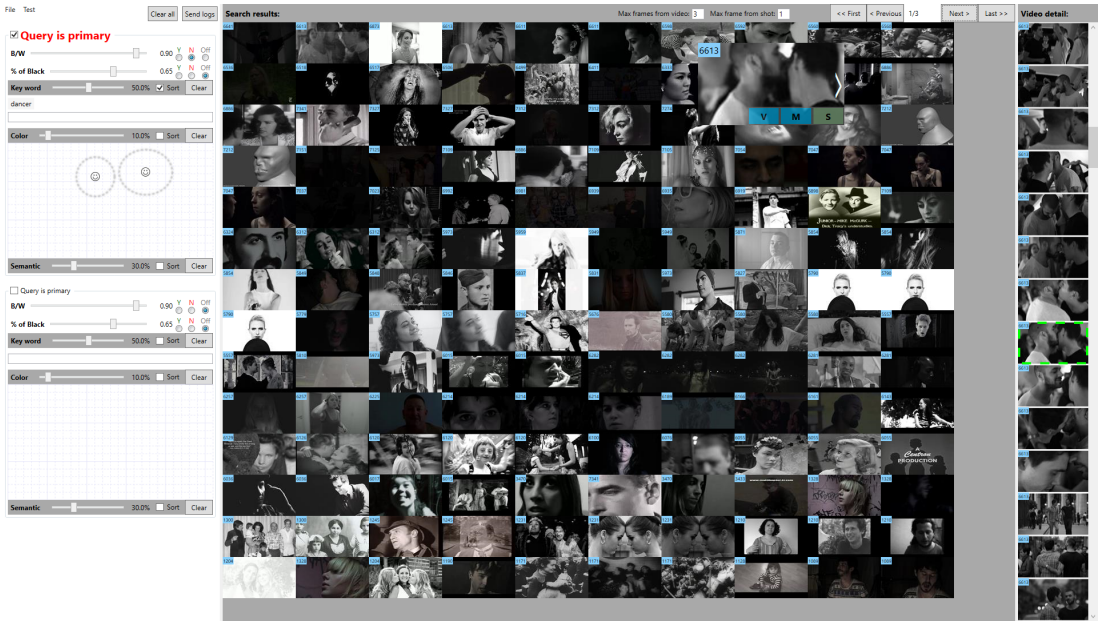


Figure 3.6: VIRET tool by Lokoc et al. [2019a] showing results for the query defined by black and white filters and two face sketches placed at the specific locations.

The application works with pre-processed data assuming the phases of shots and keyframes detection were already executed and, also, all key features for all models have been extracted. Specifically, VIRET takes advantage of four main similarity models. The first model compares the visual similarity of images based on the DCNN descriptors $x \in \mathbb{R}^n$ using the cosine distance δ. The second model employs image class labels (annotations) that were obtained by the classification procedure using a retrained neural network with 1243 image classes selected from the ImageNet database (Deng et al. [2009]). The third model uses color feature signatures representing color regions in images and the fourth model works with regions in which faces and texts were detected by deep neural networks (Hu and Ramanan [2017], Zhou et al. [2017b]). Each model accepts query inputs in a specific form and employs a similarity-based relevance scoring function $f : \mathcal{U}_Q \times \mathcal{U} \mapsto \mathbb{R}$ that assigns a score for a query $q \in \mathcal{U}_Q$ and each database object representation $x \in \mathcal{U}$. The scores are used to rank database objects and select top-ranked items for further processing. For more details (particular functions, temporal queries and multi-modal fusion) see the paper by Lokoč et al. [2019]

Prototype interface

On the left side of the VIRET tool, users can define queries for filtering and sorting all keyframes by given criteria. Every time, one model is marked as a sorting model and others are used for filtering. Filtering models always evaluate similarity of database frames to the query and return only the top $X\%$ of the frames where X is specified by the slider above each model. The intersection of filtered results is ordered by the similarity scores given by the sorting model. Final results are presented in the center panel of the VIRET tool. In Figure 3.6, you may observe results for a query specified by two face sketches and the keyword

Figure 3.7: VIRET tool by Lokoc et al. [2019a] in action. A results display for the query specified by keywords "church" and "window" with the temporal context defining the "church" is preceding a scene with the "window".

"dancer" combined with black and white filters. Visualized results are sorted by the keyword.

In VIRET tool, the keyword search model receives entered supported labels, optionally combined to more complex queries. Color sketches and face/text symbols are placed on a canvas. Each sketch element is defined by its center and a surrounding ellipse with an indicator whether the color or face/text symbol should be located in *all* space defined by the ellipse (full ellipse) or in just *any* area within the ellipse (dotted gray ellipse). Searching by the semantics (DCNN descriptors) is defined by examples selected from the result set. More examples can be selected at once. Moreover, we have also implemented the semantic search described by an external image. It is possible to drag and drop an image from a web browser found by the Google images service into our application and the image is used as a query for the semantic search. Our engine extracts the DCNN descriptors in real-time utilizing the neural network. Hence, database frames can be easily compared with the image from Google employing the same representation.

Moreover, all input fields on the left side in VIRET are duplicated. The purpose of this interface is a possibility to define temporal queries. Every query can be composed of two parts. An example is shown in Figure 3.7, in which result frames are displayed for the temporal query composed of the primary query defined by the "church" keyword followed by the latter query characterized by the keyword "window". The length of a temporal window is a parameter of the retrieval engine and is usually set to lower numbers (e.g., 10) of selected frames.

In the result view on the right side, users can inspect not only retrieved key video frames but also video context. Using mouse wheel scrolling actions on a selected frame, users can "play" temporal context from the video corresponding to the frame. Also, on the most right side, all selected frames from the same video are shown. Another possibility is to click the "V" button on a frame and the

whole set of selected frames for the video pops up in a new window.

Furthermore, the most relevant 2000 frames to the actual query can be dynamically reorganized into an image map. This action is performed by clicking on the "M" button on any retrieved frame. A new window shows up and the top 2000 frames are reorganized into a 2D layout concerning their similarity based on the semantic DCNN descriptors. This ordering algorithm is inspired by ImageMaps proposed by Barthel et al. [2015b], Barthel and Hezel [2019].

Results in interactive video search competitions

The state-of-the-art video retrieval applications are tested every year during the Video browser showdown[2] (VBS) competition (Lokoc et al. [2018], Lokoč et al. [2019]). In VBS, teams deal with KIS tasks, described by both visual scenes including the sound footage and text sentences, and also AVS tasks. The goal of each KIS task is to find a portrayed scene in the correct video as soon as possible. A team gets points for the correct submission time. Teams are penalized for wrong submissions. In 2019, the test collection for the competition counted 1000 hours

[2]http://www.videobrowsershowdown.org

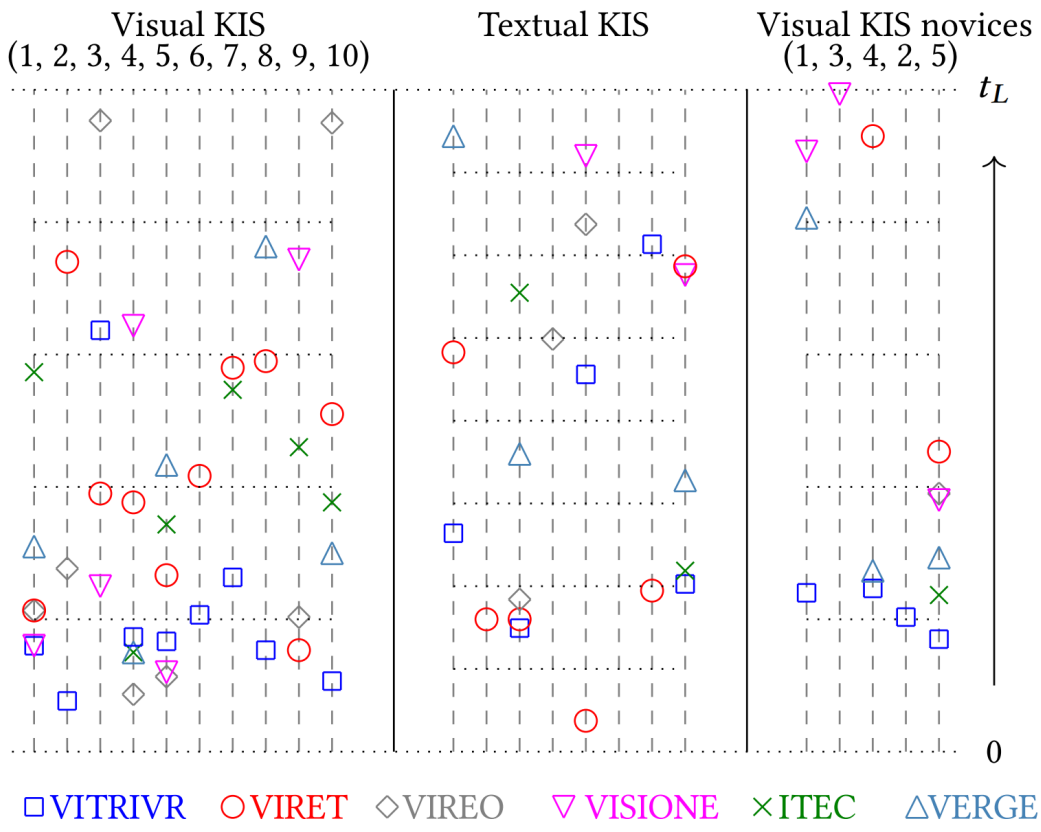


Figure 3.8: KIS results for the VBS 2019 competition by Lokoč et al. [2019]. There were all together 23 KIS tasks (one task is symbolized by one vertical line) and each team is represented by one symbol and color. For each task the lower position of symbols means faster correct submission time (the most bottom horizontal line represents starting time 0). If a team symbol is missing for a vertical line it means that the task was not successfully solved by that team.

TID	$rank_1$	t_1	t_1^s		$rank_2$	t_2	t_2^s
V_1^n	44	96s			3	92s	
V_3^n	(vid) 0	299s			1804	53s	
V_4^n	1	243s	279s		2259	3s	
V_2^n	336	70s			46	107s	
V_5^n	42	120s	136s		13	10s	
V_1	(vid) 5	28s			62	16s	64s
V_2	168	215s			33	165s	222s
V_3	(vid) 125	24s			(vid) 1	106s	117s
V_4	25	95s	113s		434	6s	
V_5	2	58s	80s		377	24s	
V_6	160	110s	125s		471	119s	
V_7	237	20s			37	20s	174s
V_8	135	167s	177s		1798	118s	
V_9	51	27s	46s		84	23s	
V_{10}	88	110s			84	141s	153s
T_1	49	35s			49	19s	290s
T_2	5	14s	96s		5	6s	
T_3	35	22s			18	10s	96s
T_4	(vid) 93	18s			7800	417s	
T_5	(vid) 78	12s			8	11s	23s
T_6	38	249s			2334	267s	
T_7	44	22s	117s		139	34s	
T_8	(vid) 8	252s			69	19s	352s
	PC1, 140 frames / page				PC2, 88 frames / page		

Table 3.1: Table showing the success rate of the VIRET tool for all KIS tasks at VBS 2019. It displays ranks of the first top occurrences of frames (or video) from searched scenes, times of the occurrences, and submission time for both PCs. The interesting observation is that several times the search scene appeared on the first display relatively soon but the task was solved substantially later (e.g., tasks V_7 or T_1) or was not solved at all (task T_6). Detail description can be found in the paper by Lokoč et al. [2019].

of video data collected from 7500 videos. The data size was almost 2 terabytes in the raw original format. In AVS tasks, the objective is to find as many scenes or shots satisfying described situations (for example find scenes where people are standing on a surf or find video shots of men jumping on a bike) as possible. It is important to submit only the correct scenes (the correctness is decided by judges). Points are obtained for correct submissions and also for total recall of submitted scenes. For the wrong submissions, a team is penalized. The time limit is usually set to 8 minutes for the KIS tasks and 5 minutes for the AVS tasks.

The effectiveness of all applications is tested not only by expert users (typically researchers who developed the tools) but also by novice users. Novices are selected from the crowd and have some time to familiarize themselves with a team and their application. However, once novice sessions start, experts cannot interfere with novices. This test case analyzes user-friendliness and comfort of presented applications.

Now, let's take a look at the results from the VBS competition in 2019. Figure

3.8 showcases KIS results of all teams for all tasks including novice sessions. In 2019, six teams participated at VBS. The vitrivr team (Rossetto et al. [2019a]) from Switzerland, VIRET team (Lokoc et al. [2019a]) from the Czech Republic, VIREO team (Nguyen et al. [2019]) from China and France, VISIONE team (Amato et al. [2019]) from Italy, ITEC team from Austria (Schoeffmann et al. [2019]) and VERGE team (Andreadis et al. [2019]) from Greece. In 2019, the overall winner was the vitrivr team who convincingly won the novice session followed closely by the VIRET team who closely won the expert session.

Regarding KIS tasks, during the expert session, most tasks were solved by the VIRET tool (Figure 3.8, more details in Table 3.1). 16 out of 18 KIS tasks were found in 1000 hours of video which is not an easy feat. However, many visual KIS tasks were solved by the vitrivr application faster. This observation could be explained also by the fact that the vitrivr team utilized the automatic speech recognition that helped in scenarios in which distinguished English speech was played in the searched video scene. Moreover, they supported OCR methods as well to search for displayed text.

In the next subsection, more detailed log analyses of our VIRET tool results from the video competition are presented.

Log analysis

During the VSB 2019 competition, we recorded (both locally and for VBS server logs) most of the actions performed by both expert and novice users. From these logs, we analyzed the effectiveness of used retrieval models, and utilization of various approaches.

Following graphs 3.9, 3.10, 3.11, 3.12 show detailed interaction analysis for the KIS tasks. Graphs are divided into two parts: graphs for the visual and textual KIS tasks. Each line is described by a combination of an action A and a user U in the format: A_U. We define five actions A: B - browsing, C - color model change, F - face model change, I - semantic image change, K - keyword model change. At the VBS competition, each team could use up to two computers for solving tasks

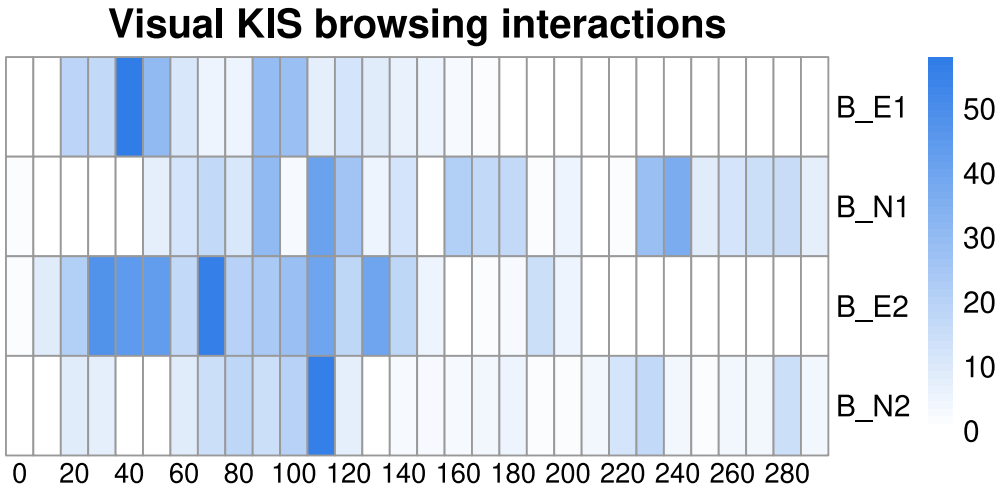


Figure 3.9: Browsing interactions for the expert and novice visual tasks during VSB 2019.

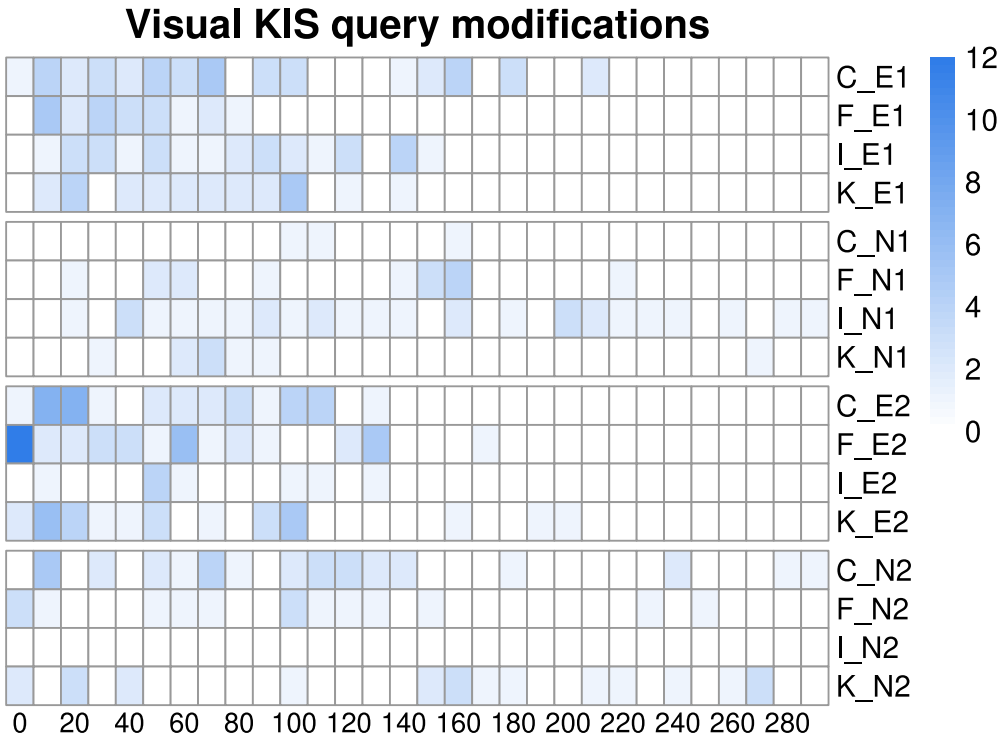


Figure 3.10: Heatmap representing query changes (Color, Face, Image, Keyword model changes) for the expert and novice visual KIS tasks during VBS 2019 by Lokoc et al. [2019b].

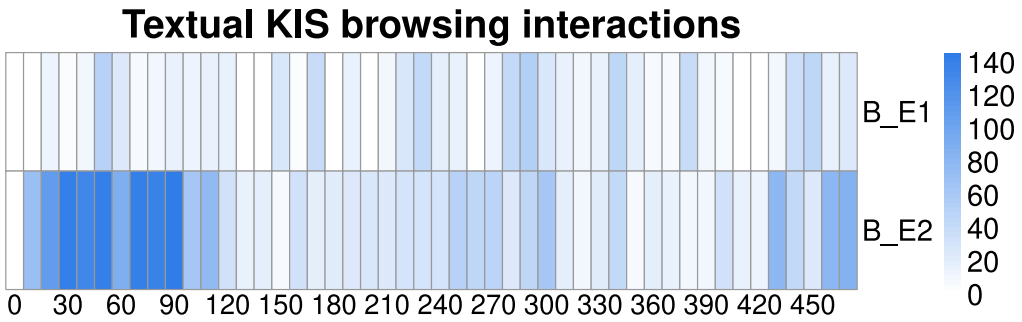


Figure 3.11: Browsing interactions for the expert textual tasks during VSB 2019.

in the video collection. Furthermore, all tasks were solved by experts and novices. Thus, the user set U consists of four entities: E1 - expert 1, E2 - expert 2, N1 - novice 1, N2 - novice 2. However, novices didn't solve textual KIS tasks and are not included in figures 3.11, 3.12.

You may notice that various users took advantage of our tool differently. For example, in Figure 3.12, the novice 1 almost did not utilize the color model while the novice 2 did not use semantic (image) model at all. In general, users tend to use only one thing that works for them and sometimes forget about other models. This behavior is more noticeable for novices because they have a relatively short time to get familiar with a video search application and it is hard for them to get a firm grasp on all features. In figures 3.9, 3.11 is depicted that browsing actions are distributed over all task time span. It means browsing is an important part of our video retrieval application and just formulating queries may not be sufficient

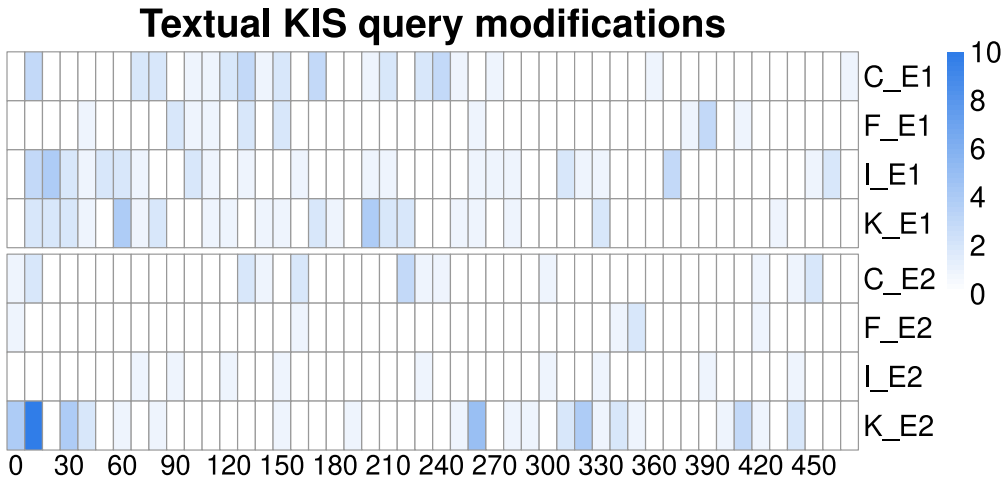
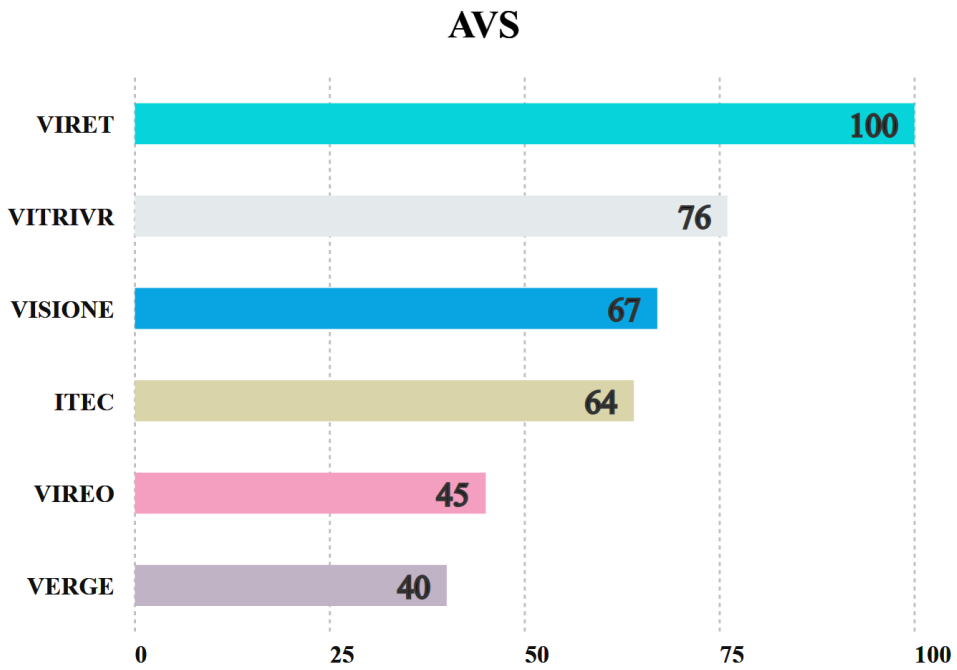


Figure 3.12: Heatmap representing query changes (Color, Face, Image, Keyword model changes) for the expert textual KIS tasks during VSB 2019 by Lokoc et al. [2019b].

to solve the search task. Nonetheless, the browsing results are only indicative due to a couple of network issues during the competition resulting in potentially incomplete server logs used for the analysis.



Source: https://videobrowsershowdown.org

Figure 3.13: The overall score of the AVS tasks at VSB 2019 during the expert session. I thank organizers of VSB 2019 for the graph.

AVS tasks were solved by our tool relatively effectively (Figure 3.13). A great benefit was the addition of the external image processing service using samples from Google images which allowed us to retrieve results for the given scenario or topic. This complemented automatic annotation used by the second member of our team. Hence, the overall recall of our search was high with respect to other

teams.

Our team also participated in the Lifelog Search Challenge (LSC) workshop during the ICMR 2019 conference (Lokoc et al. [2019c]). This competition ended up with similar results to the VSB. Team vitrivr won the competition followed by the VIRET team. VIRET closely won the expert session but lost points in the novice session. At LSC 2019, there were only KIS tasks to solve.

Discussion

Even though our demo application proved to work decently well during two competitions VBS and LSC (especially during the expert sessions), there is still room for improvement. For the novice session part, we observed that our results visualization grid is, perhaps, a little bit confusing. From our "behind seat" observations and later log analysis, we realized that novices formulated queries well, however, they sometimes did not notice retrieved correct frames.

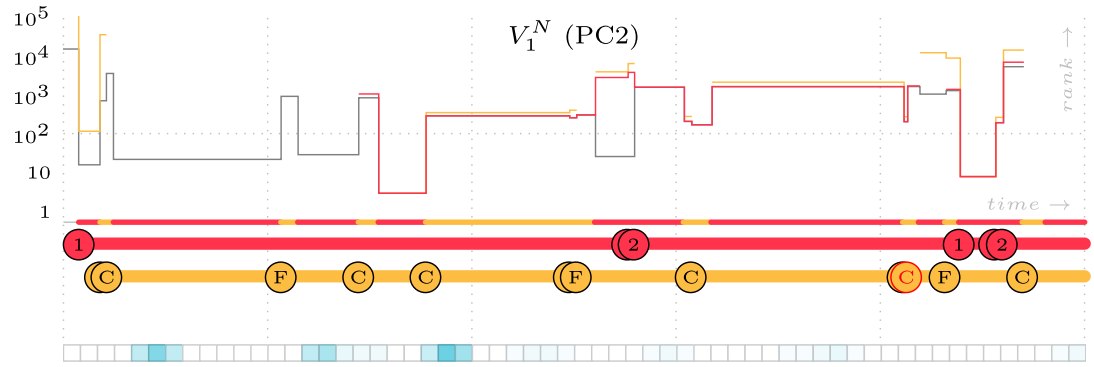


Figure 3.14: Figure displaying query changes (C - color/F - faces/KW - red numbers determining the number of active words) and position of the searched known scene in time (red line in the top). The graph was published in the work of Lokoč et al. [2019].

This observation is noticeable in Figure 3.14. A novice user formulated queries using keyword and sketch (color, face) models and the searched scene (red line in the top part of the Figure 3.14) appeared on the first display (even in the top 10 frames) few times. Nevertheless, the task wasn't successfully solved because the user did not notice the searched frames. More experimental results are presented in our paper Lokoč et al. [2019].

Besides that, from our experience, all similarity models worked adequately well. We usually started the search by typing keyword annotations or sketch that narrowed the number of frames. After that, we browsed retrieved results and refined queries. Moreover, retrieval based on temporal context also helped considerably since more frequent concepts in the dataset, that would be harder to narrow when targeting just one frame, can be further specified by additional queries targeting preceding or subsequent shots.

For the future, we would like to focus on the implementation of other similarity models such as speech recognition, scanning of text content (OCR), and local subimage search. Another challenge is to present obtained results in a more clear and organized way without restricting global results overview too much. For

instance, we would like to implement a results grid presenting retrieved frames for each video on one line. Each line would represent relevant frames to a query sorted in the time order. Moreover, each image would also show how much the particular frame is relevant to the query formulated on the left side.

In the following Chapter 4, we present similarity models and content-based retrieval algorithms and frameworks for network data that are more difficult to visually perceive by humans.

4. Models for network security domain

Besides the image and video domain, we also studied models for processing HTTPS network data with the main focus put on a network traffic classification and exploration of large databases of HTTPS requests. We worked together with researchers from the Cisco research department who provided expert domain knowledge and real use cases and applications. They mainly dedicated their research to modeling network traffic and detecting malicious communications employing different machine learning classifiers while our findings helped substantially with large scale log data processing, k nearest neighbors (k-NN) classification, and multimedia exploration that can serve as a user-assisted tool for detecting and confirming suspicious communications. In this chapter, we report primarily our contributions in this area of research.

The Hyper Text Transfer Protocol (HTTP) [Fielding et al., 1999] is one of the most popular protocols on the Internet. Over the years it has been adopted to many applications and is widely used for transferring not only web browsing context but also for exchanging large data volumes for services such as Dropbox or Skype. However, broad HTTP protocol usages attracted many malicious programs that can hide in common traffic and are hardly filtered and blocked in normal communication. Moreover, current standards force encryption and security to the HTTP protocol (shortly HTTPS). In the HTTPS traffic, malicious activities are even more easily camouflaged and hardly detected.

Our goal was to achieve the best possible precision and recall for the detection of malicious communications among the HTTPS requests. Likely, the hardest part is modeling HTTPS traffic because information about secured requests is limited.

4.1 Modeling HTTPS requests

Originally, we tried to model network data as a graph database where vertices represent servers and clients and edges are communications between them. The idea was to describe such a database as a histogram of entities called the graphlets (Przulj [2007]) that were used for biologic data. A graphlet is a simple subgraph of a typical size of 2 to 6 vertexes (nodes) with a predefined number of edges. A graph descriptor is formed as a histogram of occurrences of each defined graphlet in the whole graph. However, we realized soon that an HTTP(S) communications graph is relatively simple and is mostly only bipartite (clients usually communicate only with servers and servers only with clients). Thus, we abandoned the graphlet approach and created descriptors directly from collected network requests.

The number of properties or attributes publicly obtainable from secured HTTPS traffic is relatively small. In this work, we see a message as an individual request that establishes an encrypted TLS tunnel. Requests are typically collected by web proxies and are stored in proxy log files. The following four attributes can be obtained from each message (request) (Kohout et al. [2018]):

1. **bytes sent** m_{up} from the user's machine to the target server,

2. **bytes received** m_{down} by the client's machine from the server,

3. **duration**: m_{dur} of the tunnel, i.e., the length of the time interval for which the tunnel was active,

4. **inter-arrival time** m_{ia} (in seconds) elapsed between two consecutive requests for establishing a tunnel from the same user and the same server.

A message m is then represented as a quadruplet (a point in $\mathbb{R}^4$)

$$m = \{\log(1 + m_{\text{up}}), \log(1 + m_{\text{down}}), \log(1 + m_{\text{dur}}), \log(1 + m_{\text{ia}})\}.$$

In this paper, we present two types of feature descriptors presented in papers by Kohout and Pevny [2015a,b] and Kohout et al. [2018]. Both descriptors are modeled for an entity called a communication snapshot. Communication snapshots represent a set of HTTPS web requests (quadruplets m) for one client or for a fixed pair (server, client) for a given time frame (usually 5 minutes but it can be a whole month as well). Messages for a snapshot are aggregated into a set M from which a final descriptor is formed. The key idea behind modeling whole snapshots is that single requests may not be expressive enough to detect malicious activities, hence, we tried to monitor longer traffic (e.g., 5 minutes).

The first approach uses messages m directly and forms long sparse joint histograms from quadruplets contained in a set of messages M. For each communication snapshot, we extract one descriptor. The final descriptor is composed of a vector of real numbers of the fixed dimension 11^4 which is formed from an equidistant lattice $L = \{0, ..., 11\}^4$. Each message $m \in M$ contributes to several bins in the lattice to smoothen data noise. The final vector is produced by joining lattice indexes and normalizing all values. Each descriptor is saved in a space-saving format since many bin values equal to zero. This algorithm does not require any upfront training or parameters setup and is purely unsupervised.

The second approach utilizes Gaussian mixture model (shortly GMM) probabilistic approach (Marin et al. [2005], Duda et al. [2001]). This approach is inspired by the histogram bag of words method (Amores [2013]). The model is trained on a subset of sampled messages to create and position a predefined number of Gaussians. Then all requests $m \in M$ are mapped into space concerning Gaussians. Specifically, a descriptor has the length equal to the number of Gaussians and each message m from the set of messages M contributes to the descriptor's bin i according to a posterior probability that the message m belongs to the Gaussian i.

We have implemented a distributed MapReduce-based (described in Section 1.4) descriptor extraction algorithm that is visualized in Figure 4.1. The main purpose of this algorithm is to process a vast amount of data and obtain descriptors for the huge number of web requests generated every day. The program implemented in Hadoop operates in two phases: the map and reduce phase. In the map phase, data are loaded, processed and (key, value) pairs are emitted for each HTTPS request. The key attribute of each request (one input line) should correspond to a designated communication snapshot (e.g., one client or (server, client) pair and a time stamp). In the reduce phase, requests are grouped by the key attribute and the final descriptor is formed for a whole snapshot employing either the joint-histogram or GMM approach for modeling descriptors.

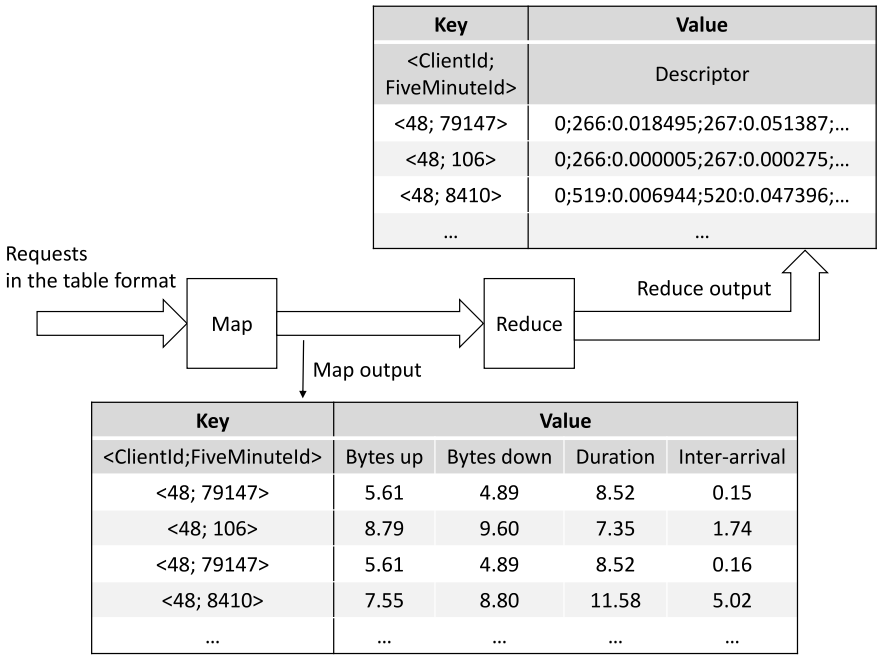


Figure 4.1: Illustration of the distributed MapReduce-based feature extraction algorithm presented by Cech et al. [2016]. In the map phase data are parsed and loaded and (key, value) pairs are formed where keys correspond to each communication snapshot. In the reduce phase, all messages for one snapshot are aggregated to form one descriptor using the joint-histogram or GMM algorithm.

4.2 Intrusion detection and data exploration

Our next focus was to detect malicious communications in HTTPS traffic using extracted feature descriptors in the form of vectors $\mathbb{R}^n$. We concentrated mainly on two tasks: classification and exploration of network data. The goal of a classification task is to design a function $\mathcal{C} : \mathbb{R}^n \mapsto [0, 1]$ that assigns a score between $[0, 1]$ to each communication snapshot. The score closing to 0 means that the snapshot is not infected (benign) while the score 1 represents malicious communication. Classification tasks usually require some kind of supervised learning approaches such as decision (regression) trees, neural networks or k nearest neighbors classifiers. On the other hand, the exploration of network data modeled by descriptors introduced previously can be computed by unsupervised techniques such as clustering or k-NN graphs and can prepare data for domain experts who decide whether some communication snapshots are infected or not.

For the classification task, we initially constructed a centralized (approximate) k-NN classifier that detects suspicious snapshots based on the nearest neighbors in a vector space modeled by extracted descriptors (Lokoc et al. [2016]). The main idea of the k-NN algorithm is to cluster objects in vector space using the Voronoi partitioning algorithm (inspired by the M-Index described in Section 1.3.5) and then to search for neighbors only in the closest partitions to each query. During the search, the metric space filtering and pruning principles (described in Section 1.3) are also employed to reduce the number of distance computations. The approximation is involved by setting up an early stop rule defining how many closest partitions are visited for query evaluation. Basically, this parameter

reflects how many database objects are considered for the k-NN result. Hence, the approximate search returns relevant results with high probability while the query evaluation time is shortened significantly.

Nevertheless, as we presented later in the work of Kohout et al. [2018], random forests and neural networks displayed better classification results for this task. Even so, a k-NN classifier can be used without a training phase required in traditional machine learning algorithms and is suitable for unsupervised tasks.

Since the need for processing and classifying large batches of communication snapshots emerged, we proposed distributed (approximate) k-NN algorithms for the MapReduce (described in Section 1.4) platform. We experimented with several approached based on a space-filling curve, local sensitive hashing or metric access methods principles. Specifically, the most promising method appeared to be the approximate pivot-based k-NN similarity join (Cech et al. [2016, 2017, 2020]) which yield high approximation precision and fast execution time. Distributed k-NN similarity join algorithms are described in full detail in the following Chapter 5.

Moreover, k-NN joins can be also used for data clustering or network data exploration. The exploration is especially useful in scenarios where an expert user searches for the most similar communication snapshots to selected queries and obtains very similar communications that can be potentially malicious. Thus, the similarity search can serve as a powerful tool for recommending servers or clients that need to be further investigated.

4.3 Experimental results and applications

In this section, we present several selected experimental results from the works of Lokoc et al. [2016, 2017], Kohout et al. [2018]. We show the effectiveness of models for communication snapshots classification and also introduce opportunities for the similarity search and multimedia exploration in this domain.

Besides traditional metrics for measuring quality of methods like precision and recall (Section 1.2.4) we focused also on another metric called FP-50 that is specific for the classification task. Formally, the value of the FP-50 error is defined as (Pevný and Ker [2015]):

$$FP50 = \frac{1}{|\mathcal{I}^-|} \sum_{i \in \mathcal{I}^-} \mathrm{I}\left[\mathcal{C}(x_i) > \mathrm{median}\{\mathcal{C}(x_j) | j \in \mathcal{I}^+\}\right], \tag{4.1}$$

where $\mathcal{I}^-$ are indexes of the negative (benign) training samples, $\mathcal{I}^+$ are indexes of the infected training samples (malware), $C(x)$ is the output of the classifier (classification score) for the sample x and I is an indicator function. The motivation is to have as low false alarm rate as possible. In other words, the FP-50 error indicates how many benign communication snapshots are present among the top half of infected snapshots according to the classifier score.

First, we compared our centralized k-NN solution with the ECM linear classifier proposed by Pevný and Ker [2015]. Experiments were measured on data collected by Cisco's cloud web security solution[1]. Our experimental dataset contains data from one day collected from many companies worldwide (altogether 145 822 799 connections to 475 605 unique servers with the total transferred data volume

[1] http://www.cisco.com/c/en/us/products/security/cloud-web-security/index.html

equal to 10 082 GB). Descriptors were formed using the joint-histogram algorithm and data were separated into 6 folds (each fold contains communication for 4 hour time period). For training purposes, communications were labeled by the `virustotal.com` service. Specifically, some messages m also contained SHA hashes of the binaries that started the HTTPS connection. If the hashes existed they were run by the service and if more than 10 anti-viruses claimed they were infected, the whole communication snapshot containing such the message m was labeled as infected.

		time of	
classifier	FP-50	training	classification
ECM	13.23%	56mins	0.3s
exact 4-NN no index	2.015%	0s	80mins
exact 4-NN	2.015%	44s	17mins
exact 4-NN with idf	2.247%	44s	23mins
approx. 4-NN (4% DB)	2.017%	44s	3mins

Table 4.1: FP-50 error and average training and classification times for data with six-fold validation presented by Lokoc et al. [2016]. Results are displayed for the 4-NN classifier that demonstrated the best FP-50 outcomes.

The Table 4.1 summarizes FP-50 error results for the centralized solution approach. We may observe that the k-NN classifier outperforms the ECM linear classifier by a significant margin. Moreover, total training and classification time of the k-NN classifier is shorter, especially for the approximate variant.

In the journal paper by Kohout et al. [2018], we have further investigated the classification experimental evaluations on bigger datasets and with more advanced machine learning approaches. We compared four classifiers consisting of neural networks, random forests, ECM linear, and XGBoost classifiers for different representations of communication snapshots (employing both variants of GMM and joint-histogram algorithms). The classification results are displayed in Figure 4.2. The experiments proved that the concept of modeling aggregated network data into communication snapshots is more efficient for classification purposes since the precision is significantly higher. Specifically, the random forest and XGBoost classifiers outperformed other methods and achieved the best accuracy. More details are presented in the previously mentioned work by Kohout et al. [2018].

4.3.1 Similarity search and exploration in network traffic

As discussed earlier, the similarity search can be used for unsupervised data organization and pre-processing. In this subsection, we describe how similarity and precision of modeled snapshot descriptors can be measured, and we showcase a demo application working with HTTPS data.

The precision of the k-NN similarity search for growing k and different data (joint-histograms and two GMM methods – Gaussians determined by either supervised and unsupervised method) is displayed in Figure 4.3. A set of queries for the k-NN search was composed only of malicious snapshots. The point of this figure is to show whether infected snapshots are close to each other in the space

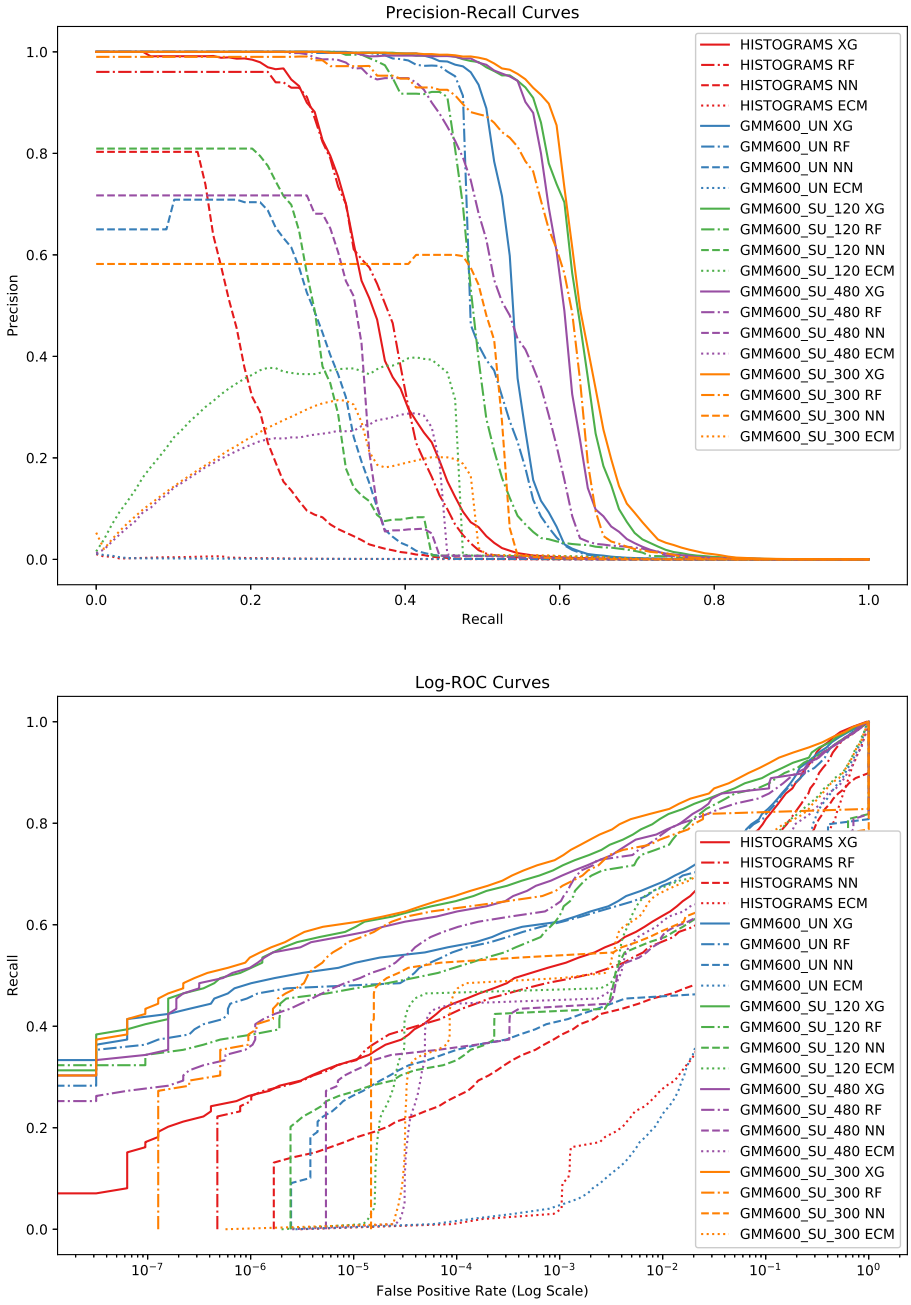


Figure 4.2: Precision, recall, and receiver operating characteristic (ROC) curves for all tested types of descriptors and classifiers. I thank Tomáš Komárek (Kohout et al. [2018]) for these images.

modeled by certain descriptors (histograms, GMM). This experiment was measured on a bigger dataset described in more detail in the paper by Kohout et al. [2018].

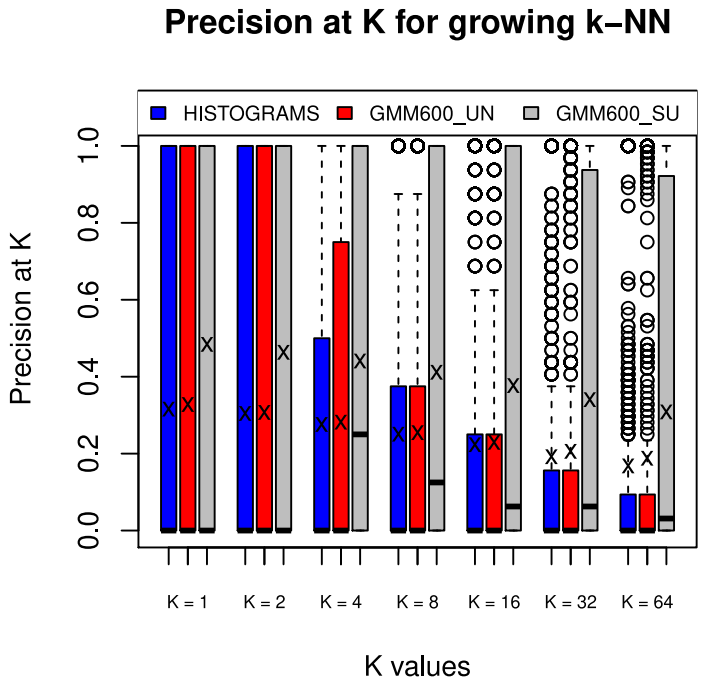


Figure 4.3: Precision for growing k for different descriptors (joint-histograms, supervised and unsupervised GMMs). The mean value is described by the X mark and median by the horizontal dash in each column (Kohout et al. [2018]).

You may observe that supervised GMM presents the best precision. However, this observation was expected because the training phase for the GMM600_SU data treated benign and malicious snapshots separately. Nonetheless, in this experiment, we focus mainly on unsupervised methods and you may observe that GMM600_UN data present slightly better precision compared to joint-histograms.

We have also implemented a visualization and support tool[2] for intrusion detection experts that can help them organize suspicious communication snapshots formed for clients and inspect potentially malicious clients or servers (Lokoc et al. [2017]). The application shows infected nodes (either servers or clients) and organizes them on a display utilizing the similarity model based on distances between sparse joint-histograms (see Figure 4.4). More similar nodes are attracted to each other while the dissimilar ones are repulsed. Users can inspect details of each node (see Figure 4.5), switch to different views as well as dynamically change parameters of the retrieval model (e.g., set weights of particular dimensions in descriptors). Moreover, the tool can be also used for exploration purposes – users can select nodes of their interest as a query and then perform the search for the most similar ones. Retrieved results can be further investigated by the domain experts who can assess their potential threat.

The next Chapter 5 describes details of algorithms for distributed computations. From yet presented algorithms and content-based retrieval systems is evident that

[2]http://herkules.ms.mff.cuni.cz/NetworkData

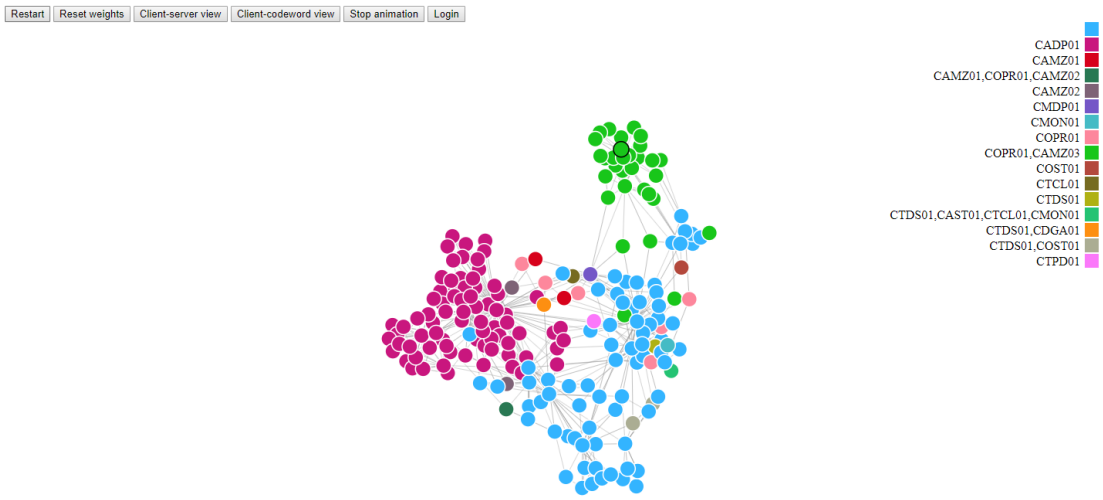


Figure 4.4: Initial display of the network data exploration tool by Lokoc et al. [2017]. Each node represents a client that can be potentially infected by one or more malware (each malware has a different color). Clients with high similarity according to the employed similarity model are organized in tighter clusters and are connected by edges.

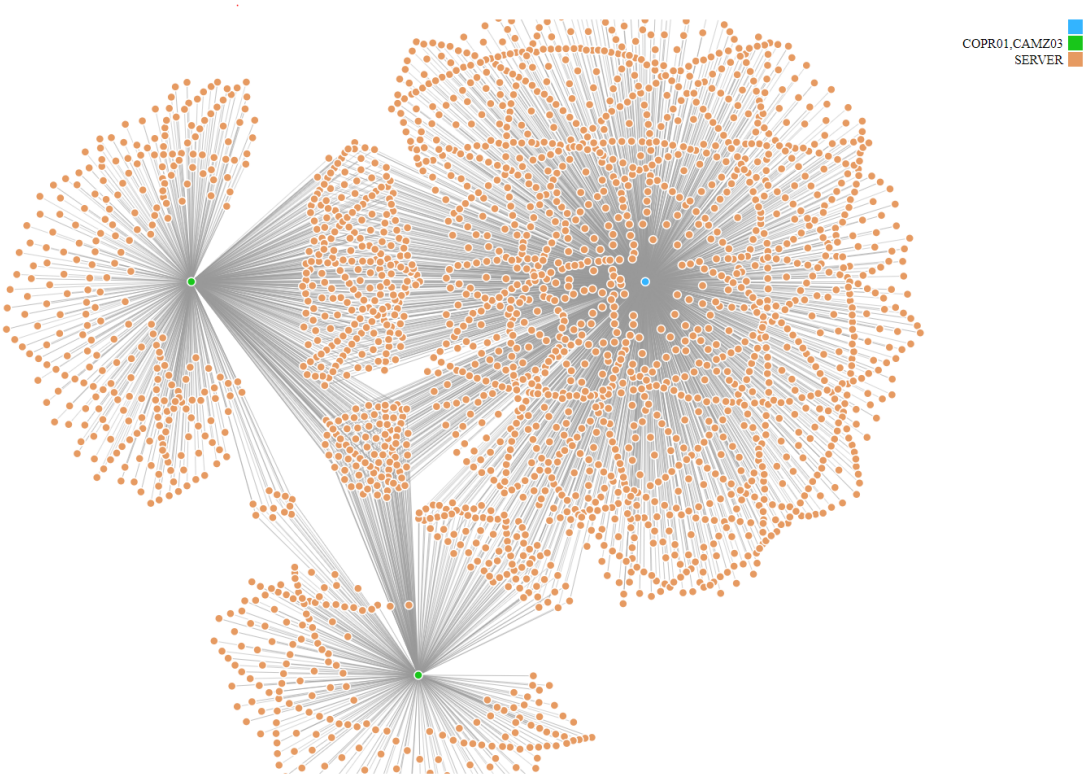


Figure 4.5: Detail view of the network data exploration tool by Lokoc et al. [2017]. For each client, servers that the client communicated with are displayed. In this view, experts can inspect if selected clients communicated with the same servers and then point to potentially suspicious servers.

scalability issues are emerging for multimedia data and systems for evaluating queries in parallel and storing data in a big cluster proved to be one of the solutions dealing with such matter.

5. Distributed similarity joins

From all presented retrieval systems studied in different domains so far, the need for effective and efficient query evaluation is evident. As discussed earlier, one of the promising ways is adopting query evaluation systems and index structures to a distributed environment to deal with scalability issues. One of the great benefits of the parallel distribution is that an evaluation cluster can be relatively easily reduced or enlarged depending on the size of a dataset and the needs of a multimedia retrieval system. In the chapter, we describe similarity joins for metric spaces that can calculate many queries altogether. Furthermore, joins are useful for pre-computing large similarity graphs and networks that can be utilized for multimedia exploration without the need for evaluating queries on the fly.

In this chapter, we introduce several algorithms working in the MapReduce environment implemented in Hadoop and Spark platforms. The chapter is heavily based on the journal paper by Cech et al. [2020] written mostly by the author of this thesis. I am very grateful for all the discussions and numerous corrections to my co-authors.

5.1 Motivation

The k-nearest neighbor (k-NN) similarity join is an asymmetric operation that returns the k most similar objects in a dataset S for each query object in a dataset R. In recent years, the study of k-NN joins attracted a considerable amount of attention due to their applicability in various domains. In the data mining and machine learning context, k-NN joins can be employed as a pre-processing step for classification or cluster analysis. In data exploration and information retrieval, similarity joins provide a similarity graph with potentially relevant entities for each object in the database. k-NN similarity join applications can be found, for example, in image and video retrieval (Ferhatosmanoglu et al. [2001], Giacinto [2007], Cobârzan et al. [2017], Lokoc et al. [2018]), spatial databases (Hjaltason and Samet [1999]), pattern recognition (Muja and Lowe [2014]), and network communication analysis and malware detection frameworks (Cech et al. [2016], Lokoc et al. [2016]).

Because data volumes are often too large to be processed on a single machine (especially for high-dimensional data), we focus on the distributed MapReduce environment Dean and Ghemawat [2008] running on Hadoop[1] and Spark[2]. MapReduce is a widely adopted framework and is considered an efficient and scalable solution for distributed big data processing. MapReduce programs are designed to run on large clusters of commodity hardware and employ a programming paradigm similar to the divide and conquer approach. Datasets are loaded, split and pre-processed in the map phase and the main execution and evaluation of an algorithm are performed in parallel on smaller data fractions in the reduce phase. More details about MapReduce and specific platforms are described in Section 1.4.

[1] `http://hadoop.apache.org/`
[2] `http://spark.apache.org/`

In this chapter, we study approximate k-NN similarity join algorithms that can provide significant speedup compared to the exact similarity join while still preserving high results precision. In many domains, the difference between exact and slightly different k nearest results is acceptable. This is particularly the case in scenarios where computing the exact similarity joins over big high-dimensional data would take significantly large execution times.

Our study focuses on similarity joins for MapReduce environments based on the metric space approach (Zezula et al. [2006]). This approach provides a universal framework for the efficient processing of various similarity models. For evaluations on vector data, we also revisited and extended two previously proposed k-NN similarity join approaches designed for vector spaces. In this chapter, we focus on algorithms employing data organizations and replication strategies initialized randomly as these techniques can be conveniently applied to Big Data in different domains. Although a study tackling related similarity joins has been previously published for Hadoop by Song et al. [2015], the study focused on low dimensional data. The subsequent journal paper by Song et al. [2016] tested data up to 386 dimensions and highlighted limitations for most k-NN join methods on such a high-dimensional dataset. The need for effective and efficient k-NN similarity joins for high-dimensional data led us to (1) design distributed similarity join techniques with thresholds or approximation guarantees, (2) revise available MapReduce algorithms integrating extensions to more efficiently handle high-dimensional data, (3) consider the implementation of such algorithms on a different platform - Spark (in addition to Hadoop), and (4) experimentally evaluate and compare the performance of the different approaches.

This chapter draws mainly from the journal paper by Cech et al. [2020], from a short conference paper that presented a comparison of our heuristic method with two previously proposed approaches on Hadoop by Cech et al. [2017] and follows the paper proposing the pivot-based heuristic k-NN join method by Cech et al. [2016] as well. We introduce several MapReduce based methods for processing k-NN similarity joins in Hadoop and Spark environments. We study exact, approximate, and ϵ-approximate versions of pivot-based algorithms and also tackle other related algorithms. Moreover, we provide implementation details and present comprehensive evaluations of all mentioned approaches.

5.1.1 Contributions

The overall contribution of our work regarding similarity joins can be summarized into four points:

- Extensions of previously proposed k-NN similarity join algorithms on MapReduce to process big high-dimensional data more efficiently.
- The introduction of pivot-based k-NN similarity join heuristic approaches on MapReduce that support approximation-related thresholds and guarantees. We analyze an approach that provides the ϵ-guarantee (which constrains the distance from each query point to its furthest neighbor returned in the k-NN join). We include a discussion of the theoretical foundations that support the proposed methods.

- The Spark and Hadoop implementation guidelines of the proposed MapReduce join methods. We point out the limitations of different platforms and show why Spark provides faster execution times. We also provide the source code of the Spark implementation of all the evaluated methods, including our new implementations of baseline related approaches based on space filling curves (Z-curve) and locality sensitive hashing.

- Thorough and extensive performance evaluation on large data with different dimensionality (from 10 to 1000 dimensions) running on fully distributed Amazon clusters, with most experiments evaluated on the Spark platform processing up to tens of millions of objects. This analysis provides guidance for selecting an appropriate algorithm for distributed k-NN join based on workload and approximation precision requirements.

The remaining part of the chapter is structured in the following way. In Section 5.2, basic formal definitions and common terms are revised. An overview of similarity joins problems, two related methods, and several proposed extensions of these methods are covered in Section 5.3. Section 5.4 presents several exact and approximate pivot-based k-NN similarity join algorithms on MapReduce and provides their implementation guidelines. Finally, in Section 5.5, the performance evaluation of all the implemented algorithms is presented and the results are discussed.

5.2 Preliminaries

The fundamental concepts and basic definitions related to approximate k-NN similarity joins are revised in the following subsections, considering the standard notations by Song et al. [2016], Zezula et al. [2006].

5.2.1 Similarity model and k-NN joins

In this work, we address the efficiency of k-NN similarity joins of complex objects obj_i (e.g., images or network traffic snapshots) modeled by high-dimensional vectors $o_i \in \mathbb{R}^n$. Unless otherwise stated, in the following text the term object denotes the vector representation. In connection with a metric distance function $\delta : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_0^+$, the tuple $M = (\mathbb{R}^n, \delta)$ forms a metric space that serves as a similarity model for retrieval (low distance means high similarity and vice versa)[3].

A popular basic similarity search operation is the k-NN query that returns k nearest neighbors from a database $S \subset \mathbb{R}^n$ to a given query object $q \in \mathbb{R}^n$.

$$kNN(q, S) = \{X \subset S; |X| = k \wedge \forall x \in X, \forall y \in S - X : \delta(q, x) \leq \delta(q, y)\} \quad (5.1)$$

A more complex k-NN similarity join operator is defined for a database $S \subset \mathbb{R}^n$ and a set of query objects $R \subset \mathbb{R}^n$ as:

$$R \bowtie S = \{(q, s) \mid q \in R, s \in kNN(q, S)\}. \quad (5.2)$$

[3]The effectiveness of the distance function and feature extraction mapping from obj to o is the subject of similarity modeling.

Because of the high computational complexity of similarity queries especially combined with similarity joins, approximations of the queries can be considered to significantly reduce computational costs (increase efficiency) while maintaining reasonable precision (trying to preserve effectiveness). An approximation can be achieved by employing many techniques and is divided into two main types: approximation with or without a guarantee. Typically, it is more difficult to achieve any form of guarantee. For example, the guarantee can be enforced by setting up a maximal distance parameter ϵ (ϵ-guarantee) or defining the worst statistical error. Apart from that, methods without guarantee, usually, only limit, for example, the maximum number of distance computation or query evaluation time. Formally, ϵ-approximation of the exact k-NN query (denoted as kNN_a) is defined as:

$$\begin{gathered} kNN_a(q,S) = \{X \subset S; |X| = k \wedge \\ \max_{x \in kNN(q,S)} \delta(q,x) \leq \max_{x \in X} \delta(q,x) \leq \epsilon \cdot \max_{x \in kNN(q,S)} \delta(q,x)\} \end{gathered} \tag{5.3}$$

where $\epsilon \geq 1$ is a constant affecting the level of approximation. The approximate k-NN similarity join is then defined as:

$$R \bowtie_a S = \{(q,s) \mid q \in R, s \in kNN_a(q,S)\}, \tag{5.4}$$

While the previous definition of the (epsilon) approximate k-NN join has been commonly used in the literature, to the best of our knowledge, no previous paper has actually implemented a MapReduce k-NN similarity join algorithm that receives ϵ as a parameter and guarantees the ϵ-related property specified in the definition. Instead, previous papers have primarily proposed heuristic methods that aim at (1) having shorter execution times than the exact k-NN join, and (2) having a relatively high result quality quantified by alternative approximation measures like precision and total distance error (these and other measures are presented in Section 5.2.2). To present a comprehensive study and the trade-offs of different types of approximate k-NN joins, in this chapter, we present a method that satisfies the ϵ guarantee and compares it with extensions of several heuristic-based methods.

5.2.2 Approximation measures

Once approximate k-NN joins are considered, the similarity join approximation quality and error have to be evaluated. For this purpose, different approximation measures (Patella and Ciaccia [2008]) can be utilized.

The k-NN query approximation precision is defined with respect to the result of exact k-NN search as:

$$precision(k,q,S) = \frac{|kNN(q,S) \cap kNN_a(q,S)|}{k} \tag{5.5}$$

An object $o_i \in kNN(q,S)$ matches an object $o_j \in kNN_a(q,S)$ iff either $ID(o_i) = ID(o_j)$ or $\delta(q,o_i) = \delta(q,o_j)$ (equal distant objects from a query q may be present in the k-NN result in an arbitrary order). Final approximate precision is computed as a sum of matching k nearest objects divided by the k.

Other approximation measures we use are the total distance ratio,

$$DR(k, q, S) = \frac{\sum_{i=1}^{k} \delta(q, kNN(q, S)[i])}{\sum_{i=1}^{k} \delta(q, kNN_a(q, S)[i])} \tag{5.6}$$

which represents a ratio between the sums of all neighbor distances in exact and approximate join results and the effective (epsilon) error for the k^{th} neighbor,

$$\epsilon_{eff}(k, q, S) = \frac{\delta(q, kNN_a(q, S)[k])}{\delta(q, kNN(q, S)[k])} \tag{5.7}$$

comparing the distances to a specific neighbor (usually at the k^{th} position).

In all definitions, the $kNN(q, S)[i]$ expression stands for the i^{th} neighbor in a sorted $kNN(q, S)$ result.

5.3 Related work on similarity joins

Many types of different similarity joins have been defined and studied over recent years. Specifically, previous work in this area studied k-Distance joins (Hjaltason and Samet [1998]) (returns the smallest k pairs between two datasets), range query joins (Silva and Reed [2012], Ma et al. [2016]) (returns all the pairs with a distance equal to or smaller than a given threshold) and k-NN similarity joins (for each record of the first dataset, it returns the k closest records in the second dataset) by Böhm and Krebs [2004], Lu et al. [2012]. Some join techniques focus just on specific data types, e.g., set-similarity joins by Vernica et al. [2010], Rong et al. [2017] or string joins with edit distance constraints (Ed-Join by Xiao et al. [2008], Trie-Join by Wang et al. [2010]). Other approaches, like QuickJoin by Jacox and Samet [2008], use pivot-based iterative space partitioning or utilize grid structure (Epsilon Grid – EGO by Böhm et al. [2001]). Similarity joins were also studied in the context of database systems and database operators (Chaudhuri et al. [2006], Silva et al. [2015]). In this chapter, we focus on the last type of similarity joins: the k-NN joins which return the k nearest neighbors for each query object. The k-NN joins are usually specified on either metric spaces (Lu et al. [2012], Jacox and Samet [2008]) or just vector data spaces (Kim et al. [2016], Tang et al. [2015]). Some centralized solutions for k-NN joins employ an index structure providing a significant evaluation speed up, e.g., the R-tree (MuX by Böhm and Krebs [2004]) or B^+-tree (iJoin by Yu et al. [2007]) based techniques.

Since sharing a complex index structure is not efficient and perhaps not even viable in a distributed environment many algorithms cannot be optimally parallelized. On the other hand, different algorithms were recently proposed directly for the MapReduce framework which is designed to work in a fully distributed environment composed of up to thousands of computing machines. Specifically, for the MapReduce paradigm, multiple methods were proposed for range query joins (Ma et al. [2016, 2017], Silva et al. [2012]) and k-NN joins utilizing pivot space partitioning (Cech et al. [2016], Lu et al. [2012], Kim et al. [2016], Hu et al. [2016]), space filling curves (Z-curve by Zhang et al. [2012]), locality sensitive hashing (Stupar et al. [2010], Zhu et al. [2015]) and Hamming distance filtering by Tang et al. [2015]. The k-NN algorithms (e.g., Zhang et al. [2012], Stupar et al. [2010]) usually work in three phases: a data partitioning

phase, a partial k-NN join computation, and intermediate results merge phase. However, not all methods need the third merge phase (e.g., Lu et al. [2012], Kim et al. [2016]) because the final results are already produced in the second phase.

Related papers by Moise et al. [2013b,a], Guðmundsson et al. [2017], Zaharia et al. [2016] have analyzed the advantages, disadvantages and bottlenecks of two of the most well-known distributed MapReduce platforms: Hadoop and Spark. Hadoop is a framework with a high focus on disk persisting operations while Spark aims to take advantage of distributed random access memory (RAM) on cluster nodes and stores data on hard drives only when the main memory space is insufficient. Since most of the previous work considering similarity k-NN joins has been proposed only for the Hadoop framework, we also integrate the implementation and performance evaluation on the Spark platform. Some of the algorithm implementation details, in fact, significantly differ between these two platforms.

In this chapter, we primarily focus on MapReduce-based general metric k-NN similarity joins described in detail later in Section 5.4. For the experiments and comparisons performed on vector data, we selected and revisited the implementation of two methods for vector spaces: space-filling Z-curve and locality sensitive hashing. Since the metric joins use randomly selected database objects for indexing, we have selected methods that also use a convenient random initialization of data partitions.

5.3.1 Space-filling curve based k-NN similarity joins

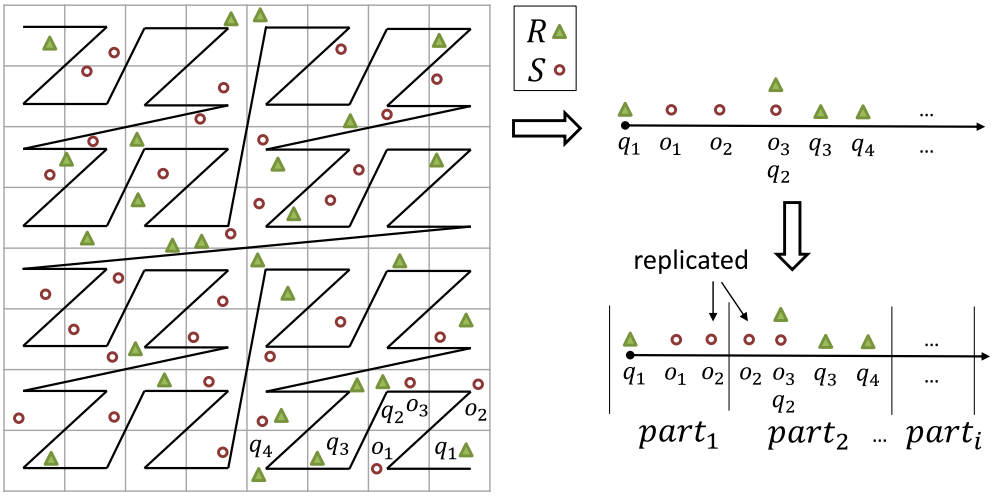


Figure 5.1: Example of the Z-curve mapping objects to the one-dimensional space and following partitioning for a parallel MapReduce processing.

A space-filling curve is an approach to map $n-$dimensional objects to a one-dimensional domain and yet preserve (to some extent) proximity of the objects. Such an approach can be used to design efficient approximate k-NN search methods using the one-dimensional values. An example of the mapping is the popular z-order curve method that easily creates so-called *z-value* from object coordinates by interleaving their binary representations (see Figure 5.1). Different types of space filling curves have been proposed for approximate k-NN search, e.g., Z-curve

by Yao et al. [2010], Z-curve with projections by Schubert et al. [2015] or Hilbert curve by Angiulli and Pizzuti [2005].

To improve the approximate search precision, Yao et al. [2010] have proposed a centralized approach zk-NN where α randomly shifted copies of the dataset (by $v_i \in \mathbb{R}^n$) are created. For each database copy S_i ($S_i = \bigcup_i \{o_i\} : o_i = o + v_i, \forall o \in S$ and for $i = 0, S_0 = S$), z-values of modified objects are computed and sorted. Similarly, all query objects in R are shifted to copies R_i. Each $q_i \in R_i$ is used to query S_i for $2 \cdot k$ objects with the k nearest lower and k nearest higher z-values from S_i to q_i. The result of a k-NN query is then refined from $2 \cdot \alpha \cdot k$ candidates using the original representations from $\mathbb{R}^n$.

In this thesis, we compare our methods to the work of Zhang et al. [2012] who adapted the zk-NN method for the MapReduce framework. For parallelization purposes, z-values are also used for dividing objects in each copy S_i and R_i into n independent partitions. Every database and query object is sent to a designated partition n_i. Moreover, we must ensure that each query has the required nearest neighbor candidates. Thus, database objects located close to partition boundaries are also replicated (copied) to neighboring partitions to guarantee that each query q will have k lower and k higher neighboring database candidates according to z-values. In ideal case, the best boundary points would be $\frac{1}{n}, \frac{2}{n}, \ldots, \frac{n-1}{n}$ quantiles of R_i to define as equal query distribution as possible. Nonetheless, precise partitioning may be very expensive due to large data volumes. Instead, objects are sampled and depending on their values and the probability model, approximate quantiles are determined.

Inside each partition, every present query object is used to find $2 \cdot k$ nearest database object candidates. Distances to the candidates are evaluated and intermediate k-NN results are returned. Each partition is processed by a separate reducer. Using a suitable number of partitions and having data equally distributed, the portion of data for each reducer is small enough to be stored in node memory. Finally, the nearest objects for each query are detected by merging the candidate k-NN results obtained from all copies S_i. The Z-curve approach overview is depicted in Algorithm 1.

Implementation revision

The Hadoop application runs in three MapReduce jobs. We came out of the Java source codes that were provided by the authors Zhang et al. [2012] of the original paper. Then we made minor modifications such as data representation changes and memory optimization which favor evaluation time. For purposes of high-dimensional data computation, we also improved the z-values serialization and implemented the z-value computation of floating point values (values are converted to integers by scaling them by the given constant). The algorithm itself has not been modified.

Our Z-curve implementation in Spark includes a few improvements and adjustments. We integrated an option named *OnlyZorder* which decides how objects with mapped z-value are stored. In case $OnlyZorder = true$ the algorithm works in the same way as Hadoop version while the $OnlyZorder = false$ option means that we store an original object's vector together with the z-value which results in a higher memory usage but the computation is faster because the back transformation from the z-value to original vector coordinates does not have to be

Algorithm 1 Z-Curve k-NN join(R, S, k, $shifts$, $samplingRate$)

1: Z_R = map objects R to z-values using $shifts$ //for all shifts
2: Z_S = map objects S to z-values using $shifts$
3: $Part_R$ = partitions of Z_R, employing $samplingRate$ //$Part_R$ – no overlap
4: $Part_S$ = partitions of Z_S, employing $samplingRate$ //$Part_S$ may overlap

5: **for** query q **in** R **do**
6: **for** $shift_i$ **in** $shifts$ **do**
7: $z_{q,i}$ = z-value corresponding to q in $shift_i$ from Z_R
8: $Part_{i,j}$ = get partition Id for $z_{q,i}$ according to $Part_R$
9: output pair $[Part_{i,j}, z_{q,i}]$ // the key = [shift id, partition index]
10: **end for**
11: **end for**

12: **for** db object o **in** S **do**
13: **for** $shift_i$ **in** $shifts$ **do**
14: $z_{o,i}$ = z-value corresponding to o in $shift_i$ from Z_S
15: $Partitions_{o,i}$ = get partitions for $z_{o,i}$ according to $Part_S$
//the replication – db objects can belong to multiple partitions in each shift
16: **for all** $Part_{i,j}$ **in** $Partitions_{o,i}$ **do**
17: output pair $[Part_{i,j}, z_{o,i}]$
18: **end for**
19: **end for**
20: **end for**

21: —*partition, shuffle, group by key*—
22: **for all** partitions $Part_{i,j}$ **do**
23: parse all query objects to $Z_{R_{i,j}}$ and db objects to $Z_{S_{i,j}}$, order $Z_{S_{i,j}}$
24: **for** z_q **in** $Z_{R_{i,j}}$ **do** //for all queries in the bucket
25: z_o = binarySearch(z_q, $Z_{S_{i,j}}$)
26: produce candidate set $C_{i,j} = \{z_o - k, z_o + k\}$ of db objects, $|C_{i,j}| = 2k$
//$C_{i,j}$ contains k lower and k higher db objects based on the z-value
27: output intermediate result $kNN_a(q, C_{i,j})$
28: **end for**
29: **end for**
30: **for** query q **in** R **do** //final k-NN results are produced
31: merge intermediate results $kNN_a(q, C_{i,j})$ into final result $kNN_a(q, S)$
32: **end for**

performed (and we can also utilize a fast L2 distance for sparse vectors which is used for other methods).

We also integrated a second parameter called *EntirePartitions* which determines how many database objects in each partition S_i are considered for real distance computations to a query $q \in R_i$. If $EntirePartitions = false$ then the algorithm uses the original $2 \cdot k$ closest neighbors by z-values from an S_i. In case the $EntirePartitions = true$, all database objects in a specific partition S_i are considered and all distances $d(q, s_i), s_i \in S_i$ are evaluated. The second option ($EntirePartitions = true$) leads to higher approximation precision but runs substantially longer. Detailed results for all options are presented in the experimental Section 5.5.

Other than that, the Spark implementation follows the original Hadoop algorithm and the best Spark programming practices. All shared data structures are broadcasted to all executors and intermediate results are kept in memory (persistence mode is set to the MEMORY_AND_DISK mode) and, usually, don't have to be written to HDFS. Here we observe the most significant speed up for a comparison between Hadoop and Spark implementations among all studied k-NN join algorithms (concrete numbers are presented in Section 5.5). This observation is explained by many disk operations performed by the Hadoop Z-curve implementation. Also, the Hadoop version is implemented in three separate MapReduce jobs and utilizes the MultipleOutputs class for storing temporary results and statistics (e.g., partitions). These results have to be merged and distributed to all mappers and reducers in the following stages which require a lot of I/O operations. Compared to that, the Spark implementation keeps all needed data in memory and does not require repetitive (expensive) disk accesses (assuming a reasonable amount of RAM is available on computing machines).

5.3.2 LSH-based k-NN similarity joins

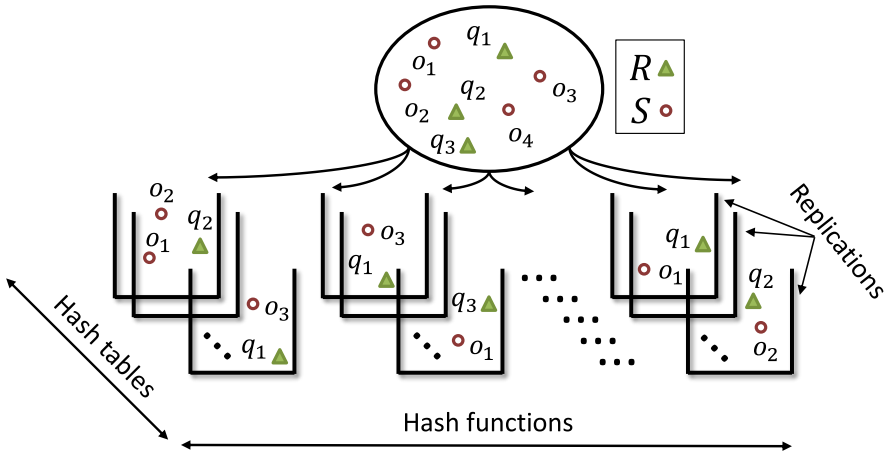


Figure 5.2: Schema of locality sensitive hashing.

Locality Sensitive Hashing (LSH, Datar et al. [2004]) is a well-known technique that hashes similar objects into the same buckets with high probability. Thanks to this property, LSH can be effectively used also for the k-NN search since the nearest neighbors are likely to be located in the same bucket. Specifically,

RankReduce proposed by Stupar et al. [2010] represents an approximate k-NN join algorithm running in MapReduce environment to simultaneously process a small number of k-NN search queries in a single MapReduce job using the LSH algorithm. The key idea behind RankReduce is to use hashing to build an index that assigns similar records to the same hash table buckets. Zhu et al. [2015] proposed an improved version of RankReduce which builds the index in more efficient way and also compares queries only with database candidates that appear more frequently in the same buckets as queries. However, both related MapReduce techniques are oriented towards long-running querying systems. They maintain the database index and the main goal is to provide fast responses to requested k-NN queries. An important property is that they assume that the number of queries is relatively small. On the other hand, many similarity join application scenarios focus on evaluating a lot of queries at once and possibly only once.

Unlike the previously mentioned methods, our LSH approach does not build persisting database indices because we focus mainly on performing independent similarity joins. Nevertheless, our algorithm could be easily adjusted to store hashed database objects for later use. Also, the previously proposed techniques are not really scalable because they assumed only a small number of queries R in the input which is a big limitation in many application scenarios. Considering this, we implemented a new LSH algorithm. From a high level point of view, our algorithm just compares database and query objects that fall into the same hash bucket after performing hashing operations on both R and S sets. To increase the approximation precision, intermediate k-NN results are produced for several independent sets of hashing functions and the final k-NN join is formed by merging the intermediate results. An illustration is depicted in Figure 5.2.

The presented method is composed of two main MapReduce jobs in Hadoop: a hashing job including k-NN evaluation and a merging job. During the map phase of the hashing job, both database (S) and query objects (R) are hashed using a set of i hash tables each containing j hash functions of the form $h_{a,B}(v) = \lfloor (a \cdot v + B)/W \rfloor$, where W is a parameter and a and b are constants generated from the p-normal distribution. For every input record $v \in S \cup R$, a set of output keys (buckets) $hash_i$ are evaluated. One $hash_i$ represents a unique string formed from j hash functions corresponding to the hash table i. The map phase emits pairs of the form $(hash_i, v)$. In the reduce phase of the hashing job, local k-NN candidates are computed for a subset of queries and database objects in every bucket identified by the key $hash_i$. In the second MapReduce job, all partial results are loaded, grouped by the query object IDs and global k-NN results for all queries are produced. The whole LSH application is also described in the Algorithm 2.

Implementation revision

We implemented this algorithm from scratch and initially followed the algorithm presented in Stupar et al. [2010] until we realized different needs and limitations of the original approach. From the original paper is unclear how to implement hashing using pre-computed hash functions $h_{a,B}(v)$. Because hashing objects directly resulted in just one big hash bucket, we implemented a pre-processing step including standard normal transformation (a value o_i is transformed to $o'_i = \frac{o_i - \mu}{\sigma}$) of all input objects.

Altogether, our LSH program is composed of three MapReduce jobs. In the

Algorithm 2 LSH k-NN join(R, S, k, $hashTables$, W)

apply standard normal transformation to R and S //helps with hashing

for hash table HT_i **in** $hashTables$ **do**
    for query q **in** R **do**
      //bucket $B_{i,j}$ contains composed results from all hash functions in HT_i
        hash q in $B_{i,j}$ using HT_i and W
        output pair $[B_{i,j}, q]$
    end for
    for db object o **in** S **do**
        hash o in $B_{i,j}$ using HT_i and W
        output pair $[B_{i,j}, o]$
    end for
end for

—*partition, shuffle, group by key*—
for all hash buckets $B_{i,j}$ **do**
    parse queries to $R_{i,j}$ and db objects to $S_{i,j}$
    for query q **in** $R_{i,j}$ **do**
        output intermediate result $kNN_a(q, S_{i,j})$
    end for
end for
for query q **in** R **do** //final k-NN results are produced
    merge intermediate results $kNN_a(q, S_{i,j})$ into the final result $kNN_a(q, S)$
end for

first job, statistics for the normal transformation are collected. The second job performs the transformation of all objects and computes the hash values in the map phase. In the reduce phase of the second job, the intermediate k-NN results are computed across all buckets. In the last job, intermediate results are merged and global k-NN results are produced.

Our Spark implementation follows the Hadoop one closely. There is no significant difference, data are also transformed using normal standard transformation, all hash tables are broadcasted to all executors, data are grouped by the output of hashing functions and intermediate results are merged to produce final k-NN join results. The only difference is that dimension statistics and intermediate results do not have to be written back to the HDFS and are kept in memory which speeds up the whole application execution.

5.4 Pivot-based k-NN similarity joins

Pivot-based methods represent a useful generic approach with the convenient random initialization, which nevertheless reflects data distribution by dividing a metric space into partitions centered around global objects (pivots) selected from the dataset. The benefits of pivot-based methods have been investigated for k-NN similarity joins on MapReduce in the work of Lu et al. [2012]. The authors describe how mappers cluster objects into groups and reducers perform the k-NN join on each group of objects separately. Distance function properties are used to define exact rules for data replication and filtering of non-relevant objects. However, in high-dimensional metric spaces the rules are not sufficiently efficient (the curse of dimensionality effect). In this section, we investigate algorithms for approximate k-NN similarity joins on MapReduce. First, the work of Lu et al. [2012] is presented including our comments on revised parts (Section 5.4.2). Then, we present our additional modifications of the exact search method to obtain a heuristic method in Section 5.4.3 and we discuss a method with the ϵ-guarantee in Section 5.4.4. For better clarity, Table 5.1 summarizes the symbols of frequently used sets in the following subsections.

$S \subset \mathbb{R}^n$	A finite set of database objects
$R \subset \mathbb{R}^n$	A finite set of query objects
$P \subset S$	A finite set of pivots selected from database objects
$C_i \subset \mathbb{R}^n$	Voronoi cell: $\{x \mid x \in \mathbb{R}^n \wedge p_i \in P \wedge \forall_{p_j \in P}(\delta(x, p_i) \leq \delta(x, p_j))\}$
$S_i = S \cap C_i$	Database objects in the Voronoi cell C_i
$R_i = R \cap C_i$	Query objects in the Voronoi cell C_i
$C = \bigcup_{i=1}^{\lvert P \rvert} \{C_i\}$	Set of all Voronoi cells for the set of pivots P
$G_1, ..., G_m \subset C$	A decomposition of C into m groups of Voronoi cells
$S_i^l \subseteq S_i$	Subset of database objects from S_i replicated to group G_l
$R_i^l = R_i$	All query objects from R_i are replicated to group G_l

Table 5.1: Symbols of frequently used sets.

5.4.1 Exact k-NN similarity join approach

The original version of the pivot-based exact k-NN join algorithm by Lu et al. [2012] (referred to as PGBJ) utilizes a Voronoi partitioning based on the set of pre-selected global pivots $p_i \in P$ and a metric distance function δ. The algorithm is composed of two main phases: data pre-processing and k-NN join evaluation. The general evaluation workflow of the algorithm is depicted in Figure 5.3.

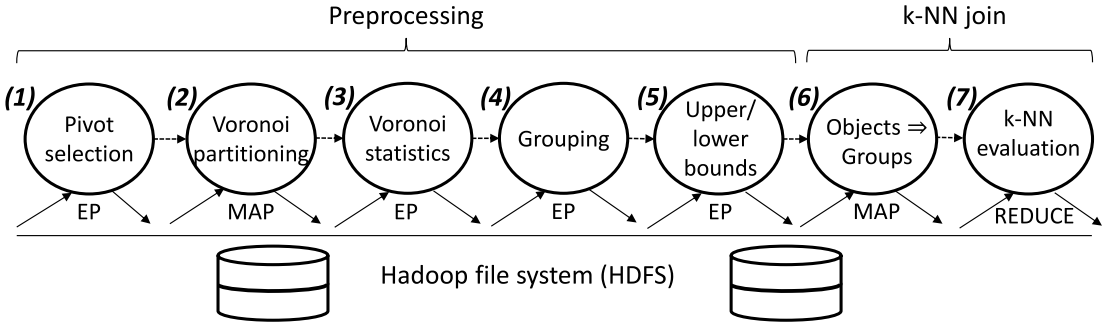


Figure 5.3: Workflow of the original implementation of the pivot-based approach for MapReduce by Lu et al. [2012]. Solid arrows represent data flow, dashed arrows represent algorithmic steps.

The pre-processing phase consists of five steps. In step **5.3.1**, pivots P are selected from the set of database objects S by an external program (EP). In step **5.3.2**, a Map job is used to evaluate sets $S_i = S \cap C_i$ and $R_i = R \cap C_i$ for all Voronoi cells C_i. Specifically, distances from all objects $S \cup R$ to all pivots P are computed and for every object the nearest pivot is identified. For database and query objects, the following records are created and stored (Table 5.2).

$$\begin{aligned} \text{database object record} \quad & rec_o = [o \in S_i, ID_{p_i}, \delta(o, p_i)] \\ \text{query object record} \quad & rec_q = [q \in R_i, ID_{p_i}, \delta(q, p_i)] \end{aligned}$$

Table 5.2: Records for database and query objects.

In step **5.3.3**, statistics are computed for every set S_i and R_i including the covering radius, the number of objects o_j and the total size of all objects o_j. Moreover, the distances of the k nearest objects $o \in S_i$ to the pivot p_i are saved for each set S_i for replication rules. In step **5.3.4**, the Voronoi cells C_i are clustered into bigger disjoint groups G_l to limit the maximum amount of replication (see step **5.3.6**). The authors proposed a grouping algorithm considering both the geometric and volume properties of cells to balance parallel workload of the k-NN join. The number of groups G_l should match the number of reducers (or executors in Spark). In step **5.3.5**, the lower and upper bounds (see Figure 5.4) are computed for replication that guarantees the correct execution of exact search (see step **5.3.6**). The following records (Table 5.3) are created and stored for sets S_i and R_i.

The second phase consists of two steps **5.3.6** and **5.3.7**, where k-NN join of the two sets (S and R) is performed as one MapReduce job. In the replication step **5.3.6**, all query objects $q \in R_i$ are assigned to a group G_l iff $C_i \in G_l$. We will denote the corresponding group in the upper index (R_i^l). Each database

record for S_i	$rec_{S_i} = [\|S_i\|, size(S_i), lb, ub, \{d_1, ..., d_k\}]$
record for R_i	$rec_{R_i} = [\|R_i\|, size(R_i), lb, ub]$

Table 5.3: Records for sets S_i and R_i including the number of objects, size of objects and the minimal (lb) and maximal (ub) distance from the pivot p_i to objects in the set. rec_{S_i} contains also distances to the k nearest objects to the pivot p_i. The records are distributed throughout the cluster (their size is relatively small).

object $o \in S$ is assigned to all groups G_l for which a lower bound $LB(o, R_i)$ is less than or equal to an upper bound $UB(R_i)$ for any $R_i \subset C_i \in G_l$. The utilized query radius upper bound $UB(R_i)$ is pre-computed in step **5.3.5** as depicted in Figure 5.4, considering the distance $R_i.ub$ of the furthest $q \in R_i$ from p_i and the stored distances of the k nearest database objects to each pivot. The lower bound distance to the furthest query object in R_i for one database object o with the closest pivot p_j is determined by the formula:

$$LB(o, R_i) = \max\{0, \delta(p_i, p_j) - R_i.ub - \delta(o, p_j)\}.$$

Technically, the database object independent value

$$v_{ij} = \max\{0, \delta(p_i, p_j) - R_i.ub - UB(R_i)\}$$

for a whole cell S_j can be pre-computed and saved in the pre-processing phase and only the comparison $v_{ij} \leq \delta(o, p_j)$ is performed for each object. A simple lower and upper bounds scheme is visualized in Figure 5.5. An important note is that when Voronoi cells are aggregated into bigger groups, the values are computed for the whole groups, selecting the minimal value across all R_i in the particular group. Objects $o \in S_i$ replicated to a group G_l form the set S_i^l.

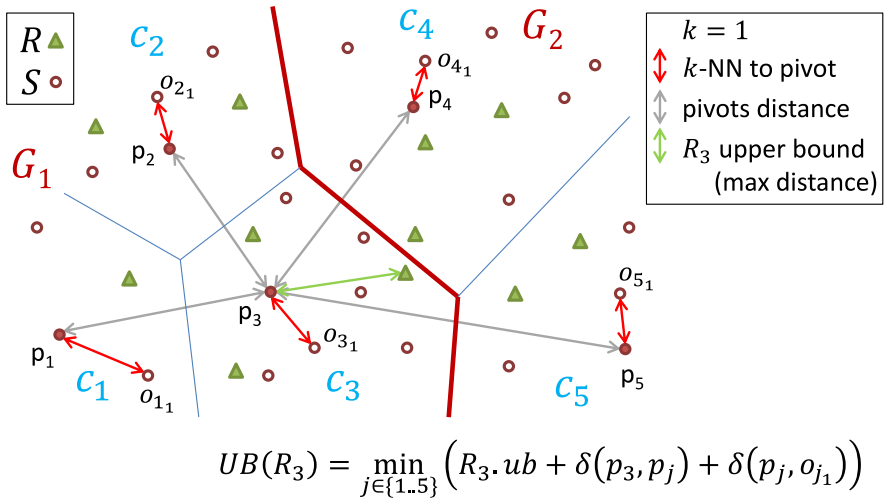


Figure 5.4: An illustration showing how to compute an upper bound for the Voronoi cell C_3 (query objects in the cell C_3 are denoted as R_3) and given $k = 1$. In this case, the result $UB(R_3) = R_3.ub + \delta(p_3, o_{3_1})$ where $R_3.ub$ is the upper bound (maximal distance to a query object $q \in R_3$ from the pivot p_3) and $o_{3_1} \in S_3$ is the closest database object (1-NN) to the pivot p_3.

Figure 5.5: Simple visualization of lower and upper bounds.

Finally, every computing unit cu_l (a reducer or executor corresponding to a group G_l) receives the query objects from all the sets R_i^l and all the sets S_j^l of database objects replicated to group G_l (step **5.3.7**). Technically, the whole records rec_o, rec_q are sent to cu_l as depicted in Algorithm 3, where the corresponding sets of records are denoted as $RecR_i^l$ and $RecS_j^l$. Then, for every query object $q \in R_i^l$, all sets S_j^l are visited according to distances between the query's nearest pivot and pivots p_j. The k-NN query is evaluated using metric space pruning techniques for each set S_j^l. The authors of the original paper by Lu et al. [2012] used parent (Equation 1.8) and cosine law filtering strategies. After all sets S_j^l are processed, the final k-NN result for the query q is produced.

5.4.2 Revisiting the exact k-NN similarity join

In this thesis, we revise some ideas regarding the exact search and the overall MapReduce k-NN join algorithm. We consider the efficient and convenient random pivot selection in step **5.3.1** given that the benefits of more expensive complex pivot selection techniques have been found with numbers of pivots significantly smaller than the ones used in our work (we usually use thousands of pivots). More details are presented in the paper by Bustos et al. [2003]. In step **5.3.4**, we implemented a grouping variant which reflects the total data size in addition to the number of objects in each group G_l. In our previous work by Cech et al. [2016], we showed that such grouping is suitable for space-saving data formats compressing sparse representations.

Regarding the k-NN evaluation step **5.3.7**, we replaced the sorting of database sets S_i^l with respect to a query q assigned to the group G_l. The new version sorts the sets based on the exact distances between the query q and the pivots p_i, unlike the original algorithm which estimated the distances using the closest pivot to q. The new ordering represents a heuristic trying to greedily reduce the actual k-NN range radius and also improve the performance of the employed approximate search heuristic. The authors of the original paper by Lu et al. [2012] used also the cosine law for more efficient filtering. However, this technique cannot be applied to all metric similarity functions so we omit this rule. On the other hand, we propose the use of the ball (Equation 1.9) and hyperplane (Equation 1.10) filtering strategies successfully employed in metric access methods (Zezula et al. [2006]).

Algorithm 3 summarizes the revised PGBJ algorithm, highlighting primarily the k-NN join steps **5.3.6** and **5.3.7**. The algorithm closely follows the description presented in the previous subsection. In the pre-processing phase, all support data structures are computed and shared across the cluster (pivots, groups, lower bounds). Specifically, the lower bound values v_{lj} (line 6) store values for whole groups in ascending order. The function for computing upper bounds is presented in Algorithm 4. This algorithm can compute upper bounds either for one query q

Algorithm 3 Exact pivot k-NN join(R, S, k)

1: *pivots* P = selected pivots from S
2: rec_S, rec_R = sets of db and query records with assigned closest pivot
3: $Stats_S, Stats_R$ = computed statistics for all Voronoi cells
4: *groups* G = grouped Voronoi cells using *stats* and *pivots*
5: *upperBounds* $UB(R_i)$ = computeUB($null$, R_i, k, P, $Stats_S, rec_{R_i}$, $true$)
6: $\forall G_l \in G$: *values* $v_{lj} = \max\{0, \min_{R_i \in G_l}\{\delta(p_i, p_j) - R_i.ub - UB(R_i)\}\}$
for simplicity: $o = rec_o.o$, $q = rec_q.q$, p_i determines $RecR_i$ or $RecS_i$

7: —*map*—
8: **for** query record rec_q **in** rec_R **do**
9: G_l = get group ID for pivot $rec_q.ID_p$
10: output pair $[G_l; rec_q]$
11: **end for**
12: **for** db record rec_o **in** rec_S **do**
//db objects can belong to multiple groups - the replication
13: **for** group ID G_l **in** groups G **do**
14: **if** $v_{li} \leq rec_o.\delta(o, p_i)$ **then** //$\delta(o, p_i)$ is pre-computed
15: output pair $[G_l; rec_o]$
16: **end if**
17: **end for**
18: **end for**

19: —*reduce*— (each group G_l on one computing unit cu_l)
20: **for** groups $G_l \in G$ **do** //all records are grouped by the group id key
21: collect all sets with query and db records to $RecR^l$ and $RecS^l$
22: **for** record rec_q **in** $RecR^l_i, \forall RecR^l_i \in RecR^l$ **do**
23: $RecS^l_{sorted}$ = sort all $RecS^l_j \in RecS^l$ based on $\delta(q, p_j)$ in asc. order
24: $kNN_{q,S} = \emptyset$
25: r_q = computeUB($rec_q, null, k, P, Stats_S, rec_{R_i}, false$)
26: **for** $RecS^l_j$ **in** $RecS^l_{sorted}$ **do**
27: **if** $\delta(q, p_j) - r_q > rec_q.\delta(q, p_i) + r_q$ **then continue** //Equation 1.10
28: **if** $\delta(q, p_j) > RecS^l_j.ub + r_q$ **then continue** //Equation 1.9
29: **for** record rec_o **in** $RecS^l_j$ **do**
30: **if** $|\delta(q, p_j) - rec_o.\delta(o, p_j)| > r_q$ **then continue** //Equation 1.8
31: update $kNN_{q,S}$ by o
32: **if** $|kNN_{q,S}| = k$ **then** $r_q = \max_{x_i \in kNN_{q,S}} \delta(q, x_i)$
33: **end for**
34: **end for**
35: output $kNN_{q,S}$
36: **end for**
37: **end for**

or for a whole query cell R_q (the only difference is the part starting on the line 4) and is used for determining replications in PGBJ and in the ϵ-guaranteed k-NN similarity join algorithm (Section 5.4.4). In the map phase of Algorithm 3, the query objects $q \in R$ are sent to their corresponding group, while the database

objects $o_i \in S$ are distributed to multiple groups G_l based on values v_{lj} and $\delta(o_i, p_j)$.

Algorithm 4 ComputeUB(rec_q, R_q, k, P, $Stats_S$, rec_{R_i}, $wholeCell$)

```
1: PQ = ∅  // priority queue in descending order by distance
2: for rec_{S_i} in Stats_S do
3:     for distance d_j in k distances to p_i in rec_{S_i}.{d_1, ..., d_k} do
           //dist is computed either for one query q or the whole cell R_q
4:         if wholeCell then
5:             dist = rec_{R_i}.ub + δ(p_q, p_i) + d_j //p_q is corresponding pivot to R_q
6:         else
7:             dist = rec_q.δ(q, p_i) + d_j
8:         end if
9:         if PQ.size < k then
10:            add dist to PQ
11:        else if PQ.peek > dist then
12:            remove the first (highest) distance from PQ
13:            add dist to PQ
14:        else
15:            break  //distances d_j are sorted in an ascending distance to p_i
16:        end if
17:    end for
18: end for
19: return PQ.peek
```

In the reduce phase, the k-NN is computed for all queries using the suggested metric space filtering methods and the exact k-NN results are produced.

Implementation revision

The Hadoop implementation is composed of two MapReduce jobs and was provided by the authors Lu et al. [2012]. First, global pivots are chosen, they are stored in HDFS and distributed via the Hadoop distributed cache class. Then, the first MapReduce job computes the distances from all objects in both sets R and S to all pivots, and collects Voronoi cell statistics in the map phase. After that, all statistics are merged together, groups are determined and upper and lower bounds are computed (these operations are implemented separately from any MapReduce job). Statistics, group information and bounds are also distributed throughout the cluster. Then, global k-NN results are computed in the second job. In the map phase, all the objects are assigned to the designated groups (including replication). In the reduce phase, the k-NN computation on a subset of queries and database objects is performed as described previously in Section 5.4.1.

We also implemented the PGBJ algorithm in the Spark environment. The main difference is that the distributed data structures (e.g., global pivots, records with statistics for all sets, group information) do not have to be stored in the HDFS but are kept in memory and broadcasted to all executors. Also, the entire algorithm does not have to be split into two jobs so the total execution time is faster. Empirical results of the comparison are presented in the evaluation Section 5.5.

5.4.3 Heuristic k-NN similarity join approach

Since the PGBJ replication algorithm uses pivot-based upper/lower bounds and the replication strategy is designed for whole Voronoi cells, almost all database objects are replicated to all groups in high-dimensional spaces. With the increasing dimensionality, also the filtering rules lose their pruning power. Such behavior is the consequence of the curse of dimensionality problem (Section 1.3.5). In high-dimensional spaces, the distances between pairs of objects are more similar and thus pivot-based lower bounds $|\delta(p, o) - \delta(p, q)|$ are usually very small, while pivot-based upper bounds are often higher than the highest distance between two objects. We build on our previous work by Cech et al. [2016, 2017], where we proposed a heuristic k-NN similarity join method on Hadoop which significantly speeds up the k-NN join time but preserves high approximation precision in the average case. The method is labeled as the pivot approximate k-NN join (PAKJ) and its high level schema is similar to the PGBJ method presented in Section 5.4.1 (Figure 5.3). Unlike the PGBJ method, the PAKJ method uses no guarantee thresholds to limit replications and the number of visited sets S_i^l by each query.

The replication step used by PAKJ is inspired by a repetitive Voronoi partitioning used by the state-of-the-art metric indexing technique M-Index by Novak et al. [2011] (more details are described in Section 1.3.5). In each set S_i, every database object o further stores a list P^o of identifiers of several closest pivots to o (i.e., pivot permutation prefix by Chavez Gonzalez et al. [2008]) detected in the pre-processing step **5.3.2**. Given the closest pivots to an object o, the proposed replication heuristic assumes that the object o should be replicated mainly to the groups containing Voronoi cells determined by the pivots (illustrated in Figure 5.6 for objects $o_1, o_2, o_3 \in S$). The heuristic has a parameter $MaxRecDepth$ controlling the number of considered closest pivots for each object $o \in S_i$. The replication heuristic utilizes directly the stored nearest pivot identifiers in step **5.3.6**. Specifically, every database object o located in a set S_i is replicated to groups $G_l \subset G$ that contain cells determined by pivots from P^o. Hence, the step **5.3.5** designed to compute upper/lower bounds can be skipped. Database records are stored in the format displayed in Table 5.4.

database object record	$rec_o = [o \in S_i, \{ID_p \mid p \in P^o\}, \{\delta(o, p) \mid p \in P^o\}]$

Table 5.4: Adjusted database records for the PAKJ algorithm.

In the k-NN evaluation step **5.3.7**, the new parameter called $FilterRatio$ is employed to determine an early stop rule. The $FilterRatio$ parameter represents the percentage of visited sets S_i^l after which the k-NN search is stopped (e.g., $FilterRatio = 0.01$ means that after visiting $0.01 \cdot |P|$ sets the k-NN search is terminated if at least k objects were found).

Our implementation of the PAKJ algorithm in Hadoop and Spark is similar to the PGBJ join. New parameters during replication step and k-NN query processing (early termination) are employed and also upper/lower bounds do not have to be computed because they are not used for the replication strategy.

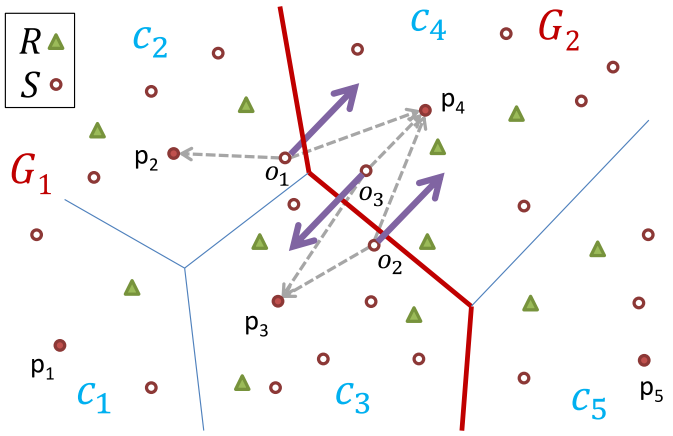


Figure 5.6: An example of PAKJ replication of database objects o_1, o_2, o_3, given $MaxRecDepth = 2$. All three objects are replicated, because their second closest pivot is located in the other group.

5.4.4 k-NN similarity join approach with the ϵ-guarantee

The PAKJ heuristic does not provide the worst case guarantees. This means that despite a good performance in the average case, some k-NN queries forming the join could result in a high effective (epsilon) error (Definition 5.7).

In this section, we investigate a MapReduce based k-NN similarity join method for metric spaces that tackles the ϵ-guaranteed approximation. Enforcing the ϵ-guarantee to support the approximately correct nearest neighbor (AC-NN) queries has been presented in the paper by Ciaccia and Patella [1999, 2000]. More specifically, given a query object $q \in R$ and the distance to its nearest neighbor $r_q \in \mathbb{R}_0^+$ the AC-NN query can return any object $o \in S$ such that $\delta(q, o) \leq \epsilon \cdot r_q$, $\epsilon \geq 1$. The authors present an exact nearest neighbor search algorithm that can be adapted for AC-NN queries by substituting the distance to the actual nearest neighbor candidate r_x by $\frac{r_x}{\epsilon}$. The same idea can be applied for approximate k-NN similarity join methods on MapReduce with the ϵ-guarantee. In the following subsection, we summarize a sound formal background inspired by Ciaccia and Patella [2000] clarifying the correctness of the utilized approach for pivot-based k-NN similarity joins on MapReduce.

Formal background

The goal of the following lemmas is to clarify that utilizing the $\frac{r_x}{\epsilon}$ radius for k-NN search in metric spaces preserves the ϵ-guarantee (Definition 5.4). Moreover, correct metric space filtering methods (object, ball and hyperplane filtering) can be used to speed up approximately correct k-NN queries.

For all the following lemmas, we work with the next premises.

Premises. *Let us assume a given metric space* $M = (\mathbb{R}^n, \delta)$, *a query object* $q \in \mathbb{R}^n$, *a finite data set* $S \subset \mathbb{R}^n$, *an arbitrary candidate result set* $X \subset S$, $|X| = k$, *an actual query radius* $r_x = \max_{x_i \in X} \delta(q, x_i)$, *an approximation parameter* $\epsilon \in \mathbb{R}^+, \epsilon \geq 1$ *and a set of pivots* $P \subset S$.

Lemma 3 clarifies the idea that database objects $o_i \in S - X$ that are in the

"ring" $< \frac{r_x}{\epsilon}, r_x >$ centered in q can be skipped and the ϵ-guarantee will not be violated.

Lemma 3. *Let* $Y = \{o_i \in S \mid \delta(q, o_i) \geq \frac{r_x}{\epsilon}\}$ *and* $r_{xy} = \max_{x_i \in kNN(q, X \cup Y)} \delta(q, x_i)$, *then it holds that* $\frac{r_x}{r_{xy}} \leq \epsilon$.

Proof 3. *Trivial.* $r_{xy} \geq \frac{r_x}{\epsilon}$, *which implies that* $\epsilon \geq \frac{r_x}{r_{xy}}$.

Note 1. *As a consequence of Lemma 3, skipping any database object* $o_i \in S - X$, $\delta(q, o_i) \geq \frac{r_x}{\epsilon}$ *during query processing does not violate the* ϵ*-guarantee condition for* ϵ*-approximate k-NN search. Nevertheless, objects* o_i *satisfying* $\delta(q, o_i) < r_x$ *can be used to update the actual query radius* r_x *and improve filtering efficiency.*

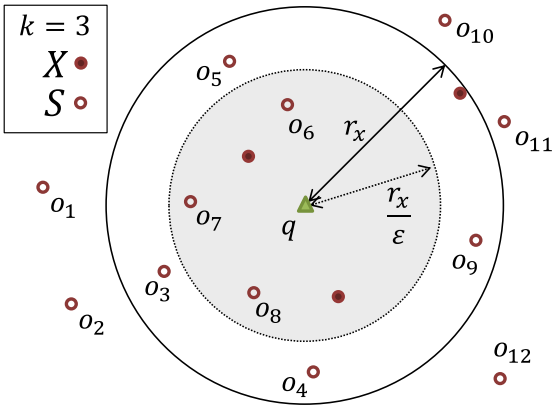


Figure 5.7: An illustration of Lemma 3. Objects outside of the gray area can be safely pruned.

Furthermore, Lemmas 4, 5, and 6 formalize the idea that filtering techniques defined in Subsection 1.3.1 (Definitions 1.8, 1.9, 1.10) do not violate the ϵ-guarantee.

Lemma 4. *Let* $Y = \{o_i \in S \mid \frac{r_x}{\epsilon} \leq \max_{\forall p \in P} |\delta(q, p) - \delta(o_i, p)|\}$ *and* $r_{xy} = \max_{x_i \in kNN(q, X \cup Y)} \delta(q, x_i)$. *Then it holds that* $\frac{r_x}{r_{xy}} \leq \epsilon$.

Proof 4. $\forall o_i \in Y$ *it holds that* $\delta(q, o_i) \geq \max_{\forall p \in P} |\delta(q, p) - \delta(o_i, p)| \geq \frac{r_x}{\epsilon}$. *In connection with Lemma 3 it holds that* $\epsilon \geq \frac{r_x}{r_{xy}}$.

Lemma 5. *Let* $p \in P$, $r_p \in \mathbb{R}^+$, $Y = \{o_i \in S \mid o_i \in Ball(p, r_p) \wedge r_p + \frac{r_x}{\epsilon} \leq \delta(q, p)\}$ *and* $r_{xy} = \max_{x_i \in kNN(q, X \cup Y)} \delta(q, x_i)$. *Then it holds that* $\frac{r_x}{r_{xy}} \leq \epsilon$.

Proof 5. $\forall o_i \in Y$ *it holds that* $\delta(q, o_i) \geq \delta(q, p) - r_p \geq \frac{r_x}{\epsilon}$. *In connection with Lemma 3 it holds that* $\epsilon \geq \frac{r_x}{r_{xy}}$.

Lemma 6. *Let* $p_1, p_2 \in P$, $S_1 = \{o_i \in S \mid \delta(p_1, o_i) \leq \delta(p_2, o_i)\}$, $Y = \{o_i \in S_1 \mid \delta(q, p_1) - \frac{r_x}{\epsilon} > \delta(q, p_2) + \frac{r_x}{\epsilon}\}$ *and* $r_{xy} = \max_{x_i \in kNN(q, X \cup Y)} \delta(q, x_i)$. *Then it holds that* $\frac{r_x}{r_{xy}} < \epsilon$.

Proof 6. *If* $\delta(q, p_1) - \frac{r_x}{\epsilon} > \delta(q, p_2) + \frac{r_x}{\epsilon}$ *then* $\forall o_j \in Ball(q, \frac{r_x}{\epsilon})$ *it holds that* $\delta(p_1, o_j) > \delta(p_2, o_j)$. *Hence,* $\forall o_i \in Y$ *it holds that* $\delta(q, o_i) > \frac{r_x}{\epsilon}$ *and so* $\epsilon > \frac{r_x}{r_{xy}}$.

The ϵ-guaranteed pivot method

Following the Lemmas 3-6, we present an algorithm called the pivot epsilon guaranteed k-NN join on MapReduce (PEGKJ). It follows the revisited exact k-NN join algorithm (Section 5.4.2) and basically uses less strict rules (employing the parameter ϵ) during replication and k-NN query evaluation phases.

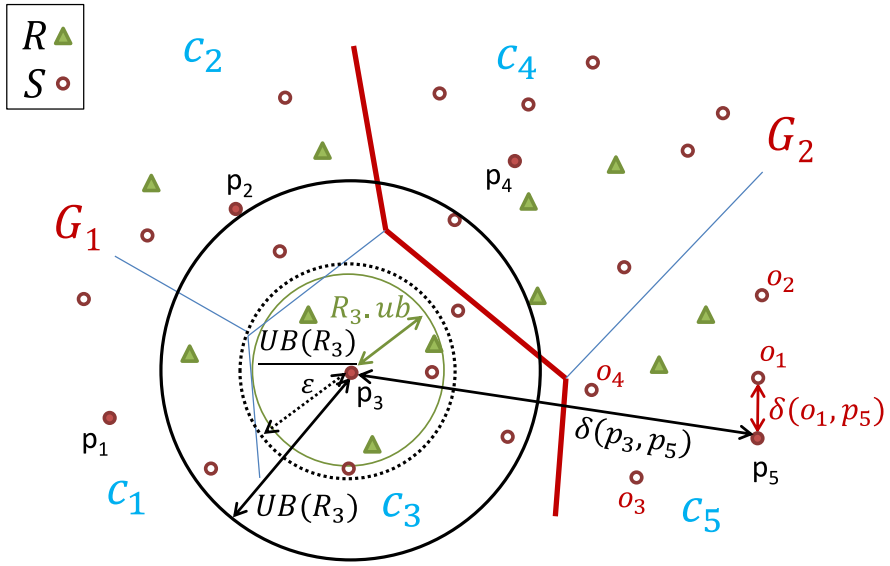


Figure 5.8: An example demonstrating the ϵ upper bound influence. Considering the queries upper bound for the set R_3 equals to $UB(R_3)$ (the black ball), for the exact k-NN join algorithm some database objects from the set S_5 are replicated to the group G_1 (depending on the distance of an object o_j to it's corresponding pivot). After the $UB(R_3)$ bound is lowered by the ϵ (the dashed circle) parameter, objects from the set S_5 are no longer replicated.

We follow the schema in Figure 5.3 with several changes involving mainly the ϵ parameter. The first change is that the upper bound $UB(R_i)$ (estimate of a query radius) for replication of objects to the group containing queries in R_i is divided by the ϵ parameter. As the original $UB(R_i)$ radius estimate guarantees at least k found objects possibly from more than one group, the PEGKJ method computes $UB(R_i)$ using only the sets from the group containing R_i. It is important to realize that computing $UB(R_i)$ from all cells (i.e., from all groups) and lowering the $UB(R_i)$ radius estimate by ϵ could result into a situation where objects used to compute the original $UB(R_i)$ are not replicated to the group containing R_i. Hence, k candidates for a query $q \in R_i$ and initial radius $UB(R_i)$ does not have to be present in the group containing R_i. Starting with a higher initial radius does not help as it could lead to the violation of the k-NN ϵ-guarantee for q because the approximate k^{th} nearest neighbor could be located further than the ϵ times the distance to the exact k^{th} nearest neighbor. Specifically, Algorithm 4 skips pivots determining cells from different groups and in the last line of Algorithm 4 the $PQ.peek/\epsilon$ value is returned. An example and the effect of lowering $UB(R_i)$ is depicted in Figure 5.8.

During the replication phase (potentially) less database objects $o \in S$ are distributed and replicated among the groups because the pre-computed values v_{ij} are greater ($UB(R_i)/\epsilon$ is subtracted). Finally, in the k-NN evaluation phase the ϵ parameter is utilized once first k nearest neighbor candidates are obtained for initial radius estimate $UB(R_i)$. The initial query radius is computed in the same way as in the original paper (using Algorithm 4 where the ϵ is not employed to guarantee a candidate set of size k). However, when k candidates are obtained, the actual radius $\frac{r_q}{\epsilon}$ is correctly used for further query processing (Algorithm 3 lines 27, 28, 30) based on previously presented Lemmas 4, 5, 6. Overall, the PEGKJ algorithm is ϵ-correct because all replication and filtering techniques utilize rules preserving the ϵ-guarantee from Definition 5.4.

In the experiments, we observed that precision drops significantly with the growing ϵ parameter for the PEGKJ method (results are presented in Section 5.5). This behavior is the consequence of too reduced actual query radius, independently on the filtering rule. However, intuitively replication rules or metric ball-ball overlap tests are less sensitive to approximation than, for example, parent filtering. To increase precision of PEGKJ, we propose a parameter called $ExactParentFiltering$ (or shortly $ExactPF$) which determines whether the parent filtering rule (Algorithm 3, line 30) considers ϵ-approximation radius $\frac{r_q}{\epsilon}$ or uses the actual radius r_q.

We also observed that the PGBJ and PEGKJ algorithms tend to replicate almost all database objects to all groups in high-dimensional spaces. Even if we assume that $UB(R_i)$ is always estimated just from the cell C_i (using the equation: $R_i.ub + \delta(p_i, o_{i_j}), o_{i_j} \in S_i$) and we inspect each set R_i in the tested group separately, then objects $o \in S_j$ are not replicated by PGBJ only if $\delta(p_i, p_j) > \delta(o, p_j) + 2 \cdot R_i.ub + \delta(p_i, o_{i_j})$ for all R_i in the group. In high-dimensional spaces, all distances between database objects are relatively similar and the probability that $\delta(p_i, p_j)$ is greater than the sum of four other involved distances in the replication rule is low. Even if two of the distances forming $UB(R_i)$ are further divided by ϵ in the PEGKJ algorithm. Actually, $LB(o, R_i) = 0$ and values $v_{lj} = 0$ were observed almost always for all datasets in our experiments (including the ϵ-approximation). In such cases, most database objects are replicated to all groups.

5.5 Experimental evaluation

In this section, the presented MapReduce k-NN similarity join algorithms are experimentally evaluated and compared. The experiments focus on scalability, precision and the overall execution time of all solutions for high-dimensional data. First, we describe the test datasets and the evaluation platform, then we compare selected methods on two MapReduce frameworks, where we present the benefits of Spark. For Spark, we investigate parameters for all the presented methods and, finally, we compare the performance of selected approaches in multiple testing scenarios. We have also published all the Spark source-codes in a publicly accessible repository on Github[4].

[4]https://github.com/PremyslCech/kNN-joins-spark

5.5.1 Description of datasets and test platform

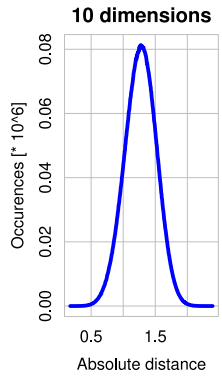


Figure 5.9: Distance distribution – 10D

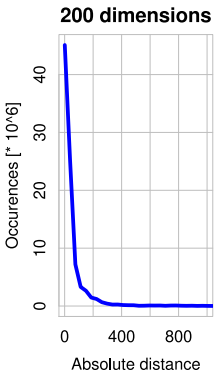


Figure 5.10: Distance distribution – 200D

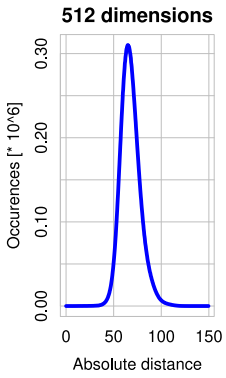


Figure 5.11: Distance distribution – 512D

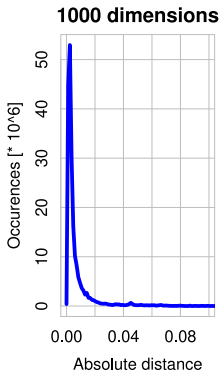


Figure 5.12: Distance distribution – 1000D

In the experiments, we perform k-NN similarity joins on four vector datasets with various number of dimensions: 10, 200, 512 and 1000. In the experiments, we often refer to the datasets by their unique dimensionality and the concatenated letter "D" (see Figures 5.9, 5.10, 5.11, and 5.12).

The 10-dimensional dataset is a synthetic dataset made by generating 200,000 objects into a uniform $[0, 1]^{10}$ cube. This artificial dataset was created to investigate and present the performance of compared methods on data with lower dimensionality.

The 200 and 1000-dimensional datasets contain histogram vectors modeling HTTPS communication (e.g., from web proxy logs). Only high-level communication features of HTTPS requests (Kohout and Pevny [2015b]) were aggregated into vectors using two techniques. The 200-dimensional dataset was created by uniform feature mapping into a 4-dimensional hypercube (Kohout and Pevny [2015b]). In the dataset with 1000 dimensions, HTTPS communication was modeled using the Gaussian Mixture Models approach (GMM) (Marin et al. [2005]). For more details, see Chapter 4, in which the feature extraction algorithms are presented in more detail.

The 512-dimensional dataset consists of 335,944 selected keyframes from the TRECVid IACC.3 video dataset (Awad et al. [2016]). The descriptors for each keyframe were extracted from the last fully connected layer of the pre-trained VGG deep neural network by Simonyan and Zisserman [2014] and further reduced to 512 dimensions using PCA.

All the datasets are divided into the database S and query points R. The distance distributions on a smaller sample for all datasets are presented in Figures 5.9, 5.10, 5.11, and 5.12. The number of database and query objects ranges in hundreds of thousands of objects for most of the experiments. Only the growing data size experiment run on tens of millions of 200D objects. Every object contains a unique object ID and a vector of values stored in the space saving format presented in the work by Cech et al. [2016]. The size of the datasets varies according to the number of dimensions from 0.5GB to 5GB of data in the space saving format. We employ the Euclidean (L_2) distance as the similarity measure.

The experiments run on a fully distributed Amazon clusters under the Elastic MapReduce and EC2 services. We used clusters of 5 to 20 computing nodes each containing the Intel Xeon processor having 4 Cores (8 threads) running at 2.5 Ghz, 15 GB RAM and 2x40 GB SSD disk (the Amazon m3.xlarge instance). Data were stored in the S3 storage system.

5.5.2 Hadoop vs Spark

In the first set of experiments, we analyze the two most popular MapReduce platforms Hadoop and Spark. We compare the platforms on three different similarity k-NN join algorithms – PAKJ and the algorithms based on Z-curves (Section 5.3.1) and locality sensitive hashing (Section 5.3.2). In Figures 5.13, 5.14 and 5.15, we compare the join evaluation time on both platforms for all datasets given the same method settings (i.e., the join result was exactly the same on both platforms). Graphs reflect the join evaluation time for the given parameter setup for each method, but similar outcomes were observed for different settings of parameters. All the tested algorithms run faster on Spark because intermediate results and shared support data structures do not have to be serialized and stored in HDFS but are kept in memory for the whole algorithm execution. Also, all k-NN join methods are implemented on Hadoop in multiple MapReduce jobs (two or three) and results from previous jobs have to be written back to HDFS and sometimes even merged or otherwise manipulated. The biggest difference is noticeable for the Z-curve algorithm which uses multiple I/O operations on Hadoop to provide proper partitioning and data pre-processing. On Spark, all operations run in memory and temporary data structures do not need to be saved on disk. Because of the convincing results on Spark for our datasets, all other experimental analyses, evaluations, and tests are presented just for the Spark platform.

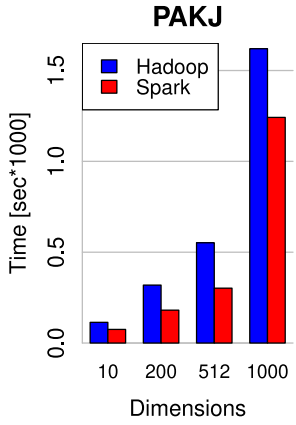


Figure 5.13: PAKJ approach – join evaluation time Hadoop vs Spark

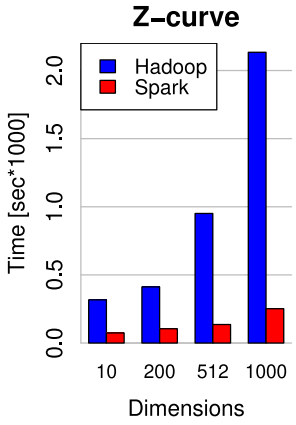


Figure 5.14: Z-curve approach – join evaluation time Hadoop vs Spark

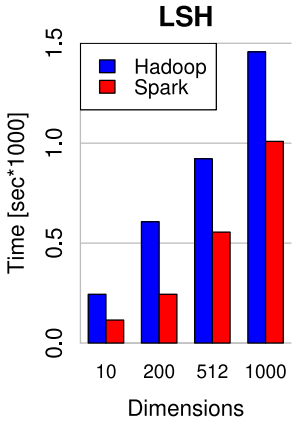


Figure 5.15: LSH approach – join evaluation time Hadoop vs Spark

5.5.3 Fine tuning of k-NN join methods

We follow with the analysis of parameters for the PAKJ approach and methods based on Z-curves and LSH, inspecting precision/speed trade-offs. We would like to emphasize that all presented time values include both the running time of the k-NN similarity join and the pre-processing time. Similar as in our previous work by Cech et al. [2017], the 1000-dimensional dataset was employed and the k was set to 5. Unless otherwise stated, similar behavior was observed also for other datasets.

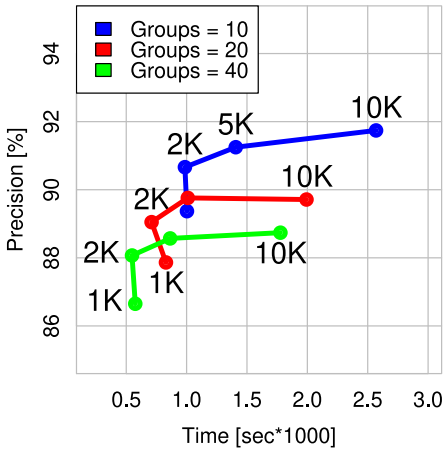


Figure 5.16: PAKJ – growing number of pivots and groups, $MaxRecDepth = 10$, $Filter = 0.01$

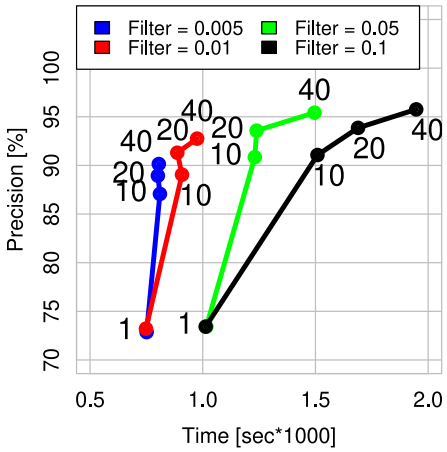


Figure 5.17: PAKJ – $MaxRecDepth$ and $FilterRatio$ parameters tuning, Pivots = 2000, Groups = 20

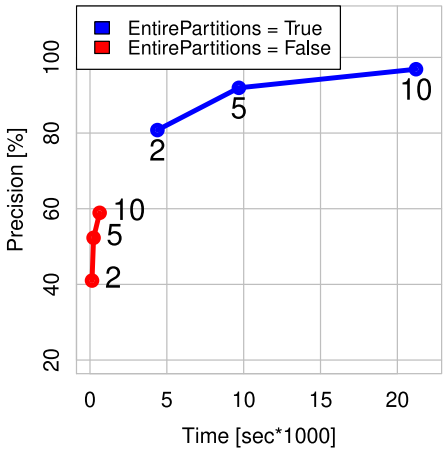


Figure 5.18: Z-curve – number of shifts parameters tuning, All in memory

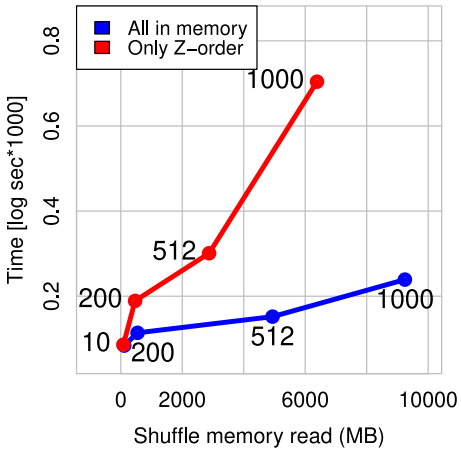


Figure 5.19: Z-curve – effect of storage options, $Shifts = 5$, $EntirePart = false$

In Figure 5.16, we study the influence of the number of groups G and randomly selected pivots P used for the Voronoi partitioning on the PAKJ algorithm performance. According to our internal test cases, other pivot selection techniques

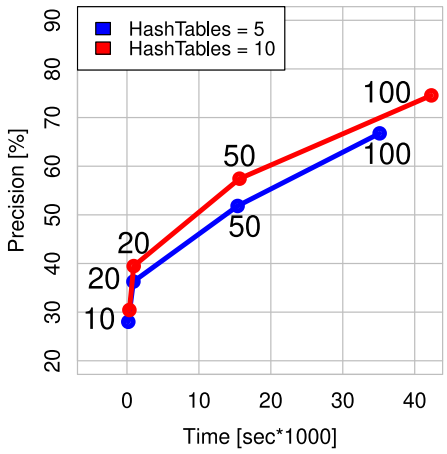


Figure 5.20: LSH approach – W parameter tuning, $HF = 20$

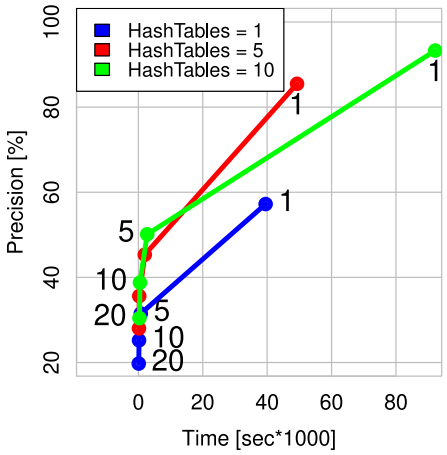


Figure 5.21: LSH approach – the effect of the number of hash functions. $W = 10$

in the original source codes provided by Lu et al. [2012] run substantially longer and did not present any significant improvements for further evaluation. Based on the results, we have fixed the number of pivots to 2,000 and the number of groups to 20 in the remaining experiments. It is suggested to set the number of groups to match the number of reducers or executors to achieve the best parallel computation balance.

Figure 5.17 shows the observed behavior of $MaxRecDepth$ and $FilterRatio$ parameters for the PAKJ method (see Section 5.4.3). As expected, lower parameter values lead to the faster processing time in most cases but achieve also limited precision. For the rest of the experiments, the $MaxRecDepth$ parameter was fixed to value 10 (if not specified otherwise) which promises a competitive precision and running time trade-off for comparisons with methods based on Z-curves and LSH. The $FilterRatio$ parameter was fixed to the values 0.01 or 0.05.

Please observe that the total k-NN join evaluation time for some lower parameter values is longer than for a bit higher values, e.g., $MaxRecDepth = 10$ and 20 for the $FilterRatio = 0.01$. A similar effect was observed also in our previous work by Cech et al. [2017]. We hypothesize that for a limited $FilterRatio$ there is a level of replication that has positive effect on efficient candidate processing on each executor, where metric filtering techniques are used (see Section 1.3.1). Please note that closer candidate/nearest objects to query points may appear in their group and so the actual ranges of k-NN queries get tighter. Hence, more candidates can be filtered out, which was revealed also by a lower number of evaluated distance computations.

For the Z-curve approach, the effect of the number of random vector shifts and the $EntirePartitions$ option (see Section 5.3.1) is presented in Figure 5.18. The experiments show that using more shifts improve approximation precision, but the running time can be extended significantly. The difference between the $EntirePartitions = true$ and $false$ options is substantial and offers a variety of the precision/running time trade-offs. For the following experiments, we usually fixed the number of shifts to 5 and used 20 partitions to fit the number of executors.

The Z-curve sampling rate parameter was set to 0.005 which influences mainly the partitions balance determining a proper level of parallelism.

Figure 5.19 displays the k-NN join evaluation time and shuffle memory usage for different levels of object storage settings described in subsection 5.3.1. While the "All in memory" option is clearly the faster option, for larger datasets it might not be viable (stored objects might not fit in memory). The "Only Z-order" approach translates Z-order back to the original coordinates and, thus, is considerably slower.

For the LSH approach (see Section 5.3.2) we investigate the effect of the number of hash tables, the number of hash functions and parameter W. Figure 5.20 shows that both precision and time increase with growing W (substantially for tested values $W > 20$). Longer running time for higher W values is mainly caused by hashing objects into bigger buckets (more objects have to be processed by the k-NN join in a large bucket). However, this parameter heavily depends on the specific dataset.

The effect of the number of hash tables and hash functions is presented in Figure 5.21 (for $W = 10$). With the growing number of hash functions both running time and precision drops because more hash bins are generated and queries do not meet all nearest objects in a bucket.

5.5.4 Approximate pivot methods comparison

In this subsection, we study different parameters and performance of both pivot-based approximate algorithms: PAKJ (the pivot approximate (heuristic) algorithm) and PEGKJ (the pivot epsilon guaranteed approach). All presented graphs were measured on 512D and 1000D datasets and for $K = 10$.

In the first four Figures 5.22, 5.23, 5.24, and 5.25, we analyze the influence of the PEGKJ parameter $ExactParentFiltering$ ($ExactPF$) on precision, k-NN join evaluation time and effective error for the growing ϵ parameter. Observe that the $ExactPF = true$ option is significantly superior in terms of precision but its running time is higher for increasing ϵ (less database objects are pruned and more distance computations are needed). Nevertheless, for $\epsilon = 5$ the precision of the variant $ExactPF = true$ is higher than the precision of the variant $ExactPF = false$ for $\epsilon = 2$, while their time is similar. Both the maximal and average effective error is also smaller for the $ExactPF = true$ parameter. Based on these observations, we conclude that once high ϵ is acceptable, the variant $ExactPF = true$ represents a better approximation option.

In Figures 5.26, 5.27, 5.28, and 5.29, we examine the same approximation measures for the growing $MaxRecDepth$ parameter. The $ExactPF$ parameter was set to $true$ for the PEAKJ method. From these graphs, it is evident that PAKJ runs considerably faster and presents only slightly lower precision. However, the epsilon guarantee (effective error) in the worst case (Figure 5.28) is violated more for the PAKJ algorithm. The average effective error corresponds to precision. In this case, PEAKJ presents lower average values.

In Figures 5.30, 5.31, 5.32, 5.33, 5.34 and 5.35, we compare different aspects of all approximate pivot-based methods. The $ExactPF$ parameter was set to $true$ for the PEGKJ method and experiments run on 512 and 1000-dimensional datasets. In Figures 5.30 and 5.31 you may observe precision/running time trade-off for all

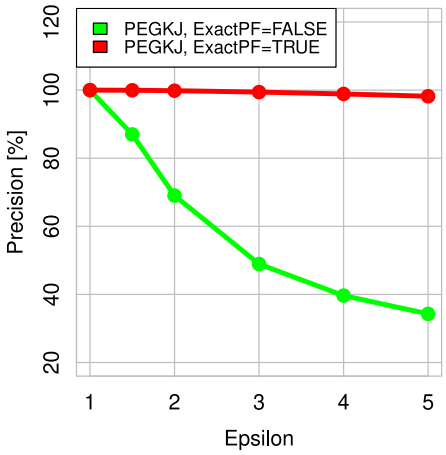


Figure 5.22: PEGKJ Exact parent filtering – Precision

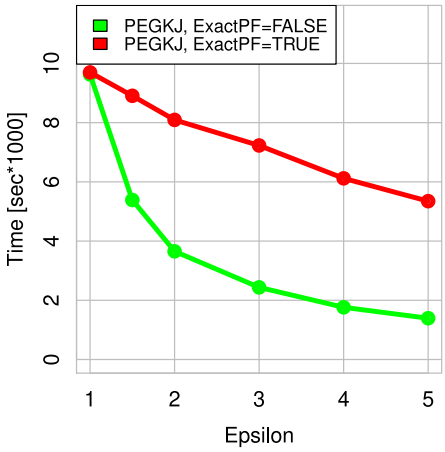


Figure 5.23: PEGKJ Exact parent filtering – Running time

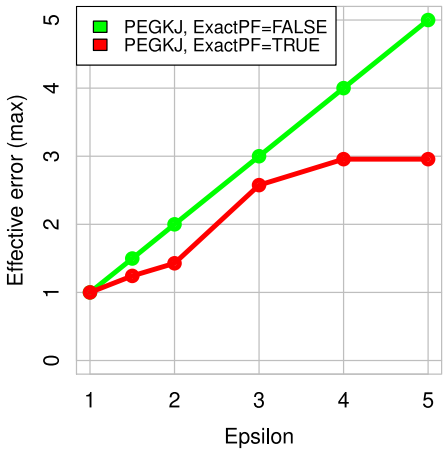


Figure 5.24: PEGKJ Exact parent filtering – Maximal Effective error

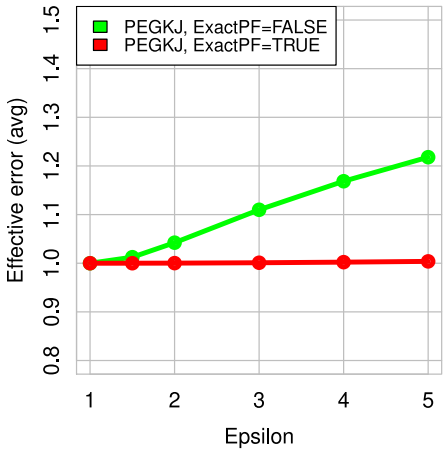


Figure 5.25: PEGKJ Exact parent filtering – Average Effective error

methods for different setup of parameters. To highlight one method, PAKJ with $Filter = 0.1$ is closing to 100% precision and is significantly faster than other variants/methods reaching high precision. In general, the PEGKJ algorithm is the slowest method and replicates objects (Figure 5.32) to almost all groups G (the reasons are explained in Section 5.4.4) but provides approximation guarantees (Figure 5.33). The replications directly affect transfered volumes of data (less is better), thus minimizing them enables a method to process larger datasets. In the last two graphs 5.34 and 5.35, we present cumulatively the number of query objects reaching an effective (real) epsilon error presented on the X-axis. Here the PEGKJ algorithm has the most objects with small effective error for lower $\epsilon = 4$, which means that more approximate k^{th} nearest neighbors from different queries are closer to exact results. Nevertheless, PAKJ methods are following PEGKJ results closely and PEGKJ for higher $\epsilon = 10$ is even worse for most queries than all presented PAKJ approaches on the 1000-dimensional dataset. Average and maximal effective errors for selected methods are presented in Table 5.5.

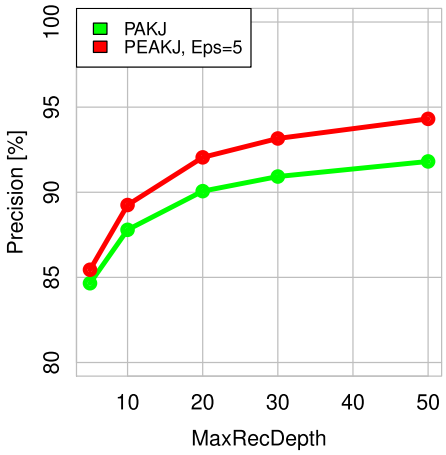


Figure 5.26: PAKJ, PEGKJ growing $MaxRecDepth$ – Precision

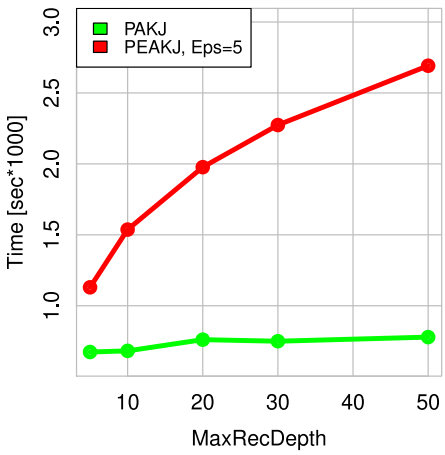


Figure 5.27: PAKJ, PEGKJ growing $MaxRecDepth$ – Running time

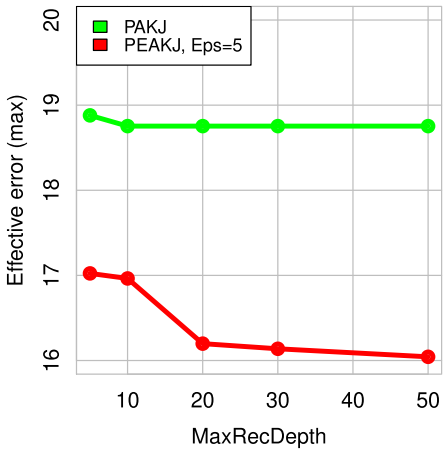


Figure 5.28: PAKJ, PEGKJ growing $MaxRecDepth$ – Maximal Effective error

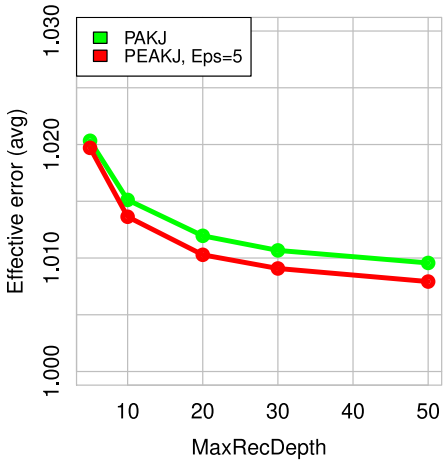


Figure 5.29: PAKJ, PEGKJ growing $MaxRecDepth$ – Average Effective error

	512D		1000D	
Method/variant	AVG	MAX	AVG	MAX
PAKJ, Filter=0.01, Depth=20	1.01329	1.95188	1.01095	6.23379
PAKJ, Filter=0.1, Depth=20	1.00400	1.81889	1.00681	6.21953
PAKJ, Filter=0.3, Depth=20	1.00371	1.81889	1.00615	6.21953
PEGKJ, EPF=true, Epsilon=4	1.00039	1.27834	1.00223	2.95698
PEGKJ, EPF=true, Epsilon=10	1.00735	1.85962	1.01325	4.48893

Table 5.5: Average and maximal effective (real) errors for pivot-based methods.

We conclude that all pivot-based algorithms perform well under different conditions and a specific algorithm utilization must be decided based on the desired approximation guarantee or an average approximation precision and running time preference.

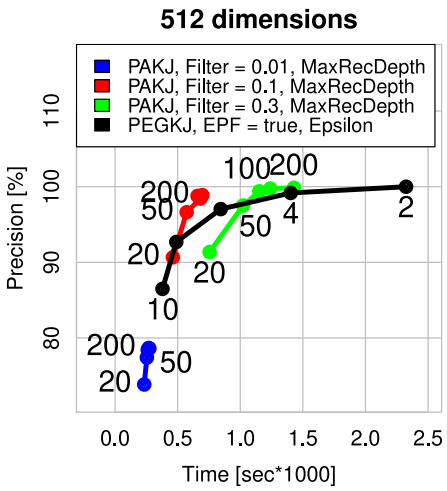


Figure 5.30: Pivot-based methods – Precision vs Time, 512D

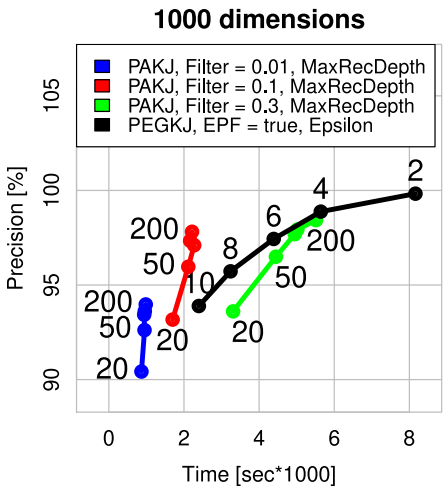


Figure 5.31: Pivot-based methods – Precis. vs Time, 1000D

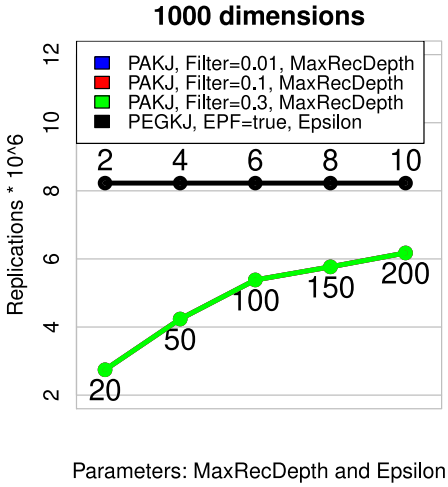


Figure 5.32: Pivot-based methods – Replications, 1000D

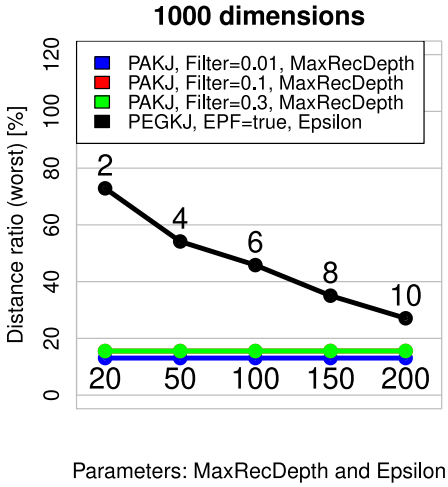


Figure 5.33: Pivot-based methods – Distance ratio, 1000D

5.5.5 Comparison of pivot-based approach with related approaches

In this section, we compare pivot-based, Z-curve and LSH k-NN approximate similarity join algorithms under several settings. Please note that all the methods use a convenient random initialization of data partitioning.

Precision and k-NN join evaluation time

First, we analyze the performance of all the compared methods on all datasets in Figures 5.36 and 5.37. Parameters were set as follows. PAKJ: $MaxRecDepth = 10$, $FilterRatio = 0.05$; Z-curve: $Shifts = 2$, $EntirePartitions = true$; LSH: W different for different datasets (20, 10, 200, 100), $HT = 5$, $HF = 20$. The results in Figure 5.36 show that for given settings the PAKJ approach has the highest precision, the Z-curve is the second best method on 200 and 1000-dimensional datasets and the LSH method displays the second best precision on the 10 and 512

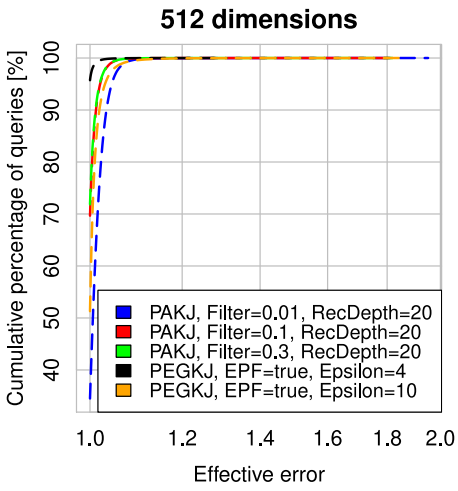


Figure 5.34: Pivot-based methods – cumul. Eff. error, 512D

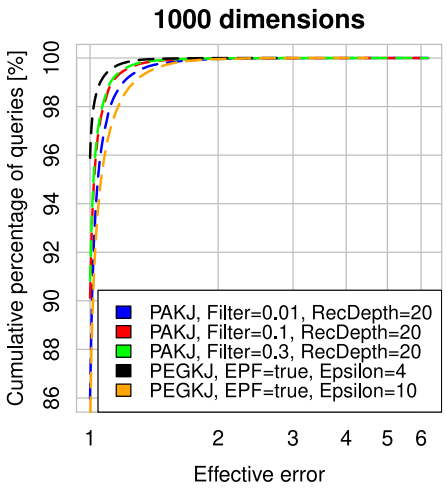


Figure 5.35: Pivot-based methods – cum. Eff. error, 1000D

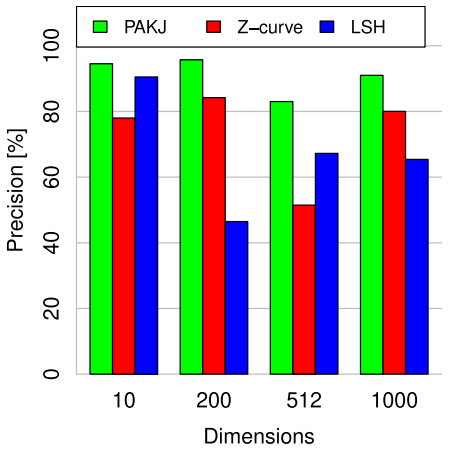


Figure 5.36: Precision for heuristic methods for all datasets

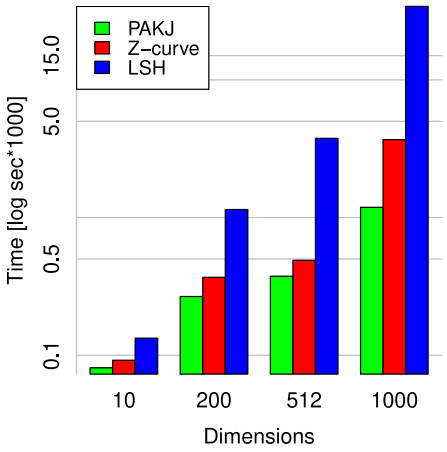


Figure 5.37: Running time for heuristic methods for all datasets

datasets. In Figure 5.37, we can observe that despite having the highest precision, the PAKJ method is also the fastest algorithm on all measured datasets. The second fastest method is the Z-curve algorithm and LSH is the approach with the longest execution time.

The effect of dimension size

In this subsection, we analyze the influence of the growing number of dimensions on the performance of three main heuristic methods with fixed parameters. Data were prepared from the 1000-dimensional dataset. For each testing scenario, first D dimensions were used for experiments, where $D \in \{100, 200, 300, 400, 500\}$. Parameters for methods were set in the following way. PAKJ: $MaxRecDepth = 10$, $FilterRatio = 0.01$; LSH: $W = 20$, $HT = 10$, $HF = 20$; Z-curve: $Shifts = 5$, $EntirePartitions = false$. You may observe in Figures 5.38 and 5.39 that the PAKJ method has the highest precision, the Z-curve approach is the fastest

and has similar precision compared to the LSH algorithm, which is the slowest method. Also, the LSH W parameter is very sensitive for a specific dataset. This observation is noticeable in Figure 5.39 where the total computation time changes significantly for different dimensions.

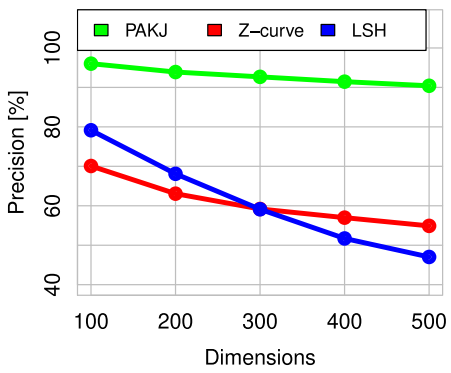


Figure 5.38: Growing dimensions – precision for all methods

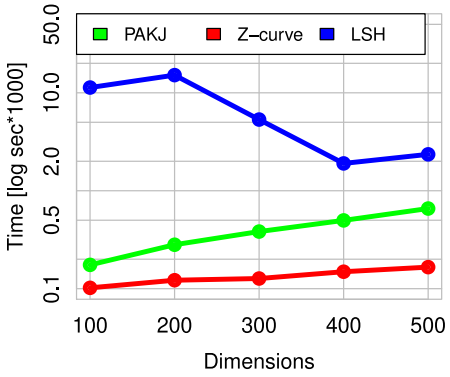


Figure 5.39: Growing dimensions – running time for all methods

The effect of dataset size

In Figures 5.40 and 5.41, we study the behavior of the compared methods for a significantly larger dataset. These tests run on the 200-dimensional dataset which consists of about 1.2 mil. database objects S and 15.3 mil. query objects R. With growing data size, we also increased the cluster size. Executors were set to {20, 40, 60, 80} and cluster size to {5, 10, 15, 20} instances (computing nodes) for objects count {4.1, 8.3, 12.4, 16.5} respectively. Despite our efforts to run experiments for all methods utilizing all objects (we even increased executor memory limits up to 4 GBs) not all methods were able to handle all test cases. The LSH algorithm ran out of memory for the highest object count 16.5 mil. For the Z-curve method, we had to set the memory saving option to the value "Only Z-order" and $EntirePartitions = false$ to satisfy the memory limits. The results show that the PAKJ method provides the highest precision while keeping reasonably fast running time, the Z-curve with fast evaluation time provides poor precision and the LSH method is both slow and has low precision.

The effect of the number of nearest neighbors

In many applications, the number of requested nearest neighbors k may reach one hundred or even more. Hence, we investigated also the effect of the increasing value of k on the precision and the similarity join evaluation time of the compared methods (see Figures 5.42 and 5.43). The experiments were performed on the 1000-dimensional dataset. The parameters of the methods were set as follows. PAKJ: $MaxRecDepth = 10$, $FilterRatio = 0.01$; Z-curve: $Shifts = 2$, $EntirePartitions = true$; LSH: $W = 50$, $HT = 10$, $HF = 20$. The precision slowly decreases for all methods, whereas the evaluation time is increasing for the PAKJ method, but for the other two methods, the evaluation time (already high) is not changing significantly. Similar to the previous graphs, the pivot space

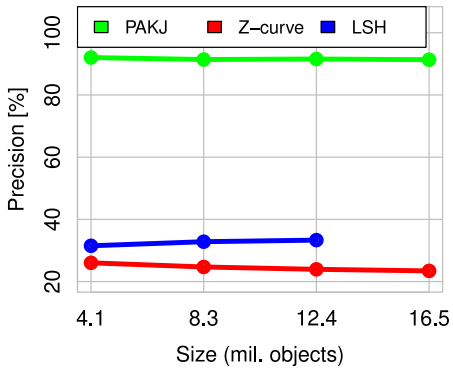


Figure 5.40: Growing dataset and cluster size – precision for all methods

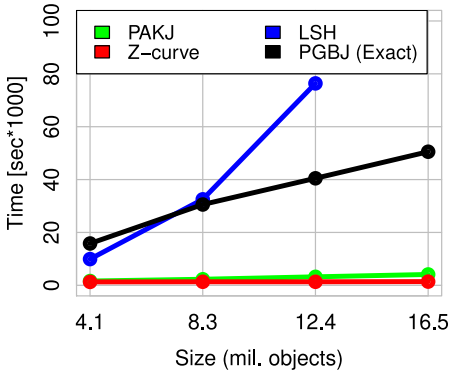


Figure 5.41: Growing dataset and cluster size – running time for all methods

approach outperforms the other two considered approaches in the precision/speed trade-off.

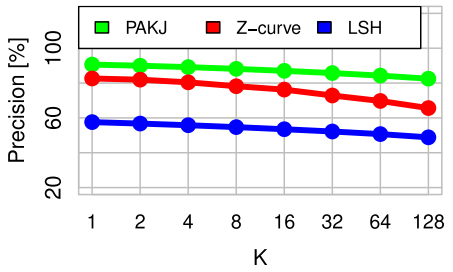


Figure 5.42: k-dependent computation – precision for all methods

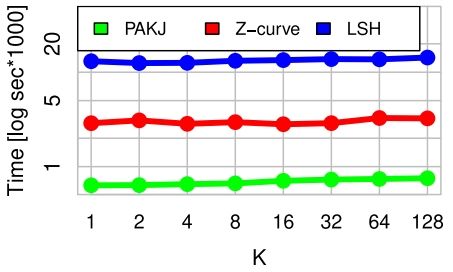


Figure 5.43: k-dependent computation – running time for all methods

5.5.6 Discussion

We analyzed the pivot-based approximate k-NN similarity join algorithms (mostly) in the Spark environment. We studied approximation quality and guarantees for pivot-based methods from the theoretical and experimental perspectives and presented two different pivot-based approximate k-NN similarity join algorithms. In experiments, we focused mainly on the approximation performance (precision, effective error, distance ratio) concerning the number of replications and the execution time of all algorithms. The PEGKJ algorithm presents a guaranteed ϵ-approximation but almost all objects are replicated to all groups on real data even with higher ϵ values, which results in longer execution times. With higher ϵ values, the approximation still provides a high precision for parameter $ExactPF = true$. The PAKJ algorithm runs faster and, in the average cases, produces the results with high precision and low effective error. However, a few PAKJ queries violate the ϵ-approximation significantly depending on a specific dataset. Nevertheless, the distance-based approach proved to be a promising approach for MapReduce-based approximate similarity joins in high-dimensional spaces.

We also compared in the Spark environment the pivot-based approximate method PAKJ with other heuristic methods designed for vector spaces – Z-curve and LSH. We focused on variants of the methods that use a simple random initialization of data partitioning structures. According to our evaluations with real data, the pivot-based approach PAKJ considering for replication a list of several nearest pivots for each database object outperformed the other two methods in the precision/efficiency trade-off. However, given just a synthetic 10-dimensional dataset the performance of the Z-curve and LSH based approaches was more competitive. We hypothesize that in high-dimensional spaces the selected variants of the related methods do not effectively reflect data distribution with random shifts or hash functions. This affects also options to finetune investigated parameters of the methods despite our effort to improve them. Hence, in the future, it would be interesting to investigate some cheap form of dataset analysis (pre-processing is part of the measured time) and use it for better initialization of the related methods.

Additional improvements to similarity k-NN joins could be achieved by implementing more sophisticated (but highly efficient) methods of space transformations and data partitioning (in our work, we focused on simple and fast random initialization methods). We would like to verify the influence of specific (trained) projections on the LSH method results, implement other types of space-filling curves, and try to control the curve's rotation direction. Moreover, we would like to evaluate the effect of managed pivot selection methods such as k-means clustering in more detail. Furthermore, restrictions on specific properties of given distance measures could also bring improvements in the form of additional saved distance computations and faster execution times.

Conclusion

In this thesis, we have studied searching, browsing, exploration and retrieval techniques for multimedia objects in multiple domains.

Beginning with preliminaries in Chapter 1, we revised the basics of similarity search and modeling, metric space postulates and methods for effective and efficient processing of similarity queries. Moreover, we showed several content-based feature and descriptor extraction examples mainly for images and we presented some well-known similarity functions that are often utilized for comparing such extracted descriptors. We debated the benefits and disadvantages of the descriptors and we also summarized basic principles of selected metric access methods (metric indexes).

Next, we have remembered retrieval/exploration strategies and challenges that researchers are currently dealing with, and we mentioned what the key components of effective and efficient content-based retrieval systems are in Chapter 2. We discussed various types of exploration structures as well as methods for arranging retrieved results in easy to read and comprehensive ways because the presentation layer of any retrieval application is essential to users too.

Furthermore, we have presented interactive prototypes and applications for image and video domain in Chapter 3. This area is concerning mainly the similarity between images since video content is typically converted to a series of images. We portrayed an array of demonstration applications and experiments for measuring effectiveness of proposed models and exploration structures. Experimental results, outcomes from user studies, and competition results brought interesting conclusions that pointed out strong and weak spots in the design and composition of used similarity models and interfaces.

In Chapter 4, we presented an approach for modeling and exploration of secured network traffic data that contain only a limited amount of information. We presented different descriptors for communication snapshots and studied methods for detecting malware activities utilizing such descriptors. Moreover, we demonstrated that such models can be used both for machine learning and exploration of large volumes of network data. Finally, we presented retrieval tools that can help domain experts with threat detection.

Since effective and efficient query evaluation needs for increasing data volumes are emerging in many domains, we proposed several algorithms working in the MapReduce distributed environment in Chapter 5. Such algorithms can process an extensive amount of data in parallel and can evaluate many queries altogether. We focused on similarity joins in metric spaces and studied various versions of them. Besides the exact similarity joins, we focused on approximate versions with or without guarantee as well. We experimentally investigated a wide variety of methods for distributed approximate similarity joins with different speed/precision trade-offs. On top of that, we publicly share source codes of all algorithms on Github[5].

Altogether, our demo tools and applications have analyzed the performance of various similarity models and exploration structures on real data in multiple domains. We focused on precision, recall, and success rate of different methods and

[5]https://github.com/PremyslCech/kNN-joins-spark

detected strong and weak points of different components of our multimedia retrieval frameworks. Based on the observed needs from all the investigated applications and areas, we developed and proposed scalable algorithms for evaluating (approximate) similarity joins in a distributed environment that could bring many similarity models for data analysis from small collections to a big data environment.

Future work

Many retrieval tasks in various domains still require more sophisticated and flexible similarity models that capture the content of multimedia objects. For example, approaches dealing with one-shot learning could bring significant improvements in terms of precision and recall of retrieval systems. Next, we would like to focus on the development of more convenient interactive search mechanisms and visualization methods, especially for the challenging video retrieval domain. From our experience from international competitions and user's feedback, it was evident that poor interface design can decrease the success rate of difficult video retrieval tasks. Finally, we want to focus on further improvements of distributed similarity joins and new applications of k-NN similarity graphs in the area of multimedia exploration.

Particularly, we plan to utilize similarity search techniques and large-scale computation algorithms in a new area concerning the detection of advertisement and consumer behavior in video content recorded by smart glasses. Employing our methods, we aim to survey media consumption (e.g., which newspapers people read, what they watch), success of advertisement campaigns (e.g., detect banners, spots or billboards that reached our respondents) and people's interests in specific products (e.g., what grocery stores are visited more frequently and what kind of products draw people's attention).

Bibliography

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In Kimberly Keeton and Timothy Roscoe, editors, *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, pages 265–283. USENIX Association, 2016. URL `https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi`.

Giuseppe Amato, Paolo Bolettieri, Fabio Carrara, Franca Debole, Fabrizio Falchi, Claudio Gennaro, Lucia Vadicamo, and Claudio Vairo. VISIONE at VBS2019. In *MultiMedia Modeling - 25th International Conference, MMM 2019, Thessaloniki, Greece, January 8-11, 2019, Proceedings, Part II*, pages 591–596, 2019. doi: 10.1007/978-3-030-05716-9_51.

Jaume Amores. Multiple instance classification: Review, taxonomy and comparative study. *Artif. Intell.*, 201:81–105, 2013. doi: 10.1016/j.artint.2013.06.003. URL `https://doi.org/10.1016/j.artint.2013.06.003`.

Stelios Andreadis, Anastasia Moumtzidou, Damianos Galanopoulos, Foteini Markatopoulou, Konstantinos Apostolidis, Thanassis Mavropoulos, Ilias Gialampoukidis, Stefanos Vrochidis, Vasileios Mezaris, Ioannis Kompatsiaris, and Ioannis Patras. VERGE in VBS 2019. In *MultiMedia Modeling - 25th International Conference, MMM 2019, Thessaloniki, Greece, January 8-11, 2019, Proceedings, Part II*, pages 602–608, 2019. doi: 10.1007/978-3-030-05716-9_53.

Fabrizio Angiulli and Clara Pizzuti. Outlier mining in large high-dimensional data sets. *IEEE Trans. Knowl. Data Eng.*, 17(2):203–215, 2005. doi: 10.1109/TKDE.2005.31. URL `https://doi.org/10.1109/TKDE.2005.31`.

Pradeep K. Atrey, M. Anwar Hossain, Abdulmotaleb El-Saddik, and Mohan S. Kankanhalli. Multimodal fusion for multimedia analysis: a survey. *Multimedia Syst.*, 16(6):345–379, 2010.

Franz Aurenhammer. Voronoi diagrams - A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, 1991.

George Awad, Jonathan Fiscus, Martial Michel, David Joy, Wessel Kraaij, Alan F. Smeaton, Georges Quénot, Maria Eskevich, Robin Aly, Gareth J. F. Jones, Roeland Ordelman, Benoit Huet, and Martha Larson. Trecvid 2016: Evaluating video search, video event detection, localization, and hyperlinking. In *Proceedings of TRECVID 2016*. NIST, USA, 2016.

Kai Uwe Barthel and Nico Hezel. Visually exploring millions of images using image maps and graphs. In Benoit Huet, Stefanos Vrochidis, and Edward Chang, editors, *Big Data Analytics for Large-scale Multimedia Search*, pages 251–275. John Wiley and Sons Inc., 2019. ISBN 978-1-119-37697-2.

Kai Uwe Barthel, Nico Hezel, and Radek Mackowiak. Graph-based browsing for large video collections. In Xiangjian He, Suhuai Luo, Dacheng Tao, Changsheng Xu, Jie Yang, and Muhammad Abul Hasan, editors, *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 237–242. Springer, 2015a. doi: 10.1007/978-3-319-14442-9_21. URL `https://doi.org/10.1007/978-3-319-14442-9_21`.

Kai Uwe Barthel, Nico Hezel, and Radek Mackowiak. Imagemap - visually browsing millions of images. In Xiangjian He, Suhuai Luo, Dacheng Tao, Changsheng Xu, Jie Yang, and Muhammad Abul Hasan, editors, *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 287–290. Springer, 2015b. doi: 10.1007/978-3-319-14442-9_30. URL `https://doi.org/10.1007/978-3-319-14442-9_30`.

Herbert Bay, Tinne Tuytelaars, and Luc J. Van Gool. SURF: speeded up robust features. In *ECCV (1)*, volume 3951 of *Lecture Notes in Computer Science*, pages 404–417. Springer, 2006.

Christian Beecks, Merih Seran Uysal, and Thomas Seidl. Signature quadratic form distances for content-based similarity. In *ACM Multimedia*, pages 697–700. ACM, 2009.

Christian Beecks, Jakub Lokoc, Thomas Seidl, and Tomás Skopal. Indexing the signature quadratic form distance for efficient content-based multimedia retrieval. In *ICMR*, page 24. ACM, 2011a.

Christian Beecks, Thomas Skopal, Klaus Schoeffmann, and Thomas Seidl. Towards large-scale multimedia exploration. In Gautam Das, Vagelis Hsristidis, and Ihab Ilyas, editors, *Proceedings of the 5th International Workshop on Ranking in Databases (DBRank 2011)*, pages 31–33, Seattle, WA, USA, aug 2011b. VLDB.

Christian Beecks, Merih Seran Uysal, Philip Driessen, and Thomas Seidl. Content-based exploration of multimedia databases. In *11th International Workshop on Content-Based Multimedia Indexing, CBMI 2013, Veszprém, Hungary, June 17-19, 2013*, pages 59–64. IEEE, 2013. doi: 10.1109/CBMI.2013.6576553. URL `https://doi.org/10.1109/CBMI.2013.6576553`.

Philip Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci.*, 337(1-3):217–239, 2005.

Christian Böhm and Florian Krebs. The k-nearest neighbour join: Turbo charging the KDD process. *Knowl. Inf. Syst.*, 6(6):728–749, 2004. URL `http://www.springerlink.com/index/10.1007/s10115-003-0122-9`.

Christian Böhm, Bernhard Braunmüller, Florian Krebs, and Hans-Peter Kriegel. Epsilon grid order: An algorithm for the similarity join on massive high-dimensional data. In Sharad Mehrotra and Timos K. Sellis, editors, *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*, pages 379–388. ACM, 2001. doi: 10.1145/375663.375714. URL `http://doi.acm.org/10.1145/375663.375714`.

Petra Budíková, Michal Batko, and Pavel Zezula. Evaluation platform for content-based image retrieval systems. In *TPDL*, volume 6966 of *Lecture Notes in Computer Science*, pages 130–142. Springer, 2011.

Petra Budíková, Michal Batko, and Pavel Zezula. Fusion strategies for large-scale multi-modal image retrieval. *T. Large-Scale Data- and Knowledge-Centered Systems*, 33:146–184, 2017.

Benjamin Bustos, Gonzalo Navarro, and Edgar Chávez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters*, 24(14): 2357–2366, 2003. doi: 10.1016/S0167-8655(03)00065-5. URL `https://doi.org/10.1016/S0167-8655(03)00065-5`.

Iain Campbell. *The ostensive model of developing information needs*. PhD thesis, Citeseer, 2000.

Iain Campbell and Cornelis J van Rijsbergen. The ostensive model of developing information needs. In *Proceedings of the 3rd international conference on conceptions of library and information science*, pages 251–268, 1996.

Přemysl Čech. Using metric indexes for effective and efficient multimedia exploration. *Charles University*, 2014.

Premysl Cech and Tomás Grosup. Comparison of metric space browsing strategies for efficient image exploration. In *13th International Workshop on Content-Based Multimedia Indexing, CBMI 2015, Prague, Czech Republic, June 10-12, 2015*, pages 1–6. IEEE, 2015. doi: 10.1109/CBMI.2015.7153631. URL `https://doi.org/10.1109/CBMI.2015.7153631`.

Premysl Cech, Jan Kohout, Jakub Lokoc, Tomás Komárek, Jakub Marousek, and Tomás Pevný. Feature extraction and malware detection on large HTTPS data using mapreduce. In Laurent Amsaleg, Michael E. Houle, and Erich Schubert, editors, *Similarity Search and Applications - 9th International Conference, SISAP 2016, Tokyo, Japan, October 24-26, 2016. Proceedings*, volume 9939 of *Lecture Notes in Computer Science*, pages 311–324, 2016. ISBN 978-3-319-46758-0. doi: 10.1007/978-3-319-46759-7_24. URL `https://doi.org/10.1007/978-3-319-46759-7_24`.

Premysl Cech, Jakub Marousek, Jakub Lokoc, Yasin N. Silva, and Jeremy Starks. Comparing mapreduce-based k-nn similarity joins on hadoop for high-dimensional data. In Gao Cong, Wen-Chih Peng, Wei Emma Zhang, Chengliang Li, and Aixin Sun, editors, *Advanced Data Mining and Applications - 13th International Conference, ADMA 2017, Singapore, November 5-6, 2017, Proceedings*, volume 10604 of *Lecture Notes in Computer Science*, pages 63–75. Springer, 2017. ISBN 978-3-319-69178-7. doi: 10.1007/978-3-319-69179-4_5. URL `https://doi.org/10.1007/978-3-319-69179-4_5`.

Premysl Cech, Jakub Lokoc, and Yasin N. Silva. Pivot-based approximate k-nn similarity joins for big high-dimensional data. *Information Systems*, 87:101410, 2020. ISSN 0306-4379. doi: https://doi.org/10.1016/j.is.2019.06.006. URL `http://www.sciencedirect.com/science/article/pii/S0306437918302473`.

Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. A primitive operator for similarity joins in data cleaning. In Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang, editors, *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 5. IEEE Computer Society, 2006. doi: 10.1109/ICDE.2006.9. URL `https://doi.org/10.1109/ICDE.2006.9`.

Edgar Chávez, J Marroquín, and Gonzalo Navarro. Overcoming the curse of dimensionality. In *European Workshop on Content-based Multimedia Indexing (CBMI'99)*, pages 57–64, 1999.

Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, September 2001. ISSN 0360-0300. doi: 10.1145/502807.502808. URL `http://doi.acm.org/10.1145/502807.502808`.

Edgar Chavez Gonzalez, Karina Figueroa, and Gonzalo Navarro. Effective proximity retrieval by ordering permutations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(9):1647–1658, September 2008. ISSN 0162-8828. doi: 10.1109/TPAMI.2007.70815.

Lu Chen, Yunjun Gao, Xinhan Li, Christian S. Jensen, and Gang Chen. Efficient metric indexing for similarity search and similarity joins. *IEEE Trans. Knowl. Data Eng.*, 29(3):556–571, 2017.

Paolo Ciaccia and Marco Patella. PAC nearest neighbor queries: Using the distance distribution for searching in high-dimensional metric spaces. In Elisa Bertino and Silvana Castano, editors, *Atti del Settimo Convegno Nazionale Sistemi Evoluti per Basi di Dati, SEBD 1999, Villa Olmo, Como, Italy, 23-25 Giugno 1999*, pages 259–273, 1999.

Paolo Ciaccia and Marco Patella. PAC nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In David B. Lomet and Gerhard Weikum, editors, *Proceedings of the 16th International Conference on Data Engineering, San Diego, California, USA, February 28 - March 3, 2000*, pages 244–255. IEEE Computer Society, 2000. doi: 10.1109/ICDE.2000.839417. URL `https://doi.org/10.1109/ICDE.2000.839417`.

Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, pages 426–435. Morgan Kaufmann, 1997.

Claudiu Cobârzan, Klaus Schoeffmann, Werner Bailer, Wolfgang Hürst, Adam Blazek, Jakub Lokoc, Stefanos Vrochidis, Kai Uwe Barthel, and Luca Rossetto. Interactive video search tools: a detailed analysis of the video browser showdown 2015. *Multimedia Tools Appl.*, 76(4):5539–5571, 2017. doi: 10.1007/s11042-016-3661-2. URL `https://doi.org/10.1007/s11042-016-3661-2`.

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, pages 253–262, New York, NY, USA, 2004. ACM. ISBN 1-58113-885-7.

Ritendra Datta, Dhiraj Joshi, Jia Li, and James Z Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys (Csur)*, 40(2): 5, 2008.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327492.

Vincent Delaitre, Ivan Laptev, and Josef Sivic. Recognizing human actions in still images: a study of bag-of-features and part-based representations. In Frédéric Labrosse, Reyer Zwiggelaar, Yonghuai Liu, and Bernie Tiddeman, editors, *British Machine Vision Conference, BMVC 2010, Aberystwyth, UK, August 31 - September 3, 2010. Proceedings*, pages 1–11. British Machine Vision Association, 2010. doi: 10.5244/C.24.97. URL `https://doi.org/10.5244/C.24.97`.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255. IEEE Computer Society, 2009.

Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 647–655. JMLR.org, 2014.

Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern classification, 2nd Edition*. Wiley, 2001. ISBN 9780471056690. URL `http://www.worldcat.org/oclc/41347061`.

H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. El Abbadi. Approximate nearest neighbor searching in multimedia databases. In *Proceedings 17th International Conference on Data Engineering*, pages 503–511, 2001. doi: 10.1109/ICDE.2001.914864.

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616, Hypertext Transfer Protocol – HTTP/1.1, 1999. URL `http://www.rfc.net/rfc2616.html`.

Raphael A. Finkel and Jon Louis Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974. doi: 10.1007/BF00288933. URL `https://doi.org/10.1007/BF00288933`.

Giorgio Giacinto. A nearest-neighbor approach to relevance feedback in content based image retrieval. In *Proceedings of the 6th ACM International Conference on Image and Video Retrieval*, CIVR '07, pages 456–463, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-733-9. doi: 10.1145/1282280.1282347.

Michael Grabner, Helmut Grabner, and Horst Bischof. Fast approximated SIFT. In P. J. Narayanan, Shree K. Nayar, and Heung-Yeung Shum, editors, *Computer Vision - ACCV 2006, 7th Asian Conference on Computer Vision, Hyderabad, India, January 13-16, 2006, Proceedings, Part I*, volume 3851 of *Lecture Notes in Computer Science*, pages 918–927. Springer, 2006. doi: 10.1007/11612032_92. URL `https://doi.org/10.1007/11612032_92`.

Tomás Grosup, Premysl Cech, Jakub Lokoc, and Tomás Skopal. A web portal for effective multi-model exploration. In Xiangjian He, Suhuai Luo, Dacheng Tao, Changsheng Xu, Jie Yang, and Muhammad Abul Hasan, editors, *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 315–318. Springer, 2015a. ISBN 978-3-319-14441-2. doi: 10.1007/978-3-319-14442-9_37. URL https://doi.org/10.1007/978-3-319-14442-9_37.

Tomás Grosup, Juraj Mosko, and Premysl Cech. Continuous hierarchical exploration of multimedia collections. In *13th International Workshop on Content-Based Multimedia Indexing, CBMI 2015, Prague, Czech Republic, June 10-12, 2015*, pages 1–4. IEEE, 2015b. doi: 10.1109/CBMI.2015.7153621. URL https://doi.org/10.1109/CBMI.2015.7153621.

Sudipto Guha, H. V. Jagadish, Nick Koudas, Divesh Srivastava, and Ting Yu. Approximate XML joins. In *SIGMOD Conference*, pages 287–298. ACM, 2002.

Gylfi Þór Guðmundsson, Laurent Amsaleg, Björn Þór Jónsson, and Michael J. Franklin. Towards engineering a web-scale multimedia service: A case study using spark. In *Proceedings of the 8th ACM on Multimedia Systems Conference, MMSys 2017, Taipei, Taiwan, June 20-23, 2017*, pages 1–12, 2017. doi: 10.1145/3083187.3083200.

Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In Beatrice Yormark, editor, *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, USA, June 18-21, 1984*, pages 47–57. ACM Press, 1984. doi: 10.1145/602259.602266. URL https://doi.org/10.1145/602259.602266.

Michael Gygli. Ridiculously fast shot boundary detection with fully convolutional neural networks. In *2018 International Conference on Content-Based Multimedia Indexing, CBMI 2018, La Rochelle, France, September 4-6, 2018*, pages 1–4, 2018. doi: 10.1109/CBMI.2018.8516556.

Samantha Kelly Hastings. Evaluation of image retrieval systems: Role of user feedback. *Library Trends*, 48(2), 1999. URL http://alexia.lis.uiuc.edu/puboff/catalog/trends/48_2abs.html#hastings.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Daniel Heesch. A survey of browsing models for content based image retrieval. *Multimedia Tools Appl.*, 40(2):261–284, 2008.

Gísli R. Hjaltason and Hanan Samet. Incremental distance join algorithms for spatial databases. In Laura M. Haas and Ashutosh Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA.*, pages 237–248. ACM Press, 1998. doi: 10.1145/276304.276326. URL http://doi.acm.org/10.1145/276304.276326.

Gísli R. Hjaltason and Hanan Samet. Distance browsing in spatial databases. *ACM Trans. Database Syst.*, 24(2):265–318, 1999. doi: 10.1145/320248.320255. URL http://doi.acm.org/10.1145/320248.320255.

Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6):417, 1933.

Peiyun Hu and Deva Ramanan. Finding tiny faces. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 1522–1530. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.166. URL https://doi.org/10.1109/CVPR.2017.166.

Weiming Hu, Nianhua Xie, Li Li, Xianglin Zeng, and Stephen J. Maybank. A survey on visual content-based video indexing and retrieval. *IEEE Trans. Systems, Man, and Cybernetics, Part C*, 41(6):797–819, 2011. doi: 10.1109/TSMCC.2011.2109710. URL https://doi.org/10.1109/TSMCC.2011.2109710.

Yupeng Hu, Chong Yang, Cun Ji, Yang Xu, and Xueqing Li. Efficient snapshot KNN join processing for large data using mapreduce. In *22nd IEEE International Conference on Parallel and Distributed Systems, ICPADS 2016, Wuhan, China, December 13-16, 2016*, pages 713–720. IEEE Computer Society, 2016. doi: 10.1109/ICPADS.2016.0098. URL https://doi.org/10.1109/ICPADS.2016.0098.

Edwin H. Jacox and Hanan Samet. Metric space similarity joins. *ACM Trans. Database Syst.*, 33(2):7:1–7:38, 2008. doi: 10.1145/1366102.1366104. URL http://doi.acm.org/10.1145/1366102.1366104.

Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Hamming embedding and weak geometric consistency for large scale image search. In David A. Forsyth, Philip H. S. Torr, and Andrew Zisserman, editors, *Computer Vision - ECCV 2008, 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part I*, volume 5302 of *Lecture Notes in Computer Science*, pages 304–317. Springer, 2008. doi: 10.1007/978-3-540-88682-2_24. URL https://doi.org/10.1007/978-3-540-88682-2_24.

Björn Þór Jónsson, Marcel Worring, Jan Zahálka, Stevan Rudinac, and Laurent Amsaleg. Ten research questions for scalable multimedia analytics. In Qi Tian, Nicu Sebe, Guo-Jun Qi, Benoit Huet, Richang Hong, and Xueliang Liu, editors, *MultiMedia Modeling - 22nd International Conference, MMM 2016, Miami, FL, USA, January 4-6, 2016, Proceedings, Part II*, volume 9517 of *Lecture Notes in Computer Science*, pages 290–302. Springer, 2016. doi: 10.1007/978-3-319-27674-8_26. URL https://doi.org/10.1007/978-3-319-27674-8_26.

M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015. ISSN 0036-8075. doi: 10.1126/science.aaa8415. URL https://science.sciencemag.org/content/349/6245/255.

Dhiraj Joshi and James Ze Wang. Multimedia systems and content-based image retrieval. by sagarmay deb, idea group publishing, 2004, $79.95 ISBN 1-59140-156-9. *Inf. Process. Manage.*, 41(2):406–407, 2005.

Jayanthi Karuppusamy and Karthikeyan Marappan. Efficient color and texture feature extraction technique for content based image retrieval system. *Int. Arab J. Inf. Technol.*, 13(6A):784–790, 2016.

Wooyeol Kim, Younghoon Kim, and Kyuseok Shim. Parallel computation of k-nearest neighbor joins using mapreduce. In James Joshi, George Karypis, Ling Liu, Xiaohua Hu, Ronay Ak, Yinglong Xia, Weijia Xu, Aki-Hiro Sato, Sudarsan Rachuri, Lyle H. Ungar, Philip S. Yu, Rama Govindaraju, and Toyotaro Suzumura, editors, *2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016*, pages 696–705. IEEE, 2016. doi: 10.1109/BigData.2016.7840662. URL `https://doi.org/10.1109/BigData.2016.7840662`.

Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

J. Kohout and T. Pevny. Automatic discovery of web servers hosting similar applications. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, 2015a.

Jan Kohout and Tomas Pevny. Unsupervised detection of malware in persistent web traffic. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, 2015b.

Jan Kohout, Tomás Komárek, Premysl Cech, Jan Bodnár, and Jakub Lokoc. Learning communication patterns for malware discovery in https data. *Expert Syst. Appl.*, 101:129–142, 2018. doi: 10.1016/j.eswa.2018.02.010. URL `https://doi.org/10.1016/j.eswa.2018.02.010`.

Miroslav Kratochvil, Abhishek Koladiya, Jana Balounova, Vendula Novosadova, Radislav Sedlacek, Karel Fišer, Jiří Vondrášek, and Karel Drbal. Som-based embedding improves efficiency of high-dimensional cytometry data analysis. *bioRxiv*, page 496869, 2019.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, 2017.

Martin Krulis, Jakub Lokoc, and Tomás Skopal. Efficient extraction of clustering-based feature signatures using GPU architectures. *Multimedia Tools Appl.*, 75 (13):8071–8103, 2016. doi: 10.1007/s11042-015-2726-y. URL `https://doi.org/10.1007/s11042-015-2726-y`.

Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845—848, 1965.

Michael S. Lew, Nicu Sebe, Chabane Djeraba, and Ramesh Jain. Content-based multimedia information retrieval: State of the art and challenges. *TOMCCAP*, 2(1):1–19, 2006. doi: 10.1145/1126004.1126005. URL `https://doi.org/10.1145/1126004.1126005`.

Jakub Lokoc, Premysl Cech, Jiri Novák, and Tomás Skopal. Cut-region: A compact building block for hierarchical metric indexing. In Gonzalo Navarro and Vladimir Pestov, editors, *Similarity Search and Applications - 5th International Conference, SISAP 2012, Toronto, ON, Canada, August 9-10, 2012. Proceedings*, volume 7404 of *Lecture Notes in Computer Science*, pages 85–100. Springer, 2012. ISBN 978-3-642-32152-8. doi: 10.1007/978-3-642-32153-5_7. URL https://doi.org/10.1007/978-3-642-32153-5_7.

Jakub Lokoc, Tomás Grosup, Premysl Cech, and Tomás Skopal. Towards efficient multimedia exploration using the metric space approach. In *12th International Workshop on Content-Based Multimedia Indexing, CBMI 2014, Klagenfurt, Austria, June 18-20, 2014*, pages 1–4. IEEE, 2014a. ISBN 978-1-4799-3990-9. doi: 10.1109/CBMI.2014.6849851. URL https://doi.org/10.1109/CBMI.2014.6849851.

Jakub Lokoc, Juraj Mosko, Premysl Cech, and Tomás Skopal. On indexing metric spaces using cut-regions. *Inf. Syst.*, 43:1–19, 2014b. doi: 10.1016/j.is.2014.01.007. URL https://doi.org/10.1016/j.is.2014.01.007.

Jakub Lokoc, Jan Kohout, Premysl Cech, Tomás Skopal, and Tomás Pevný. k-nn classification of malware in HTTPS traffic using the metric space approach. In Michael Chau, G. Alan Wang, and Hsinchun Chen, editors, *Intelligence and Security Informatics - 11th Pacific Asia Workshop, PAISI 2016, Auckland, New Zealand, April 19, 2016, Proceedings*, volume 9650 of *Lecture Notes in Computer Science*, pages 131–145. Springer, 2016. ISBN 978-3-319-31862-2. doi: 10.1007/978-3-319-31863-9_10. URL https://doi.org/10.1007/978-3-319-31863-9_10.

Jakub Lokoc, Tomás Grosup, Premysl Cech, Tomás Pevný, and Tomás Skopal. Malware discovery using behaviour-based exploration of network traffic. In Christian Beecks, Felix Borutta, Peer Kröger, and Thomas Seidl, editors, *Similarity Search and Applications - 10th International Conference, SISAP 2017, Munich, Germany, October 4-6, 2017, Proceedings*, volume 10609 of *Lecture Notes in Computer Science*, pages 315–323. Springer, 2017. ISBN 978-3-319-68473-4. doi: 10.1007/978-3-319-68474-1_22. URL https://doi.org/10.1007/978-3-319-68474-1_22.

Jakub Lokoc, Werner Bailer, Klaus Schoeffmann, Bernd Münzer, and George Awad. On influential trends in interactive video retrieval: Video browser showdown 2015-2017. *IEEE Trans. Multimedia*, 20(12):3361–3376, 2018.

Jakub Lokoč, Gregor Kovalčík, Bernd Münzer, Klaus Schöffmann, Werner Bailer, Ralph Gasser, Stefanos Vrochidis, Phuong Anh Nguyen, Sitapa Rujikietgumjorn, and Kai Uwe Barthel. Interactive search or sequential browsing? a detailed analysis of the video browser showdown 2018. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 15(1):29, 2019.

Jakub Lokoc, Gregor Kovalcík, Tomás Soucek, Jaroslav Moravec, Jan Bodnár, and Premysl Cech. VIRET tool meets nasnet. In Ioannis Kompatsiaris, Benoit Huet, Vasileios Mezaris, Cathal Gurrin, Wen-Huang Cheng, and Stefanos Vrochidis, editors, *MultiMedia Modeling - 25th International Conference, MMM*

2019, Thessaloniki, Greece, January 8-11, 2019, Proceedings, Part II, volume 11296 of *Lecture Notes in Computer Science*, pages 597–601. Springer, 2019a. ISBN 978-3-030-05715-2. doi: 10.1007/978-3-030-05716-9_52. URL https://doi.org/10.1007/978-3-030-05716-9_52.

Jakub Lokoc, Gregor Kovalcík, Tomás Soucek, Jaroslav Moravec, and Premysl Cech. VIRET: A video retrieval tool for interactive known-item search. In Abdulmotaleb El-Saddik, Alberto Del Bimbo, Zhongfei Zhang, Alexander G. Hauptmann, K. Selçuk Candan, Marco Bertini, Lexing Xie, and Xiao-Yong Wei, editors, *Proceedings of the 2019 on International Conference on Multimedia Retrieval, ICMR 2019, Ottawa, ON, Canada, June 10-13, 2019.*, pages 177–181. ACM, 2019b. doi: 10.1145/3323873.3325034. URL https://doi.org/10.1145/3323873.3325034.

Jakub Lokoc, Tomás Soucek, Premysl Cech, and Gregor Kovalcík. Enhanced VIRET tool for lifelog data. In Cathal Gurrin, Klaus Schöffmann, Hideo Joho, Duc-Tien Dang-Nguyen, Michael Riegler, and Luca Piras, editors, *Proceedings of the ACM Workshop on Lifelog Search Challenge, LSC@ICMR 2019, Ottawa, ON, Canada, 10 June 2019.*, pages 25–26. ACM, 2019c. doi: 10.1145/3326460.3329159. URL https://doi.org/10.1145/3326460.3329159.

Jakub Lokoč, Gregor Kovalčík, Tomáš Souček, Jaroslav Moravec, and Přemysl Čech. A framework for effective known-item search in video. In *Proceedings of the 27th ACM International Conference on Multimedia (MM'19), October 21–25, 2019, Nice, France*, ACM, New York, NY, USA, pages 1–9. ACM, 2019. doi: 10.1145/3343031.3351046. URL https://doi.org/10.1145/3343031.3351046.

David G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, pages 1150–1157, 1999.

David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

Wei Lu, Yanyan Shen, Su Chen, and Beng Chin Ooi. Efficient processing of k nearest neighbor joins using mapreduce. *Proc. VLDB Endow.*, 5(10):1016–1027, June 2012. ISSN 2150-8097. doi: 10.14778/2336664.2336674.

Youzhong Ma, Xiaofeng Meng, and Shaoya Wang. Parallel similarity joins on massive high-dimensional data using mapreduce. *Concurrency and Computation: Practice and Experience*, 28(1):166–183, 2016. doi: 10.1002/cpe.3663. URL https://doi.org/10.1002/cpe.3663.

Youzhong Ma, Shijie Jia, and Yongxin Zhang. A novel approach for high-dimensional vector similarity join query. *Concurrency and Computation: Practice and Experience*, 29(5), 2017. doi: 10.1002/cpe.3952. URL https://doi.org/10.1002/cpe.3952.

Miroslav Macik, Jakub Lokoc, Premysl Cech, and Tomás Skopal. Particle physics model for content-based 3d exploration. In *14th International Workshop on Content-Based Multimedia Indexing, CBMI 2016, Bucharest, Romania, June 15-17, 2016*, pages 1–6. IEEE, 2016. ISBN 978-1-4673-8695-1. doi: 10.1109/CBMI.2016.7500259. URL https://doi.org/10.1109/CBMI.2016.7500259.

Jean-Michel Marin, Kerrie Mengersen, and Christian P. Robert. Bayesian modelling and inference on mixtures of distributions. In D.K. Dey and C.R. Rao, editors, *Bayesian ThinkingModeling and Computation*, volume 25 of *Handbook of Statistics*, pages 459 – 507. Elsevier, 2005.

Diana Moise, Denis Shestakov, Gylfi Þór Gudmundsson, and Laurent Amsaleg. Terabyte-scale image similarity search: Experience and best practice. In *Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, Santa Clara, CA, USA*, pages 674–682, 2013a. doi: 10.1109/BigData.2013.6691637.

Diana Moise, Denis Shestakov, Gylfi Þór Gudmundsson, and Laurent Amsaleg. Indexing and searching 100m images with map-reduce. In *International Conference on Multimedia Retrieval, ICMR'13, Dallas, TX, USA, April 16-19, 2013*, pages 17–24, 2013b. doi: 10.1145/2461466.2461470.

Juraj Moško. Exploration of multimedia collections. *Charles University*, 2016.

Juraj Mosko, Jakub Lokoc, Tomás Grosup, Premysl Cech, Tomás Skopal, and Jan Lansky. MLES: multilayer exploration structure for multimedia exploration. In Tadeusz Morzy, Patrick Valduriez, and Ladjel Bellatreche, editors, *New Trends in Databases and Information Systems - ADBIS 2015 Short Papers and Workshops, BigDap, DCSA, GID, MEBIS, OAIS, SW4CH, WISARD, Poitiers, France, September 8-11, 2015. Proceedings*, volume 539 of *Communications in Computer and Information Science*, pages 135–144. Springer, 2015a. ISBN 978-3-319-23200-3. doi: 10.1007/978-3-319-23201-0_16. URL `https://doi.org/10.1007/978-3-319-23201-0_16`.

Juraj Mosko, Jakub Lokoc, Tomás Grosup, Premysl Cech, Tomás Skopal, and Jan Lánský. Evaluating multilayer multimedia exploration. In Giuseppe Amato, Richard C. H. Connor, Fabrizio Falchi, and Claudio Gennaro, editors, *Similarity Search and Applications - 8th International Conference, SISAP 2015, Glasgow, UK, October 12-14, 2015, Proceedings*, volume 9371 of *Lecture Notes in Computer Science*, pages 162–169. Springer, 2015b. ISBN 978-3-319-25086-1. doi: 10.1007/978-3-319-25087-8_15. URL `https://doi.org/10.1007/978-3-319-25087-8_15`.

Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(11):2227–2240, 2014. doi: 10.1109/TPAMI.2014.2321376. URL `https://doi.org/10.1109/TPAMI.2014.2321376`.

Stefan Müller, Stefan Eickeler, and Gerhard Rigoll. Multimedia database retrieval using hand-drawn sketches. In *Fifth International Conference on Document Analysis and Recognition, ICDAR 1999, 20-22 September, 1999, Bangalore, India*, pages 289–292. IEEE Computer Society, 1999. doi: 10.1109/ICDAR.1999.791781. URL `https://doi.org/10.1109/ICDAR.1999.791781`.

Phuong Anh Nguyen, Chong-Wah Ngo, Danny Francis, and Benoit Huet. VIREO @ video browser showdown 2019. In *MultiMedia Modeling - 25th International Conference, MMM 2019, Thessaloniki, Greece, January 8-11, 2019, Proceedings, Part II*, pages 609–615, 2019. doi: 10.1007/978-3-030-05716-9_54.

David Novak, Michal Batko, and Pavel Zezula. Metric index: An efficient and scalable solution for precise and approximate similarity search. *Inf. Syst.*, 36 (4):721–733, 2011.

David Novak, Michal Batko, and Pavel Zezula. Large-scale similarity data management with distributed metric index. *Inf. Process. Manage.*, 48(5):855–872, 2012.

Marco Patella and Paolo Ciaccia. The many facets of approximate similarity search. In *Proceedings of the 24th International Conference on Data Engineering Workshops, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 308–319. IEEE Computer Society, 2008. doi: 10.1109/ICDEW.2008.4498340. URL https://doi.org/10.1109/ICDEW.2008.4498340.

Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011. URL http://dl.acm.org/citation.cfm?id=2078195.

Tomás Pevný and Andrew D. Ker. Towards dependable steganalysis. In Adnan M. Alattar, Nasir D. Memon, and Chad Heitzenrater, editors, *Media Watermarking, Security, and Forensics 2015, San Francisco, CA, USA, February 9-11, 2015, Proceedings*, volume 9409 of *SPIE Proceedings*, page 94090I. SPIE, 2015. doi: 10.1117/12.2083216. URL https://doi.org/10.1117/12.2083216.

Olivier Pietquin and Helen F. Hastie. A survey on metrics for the evaluation of user simulations. *Knowledge Eng. Review*, 28(1):59–73, 2013. doi: 10.1017/S0269888912000343. URL https://doi.org/10.1017/S0269888912000343.

Natasa Przulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):177–183, 2007. doi: 10.1093/bioinformatics/btl301. URL https://doi.org/10.1093/bioinformatics/btl301.

David Reinsel, John Gantz, and John Rydning. Data age 2025: The evolution of data to life-critical. *Don't Focus on Big Data*, 2017.

Chuitian Rong, Chunbin Lin, Yasin N. Silva, Jianguo Wang, Wei Lu, and Xiaoyong Du. Fast and scalable distributed set similarity joins for big data analytics. In *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017*, pages 1059–1070. IEEE Computer Society, 2017. doi: 10.1109/ICDE.2017.151. URL https://doi.org/10.1109/ICDE.2017.151.

Luca Rossetto, Mahnaz Amiri Parian, Ralph Gasser, Ivan Giangreco, Silvan Heller, and Heiko Schuldt. Deep learning-based concept detection in vitrivr. In *MultiMedia Modeling - 25th International Conference, MMM 2019, Thessaloniki,*

Greece, January 8-11, 2019, Proceedings, Part II, pages 616–621, 2019a. doi: 10.1007/978-3-030-05716-9_55.

Luca Rossetto, Heiko Schuldt, George Awad, and Asad A. Butt. V3C - A research video collection. In Ioannis Kompatsiaris, Benoit Huet, Vasileios Mezaris, Cathal Gurrin, Wen-Huang Cheng, and Stefanos Vrochidis, editors, *MultiMedia Modeling - 25th International Conference, MMM 2019, Thessaloniki, Greece, January 8-11, 2019, Proceedings, Part I*, volume 11295 of *Lecture Notes in Computer Science*, pages 349–360. Springer, 2019b. doi: 10.1007/978-3-030-05710-7_29. URL https://doi.org/10.1007/978-3-030-05710-7_29.

Yossi Rubner and Carlo Tomasi. *Perceptual metrics for image database navigation*, volume 594. Springer Science & Business Media, 2013.

Yossi Rubner, Leonidas J Guibas, and Carlo Tomasi. The earth mover's distance, multi-dimensional scaling, and color-based image retrieval. In *Proceedings of the ARPA image understanding workshop*, volume 661, page 668, 1997.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Simone Santini and Ramesh C. Jain. Beyond query by example. In Wolfgang Effelsberg and Brian C. Smith, editors, *Proceedings of the 6th ACM International Conference on Multimedia '98, Bristol, England, September 12-16, 1998.*, pages 345–350. ACM, 1998. doi: 10.1145/290747.290800. URL https://doi.org/10.1145/290747.290800.

Raimondo Schettini, Gianluigi Ciocca, and Isabella Gagliardi. Feature extraction for content-based image retrieval. In *Encyclopedia of Database Systems (2nd ed.)*. Springer, 2018.

Klaus Schoeffmann, David Ahlström, and László Böszörményi. A user study of visual search performance with interactive 2d and 3d storyboards. In Marcin Detyniecki, Ana García-Serrano, Andreas Nürnberger, and Sebastian Stober, editors, *Adaptive Multimedia Retrieval. Large-Scale Multimedia Retrieval and Evaluation - 9th International Workshop, AMR 2011, Barcelona, Spain, July 18-19, 2011, Revised Selected Papers*, volume 7836 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 2011. doi: 10.1007/978-3-642-37425-8_2. URL https://doi.org/10.1007/978-3-642-37425-8_2.

Klaus Schoeffmann, Marco A. Hudelist, and Jochen Huber. Video interaction tools: A survey of recent work. *ACM Comput. Surv.*, 48(1):14:1–14:34, September 2015. ISSN 0360-0300. doi: 10.1145/2808796. URL http://doi.acm.org/10.1145/2808796.

Klaus Schoeffmann, Bernd Münzer, Andreas Leibetseder, Jürgen Primus, and Sabrina Kletz. Autopiloting feature maps: The deep interactive video exploration (divexplore) system at VBS2019. In *MultiMedia Modeling - 25th International Conference, MMM 2019, Thessaloniki, Greece, January 8-11, 2019, Proceedings, Part II*, pages 585–590, 2019. doi: 10.1007/978-3-030-05716-9_50.

Erich Schubert, Arthur Zimek, and Hans-Peter Kriegel. Fast and scalable outlier detection with approximate nearest neighbor ensembles. In Matthias Renz, Cyrus Shahabi, Xiaofang Zhou, and Muhammad Aamir Cheema, editors, *Database Systems for Advanced Applications - 20th International Conference, DASFAA 2015, Hanoi, Vietnam, April 20-23, 2015, Proceedings, Part II*, volume 9050 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 2015.

S. Sclaroff, L. Taycher, and M. La Cascia. Imagerover: a content-based image browser for the world wide web. In *1997 Proceedings IEEE Workshop on Content-Based Access of Image and Video Libraries*, pages 2–9, June 1997. doi: 10.1109/IVL.1997.629714.

Yasin N. Silva and Jason M. Reed. Exploiting mapreduce-based similarity joins. In K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 693–696. ACM, 2012. doi: 10.1145/2213836.2213935. URL `http://doi.acm.org/10.1145/2213836.2213935`.

Yasin N. Silva, Jason M. Reed, and Lisa M. Tsosie. Mapreduce-based similarity join for metric spaces. In *Proceedings of the 1st International Workshop on Cloud Intelligence*, Cloud-I '12, pages 3:1–3:8, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1596-8. doi: 10.1145/2347673.2347676. URL `http://doi.acm.org/10.1145/2347673.2347676`.

Yasin N. Silva, Spencer S. Pearson, Jaime Chon, and Ryan Roberts. Similarity joins: Their implementation and interactions with other database operators. *Inf. Syst.*, 52:149–162, 2015. doi: 10.1016/j.is.2015.01.008. URL `https://doi.org/10.1016/j.is.2015.01.008`.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL `http://arxiv.org/abs/1409.1556`.

Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *9th IEEE International Conference on Computer Vision (ICCV 2003), 14-17 October 2003, Nice, France*, pages 1470–1477. IEEE Computer Society, 2003. doi: 10.1109/ICCV.2003.1238663. URL `https://doi.org/10.1109/ICCV.2003.1238663`.

Tomás Skopal. Pivoting m-tree: A metric access method for efficient similarity search. In *DATESO*, volume 98 of *CEUR Workshop Proceedings*, pages 27–37. CEUR-WS.org, 2004.

Arnold W. M. Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh C. Jain. Content-based image retrieval at the end of the early years. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(12):1349–1380, 2000.

G. Song, J. Rochas, F. Huet, and F. Magoulès. Solutions for processing k nearest neighbor joins for massive data on mapreduce. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 279–287, March 2015. doi: 10.1109/PDP.2015.79.

Ge Song, Justine Rochas, Lea El Beze, Fabrice Huet, and Frédéric Magoulès. K nearest neighbour joins for big data on mapreduce: A theoretical and experimental analysis. *IEEE Trans. Knowl. Data Eng.*, 28(9):2376–2392, 2016. doi: 10.1109/TKDE.2016.2562627. URL https://doi.org/10.1109/TKDE.2016.2562627.

Henrik Stewénius, Steinar H. Gunderson, and Julien Pilet. Size matters: Exhaustive geometric verification for image retrieval accepted for ECCV 2012. In Andrew W. Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision - ECCV 2012 - 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part II*, volume 7573 of *Lecture Notes in Computer Science*, pages 674–687. Springer, 2012. doi: 10.1007/978-3-642-33709-3_48. URL https://doi.org/10.1007/978-3-642-33709-3_48.

Aleksandar Stupar, Sebastian Michel, and Ralf Schenkel. Rankreduce - processing k-nearest neighbor queries on top of mapreduce. In Roi Blanco, Berkant Barla Cambazoglu, and Claudio Lucchese, editors, *Proceedings of the 8th Workshop on Large-Scale Distributed Systems for Information Retrieval, Geneva, Switzerland, July 23, 2010*, volume 630 of *CEUR Workshop Proceedings*, pages 13–18. CEUR-WS.org, 2010. URL http://ceur-ws.org/Vol-630/lsdsir2.pdf.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

MingJie Tang, Yongyang Yu, Walid G. Aref, Qutaibah M. Malluhi, and Mourad Ouzzani. Efficient processing of hamming-distance-based similarity-search queries over mapreduce. In Gustavo Alonso, Floris Geerts, Lucian Popa, Pablo Barceló, Jens Teubner, Martín Ugarte, Jan Van den Bussche, and Jan Paredaens, editors, *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015.*, pages 361–372. OpenProceedings.org, 2015. doi: 10.5441/002/edbt.2015.32. URL https://doi.org/10.5441/002/edbt.2015.32.

Rares Vernica, Michael J. Carey, and Chen Li. Efficient parallel set-similarity joins using mapreduce. In Ahmed K. Elmagarmid and Divyakant Agrawal, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*, pages 495–506. ACM, 2010. doi: 10.1145/1807167.1807222. URL http://doi.acm.org/10.1145/1807167.1807222.

Jiannan Wang, Guoliang Li, and Jianhua Feng. Trie-join: Efficient trie-based string similarity joins with edit-distance constraints. *PVLDB*, 3(1):1219–1230, 2010. URL http://www.comp.nus.edu.sg/~vldb2010/proceedings/files/papers/R108.pdf.

Marcel Worring, Dennis Koelma, and Jan Zahálka. Multimedia pivot tables for multimedia analytics on image collections. *IEEE Trans. Multimedia*, 18(11):

2217–2227, 2016. doi: 10.1109/TMM.2016.2614380. URL https://doi.org/10.1109/TMM.2016.2614380.

Chuan Xiao, Wei Wang, and Xuemin Lin. Ed-join: an efficient algorithm for similarity joins with edit distance constraints. *PVLDB*, 1(1):933–944, 2008. URL http://www.vldb.org/pvldb/1/1453957.pdf.

Bin Yao, Feifei Li, and Piyush Kumar. K nearest neighbor queries and knn-joins in large relational databases (almost) for free. In Feifei Li, Mirella M. Moro, Shahram Ghandeharizadeh, Jayant R. Haritsa, Gerhard Weikum, Michael J. Carey, Fabio Casati, Edward Y. Chang, Ioana Manolescu, Sharad Mehrotra, Umeshwar Dayal, and Vassilis J. Tsotras, editors, *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 4–15. IEEE Computer Society, 2010. doi: 10.1109/ICDE.2010.5447837. URL https://doi.org/10.1109/ICDE.2010.5447837.

Alexei Yavlinsky, Edward James Schofield, and Stefan M. Rüger. Automated image annotation using global features and robust nonparametric density estimation. In Wee Kheng Leow, Michael S. Lew, Tat-Seng Chua, Wei-Ying Ma, Lekha Chaisorn, and Erwin M. Bakker, editors, *Image and Video Retrieval, 4th International Conference, CIVR 2005, Singapore, July 20-22, 2005, Proceedings*, volume 3568 of *Lecture Notes in Computer Science*, pages 507–517. Springer, 2005. doi: 10.1007/11526346_54. URL https://doi.org/10.1007/11526346_54.

Kwang Moo Yi, Eduard Trulls, Vincent Lepetit, and Pascal Fua. LIFT: learned invariant feature transform. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VI*, volume 9910 of *Lecture Notes in Computer Science*, pages 467–483. Springer, 2016. doi: 10.1007/978-3-319-46466-4_28. URL https://doi.org/10.1007/978-3-319-46466-4_28.

Forrest W Young. *Multidimensional scaling: History, theory, and applications*. Psychology Press, 2013.

Cui Yu, Bin Cui, Shuguang Wang, and Jianwen Su. Efficient index-based KNN join processing for high-dimensional data. *Information & Software Technology*, 49(4):332–344, 2007. doi: 10.1016/j.infsof.2006.05.006. URL https://doi.org/10.1016/j.infsof.2006.05.006.

Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: A unified engine for big data processing. *Commun. ACM*, 59(11):56–65, October 2016. ISSN 0001-0782. doi: 10.1145/2934664.

Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search - The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Kluwer, 2006. ISBN 978-0-387-29146-8. doi: 10.1007/0-387-29151-2. URL https://doi.org/10.1007/0-387-29151-2.

Chi Zhang, Feifei Li, and Jeffrey Jestes. Efficient parallel knn joins for large data in mapreduce. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, pages 38–49, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-0790-1. doi: 10.1145/2247596.2247602.

Ruofei Zhang, Zhongfei (Mark) Zhang, Mingjing Li, Wei-Ying Ma, and HongJiang Zhang. A probabilistic semantic model for image annotation and multi-modal image retrieval. *Multimedia Syst.*, 12(1):27–33, 2006. doi: 10.1007/s00530-006-0025-1. URL `https://doi.org/10.1007/s00530-006-0025-1`.

Wengang Zhou, Houqiang Li, and Qi Tian. Recent advance in content-based image retrieval: A literature survey. *arXiv preprint arXiv:1706.06064*, 2017a.

Xinyu Zhou, Cong Yao, He Wen, Yuzhi Wang, Shuchang Zhou, Weiran He, and Jiajun Liang. EAST: an efficient and accurate scene text detector. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 2642–2651. IEEE Computer Society, 2017b. doi: 10.1109/CVPR.2017.283. URL `https://doi.org/10.1109/CVPR.2017.283`.

Pingfei Zhu, Xiangwen Zhan, and Wenming Qiu. Efficient k-nearest neighbors search in high dimensions using mapreduce. In Keqiu Li, Heng Qi, Jean-Luc Gaudiot, Jay Kishigami, Hongyi Wu, Kuan-Ching Li, and Yongwei Wu, editors, *Fifth IEEE International Conference on Big Data and Cloud Computing, BDCloud 2015, Dalian, China, August 26-28, 2015*, pages 23–30. IEEE Computer Society, 2015. doi: 10.1109/BDCloud.2015.51. URL `https://doi.org/10.1109/BDCloud.2015.51`.

List of Figures

List of Tables

List of Abbreviations

AC-NN – approximately correct nearest neighbor
ASR – automatic speech recognition
AVS – ad-hoc video search
BoF – bag of features
BoW – bag of words
BR – ball region
CBIR – content-based image retrieval
CBMR – content-based multimedia retrieval
CBVR – content-based video retrieval
CR – cut region
DCNN – deep convolutional neural networks
DR – distance ratio
ECM – exponential Chebyshev minimizer
EP – external program
FP50 – false positive rate at 50% of recall
FPS – frames per second
FTI – find-the-image application
GMM – gaussian mixture model
HD – high definition
HDFS – Hadoop distributed file system
HE – Hamming embedding
HTTP – hyper text transfer protocol
HTTPS – hyper text transfer protocol secured
I/O – input/output
KIS – known item search
kNN – k nearest neighbors
LB – lower bound
LIFT – learned invariant feature transform
LSC – lifelog search challenge
LSH – locality sensitive hashing
MDS – multidimensional scaling
M-Index – metric index
MLES – multilayer exploration structure
M-Tree – metric tree
OCR – optical character recognition
OODB – object oriented database
ORDBMS – object relational database management system
PAKJ – pivot-based approximate kNN join
PCA – principal component analysis
PEGKJ – pivot-based epsilon guaranteed kNN join
PGBJ – pivot-based extact kNN join
PM-Tree – pivoting metric tree
RAM – random access memory
RDD – resilient distributed dataset
ROC – receiver operating characteristic

SIFT – scale invariant features
SOM – self-organizing map
SQFD – signature quadratic form distance
SSM – self-sorting map
SURF – speed up robust features
TSL – transport layer security
UB – upper bound
VBS – video browser showdown
VIRET – video retrieval tool
YARN – yet another resource negotiator

List of publications

Jakub Lokoc, Premysl Cech, Jiri Novák, and Tomás Skopal. Cut-region: A compact building block for hierarchical metric indexing. In Gonzalo Navarro and Vladimir Pestov, editors, *Similarity Search and Applications - 5th International Conference, SISAP 2012, Toronto, ON, Canada, August 9-10, 2012. Proceedings*, volume 7404 of *Lecture Notes in Computer Science*, pages 85–100. Springer, 2012. ISBN 978-3-642-32152-8. doi: 10.1007/978-3-642-32153-5_7. URL https://doi.org/10.1007/978-3-642-32153-5_7

Jakub Lokoc, Juraj Mosko, Premysl Cech, and Tomás Skopal. On indexing metric spaces using cut-regions. *Inf. Syst.*, 43:1–19, 2014b. doi: 10.1016/j.is.2014.01.007. URL https://doi.org/10.1016/j.is.2014.01.007

Jakub Lokoc, Tomás Grosup, Premysl Cech, and Tomás Skopal. Towards efficient multimedia exploration using the metric space approach. In *12th International Workshop on Content-Based Multimedia Indexing, CBMI 2014, Klagenfurt, Austria, June 18-20, 2014*, pages 1–4. IEEE, 2014a. ISBN 978-1-4799-3990-9. doi: 10.1109/CBMI.2014.6849851. URL https://doi.org/10.1109/CBMI.2014.6849851

Juraj Mosko, Jakub Lokoc, Tomás Grosup, Premysl Cech, Tomás Skopal, and Jan Lánský. Evaluating multilayer multimedia exploration. In Giuseppe Amato, Richard C. H. Connor, Fabrizio Falchi, and Claudio Gennaro, editors, *Similarity Search and Applications - 8th International Conference, SISAP 2015, Glasgow, UK, October 12-14, 2015, Proceedings*, volume 9371 of *Lecture Notes in Computer Science*, pages 162–169. Springer, 2015b. ISBN 978-3-319-25086-1. doi: 10.1007/978-3-319-25087-8_15. URL https://doi.org/10.1007/978-3-319-25087-8_15

Tomás Grosup, Premysl Cech, Jakub Lokoc, and Tomás Skopal. A web portal for effective multi-model exploration. In Xiangjian He, Suhuai Luo, Dacheng Tao, Changsheng Xu, Jie Yang, and Muhammad Abul Hasan, editors, *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 315–318. Springer, 2015a. ISBN 978-3-319-14441-2. doi: 10.1007/978-3-319-14442-9_37. URL https://doi.org/10.1007/978-3-319-14442-9_37

Tomás Grosup, Juraj Mosko, and Premysl Cech. Continuous hierarchical exploration of multimedia collections. In *13th International Workshop on Content-Based Multimedia Indexing, CBMI 2015, Prague, Czech Republic, June 10-12, 2015*, pages 1–4. IEEE, 2015b. doi: 10.1109/CBMI.2015.7153621. URL https://doi.org/10.1109/CBMI.2015.7153621

Premysl Cech and Tomás Grosup. Comparison of metric space browsing strategies for efficient image exploration. In *13th International Workshop on Content-*

Based Multimedia Indexing, CBMI 2015, Prague, Czech Republic, June 10-12, 2015, pages 1–6. IEEE, 2015. doi: 10.1109/CBMI.2015.7153631. URL https://doi.org/10.1109/CBMI.2015.7153631

Juraj Mosko, Jakub Lokoc, Tomás Grosup, Premysl Cech, Tomás Skopal, and Jan Lansky. MLES: multilayer exploration structure for multimedia exploration. In Tadeusz Morzy, Patrick Valduriez, and Ladjel Bellatreche, editors, *New Trends in Databases and Information Systems - ADBIS 2015 Short Papers and Workshops, BigDap, DCSA, GID, MEBIS, OAIS, SW4CH, WISARD, Poitiers, France, September 8-11, 2015. Proceedings*, volume 539 of *Communications in Computer and Information Science*, pages 135–144. Springer, 2015a. ISBN 978-3-319-23200-3. doi: 10.1007/978-3-319-23201-0_16. URL https://doi.org/10.1007/978-3-319-23201-0_16

Jakub Lokoc, Jan Kohout, Premysl Cech, Tomás Skopal, and Tomás Pevný. k-nn classification of malware in HTTPS traffic using the metric space approach. In Michael Chau, G. Alan Wang, and Hsinchun Chen, editors, *Intelligence and Security Informatics - 11th Pacific Asia Workshop, PAISI 2016, Auckland, New Zealand, April 19, 2016, Proceedings*, volume 9650 of *Lecture Notes in Computer Science*, pages 131–145. Springer, 2016. ISBN 978-3-319-31862-2. doi: 10.1007/978-3-319-31863-9_10. URL https://doi.org/10.1007/978-3-319-31863-9_10

Premysl Cech, Jan Kohout, Jakub Lokoc, Tomás Komárek, Jakub Marousek, and Tomás Pevný. Feature extraction and malware detection on large HTTPS data using mapreduce. In Laurent Amsaleg, Michael E. Houle, and Erich Schubert, editors, *Similarity Search and Applications - 9th International Conference, SISAP 2016, Tokyo, Japan, October 24-26, 2016. Proceedings*, volume 9939 of *Lecture Notes in Computer Science*, pages 311–324, 2016. ISBN 978-3-319-46758-0. doi: 10.1007/978-3-319-46759-7_24. URL https://doi.org/10.1007/978-3-319-46759-7_24

Jakub Lokoc, Tomás Grosup, Premysl Cech, Tomás Pevný, and Tomás Skopal. Malware discovery using behaviour-based exploration of network traffic. In Christian Beecks, Felix Borutta, Peer Kröger, and Thomas Seidl, editors, *Similarity Search and Applications - 10th International Conference, SISAP 2017, Munich, Germany, October 4-6, 2017, Proceedings*, volume 10609 of *Lecture Notes in Computer Science*, pages 315–323. Springer, 2017. ISBN 978-3-319-68473-4. doi: 10.1007/978-3-319-68474-1_22. URL https://doi.org/10.1007/978-3-319-68474-1_22

Premysl Cech, Jakub Marousek, Jakub Lokoc, Yasin N. Silva, and Jeremy Starks. Comparing mapreduce-based k-nn similarity joins on hadoop for high-dimensional data. In Gao Cong, Wen-Chih Peng, Wei Emma Zhang, Chengliang Li, and Aixin Sun, editors, *Advanced Data Mining and Applications - 13th International Conference, ADMA 2017, Singapore, November 5-6, 2017, Proceedings*, volume 10604 of *Lecture Notes in Computer Science*, pages 63–75. Springer, 2017. ISBN 978-3-319-69178-7. doi: 10.1007/978-3-319-69179-4_5. URL

https://doi.org/10.1007/978-3-319-69179-4_5

Jan Kohout, Tomás Komárek, Premysl Cech, Jan Bodnár, and Jakub Lokoc. Learning communication patterns for malware discovery in https data. *Expert Syst. Appl.*, 101:129–142, 2018. doi: 10.1016/j.eswa.2018.02.010. URL https://doi.org/10.1016/j.eswa.2018.02.010

Jakub Lokoc, Gregor Kovalcík, Tomás Soucek, Jaroslav Moravec, Jan Bodnár, and Premysl Cech. VIRET tool meets nasnet. In Ioannis Kompatsiaris, Benoit Huet, Vasileios Mezaris, Cathal Gurrin, Wen-Huang Cheng, and Stefanos Vrochidis, editors, *MultiMedia Modeling - 25th International Conference, MMM 2019, Thessaloniki, Greece, January 8-11, 2019, Proceedings, Part II*, volume 11296 of *Lecture Notes in Computer Science*, pages 597–601. Springer, 2019a. ISBN 978-3-030-05715-2. doi: 10.1007/978-3-030-05716-9_52. URL https://doi.org/10.1007/978-3-030-05716-9_52

Jakub Lokoc, Gregor Kovalcík, Tomás Soucek, Jaroslav Moravec, and Premysl Cech. VIRET: A video retrieval tool for interactive known-item search. In Abdulmotaleb El-Saddik, Alberto Del Bimbo, Zhongfei Zhang, Alexander G. Hauptmann, K. Selçuk Candan, Marco Bertini, Lexing Xie, and Xiao-Yong Wei, editors, *Proceedings of the 2019 on International Conference on Multimedia Retrieval, ICMR 2019, Ottawa, ON, Canada, June 10-13, 2019.*, pages 177–181. ACM, 2019b. doi: 10.1145/3323873.3325034. URL https://doi.org/10.1145/3323873.3325034

Jakub Lokoc, Tomás Soucek, Premysl Cech, and Gregor Kovalcík. Enhanced VIRET tool for lifelog data. In Cathal Gurrin, Klaus Schöffmann, Hideo Joho, Duc-Tien Dang-Nguyen, Michael Riegler, and Luca Piras, editors, *Proceedings of the ACM Workshop on Lifelog Search Challenge, LSC@ICMR 2019, Ottawa, ON, Canada, 10 June 2019.*, pages 25–26. ACM, 2019c. doi: 10.1145/3326460.3329159. URL https://doi.org/10.1145/3326460.3329159

Jakub Lokoč, Gregor Kovalčík, Tomáš Souček, Jaroslav Moravec, and Přemysl Čech. A framework for effective known-item search in video. In *Proceedings of the 27th ACM International Conference on Multimedia (MM'19), October 21–25, 2019, Nice, France*, ACM, New York, NY, USA, pages 1–9. ACM, 2019. doi: 10.1145/3343031.3351046. URL https://doi.org/10.1145/3343031.3351046

Premysl Cech, Jakub Lokoc, and Yasin N. Silva. Pivot-based approximate k-nn similarity joins for big high-dimensional data. *Information Systems*, 87: 101410, 2020. ISSN 0306-4379. doi: https://doi.org/10.1016/j.is.2019.06.006. URL http://www.sciencedirect.com/science/article/pii/S0306437918302473