Database Support for Large-Scale Multimedia Retrieval

Inauguraldissertation

zur
Erlangung der Würde eines Doktors der Philosophie
vorgelegt der
Philosophisch-Naturwissenschaftlichen Fakultät
der Universität Basel

von

Ivan Giangreco

aus Italien

Basel, 2018

Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät
auf Antrag von

Prof. Dr. Heiko Schuldt, Universität Basel, Dissertationsleiter
Prof. Dr. Michael Grossniklaus, Universität Konstanz, Korreferent

Basel, den 27.02.2018

Prof. Dr. Martin Spiess, Dekan

Tut das Unnütze,
singt die Lieder,
die man aus eurem Mund
nicht erwartet!
Seid unbequem,
seid Sand,
nicht das Öl im
Getriebe der Welt!

— Günter Eich,
Träume

Everything not saved
will be lost.

Quit Screen Message

Zusammenfassung

Die Verwaltung multimedialer Datensammlungen stellt aufgrund der zunehmenden Verbreitung moderner Aufnahmetechnologien und den daraus resultierenden, stetig wachsenden Multimediasammlungen nicht nur wegen deren Grösse, sondern auch bedingt durch die Komplexität der zu speichernden Daten und assoziierten Metadaten vermehrt eine grosse Herausforderung dar. Aufgrund der fehlenden Unterstützung für die gegebenen Daten und die entsprechenden Suchparadigmen ist konventionellen Ansätzen der Datenspeicherung nur begrenzt Erfolg beizumessen. Zuletzt hat die Multimediaforschung das Fehlen einer Lösung für eine effiziente und effektive Datenverwaltung von multimedialen Inhalten als bedeutendes Hindernis für weitere Entwicklungen in diesem Forschungsfeld eingestuft.

Die vorliegende Dissertation schliesst die Lücke zwischen den Forschungsfeldern der Datenbanksysteme und des Multimedia-Retrievals und stellt einen Ansatz zur Verwaltung grosser multimedialer Datensammlungen sowie die entsprechenden Suchparadigmen vor. Hierfür werden die notwendigen Basiskomponenten für ein Multimedia-Datenbanksystem betrachet, welches auf dem relationalen Datenmodell und dem Vektorraummodell basiert. Die Arbeit präsentiert die folgenden Beiträge zur Entwicklung eines ganzheitlichen Modells für ein Datenverwaltungssystem für Multimediadaten: Ein Architekturmodell wird eingeführt, das ein Datenverwaltungssystem für multimediale Daten aus systemarchitektonischer Sicht beschreibt. Darüber hinaus wird ein Datenmodell vorgestellt, das einerseits Unterstützung für die Speicherung von Multimediadaten und deren Metadaten bietet und andererseits auf Ähnlichkeit basierte Suchen erlaubt. Es wird ein ausführliches Anfragemodell für eine breite Auswahl an Suchparadigmen beschrieben, welches Anfragen sowohl aus logischer Hinsicht als auch im Hinblick auf deren Ausführung zu spezifizieren vermag. Zudem wird die Effizienz und Skalierbarkeit des Systems aus der Perspektive eines Verteilungs- und eines Speichermodells betrachtet. Eine Reihe verschiedener Indexstrukturen für hochdimensionale Daten aus dem Vektorraummodell wird überdies präsentiert und bereitgestellt.

Die entwickelten Modelle werden im Multimedia-Datenverwaltungssystem ADAM$_{pro}$ innerhalb des iMotion/vitrivr Systems implementiert und quantitativ unter Beizug von Datensammlungen evaluiert, die in ihren Grössen den aktuellen Stand der Forschung übersteigen. Die Resultate bekräftigen die vorgestellten Modelle und erlauben es, die Wirksamkeit der eingeführten Konzepte hervorzuheben. Aus den Ergebnissen lassen sich zudem wichtige Implikationen für die zukünftige Forschung im Bereich der Datenverwaltung für multimediale Daten ableiten.

Abstract

With the increasing proliferation of recording devices and the resulting abundance of multimedia data available nowadays, searching and managing these ever-growing collections becomes more and more difficult. In order to support retrieval tasks within large multimedia collections, not only the sheer size, but also the complexity of data and their associated metadata pose great challenges, in particular from a data management perspective. Conventional approaches to address this task have been shown to have only limited success, particularly due to the lack of support for the given data and the required query paradigms. In the area of multimedia research, the missing support for efficiently and effectively managing multimedia data and metadata has recently been recognised as a stumbling block that constraints further developments in the field.

In this thesis, we bridge the gap between the database and the multimedia retrieval research areas. We approach the problem of providing a data management system geared towards large collections of multimedia data and the corresponding query paradigms. To this end, we identify the necessary building-blocks for a multimedia data management system which adopts the relational data model and the vector-space model. In essence, we make the following main contributions towards a holistic model of a database system for multimedia data: We introduce an architectural model describing a data management system for multimedia data from a system architecture perspective. We further present a data model which supports the storage of multimedia data and the corresponding metadata, and provides similarity-based search operations. This thesis describes an extensive query model for a very broad range of different query paradigms specifying both logical and executional aspects of a query. Moreover, we consider the efficiency and scalability of the system in a distribution and a storage model, and provide a large and diverse set of index structures for high-dimensional data coming from the vector-space model.

The developed models crystallise into the scalable multimedia data management system ADAM$_{pro}$ which has been implemented within the iMotion/vitrivr retrieval stack. We quantitatively evaluate our concepts on collections that exceed the current state of the art. The results underline the benefits of our approach and assist in understanding the role of the introduced concepts. Moreover, the findings provide important implications for future research in the field of multimedia data management.

Acknowledgements

This thesis is the product of four years of intense work at the University of Basel. It completes my time as a student at the university which started back in 2006. Along this way, throughout my studies as Bachelor, Master and PhD student, I got to know many great and kind people who accompanied me in this wonderful time and to whom I am honestly grateful, despite I may not mention them here by name. Nevertheless, I would like to express my gratitude towards a few particular people.

First, a special thanks goes to Prof. Dr. Heiko Schuldt, not only for giving me the chance – both throughout my studies and during my PhD – to work on very interesting projects and topics, but also for the countless opportunities I received during all this time. Thanks for all!

I wish to also thank Prof. Dr. Michael Grossniklaus for his willingness to review my thesis, for his time and effort in doing this and for inviting me to the University of Konstanz to present my work.

During my PhD, I had the great pleasure to work together with the members of the Databases and Information Systems research group. A special thanks deserve Claudiu Tănase and, in particular, Luca Rossetto who have also been part of the iMotion/vitrivr project. Thanks for all the interesting discussions we had and for being great team members. Moreover, I would like to mention the former PhD students Filip-Martin Brinkmann, Ihab Al Kabary, Ilir Fetai, Lukas Beck, Nenad Stojnic and the current PhD students Alexander Stiemer, Lukas Probst, Marco Vogt, Ralph Gasser and Silvan Heller with whom I shared many hours throughout those years both in talks and discussions, but also when just having a coffee. Thanks for the pleasant time we had together.

Thanks to all the students I had the pleasure to work with throughout the past years and who contributed to this thesis in one or the other way. I would like to take the opportunity to thank the many other people at the Department of Mathematics and Computer Science who are only thanked too seldom, be it the management, the secretariat or the IT administrators. Thanks also to all the other colleagues at the department, the professors, the postdocs and PhD students, the Bachelor and Master students with whom I had the pleasure to interact with through the past years. Thank you to my former colleagues of study Céline Kellner, Daniel Kohler, Ferdinand Niedermann, Florian Lindörfer, Marcel Büchler, Martin Spielmann, Steven Rose, Thomas Ritter and Urs Schnurrenberger for the good times we had together at Bernoullistrasse.

A big thank you also to my dear – often longtime – choir friends, my former neighbours and childhood friends, my friends from school, and friends I know from here and there,

who all keep making my life sweeter and sweeter. I consider myself lucky to have a circle of friends full of such fantastic people. A special thanks deserves Raphaela Gisi for designing the cover of this thesis.

Finally, I owe a big thanks to my family, in particular, my parents and my brother, my godmother, but also my father/mother/brother/sisters in-laws and nieces and nephews for their endless help, support, understanding, time, food, love and – most important – for dragging me lovingly from work so that I could also just enjoy from time to time *il dolce far niente*. Last, but in no way least, thank you, dearest Noemi, for your never-ending support and the wonderful time we have together. You are the most wonderful companion one can imagine!

"I can no other answer make
but thanks, and thanks, and
ever thanks."

— William Shakespeare,
Twelfth Night

This work was partly supported by the Swiss National Science Foundation in the context of the CHIST-ERA project IMOTION, contract no. 20CH21_151571, which is also thankfully acknowledged.

Contents

List of Figures

List of Tables

List of Acronyms

ACID	Atomicity, Consistency, Isolation, Durability
HDFS	Hadoop Distributed File System
JDBC	Java Database Connectivity
MPEG	Moving Picture Experts Group
NN	Nearest Neighbour (in particular, κNN and εNN)
ODBC	Open Database Connectivity
P2P	Peer-to-Peer
PCA	Principal Component Analysis
QbE	Query-by-Example
QbS	Query-by-Sketch
RBO	Rank-biased Overlap
XML	Extensible Markup Language

List of Symbols

The list below summarises symbols occurring frequently throughout this thesis. Additional notation is introduced as needed.

$o \in \mathcal{O}$	multimedia document
dim	dimensionality
n	collection size, cardinality of relation
$\boldsymbol{u} := \left(u_1, u_2, \cdots, u_{dim}\right) \in \mathbb{U}$	data space
$\boldsymbol{d} := \left(d_1, d_2, \cdots, d_{dim}\right) \in \mathcal{D}$	representation of multimedia documents
$qi \in \mathcal{QI}$	user query intent
$\boldsymbol{q} := \left(q_1, q_2, \cdots, q_{dim}\right) \in \mathcal{Q}$	representations of query
$\delta(\cdot, \cdot)$	distance function, e.g., Minkowski distance $\delta_{L_p}(\cdot, \cdot)$
$\bar{\delta}$	evaluated distance function
$a \in A$	attributes
$\textsc{dom}(\cdot)$	data domain of attribute
$t \in R$	tuple of relation
$\textsc{sch}(\cdot)$	schema of relation
$\textsc{val}(\cdot)$	extent of relation
q	(database) query
$\textsc{tid} \in \mathrm{TID}$	tuple identifier
$(\textsc{tid}, \ddot{\imath}) \in I_R$	index to relation R
$\pi_A(\cdot)$	projection function based on attributes A
$\sigma_\varphi(\cdot)$	selection operation with filtering predicate φ
$\mathcal{A}$	candidate set of results
$\mathcal{R}$	result set
$\tau_\psi(\cdot)$	similarity operation with similarity predicate ψ
$\tilde{R}$	similarity-based relation
ϑ	limiting predicate

Part I

Introduction and Background

I have nothing to offer but blood, toil, tears, and sweat.

— Winston Churchill

1

Introduction

Vannevar Bush published in 1945 a visionary essay entitled *As We May Think* [Bus45] presenting an utopian machine named *Memex* which allows to cope with the endlessly increasing amount of information and knowledge available: Memex, an extension and index to the human memory, was envisioned as an analogue device able to store all of the human knowledge in one collective memory with the form factor of a desk. By associating and interlinking pages, the fabulous machine allows the user to navigate through pages which are stored on microfilm. This early vision has become – albeit in a different form – to large extents reality in what is known as the *world wide web* composed of its endless amounts of hyper-linked websites. However, today's web is no longer a composition of static text documents only. A great share of the web – more than it was possibly envisioned at the time of writing of Vannevar Bush's article – is multimedia data such as audio data, images, videos, etc.[1]

The term *multimedia* has come to denote a broad concept which can be summarised as *"any combination of text, art, sound, animation, and video delivered to you by computer or other electronic or digitally manipulated means"* [Vau11, p. 0]. Not only does the term carry important technological meaning, but it has also had great sociological implications in the past decades. It is, hence, not surprising that in 1995 the term *multimedia* was awarded being the word of the year of the German language [GfdS17]. In the same year, the German magazine *Der Spiegel* declared the future to be an era of multimedia [Spi95]. The proliferation of smartphones in recent years has added to the multimedia deluge we are confronted with nowadays. The resulting ocean of multimedia data accommodates private and professional content, both worthless and valuable, sometimes curated and often just randomly collected.

[1] For example, [LS16], predicted in the beginning of 2016 that 2.5 trillion photos will have been shared online in 2016. Consider, for instance, [RS17] for further insights with regards to web video data.

To be able to manage this data, the last decade has seen a rise of digital libraries which allow to store all sorts of digital data. Consider, for example, image archives storing professional photography (e.g., the Getty archive[2]), archives with historic photographs (e.g., the ETH image archive[3]), digitised art collections (e.g., the Google Arts & Culture project[4]), archives of internet content (e.g., archive.org[5]), or simply an online encyclopaedia (e.g., wikipedia.org[6]). These archives collect and store a great share of human knowledge often in form of multimedia data.

However, more than ever, to cope with the increasing size of collections, it is crucial to make such collections accessible to retrieval. Given the unbalance that is evidently present in digital libraries nowadays between the amount of data produced and the amount of data processed, services that provide the user with retrieval functionalities are becoming indispensable. As the authors of [LSE02] note, *"what use is the sum of human knowledge if nothing can be found?"* [LSE02]

A librarian who curates data collections and helps in searching items, is no longer able to match today's data flood. Instead, to navigate and search modern times data, sophisticated search engines, such as Google[7] and Bing[8], provide the necessary support. Today, both search engines which started as textual search engines for static web contents, are powered by elaborate, complex algorithms to manage large networks of information and knowledge, and search in a great variety of data including images and videos.

The list of potential applications requiring multimedia retrieval capabilities is long and there seems to be a general need in both the business world and in the everyday, personal context for approaching the problem. The applications relying on multimedia data are manifold and include, for instance,

- medical applications, for instance, to retrieve video recordings from endoscopic surgery for providing explanations to patients and for follow-up operations [SBL+16];
- archaeological applications, for example, for the identification of ancient coins used for research purposes [KHZ09];
- sports applications, for instance, for educational and analysis purposes for coaches of soccer teams [AS13];
- geographical applications, for example, for searching in satellite imagery for specific buildings [CCM17];

[2] http://www.gettyimages.com/
[3] http://www.e-pics.ethz.ch/
[4] http://www.google.com/culturalinstitute/
[5] http://www.archive.org/
[6] http://www.wikipedia.org/
[7] http://www.google.com/
[8] http://www.bing.com/

- musical applications, for instance, for retrieving the name of a song being played [Wan03];
- journalism applications, for example, to be able to find illustrations for a newspaper [JJ05; MS98];
- museum applications, for instance, for providing museum visitors with additional information based on new interaction methods [TLS+16; BMR+07b];
- art applications, for example, to assert pieces of art as not being stolen objects [Art16];
- personal applications, for example, to manage personal photo collections which are cleaned from bad pictures, de-duplicated and made searchable [KSR+06].

In recent years, such applications have spurred research in many fields of computer science and beyond. Multimedia retrieval has become a great challenge to tackle for various areas of research, including

- library research considering, for instance, the modelling of queries and content;
- computer vision research focusing, for example, on the crafting of visual features from image and video data, or approaches for face detection;
- machine learning research, for instance, to learn feature extraction functions to detect discriminable properties of the multimedia documents, to extract semantic concepts or to reason on content and context;
- database research focusing, for example, on the storage of data to allow for an efficient retrieval, on index structures for fast retrieval;
- network research considering, for instance, the provisioning of multimedia content and the distribution over the internet;
- information retrieval research on, for example, the handling of textual documents, etc.;
- user interface research investigating, for instance, approaches for appropriate user interfaces supporting users in their retrieval tasks.

Recently, the field of machine learning has received particular public attention. *Deep Learning* has greatly influenced research in the field of multimedia research and opened up unimagined possibilities in multimedia retrieval. However, these new possibilities form only one side of the coin. While machine learning may publicly be perceived as being the only driving force of multimedia research, in reality, it is only one part of a big puzzle of

equally important research areas with multimedia as the common denominator. Instead, considering multimedia retrieval from a holistic perspective, it also requires support from a data management perspective for the storage of the data, user interfaces that allow to appropriately specify queries for searching in the data, etc. The need for such a holistic perspective has been recognised in the research community; in the Video Browser Showdown (VBS)[9], an international competition on multimedia retrieval systems, the scenario of a user searching for a known item is implemented as a basis for the evaluation. Rather than focusing on only one aspect of the retrieval problem, the competition, hence, compares the full stack of the competing systems. The contest, hence, attempts at comparing the systems based on the efficiency of the system, the extracted data used for searching, the user interface put at the user's disposal, etc.

1.1 Multimedia Data

The different nature of multimedia data when compared to textual data leads to new challenges important to tackle. We identify the following challenges posed by multimedia data:

self-descriptiveness Multimedia data does not provide an explicit content description and must, hence, be annotated to be searchable. Manually annotating data is a time-consuming, gargantuan task. As in the proverb *"a picture is worth a thousand words"*, manually annotating multimedia data will yield subjective, incomplete and inaccurate annotations [DZS+02].

diversity and complexity Because of its very diverse nature, multimedia data cannot be subsumed under one model or structure. Depending on the type of multimedia document, the characteristics of it are very different, and so is the complexity. Contrast, for example, an audio document, which comes with a time dimension, with an image, which has no time information but stores a colour information within two dimensions.

Big Data From a data perspective, multimedia data adheres to the characterisation of Big Data based on the three V's from [Lan01], namely volume, velocity and variety (cf. [MSG+13a; MSG+13b; JWZ+16]). In terms of volume, multimedia data is generally larger in size compared to textual documents (while for example, all works of Shakespeare in textual form will use only a few megabytes, the same space is filled by just a few minutes of an audio recording). Real-time multimedia data and multimedia streams form the velocity part of the definition. With regards to variety, the heterogeneous nature of multimedia data (and the queries), as introduced previously, can be mentioned.

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

relationships Multimedia data comes – depending on the multimedia type – with spatial and temporal relationships [Özs99]. In images, for example, the spatial relationships between objects are possibly very important. Videos additionally have to consider the relationships between these objects over time.

interpretation Multimedia data is generally amenable to multiple interpretations which are dependent on the context. Similarly, queries for retrieving multimedia data are subjective and their interpretation is comparably fuzzy [Nar96].

user access As a consequence of multimedia data being open to interpretation, user access becomes complex and subjective [Özs99]. Queries only seldom need the consideration of full equality between the query object and a multimedia document of the collection. Instead, querying for multimedia data is based on incomplete queries (otherwise the user would anticipate the full result and, hence, would not require querying for it) and the notion of similarity.

These properties of multimedia result in great discrepancies between the various actors and steps involved in the process of recording and searching multimedia data. We identify the following gaps present in the handling of multimedia data (see Figure 1.1):

sensory gap The first challenge arises from the *sensory gap* denoting the fact that a (computational) description from recording a scene may vary due to changes in the context [DJL+08]. For instance, a two-dimensional recording of a three-dimensional scene may yield varying results based on the camera viewpoint, illumination, presence or absence of occlusion, etc.

semantic gap The lack of self-descriptiveness results in what is referred to as the *semantic gap* [SWS+00], which denotes the discrepancy between the low-level content (the simple pixels of an image or samples in audio recording) and the higher-level semantics and interpretations of the content. More precisely, it denotes the lack of coincidence between the extractable visual information and the interpretation in a given context [SWS+00].

expressive gap Considering the user, we identify a discrepancy between the content a user perceives in a multimedia object (based on their personal experiences and expectations), the concepts which are ultimately detectable by the system and, moreover, the user's ability to express these concepts within a query. We refer to this discrepancy as the *expressive gap*. This gap is particularly important for high-level concepts such as emotions, feelings, smells, etc.

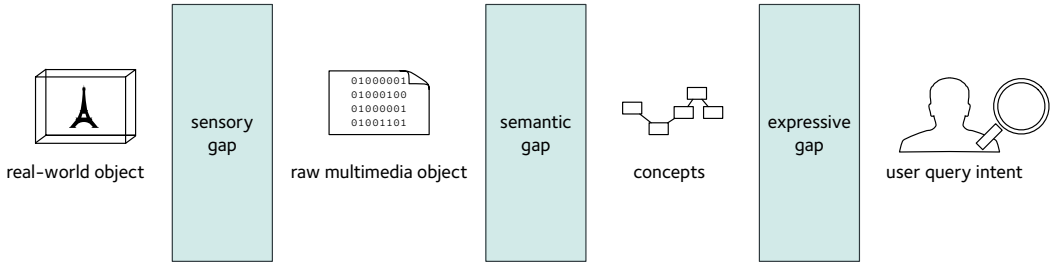


Figure 1.1 Gaps in a multimedia retrieval system: The visualisation depicts the sensory gap between a real-world object and a recorded multimedia content; the semantic gap denotes the discrepancy between the detectable concepts and the raw multimedia data; the expressive gap means the gap between the user intent and the concepts used internally by the system.

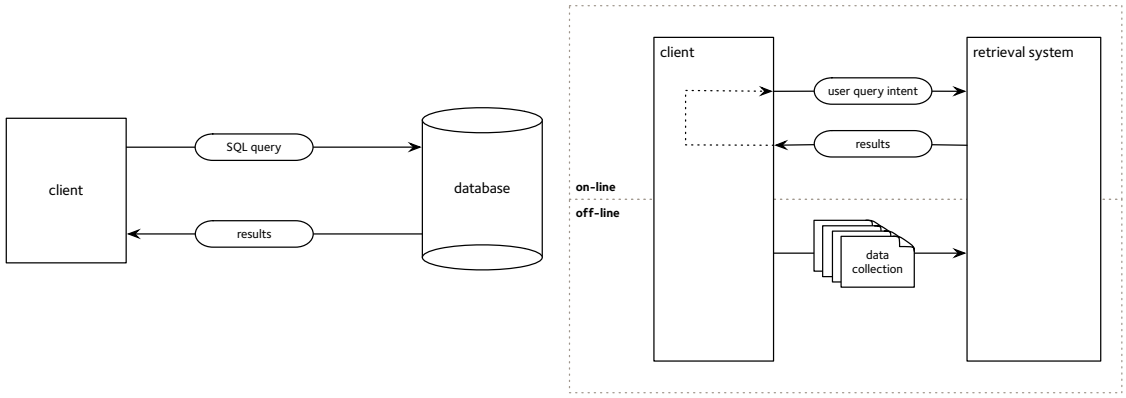
Taking all these properties into consideration, it becomes evident that multimedia data is more complex than simple textual data. This complexity is carried throughout all phases of the recording, processing and – most importantly – retrieval pipeline for multimedia data.

1.2 Focus and Significance of Research

In this thesis, we focus on the data management aspects of multimedia data, in particular in the context of retrieval.

Throughout research literature, it can be noted that in many retrieval applications the main approach to storing multimedia data and metadata is to only make use of the means provided by the file system [AN97] and integrate the data management aspects directly into the retrieval application [Fuh12; Fuh14]. This approach has obviously severe drawbacks as it does not consider the separation of data management aspects from the retrieval logic. Moreover, given that data elements are defined and implemented freely based on the current requirements, the data in retrieval systems does not have a proper structure and is, hence, not self-describing.

In database systems, on the other hand, names and labels are defined at creation time and used throughout [EN11, p. 10]. A database is self-describing in nature providing insulation between programs and data and data abstraction [EN11, pp. 10]. However, given the properties of multimedia data, it is commonly held that there are weaknesses in storing such data in conventional database systems: Traditional data management systems are limited in their support for multimedia data and queries as they have been catered to a very specific setting of structured data and exact-matching queries. Their lack of specialised index structures, missing query paradigms and support for long-running queries makes conventional databases only a second choice as a means for storing multimedia data.



(a) General database processing of a query.

(b) On-line and off-line distinction of a retrieval system.

Figure 1.2 High-level view of a database and a retrieval system.

Thus, while both, database and retrieval systems, grew out of the same need to manage data and make it searchable, there are great differences between both kind of systems. Figure 1.2(a), for example, shows the processing of a general database query in a traditional system. We contrast this view with Figure 1.2(b), a general system's view of an information retrieval system, which distinguishes the off-line loading phase and the on-line querying phase. Starting from these illustrations, in the following, we list differences between traditional databases and retrieval systems (see Table 1.1 for a summary).

Pragmatics In a classical application using a database system, a clear separation of concerns is existent in that the pragmatic aspects (the application logic) are located in the application, while the database system is responsible for data management tasks [Fuh12; Fuh14]. This separation is not present in retrieval systems; instead, information retrieval systems mix application logic and data management aspects.

User As a consequence of the shift in pragmatics, while a user may only seldom access directly a database system, users are envisioned in a retrieval system to interact with it. Hence, in the retrieval system setting, the user is considered to be a non-technical user. In a database setting, instead, a user is an application developer (or an application) using SQL [Wei07].

Data With the advent of more complex data types such as image, video or music data and free text documents, the data is no longer fully structured as it is in traditional database management systems. Such data cannot generally be broken down to a schema or data model with a clear syntax or semantics, particularly as it is often not alphanumeric. Instead,

the data is said to be semi-structured or even unstructured. The tabular view often used in traditional database systems does not apply to such data.

Query Database systems support searching based on single attributes which are compared using a Boolean, exact-matching query. In the retrieval setting, selecting data items only based on Boolean predicates is no longer reasonable: Retrieving a text document using a web search engine which necessitates having as input the full document, or using an image search system that only considers pixel-wise equality, would defeat the purpose of the search engine to find the corresponding item. Retrieval systems, hence, rather support similarity searches which are specified in an imprecise and incomplete way and search within a full document [Son96; ZAD+06, pp. 3]. Given that equality comparisons do not hold, queries in retrieval systems are generally more complex to execute and, therefore, comparably long-running.

Results As a consequence of the supported query paradigms, the results to a retrieval task are said to be relevant to the query posed by the user (rather than matching the query) [Rij79]. The relevancy is often denoted by a score which measures the similarity between the query object and the multimedia document at hand and introduces an ordering in the results [FLN01].

Interaction Rather than following a single-query-single-answer approach as known from database systems, retrieval systems are based on interaction sequences built up by multiple querying steps which make use of techniques such as query expansion, query refinement and relevance feedback to get to the most relevant results for the user. These techniques make querying an explorative task rather than a true search as it is generally found in conventional database systems.

Updates Finally, while database systems can easily handle real-time updates, retrieval systems are more designed to receive updates at off-line time given that the addition of new data items is computationally more intensive. This crystallises into the distinction of two phases within retrieval systems: an on-line, query phase which is intensive in read operations, and an off-line phase which is more focused on write operations to load the system. Figure 1.2(b) summarises both phases.

Table 1.1 Comparison of a database and a retrieval system (based on [Son96; Rij79, pp. 1; ZAD+06, pp. 3; Wei07; Fuh12; Fuh14]).

	Database system	Retrieval system
Pragmatics	separation of concerns	application and data management within the same system
User	developer or application	non-technical, real-world user
Data	structured data following a data model with clear syntax and semantics	semi-structured/unstructured data (e.g., free text document, images, etc.)
Query	unambiguous, attribute-based, Boolean matching	imprecise and incomplete, document-based, similarity matching, long-running
Results	exactly matching w.r.t. query	relevant w.r.t. query; not equal to query
Interaction	single query produces single answer (request/response)	interaction sequence, possibly using relevance feedback
Updates	real-time updates	updates at off-line time, update index

As a consequence of these differences, the predominant approach for multimedia retrieval systems so far has been not to make use of any dedicated data management system for storing the data, but to have an integrated application combining data storage, retrieval and application aspects into one custom-built system [Fuh12; Fuh14] which makes use of the operating system means for storage. This disparity between the approaches (and ultimately also between the research areas) has led to the situation, where advances in the field of databases are not easily applicable to retrieval systems and have therefore mostly been applied only in narrow scenarios or in niche applications. To overcome the current situation, an integrated approach to data management for multimedia data is necessary. The call for new approaches has been raised in the database community already in the 2005 Lowell report on the future of database research [AGG+05], which lists multimedia data management as an important problem to tackle. Following the report, the need has been re-iterated in the database community very many times (e.g., [ACR+05; AGG+05; WLL+15; Wei07; Fuh12; Fuh14]). However, as this call has not yet resulted in any fruitful approaches to solve the data management problem for multimedia retrieval data, the need was raised again in 2016 [JWZ+16] by the multimedia retrieval research community in ten questions for future research. Out of these ten questions five are closely linked to the problem of data management. In particular, the authors call for research in the following areas [JWZ+16]:

- the use of data management techniques for increasingly large collections of multimedia items and the corresponding metadata;

- the applicability of existing query languages to support multimedia queries and the need for novel query languages;

- the support of different workloads by the data management system for performing varying exploration/search tasks;

- the use of data management techniques to improve the quality of result fusions;

- the application of data management techniques to improve the user's interactive experience.

As a consequence of these research questions, the authors note that it is the data management system which makes research in multimedia analytics ultimately scalable.

1.3 Contributions

The disparity between the fields of databases and retrieval systems has lead to a gap in systems for managing multimedia data. In this thesis, we consider the efforts made in both fields of research and attempt at bridging the gap between research in the field of databases and (information) retrieval. Our endeavour will focus on the data management aspects of multimedia data with the objective of

> *organising the data management and storage of multimedia data and the corresponding metadata providing an efficient and effective retrieval.*

This task stands obviously at the intersection and the boundaries of various fields of research, not least the field of computer vision providing the data to store, the field of retrieval desiring a set of query paradigms for providing appropriate answers to users information needs, the field of machine learning using the data for gaining new insights, etc. This situation accumulates a large set of requirements which are not possibly answerable in one single thesis, particularly as there is a multitude of ways in which these requirements can be satisfied. Our objective can, hence, not be to provide a universal, general-purpose approach for multimedia databases, but rather to lay the foundations for a multimedia data management system and to open up an even greater set of research questions in this direction. We will particularly focus on image and video multimedia documents and put less emphasis on systems for text retrieval, as the set of problems for text-based systems is to some extent very different from the challenges present in image and video collections. In this thesis, we consider in particular the following aspects (also based on [JWZ+16]), which have so far only received limited attention in research:

- the logical data model which can be used to model both the structured information and the (unstructured) multimedia document within our data management system;
- the query model allowing to search within the structured information and to perform retrieval based on the multimedia document;
- the processing and execution of a query in an effective and efficient way, particularly in light of the long-running nature of similarity-based queries;
- approaches to store and manage the data from a physical perspective and means to increase the system efficiency, for instance, by means of index structures and the application of distribution.

We distill our approach from the needs for a data management system within a full multimedia retrieval stack and we study, based on the identified requirements, the ingredients for a multimedia data management system. Our approach crystallises into a fully working system which is used within the iMotion/vitrivr multimedia retrieval stack. In this thesis, we do not attempt to answer questions with respect to transactions, recovery, security, etc. in the context of multimedia retrieval, given their secondary importance within our application. Instead, we mainly focus on the integration, data independence, persistence and query support aspects of such a system. This thesis makes, in particular, the following contributions:

1. We identify **the necessary building-blocks** for supporting multimedia data and queries in a multimedia data management system; we present and analyse selected aspects of database and retrieval systems.
2. We define a **blueprint for a multimedia data management system** and its components focused on the retrieval context.
3. We adopt the **relational data model and the vector-space model** to the end of creating an integrated data model for both structured and unstructured data.
4. We put strong emphasis on the **query model** supporting multimedia data and its corresponding metadata and which is specified on a logical and an executional level.
5. We focus on the **efficiency of the system** by considering both distribution and indexing techniques. In particular, we present and compare a large set of state-of-the-art index structures for high-dimensional vector data which we employ in our system.
6. This thesis presents a **working implementation** which has successfully been used in the iMotion/vitrivr project. Moreover, we present a novel user interface which allows to explore the supported query paradimgs in more detail.
7. We present an **evaluation of the introduced concepts** with both synthetic and real data to provide a basis for discussion of the applicability of the concepts presented.

The contributions made are novel with respect to their holistic view our approach takes: Previous attempts have often only considered a very limited scope and only focused on limited aspects of the data or query model, single index structures for high-dimensional data, etc. This thesis considers the problem of multimedia data management from a variety of angles and puts also a strong emphasis on the efficiency of the system by considering both distribution and a large set of index structures for high-dimensional data. Moreover, in contrast to many previous approaches, in this thesis, we start with the supposition that the separation of data management questions from the pragmatics of the retrieval application bears a lot of potential. Research in both multimedia retrieval research and databases has called for such an approach and for the support from the corresponding communities (e.g., [AGG+05; JWZ+16]), however, only little efforts have been made in the past decade. Multimedia databases have remained niche applications in both research and commercially available products as already noted in [Vri99, pp. 7]. This thesis ultimately bridges the gap between the research communities and proposes a solution to the data management problem for the field of multimedia retrieval.

1.4 Outline

This thesis is largely structured in four parts: an introductory part; a part on the foundations with respect to the topic from the multimedia retrieval and the database perspective; a conceptual part introducing the underlying models for a multimedia data management system; a discussion-oriented part which presents the evaluation and related work. In more detail, this thesis is structured as follows:

- Within the first, introductory part of this thesis, we discuss in more detail the aspects of a multimedia data management system and identify the requirements of it (Chapter 2).
- In the foundations part, we introduce in Chapter 3 basic concepts of multimedia retrieval. Chapter 4, on the other hand, presents the foundations of database systems with a large focus on index structures for high-dimensional vector data.
- We model a multimedia data management system in the third part of this thesis from a holistic perspective (Chapter 5): We introduce a blueprint, the data and query model adopted in our system, and consider distribution and storage aspects. Our implementation of the presented concepts in the prototype ADAM$_{pro}$ (*A Da*tabase for *M*ultimedia) is discussed in Chapter 6.
- In the last part of our thesis we present the evaluation of our system (Chapter 7). Chapter 8 discusses related, scientific literature and compares our approach to existing approaches. This thesis concludes with Chapter 9 and an outlook to future work.

I wanted to separate data from programs, because data and instructions are very different.

— Ken Thompson

2

Multimedia Data Management

Increasing collection sizes and data volumes make the situation of managing multimedia data more and more precarious and heighten the need for systems that efficiently and effectively manage such data. In this chapter, we give a bird's eye view and present important aspects and requirements of a multimedia data management system. We first consider in more detail the data and the queries involved, and focus on the overall system design. Following that, we discuss the purpose and the requirements of a multimedia data management system. We conclude this chapter by a number of scenarios that a multimedia data management system should be able to handle.

2.1 Multimedia Data and Metadata

In Section 1.1, we have presented properties that reflect the nature of multimedia data. We have pointed out that the diversity of data is one of many stumbling blocks to solving the problem of multimedia data management. This diversity is due to the various types of data that multimedia data management systems must be able to handle, including [AN97]

- images (photographs, maps, paintings),
- graphic objects (sketches, illustrations, 3D objects),
- animation sequences,
- videos,
- audio,
- texts,
- composite multimedia formed as a combination of two or more of the aforementioned data types.

The very different nature of each multimedia type makes it difficult to define a universal approach compatible with all data types. For example, while video and audio data have

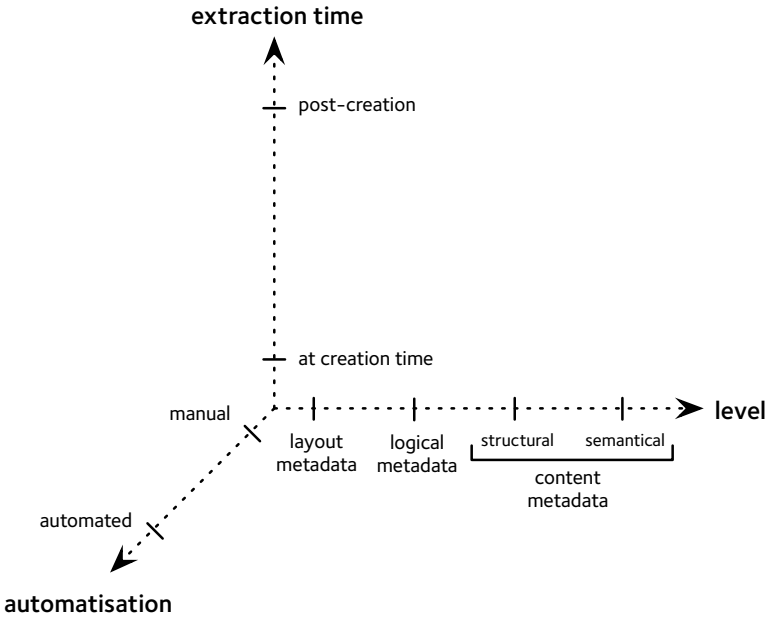


Figure 2.1 Dimensions of the metadata of a multimedia object: The visualisation distinguishes the level of the metadata (layout, logical, content metadata), the time the metadata was extracted (at creation time or post-creation) and the degree of automatisation of the extraction (automated or manual).

a temporal characteristic which has to be considered, audio lacks the spatial component possibly available in both, image and video data.

When considering multimedia data, it is imperative to distinguish the true content data (e.g., image data, audio data, etc.) from the metadata which is often as indispensable as the multimedia object and, hence, also important for the retrieval phase. Based on [VB98; MRT91; DM11], we propose the classification of metadata along the following dimensions, as summarised in Figure 2.1.

Level of metadata The level of metadata distinguishes what the metadata is related to [VB98; MRT91]:

layout metadata The layout metadata describes the document presentation information (e.g., the image type, the colour space, the audio length).

logical metadata The logical metadata describes contextual catalogue information (e.g., recording date, location, content creator).

content metadata The content metadata describes the content of the multimedia object in terms of *structure* (e.g., colour distribution, motion descriptions) and *semantics* (e.g., who/what/when/where information about objects or events).

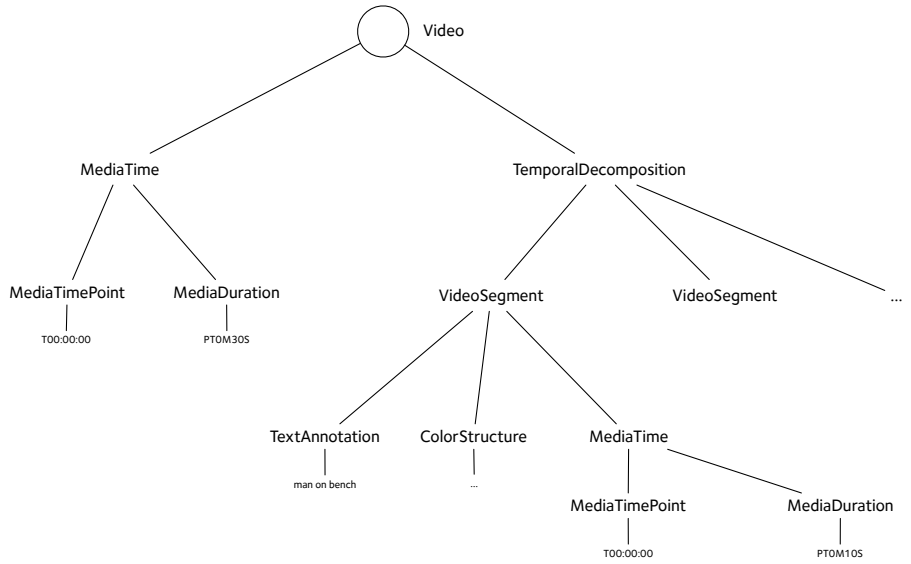


Figure 2.2 MPEG-7 description tree: This example visualises a description tree for a video showing the description of a segment using a textual annotation and colour structure information (based on [HBH+04]).

Extraction time We distinguish metadata generated and recorded at creation time (often automatically) of the multimedia object or extracted post-creation. For instance, the recording time is stored automatically at creation time, while semantical information may be added at a later time when cataloguing the multimedia document.

Automatisation The third dimensionality with respect to the extractable metadata considers the automatisation of the extraction. Certain information can be extracted automatically from a multimedia object (e.g., colour distributions, motion descriptions, etc.), for semantical information the automation can only be achieved in limited ways and manual tagging is necessary.

As a means to standardise multimedia metadata, the MPEG-7 standard was developed. The MPEG-7 standard [DM11] is an ISO/IEC standard [ISO15938-3:2002] proposed by the Moving Picture Experts Group (MPEG) for describing the metadata of a multimedia object. It allows to store the metadata described above, i.e., the layout and logical metadata, together with structural and semantic content metadata. The standard defines *descriptors* allowing to extract *features* or descriptions (e.g., colour distributions, motion descriptions) which characterise a multimedia object, *descriptor schemes* denoting the structure and semantics of the descriptions, and a *description definition language* for the extension of the descriptor schemes. MPEG-7 uses the Extensible Markup Language (XML) for the content description and can be represented as a tree [HBH+04] as shown exemplary in Figure 2.2.

2.2 Multimedia Queries

Querying multimedia metadata necessitates various approaches. While traditional database systems are tailored to searching in the structured information coming from the layout and logical metadata [VB98], this approach is not sufficient for addressing all types of queries involved when providing a database for multimedia data. We distinguish three approaches for retrieval (based on [Rüg10, pp. 13; WNM+95]):

search by attribute Searching by attribute denotes the conventional approach to searching in databases by making use of Boolean predicates which are exactly matched by the results.

retrieval using free text A further approach for retrieval involves searching in the textual data using free text input. A source for searching may either be textual data that has been collected by automatically analysing a multimedia document (e.g., by performing object-character recognition (OCR) or automatic speech recognition (ASR)) or the recognition of concepts within the multimedia document.

similarity retrieval Similarity-based retrieval allows searching in the data based on the contents of a multimedia object. Generally, we distinguish fingerprinting which allows to search in the content of a multimedia document based on a small, unique fingerprint, and content-based searches which allow to search in extracted visual/auditorial properties of a document [DJL+08].

The MPEG-7 standard defines, similar to the description tree of a multimedia document depicted in Figure 2.2, a standard for query trees. A query in an MPEG-7 system should not only return results that exactly match a given query, but also allow to search based on similarity given the conditions specified by the query.

2.3 System Architecture

The MPEG-7 standard defines a framework for processing a query (without defining its components in detail). Figure 2.3 gives an overview of the standardised query processing in an MPEG-7 system. The visualisation displays a user query which is transformed by a query generator into an MPEG-7 query which is then transformed into a query tree that can be executed by a query processor accessing a database storing both the layout and logical metadata (meta base) and the content metadata (feature base). The database stores MPEG-7 data descriptions which have been created through processing multimedia documents by a metadata generator.

While such a solution may certainly be built into one single system, in the following, we project the system architecture – also following to some extent the ideas of the MPEG-7

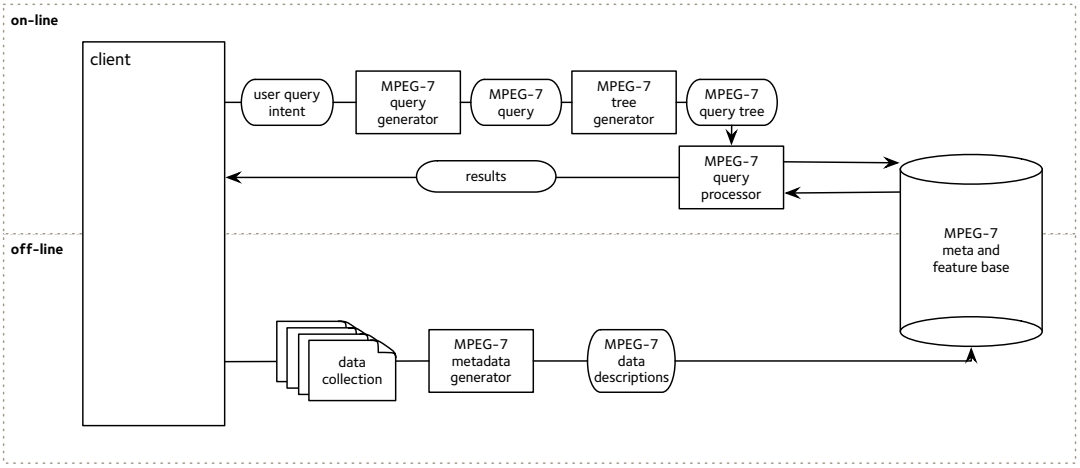


Figure 2.3 Query processing in an MPEG-7 system (based on [KAG10; HBH+04]).

standard – onto multiple systems with well-defined requirements. Generally, the separation between application and core retrieval system, has been advocated, for example, in [Fuh12; Fuh14; JWZ+16] for modern multimedia retrieval systems. Following the ideas mentioned in [Fuh14], we move the pragmatics pertaining to the application logic to a retrieval engine, while the aspects of data management are segregated from the application and only reachable via a generic data management interface. The retrieval engine becomes, hence, responsible for extracting and preparing metadata from the multimedia documents which are then stored in the data management system. The data management system, on the other hand, stores and searches the extracted metadata. The results of the queries are fused by the retrieval engine and returned to the user, while providing access to the multimedia object. Figure 2.4 gives an overview of the positioning of a multimedia data management system within a full retrieval stack.

2.4 Purpose and Requirements of a Multimedia Data Management System

Based on [AN97; Kal00; Vri99, pp. 45], in this section, we analyse the purpose and the requirements of a multimedia data management system.

First, we consider services that are provided by traditional database systems. These include in particular (based on [AN97; Kal00])

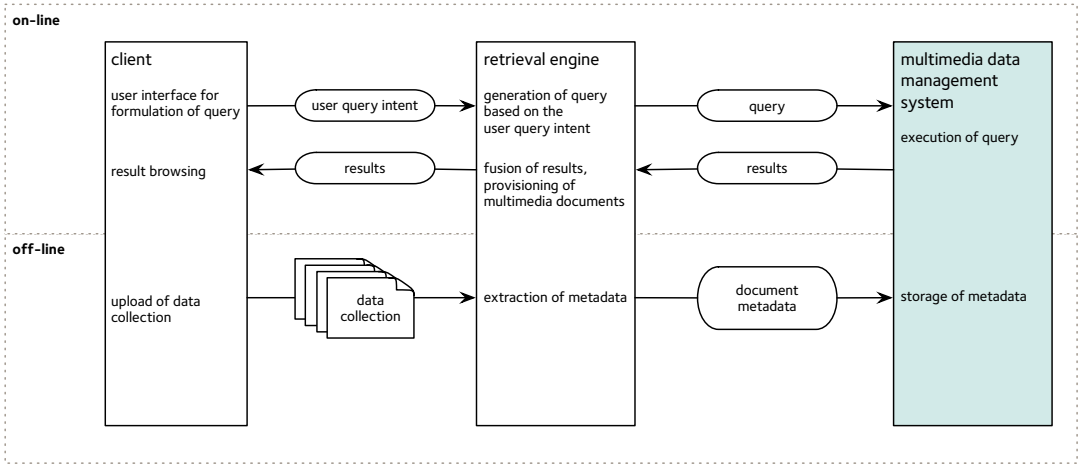


Figure 2.4 Positioning of the data management system within the full retrieval stack: In our system view, we separate the retrieval engine, which we consider being part of the application logic, from the data management logic. The retrieval engine is responsible for extracting metadata and generating queries for the multimedia data management system; the multimedia data management system, on the other hand, stores the metadata and executes queries on it.

- data abstraction ensuring the separation of the logical and the physical models;
- application neutrality ensuring the separation of the database functionalities and the application pragmatics;
- integration of data ensuring the embedding of various data into a single database, without the need to be duplicated for different instantiations;
- integrity control imposes consistency rules from one transaction to another;
- persistence providing the ability to permanently store data objects;
- concurrency control imposing certain guarantees on concurrent transactions and allowing multi-user access;
- fault tolerance ensuring that failing transactions have no effect on the persisted data;
- privacy and authorisation ensure that only authorised accesses and modifications of the data are possible;
- high-level access through query languages.

These services for traditional database systems may be extended by the following services special for multimedia data management:

- support for multimedia data and the corresponding metadata in form of retrievable objects [Vri99, pp. 49];
- retrieval capabilities for both exactly matching queries and similarity-based queries;
- support for large capacity storage management to be able to handle great amounts of multimedia data [Kal00];

- presentation capabilities and support for multimedia interfaces, also for the query formulation [Kaloo];
- support for interactive processes which allow the user to refine queries and results [Kaloo];
- content independence [Vri99, p. 60] denoting the independence of the content metadata extracted and the means to formulate queries on the contents.

Starting from this list of favourable services of a data management system for multimedia data, in the following, we crystallise important high-level requirements of the data management system within the full multimedia retrieval stack. In this thesis, we focus on the following requirements for a multimedia data management system (based on [AN97; KB96, pp. 26]):

- support for both structured and unstructured data, i.e., for data with a fixed schema and data coming from a multimedia document, respectively;
- support for query paradigms for multimedia data, including similarity searching, particularly also in combination with attribute-based, Boolean querying;
- support for scalability to large collections of multimedia data while still providing a high degree of performance.

We note that while the full capabilities of traditional databases are favourable to have for multimedia data management systems, certain topics, such as transaction management, recovery, concurrency control, etc. are only secondary to such systems. Multimedia data management systems are generally not used as full-fledged databases with a high write throughput; instead such systems clearly distinguish the off-line, loading phase which is write-intensive, from the on-line query phase which focuses more on reading data (see Section 1.2). For the remainder of this thesis, our assumption is, hence, that we can clearly distinguish the off-line, loading time from the on-line, querying time. Moreover, we assume that operations performed in the off-line time are not time critical, while the operations performed at query time are.

2.5 Retrieval Scenarios

In the following, we present two exemplary scenarios which we will use in the course of this thesis for presenting and discussing the concepts introduced throughout. Both scenarios build upon the ideas of the iMotion/vitrivr application, which we will introduce as well in this section.

Both scenarios can be categorised as supporting *aimed* [DJL+08] or *targeted search* [SWS+00] being a search where a specific result – a *known item* –, known to exist in the

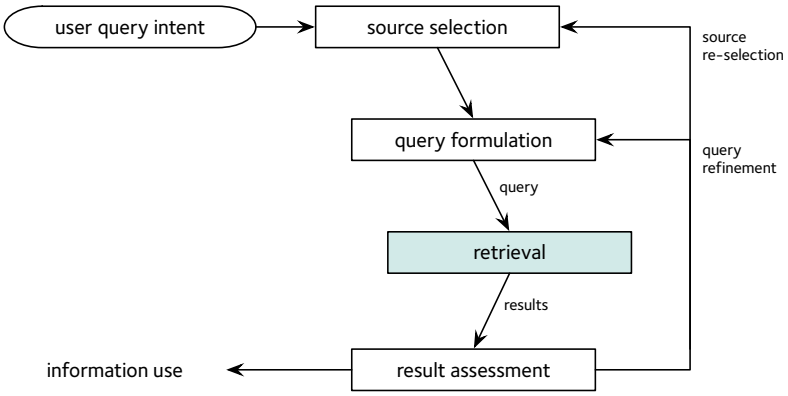


Figure 2.5 Retrieval cycle in answering a user query intent (based on [Rüg10, p. 11; Lin08, p. 29]).

collection, is sought. In contrast to randomly browsing through a list, in an aimed search task, the user has a clear search intent and the accomplishment of the task – by finding the defined end-result – can be clearly stated and its success verified [DJL+08].

Considering the interaction with the system, following [Rüg10, p. 11; Lin08, p. 29], the stages for querying involve a number of interactions between the user and the system as shown in Figure 2.5. More precisely, a user starting with a search intent based on a specific information need may choose a source for answering the question at hand and formulate the query according to the information need. The query is posed to the system performing a retrieval. The results are presented to the user, who may select and assess the returned documents. Possibly, the query might be further refined by reformulating it or selecting a new source, if the currently used source does not provide promising results. This brings the user back to the source selection and the query formulation stage, respectively. The user may loop through the stages until results which satisfy the information need are found (or quit unsatisfied at some point in time the interaction with the system). We will use this interaction framework in the following to describe two retrieval scenarios.

2.5.1 Scenario 1: Film Scenario

Alice is watching a film on the television which she, however, cannot assign to any film she knows. How could she find out what film the scenes she is watching belong to? To address this task, in the most straightforward case, she may use available contextual information stored as logical metadata (e.g., current television program information) to get to an answer to the aforementioned question. However, such information is not always available and, moreover, may require the manual preparation of such data (e.g., making the information available in the television program).

We envision a smartphone application that may help prospective users to answer the

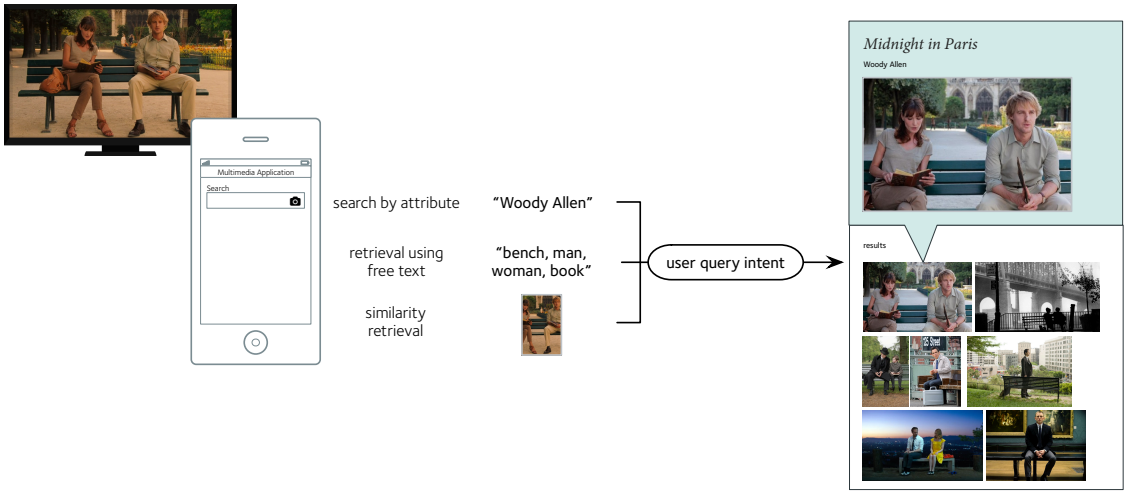


Figure 2.6 **Exemplary film application:** It allows to search based on an attribute, e.g., the director, to search using keywords in the scene descriptions or to take a picture or video of the film.[A]

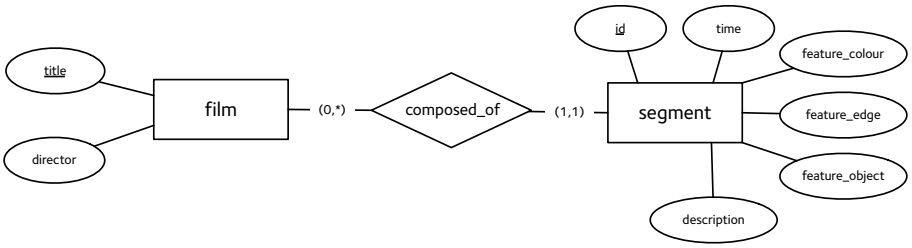


Figure 2.7 **Entity-relationship model of a film retrieval application.**

question by using any kind of available information (see Figure 2.6): Alice may recognise the director by the style of the film (e.g., "Woody Allen"), and search in the logical metadata based on this information. The user may rely on automatically extracted information and search using keywords or free text describing the content of the video scene (e.g., "bench, man, woman, book") or on the basis of metadata collected by automatically analysing a multimedia object (e.g., OCR data or data from the automatic speech recognition). However, such queries pose a media discontinuity and require searching cross-modal (for instance, by using text to search in image data). To stay within the same media, the application could, for example, allow the user to take a picture or a video of the scene using her smartphone and search – based on the snippet taken – in a database of videos for films containing similar scenes.

Figure 2.7 presents a very simplified excerpt of a conceptual view of the data in an entity-relationship model: For each *film* the title and the director is stored. For each visually similar scene extracted from the whole film which we denote as *segment*, we store the timing within the film and a number of extracted visual properties, such as colour features, edge features and object features (see Section 3.3.2).

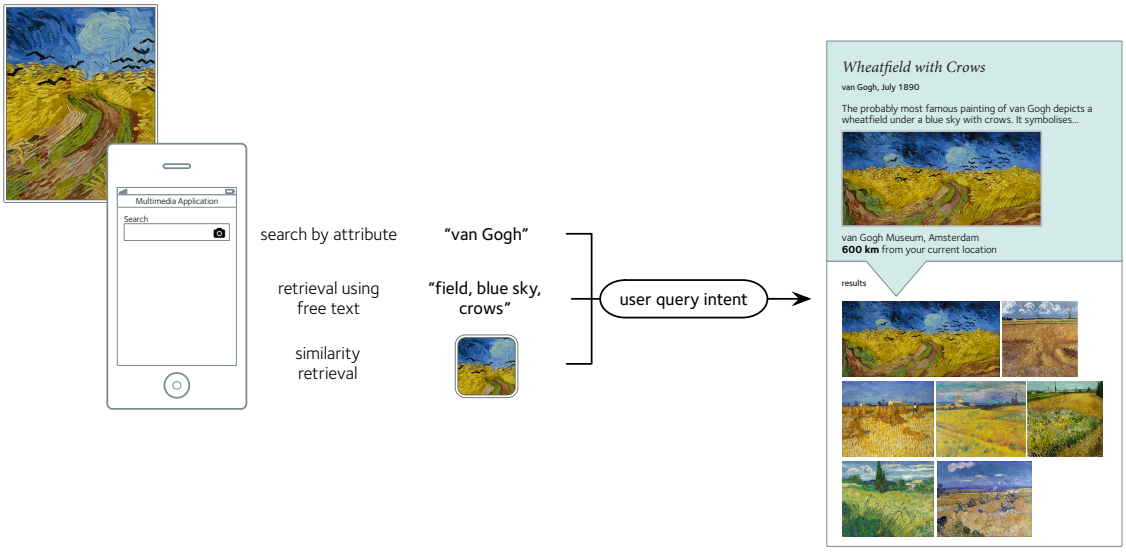


Figure 2.8 **Exemplary art application:** It allows to search based on the painter's name, search using keywords based on the description of the painting or using a picture of the painting.[B]

In this scenario, the following interaction sequence is involved in the process of the use case:

1. To satisfy her user information need, Alice may start her film retrieval application. She formulates the query according to her needs and context, e.g., by specifying the director's name, by describing the objects recognised in the scene, or by taking a photography of the television screen.
2. The system answers with a result list of result elements ordered by similarity with respect to the query. For each result element a preview image is displayed together with the name of the film and the director's name.
3. Alice assesses the quality of the results and decides on refining the query or exiting the query loop.

2.5.2 Scenario 2: Art Scenario

Bob sees while walking in Basel a poster for a van Gogh exhibition starting soon. He likes the painting, but does not know its name or where it is exposed. He may use his smartphone application to search based on the painter's name or based on a photograph of the poster for more information on the painting and where it is currently shown. Moreover, he may look at similar paintings in museums which are as close as possible to him. Figure 2.8 summarises the use case.

In Figure 2.9, we show a conceptual view of the data model. The entity-relationship model stores for a *painting* its title, the name of the painter and a description; moreover,

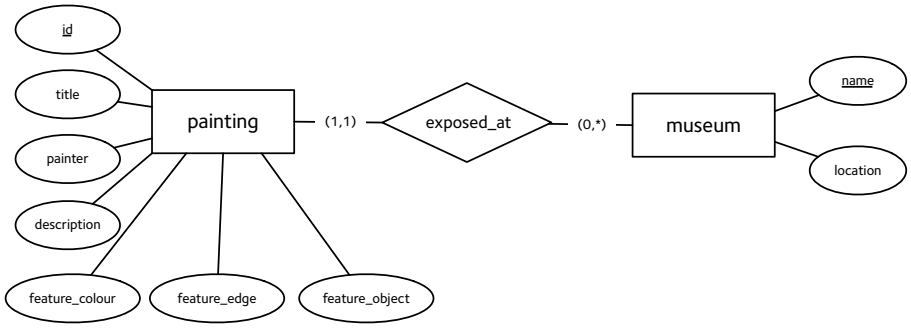


Figure 2.9 Entity-relationship model of an art retrieval application.

certain features are extracted describing the visual characteristics of the paintings. For each painting, we store at which *museum* it is exposed. The museums are identified by their name; we store the location of the museum (e.g., as a GPS coordinate).

The following steps are involved in the use case:

1. To satisfy his user information need, Bob opens the art application. He formulates the query according to his information need and context, e.g., by means of a photography of the painting, or by using textual descriptions of visible concepts or based on the metadata (e.g., the painter's name as recognised by Bob), and submits the query to the retrieval system. His current location is submitted to the system as well.
2. The system answers with a list of result elements ordered by similarity with respect to the query. For each result element, he receives together with a preview image more information (the title, the painter's name and a description) on the painting. Moreover, for each result, the distance to the museum exposing the painting is displayed.
3. Bob assesses the quality of the retrieval results and decides on refining the query or exiting the query loop.

2.5.3 iMotion/vitrivr System

The iMotion/vitrivr system is a large-scale multimedia retrieval system [RGS+15; RGT+16a; RGH+16b; RGT+17b; GRT+17; TRG+16; RGG+18; Gas17; RGG+17] which comes with a great variety of different query paradigms for searching in multimedia collections. The system is able to handle not only video data, but also image and audio data, together with three-dimensional models.

With iMotion/vitrivr, we present a fully working system which provides, on the one hand, searching on the basis of automatically collected metadata (e.g., content author, video length, etc.) or manually added tags describing the content of the multimedia object. On the other hand, it supports similarity-based queries on the basis of the content metadata, for instance, by *Query-by-Sketch (QbS)*, *Query-by-Example (QbE)*, querying by motion, querying by audio and concept-based retrieval. For the retrieval, this means that users can

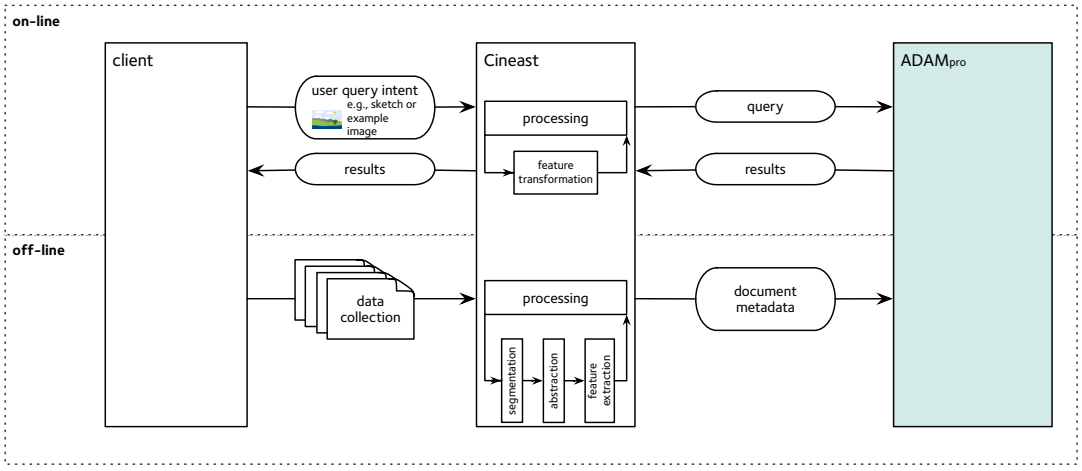


Figure 2.10 Architecture of the iMotion/vitrivr system: The system is divided into the retrieval engine Cineast and the data management system ADAM$_{pro}$.

specify either an existing image or a video snippet as a query, or provide the system with a hand-drawn sketch of the most relevant items, or draw motion or record an audio snippet to search for. Moreover, users can specify the concepts (e.g., detected objects) that should be present in the results.

For supporting retrieval, iMotion/vitrivr makes use of a large number of descriptors [RGS+15] which extract from the multimedia object inherent visual properties. The descriptors include a colour moments descriptor, a colour layout descriptor, an edge histogram descriptor, a directional motion histogram descriptor, etc. [RGS14]. The content metadata resulting from these descriptors are stored as high-dimensional dense data points (with up to over several hundred dimensions) and used at query time for comparison to the query object.

From an architectural perspective (see Figure 2.10), the iMotion/vitrivr system is composed of a retrieval engine, Cineast [RGS14], which is responsible for extracting the visual properties from the multimedia objects. The extracted metadata is stored within ADAM$_{pro}$ which forms the data management layer of the retrieval stack described in more detail in this thesis. ADAM$_{pro}$ is responsible for both, storing and retrieving the metadata.

A screenshot of the iMotion/vitrivr user interface for posing queries to the system is shown in Figure 2.11.

Figure 2.11 **Screenshot of the iMotion/vitrivr application:** The application allows to input a sketch and search using the sketch for videos depicting similar scenes to the one drawn.

Part II

Foundations

"And what is the use of a book," thought Alice, "without pictures or conversations?"

— Lewis Carroll, *Alice's Adventures in Wonderland*

3

Foundations of Multimedia Retrieval

In the following chapter, we focus on the foundations of multimedia retrieval systems and particularly highlight similarity-based searches on the content level which form a valuable property of retrieval systems.

In order to make a collection searchable from a content point of view, we have to consider how to compare incomplete and imprecise queries to multimedia documents. In the light of this objective, the straightforward way would be to perform a full equality comparison between the query and a multimedia document. However, as noted previously, this approach is not feasible. Consider, for example, searching in an image collection using a sketch; a pixel-wise equality comparison would not be successful, as no image would exactly match the sketch at hand. Moreover, a pixel-wise comparison would put a high burden on the system and prevent to scale to large collection sizes. Instead, in the following chapter, we elaborate on the necessary aspects to perform similarity queries.

We first start by presenting a general architectural view of a retrieval system and then focus on a generic retrieval model which we expound on in this chapter. The field of information retrieval knows a few classical retrieval models which are very well documented in literature (e.g., [BR11, pp. 66]). The most common models include, for instance, the Boolean retrieval model, the probabilistic retrieval model, the fuzzy retrieval model and the *vector-space retrieval model* which we will concentrate on in this chapter. We will present applications of the vector-space retrieval model using various types of multimedia and detail how to approach the problem of multimedia retrieval. For this, we focus on image and video data and skip details on audio processing, which are, however, also compatible with the selected model. This chapter closes by a survey of related work in multimedia applications.

3.1 Retrieval System Architecture

From an architectural point of view, a retrieval system can be separated into two parts: The *off-line*, loading-centric part, is characterised by the system being loaded with data and the *on-line* part, on the other hand, being the query-centric phase.

In Figure 3.1, we present an architectural view of a retrieval system. It depicts the off-line processing step, which produces comparable representations of multimedia documents. The processing step may involve some pre-processing (e.g., the segmentation of a video) and a feature extraction step in which a set of inherent properties, generally termed *features* or *descriptions*, of the multimedia document are extracted. The extracted features are later on used for comparison between a query and a multimedia document. The document collection stores the extracted features in a retrieval index[1]; additionally, the document collection may also store logical or structural metadata on the multimedia document, for instance, for presentational purposes. At query time the extracted features are queried using a query which has been processed as well to be comparable to the document representations; the information on the possibly relevant documents is retrieved and presented in a ranked list to the client.

In the following, we detail both the off-line loading and the on-line querying process. Consider a user (or, for instance, a web crawler) adding a document to the retrieval system.

1. The retrieval system processes the document and extracts the relevant information.
2. The document is stored together with the extracted features into the document collection.

As soon as the data is inserted into the document collection, a user is able to perform queries on the system:

1. A user of the retrieval system formulates a query which is sent to the retrieval system.
2. The user query intent is transformed into a query representation which is comparable to the representation of documents stored at off-line time.
3. The document collection is queried using the query representation; the relevant documents are retrieved and ranked according to the query.
4. The results are returned, possibly with additional information, to the client.

[1] The term *index* has a slightly different meaning in the context of retrieval than in the database context. While in the database discourse it denotes an auxiliary structure which allows to increase the performance of the system, in the retrieval context it has come to describe the primary storage data structure for the extracted content metadata.

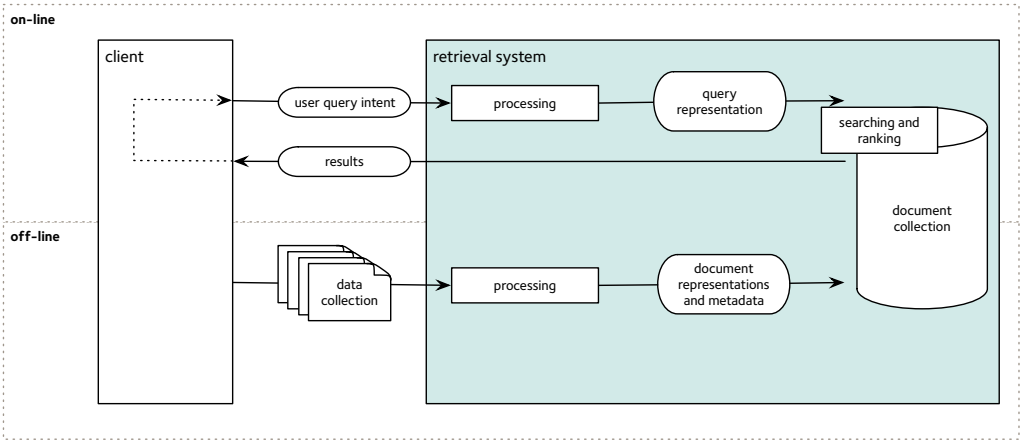


Figure 3.1 Architecture of a retrieval system: In the off-line phase, the multimedia documents added to the system are processed and stored in the document collection; in the on-line phase, the document collection is searched to answer the user query with ranked results.

3.2 Retrieval Models

Retrieval models provide a theoretical framework and mathematical basis for querying unstructured data as found, for example, in the context of multimedia. We start from the notion of a document (e.g., a textual document, an image, a video) to be stored within a retrieval system. A document in the given context is processed in its entirety; in particular, it is generally considered not to have any special internal structure.

The multimedia document $o \in \mathcal{O}$ is embedded into a feature space using a *feature transformation function* $f \in \mathcal{F}$ into a unified space $\mathbb{U}$ and stored as a (simplified) *representation of a document* $\boldsymbol{d}$ in a document collection $\mathcal{D}$. Similarly, the user query intents $qi \in \mathcal{QI}$ are transformed by using a feature transformation function into the same feature space $\mathbb{U}$. A *query representation* $\boldsymbol{q} \in \mathcal{Q}$ is used as reference against which the document representations are compared. Hence, both the set of representations of documents and the set of representations of queries need to be embedded into the same space. We, hence, ultimately require that

$$\mathcal{D} \subset \mathbb{U} \quad \text{and} \quad \mathcal{Q} \subset \mathbb{U} \tag{3.1}$$

i.e., both the documents and the queries are drawn from the same universe of valid representations. With the multimedia objects and the queries being translated into the unified retrieval model, the query is compared to the stored documents using a comparison function ($\varsigma(\cdot,\cdot)$ or $\delta(\cdot,\cdot)$) which returns a similarity or distance measure denoting the similarity or distance, respectively, of the query to the multimedia document. Considering these elements, in Definition 3.1, we present a definition of a generic retrieval model.

Definition 3.1 Generic retrieval model (based on [BR11, p. 58]).

A retrieval model is a tuple

$$[\mathcal{D}, \mathcal{Q}, \mathcal{F}, \varsigma(\cdot,\cdot)]$$

where

$\mathcal{D}$ denotes a set of representations of documents in a collection, with $\boldsymbol{d} \in \mathcal{D} \subset \mathbb{U}$;

$\mathcal{Q}$ denotes a set of representations of user query intents, with $\boldsymbol{q} \in \mathcal{Q} \subset \mathbb{U}$;

$\mathcal{F}$ denotes a framework to model the document representations and the queries with a feature transformation function $f(\cdot) \in \mathcal{F}$;

$\varsigma(\cdot,\cdot)$ or $\delta(\cdot,\cdot)$ is a comparison function, given as a similarity function or a distance function, respectively.

Representations of documents The set of representations of documents (*document collection*) $\mathcal{D}$ constitutes the collection that is indexed by the retrieval system. A representation of a document $\boldsymbol{d} \in \mathcal{D}$ is often also termed *feature* or *description*. A feature is a concise and aggregated description of an inherent property reflecting the content of a multimedia object; moreover, it is supposed to be diverse enough throughout the collection to distinguish the multimedia objects from each other. For transforming a document to its representation within the retrieval system there exists a feature transformation function

$$f_{\mathcal{O}} : \mathcal{O} \to \mathbb{U} \tag{3.2}$$

which is strongly dependent on the multimedia type; a set of standardised feature extractors is available in literature and we will give more details in Section 3.3.

Representations of user query intents The user query intents $qi \in \mathcal{QI}$ result in query representations $\boldsymbol{q} \in \mathcal{Q}$, where $\mathcal{Q}$ is a subset of the universe of valid representations $\mathbb{U}$, when transformed using the feature transformation function. There exists a mapping

$$f_{\mathcal{QI}} : \mathcal{QI} \to \mathbb{U} \tag{3.3}$$

which maps a user query intent to a query specification in the retrieval model at hand. Note that a query object is not necessarily an element of $\mathcal{D}$.

Feature transformation The creation of a representation of a document or query is a task known as *feature transformation*. The class of functions to perform a feature transformation is composed of generally non-isomorphic functions which allow to adapt a document to

the retrieval framework in use. We define the feature transformation as a function

$$f : \left.\begin{matrix} \mathcal{O} \\ \mathcal{QI} \end{matrix}\right\} \to \mathbb{U} \tag{3.4}$$

The feature transformation step may involve complex transformations, for example, the segmentation of a multimedia object into smaller parts, a feature extraction step, normalisations, etc. The feature transformation not only allows the adaption of a document to the retrieval model used. Moreover, it reduces the search complexity since it avoids the full inspection of documents, but only considers the extracted features for comparison.

Comparison function We have noted previously that equality comparisons are not sensible in the given retrieval context, but rather comparison functions denoting the (dis-)similarity between the query object and a multimedia document are more useful. The comparison measure, which is is highly dependent on the needs of the application and the data domain, allows to compare document and query representations and assign a numeric value to the comparison. In this way, it creates an ordering amongst the document collection with respect to a query. In its most general form, a comparison function is a function

$$\mathbb{U} \times \mathbb{U} \to \mathbb{R} \tag{3.5}$$

allowing to compare queries to documents, but also documents among each other. A similarity function is a function of the form

$$\varsigma : \mathbb{U} \times \mathbb{U} \to [0, 1] \tag{3.6}$$

with 0 being maximally dissimilar and 1 denoting equality. Often, a distance function $\delta(\cdot, \cdot)$, differently said, a measure of *dis*similarity rather than a similarity measure, is used to measure resemblance and create a ranking in the documents of a collection. A distance function $\delta(\cdot, \cdot)$ takes the form

$$\delta : \mathbb{U} \times \mathbb{U} \to [0, \infty) = \mathbb{R}_0^+ \tag{3.7}$$

where a distance of 0 denotes equality and a distance of ∞ denotes maximal dissimilarity. We use $\bar{\delta}$ to denote the evaluation of a distance function $\delta(\cdot, \cdot)$. In the remainder of this thesis, we will mostly concentrate on the use of distance functions rather than similarity functions.

For two elements $\boldsymbol{u}_i, \boldsymbol{u}_j \in \mathbb{U}$ (e.g., a query and a document representation), the relation between a similarity and a distance function is given by

$$\delta(\boldsymbol{u}_i, \boldsymbol{u}_j) = 0 \Leftrightarrow \varsigma(\boldsymbol{u}_i, \boldsymbol{u}_j) = 1 \tag{3.8}$$

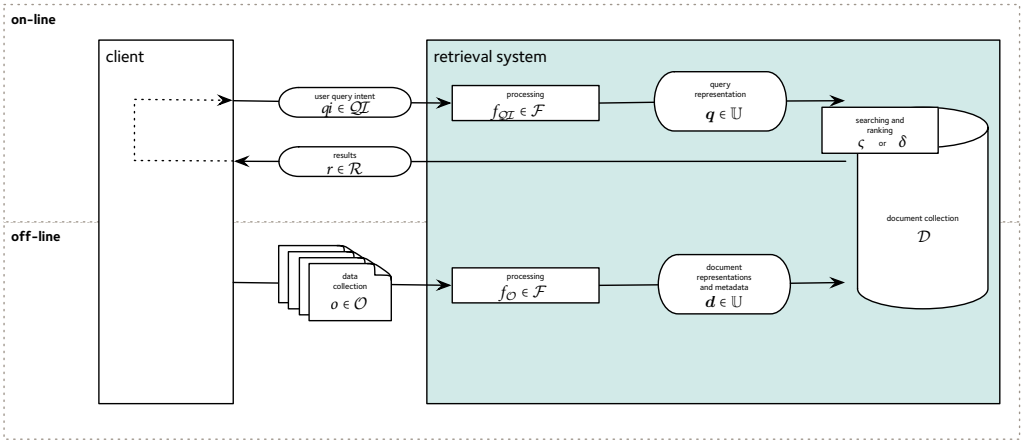


Figure 3.2 Architectural view with the parts of retrieval model specified (based on Figure 3.1).

The transformation from a distance to a similarity score is not straightforward and difficult to generalise, as a distance to similarity correspondence function is dependent on both the data and the distance measure; in particular, the fact that the distance measure is generally not bounded makes the translation from the one score to the other a difficult task to achieve.

A distance metric should generally satisfy the following properties [Spr14, pp. 127]:

- $\forall \boldsymbol{u}_i, \boldsymbol{u}_j \in \mathbb{U} : \delta(\boldsymbol{u}_i, \boldsymbol{u}_j) \geq 0$ non-negativity;
- $\forall \boldsymbol{u}_i, \boldsymbol{u}_j \in \mathbb{U} : \delta(\boldsymbol{u}_i, \boldsymbol{u}_j) = 0 \Leftrightarrow \boldsymbol{u}_i = \boldsymbol{u}_j$ zero for identity, i.e., only if two objects are exactly equal the distance should yield zero, otherwise it should not;
- $\forall \boldsymbol{u}_i, \boldsymbol{u}_j \in \mathbb{U} : \delta(\boldsymbol{u}_i, \boldsymbol{u}_j) = \delta(\boldsymbol{u}_j, \boldsymbol{u}_i)$ symmetry, i.e., a distance function should reflect the real world notion of distances by which it does not matter whether going from point i to point j or vice-versa;
- $\forall \boldsymbol{u}_i, \boldsymbol{u}_j, \boldsymbol{u}_k \in \mathbb{U} : \delta(\boldsymbol{u}_i, \boldsymbol{u}_k) \leq \delta(\boldsymbol{u}_i, \boldsymbol{u}_j) + \delta(\boldsymbol{u}_j, \boldsymbol{u}_k)$ triangle inequality, i.e., a distance function can be bounded, as the direct distance between two points should always be smaller/equal to a distance between the two points with an additional hop.

To summarise, in Figure 3.2, we extend the architectural view presented in Figure 3.1, and specify the parts of the model within the illustration.

3.2.1 Retrieval Operations

Intuitively, a good retrieval system should present relevant documents first, followed by less relevant documents. By its definition, a similarity-based search will have to return all elements of the collection, as all elements "match" to a certain degree the query (given by the comparison function). However, returning all documents of a database – even if

sorted in ascending order by distance – to the user is not sensible, because the user will most probably not be interested in browsing through the whole collection. Similarly, for comparison, a web search engine returns the results – in paged form – only up to a certain number of total results as well. For a query vector $\boldsymbol{q}$ and a collection $\mathcal{D}$, we, hence, return a result set $\mathcal{R}$

$$(\boldsymbol{q}, \mathcal{D}) \rightarrow \mathcal{R} \tag{3.9}$$

which is smaller than the full set of documents ($|\mathcal{R}| \ll |\mathcal{D}|$) and for which the contents of it have a score attached according to which the set can be sorted. Depending on the user need, the returned results may satisfy the similarity predicate up to a certain constraint specified as a limiting predicate, for example, absolutely by a maximum distance ε, or relatively by a maximum number of elements to return κ. In the following, we present the two most prominent query modes, ε nearest neighbour querying and κ nearest neighbour querying. We refer to [SHS+08] for an overview of further – less prominent – retrieval operations, including reverse nearest neighbour, continuous nearest neighbour, group nearest neighbour, etc.

3.2.1.1 ε Nearest Neighbour Query

The ε nearest neighbour (εNN) query is defined by an absolutely specified distance threshold which delimits the region of the data space interesting to the user. Considering the use of a distance measure, the response of such a query returns objects of the collection that are dissimilar from the reference up to a given threshold ε.

$$\varepsilon NN(\boldsymbol{q}, \mathcal{D}, \varepsilon) = \{\boldsymbol{d} \mid \forall \boldsymbol{d} \in \mathcal{D} \wedge \delta(\boldsymbol{q}, \boldsymbol{d}) < \varepsilon\} \tag{3.10}$$

εNN queries can be extended to range queries by not only specifying upper distance bounds, but also lower distance bounds, i.e.,

$$\varepsilon NN(\boldsymbol{q}, \mathcal{D}, \varepsilon_{\text{lower}}, \varepsilon_{\text{upper}}) = \{\boldsymbol{d} \mid \forall \boldsymbol{d} \in \mathcal{D} \wedge \delta(\boldsymbol{q}, \boldsymbol{d}) > \varepsilon_{\text{lower}} \ \wedge \ \delta(\boldsymbol{q}, \boldsymbol{d}) < \varepsilon_{\text{upper}}\} \tag{3.11}$$

εNN queries are usually applied in geographical applications, in which a user might be interested in all items within a certain distance. In multimedia retrieval applications fixing an absolute distance is, in contrast, only difficult to achieve as the distance value is very much dependent on the data and the distance measure and not intuitively comprehensible. Figure 3.3(a) gives a visualisation of an εNN query.

Example 3.1 ε **nearest neighbour query:** Application in a geographical context.

Bob is standing at the Marktplatz in Basel and looking for all museums within one kilometre. As his application stores the location coordinates for the museums, he is able to compute the distance from his current location to all museums. He filters for all museums which are within 1 km and, hence, he performs an εNN query with $\varepsilon = 1$ km. The following table up to rank four exemplifies the data selected.

Rank	Museum Name	Distance
1	Pharmazie-Historisches Museum	150 m
2	Naturhistorisches Museum	290 m
3	Museum der Kulturen	400 m
4	Kunstmuseum Basel	800 m
...	...	...
...	Fondation Beyeler	6.4 km

3.2.1.2 κ Nearest Neighbour Query

As noted previously, the absolute specification of a maximum distance often poses difficulties, in particular if there exists no knowledge on the distance function and the data distribution. The κ nearest neighbour (κNN) query type, hence, considers rather the number of results to return and specifies a cut-off value κ after which the sorted list of results becomes uninteresting to the user [CNB+01], rather than the absolute distance.

For introducing the κNN search, we start from the definition of a (single) nearest neighbour defined as

$$NN(\boldsymbol{q}, \mathcal{D}) = \{\boldsymbol{d} \mid \forall \boldsymbol{d} \in \mathcal{D} \wedge \nexists \boldsymbol{d}' \in \mathcal{D} \ \wedge \ \boldsymbol{d} \neq \boldsymbol{d}' \ \wedge \ \delta(\boldsymbol{q}, \boldsymbol{d}') < \delta(\boldsymbol{q}, \boldsymbol{d})\} \tag{3.12}$$

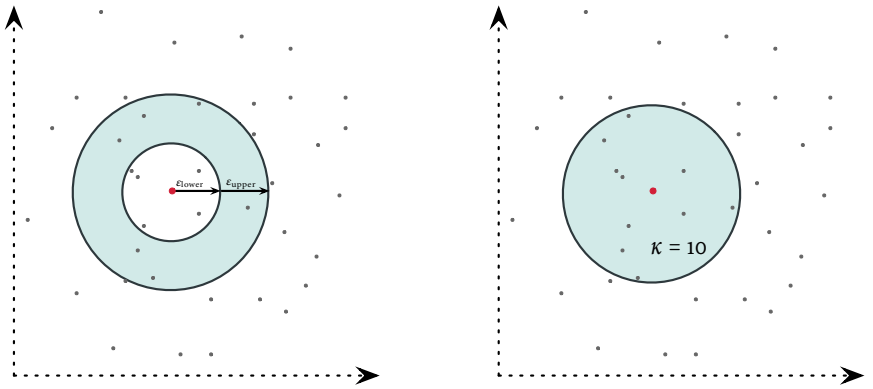
i.e., it finds the data point $\boldsymbol{d}$ which is closer to query vector $\boldsymbol{q}$ than any other data point $\boldsymbol{d}'$. In a simpler notation, we may say

$$NN(\boldsymbol{q}, \mathcal{D}) = \underset{\boldsymbol{d} \in \mathcal{D}}{\operatorname{argmin}}\, \delta(\boldsymbol{d}, \boldsymbol{q}) \tag{3.13}$$

We extend the definition of the nearest neighbour to consider the κ "closest" documents which might be interesting to the user:

$$\kappa NN(\boldsymbol{q}, \mathcal{D}, \kappa) = \{\boldsymbol{d} \mid \forall \boldsymbol{d} \in \mathcal{R} \subset \mathcal{D} \ \wedge \ |\mathcal{R}| = \kappa \ \wedge \ \nexists \boldsymbol{d}' \in (\mathcal{D} - \mathcal{R}) \wedge \delta(\boldsymbol{q}, \boldsymbol{d}') < \delta(\boldsymbol{q}, \boldsymbol{d})\} \tag{3.14}$$

In the case of ties, we are satisfied with any set of κ elements which satisfies the conditions. Figure 3.3(b) gives a visualisation of a κNN query.



(a) Visualisation of a range query based on εNN queries.

(b) Visualisation of a κNN query.

Figure 3.3 Visualisation of retrieval operations in retrieval systems.

Example 3.2 κ nearest neighbour query.

Alice uses her smartphone application to identify the film currently being played on TV. She does not want to spend too much time in looking at the result list and, hence, decides that she will consider at most 5 result items, as it sounds like a reasonable number of elements to scan through. She, hence, performs a similarity search for at most $\kappa = 3$ items. The following table exemplifies the data selected.

Rank	Film Title	Distance
1	Midnight in Paris	0.2
2	Finding Neverland	0.7
3	500 Days of Summer	1.6
...	...	...
...	Manhattan	12.8

In certain applications, it might be sufficient to retrieve *any one* of the closest elements to the query, rather than the nearest neighbour. We define such an approximative nearest neighbour query $\widetilde{NN}(\boldsymbol{q})$ as

$$\widetilde{NN}(\boldsymbol{q}) = \{\boldsymbol{d} \in \mathcal{D} \mid \forall \boldsymbol{d}' \in \mathcal{D} : \boldsymbol{d} \neq \boldsymbol{d}' \wedge \delta(\boldsymbol{d}, \boldsymbol{q}) < (1 + \chi)\delta(\boldsymbol{d}', \boldsymbol{q})\} \tag{3.15}$$

where χ denotes the degree of inaccuracy. While obviously multiple elements from the collection may satisfy this condition, it is sufficient to return one element out of all. This definition can obviously be extended to also accommodate approximate κNN queries.

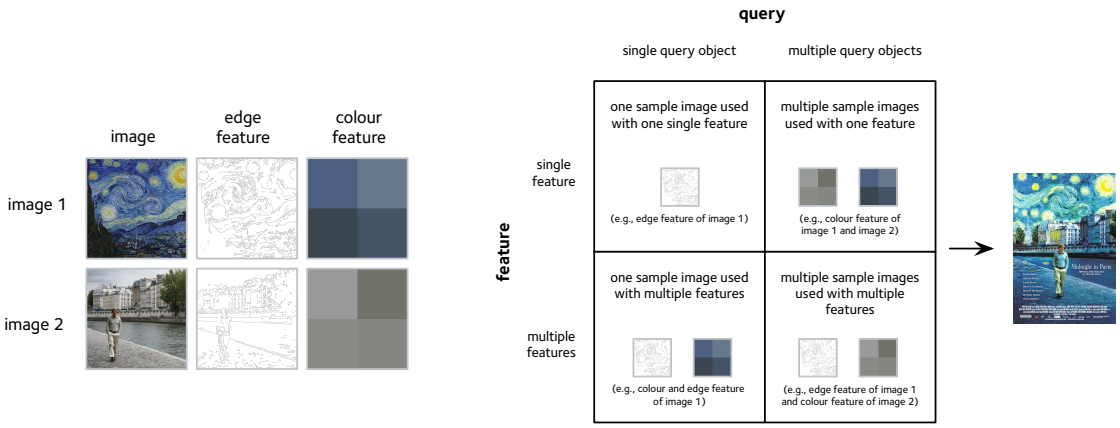


Figure 3.4 Matrix of complex similarity queries in the context of image retrieval: By combining query objects and features, possibly the results may be adjusted to satisfy more complex user query intents (based on [BMS+01]).[C]

3.2.2 Complex Queries

So far, we have only considered simple queries based on a single query vector $\boldsymbol{q} \in \mathcal{Q}$

$$(\boldsymbol{q}, \mathcal{D}) \rightarrow \mathcal{R} \tag{3.16}$$

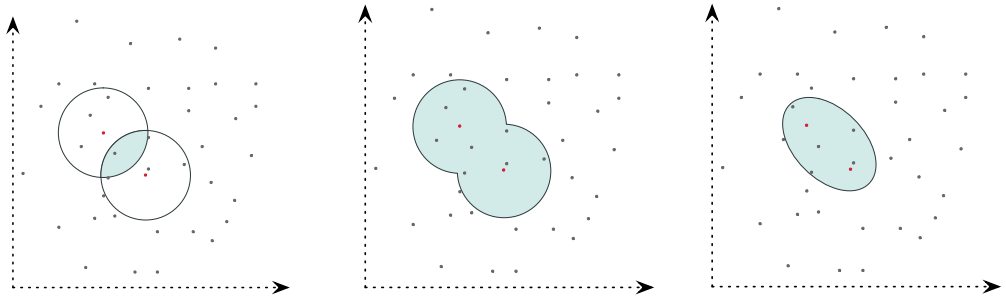
In the following, we address queries which are more complex in their nature and are composed of multiple query objects or multiple features. We consider a set of query vectors $\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_m\}$ (e.g., denoting different objects or different features), possibly applied on different collections $\{\mathcal{D}_1, \ldots, \mathcal{D}_m\}$, i.e.,

$$(\boldsymbol{q}_1, \mathcal{D}_1) \times (\boldsymbol{q}_2, \mathcal{D}_2) \times \cdots \times (\boldsymbol{q}_m, \mathcal{D}_m) \rightarrow \mathcal{R} \tag{3.17}$$

To retrieve a result set which satisfies all similarity predicates, it is ultimately necessary to define a distance combining function $\dot{\delta}$ which aggregates multiple distances $\delta \in \mathbb{R}_o^+$ to one single distance, i.e.,

$$\dot{\delta} : \mathbb{R}_o^+ \times \mathbb{R}_o^+ \times \cdots \times \mathbb{R}_o^+ \rightarrow \mathbb{R}_o^+ \tag{3.18}$$

We follow the taxonomy as introduced in [BMS+01] and illustrated in Figure 3.4 with respect to multi-object and multi-feature queries. Distance combining functions can operate on multiple query objects and features and combine the results from the single scans into one coherent result set. For the distance combining function, the interpretation whether multiple objects or multiple features are used in the query is not relevant. However, by the choice of the distance combining function a desired semantics can be expressed. Based on [BMS+01], we introduce three distance combining functions. In Figure 3.5, we illustrate the result of these functions.



(a) Fuzzy-and. (b) Fuzzy-or. (c) Weighted average.

Figure 3.5 Visualisation of distance combining functions: The visualisation uses the Euclidean distance (i.e., a Minkowski distance with order $p = 2$) to present the results (highlighted) of the distance combining function (based on [Spr14, p. 135]).

Fuzzy-and Consider a set of query points $\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_m\} \subset \mathcal{Q}'$; using a fuzzy-and operation, a data point should be close to all query points (as shown in Figure 3.5(a)). More precisely, the maximum distance to any of the query points should be minimised, as it is not sufficient to be only similar to one of the query points. Formally,

$$\dot{\delta}_{\text{and}} = \max_{i=0}^{m} \bar{\delta}_i \tag{3.19}$$

Fuzzy-or On the other hand, using a fuzzy-or a data point should be close to any one of the query points. Hence, it is sufficient to be only similar to one of the query points (see Figure 3.5(b)).

$$\dot{\delta}_{\text{or}} = \min_{i=0}^{m} \bar{\delta}_i \tag{3.20}$$

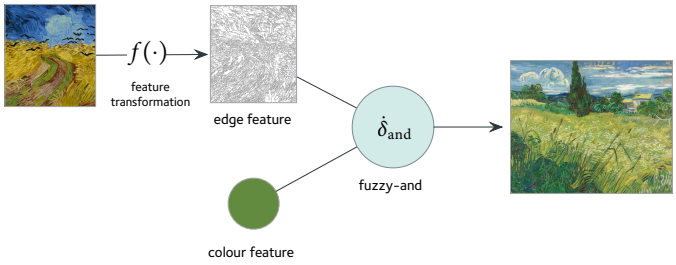
Weighted average A weighted average distance combining function will weight the proximity according to a user preference. As a consequence, a data point in the results should be close in some respect to all query points (see Figure 3.5(c)). For each query object, a weight denotes the importance of it; the distance combining function is given as

$$\dot{\delta}_{\text{wavg}} = \frac{\sum_{i=0}^{m} w_i \bar{\delta}_i}{\sum_{i=0}^{m} w_i} \tag{3.21}$$

where $w_i \in \mathbb{R}$ denotes a weight for distance $\bar{\delta}_i$.

Example 3.3 Complex query: Image retrieval.

Bob is interested in Vincent van Gogh's paintings. He is looking for one of van Gogh's famous paintings of a wheat field. Van Gogh has painted a great series of paintings which depict a wheat field, however, often in varying colours. Most of his paintings use a yellowish colour, sometimes, however, he depicts the wheat field in different shades.
To search for the variations of the wheat field, Bob uses the edge information of an image of the painting in yellow shades and defines the colour green being the average colour for the image, as visualised in the following. By employing a *fuzzy-and* operation, Bob is able to search for the paintings he is looking for, as the resulting image should satisfy both the edge features (representing the content of the image) and the colour feature. D



Example 3.4 Complex query: Image retrieval combined with geographical search.

Bob is interested to find all museums which are as close as possible to his current location and which display paintings as similar as possible to the poster he has taken a picture of. For performing such a query, Bob uses the colour information of the photograph and the distance based on his current location. Following the taxonomy introduced previously, he performs a multi-object-multi-feature query: In the case of the photograph, he uses a colour feature based on the colour features; in the case of the distance, he uses the distance computed using his current location. By employing again a *fuzzy-and* operation, Bob is able to combine both similarity predicates.

3.2.3 Vector-Space Retrieval

While the vector-space retrieval model [SWY75] was originally developed for text retrieval, its application nowadays for various forms of retrieval is rampant (e.g., for music retrieval [KNK+99], for protein classification [AKK+99], for human motion retrieval [KPZ+04]).

The vector-space retrieval model considers documents and queries as points in a high-dimensional, real-valued space in which a metric can be computed. Vector spaces can be considered metric spaces [CN06] for which a distance function, as a means for comparison, is defined.

Representations The feature transformation step in the vector-space retrieval setting creates real-valued vectors with a fixed number of dimensions ($f : \mathcal{O} \to \mathbb{R}^{dim}$). The space underlying the vector-space model is hence the *dim*-dimensional space of real values, i.e., $\mathbb{U} = \mathbb{R}^{dim}$, with *dim* being often comparably large ($dim > 50$) and, hence, the space considered high-dimensional. An element $\boldsymbol{u}_i \in \mathbb{U}$ in the space is given as

$$\boldsymbol{u}_i \in \mathbb{U} \text{ with } \boldsymbol{u}_i := \begin{pmatrix} u_{i,1} \\ u_{i,2} \\ \vdots \\ u_{i,dim} \end{pmatrix} \text{ where } u_{i,1}, u_{i,2}, \ldots u_{i,dim} \in \mathbb{R} \tag{3.22}$$

We use the term *data point* or *feature vector* to denote a document representation in the vector-space retrieval model; similarly, we refer to a query representation as a *query point*.

Comparison functions Vector spaces can be seen as special cases of metric spaces [CN06]. Hence, a distance measure exists which allows to compute a score between two data points in the space, i.e.,

$$\delta : \mathbb{R}^{dim} \times \mathbb{R}^{dim} \to \mathbb{R}_o^+ \tag{3.23}$$

satisfying the properties for distance functions mentioned previously. As previously, for a distance value of zero, two objects are said to be equal (in the extracted features); for a distance value towards infinity, two objects become more and more dissimilar.

A group of distance measures used often in vector-space retrieval is based on the Minkowski norm. The Minkowski norm of order p is defined as

$$L_p : \mathbb{R}^{dim} \to \mathbb{R}_o^+ \tag{3.24}$$

$$\boldsymbol{u} \mapsto (\sum_{i=1}^{n} = |u_i|^p)^{\frac{1}{p}} \tag{3.25}$$

leading to the class of Minkowski distances defined by

$$\delta_{L_p}(\boldsymbol{d}, \boldsymbol{q}) = \sqrt[p]{\sum_{j=1}^{dim} (d_j - q_j)^p} \tag{3.26}$$

where p denotes the order of the Minkowski distance.

An adjusted formulation of the Minkowski distance considers weighting the dimensions of the vector to give more weight to selected dimensions while lowering the importance of other dimensions. Given a weighting vector $\boldsymbol{w} = \{w_1, \ldots, w_{dim}\}$ with $w_1, \ldots, w_{dim} \in \mathbb{R}$, a weighted Minkowski distance is defined as

$$\delta_{L_p}(\boldsymbol{d}, \boldsymbol{q}, \boldsymbol{w}) = \frac{1}{\sum_{j=1}^{dim} w_j} \sqrt[p]{\sum_{j=1}^{dim} (w_j (d_j - q_j))^p} \tag{3.27}$$

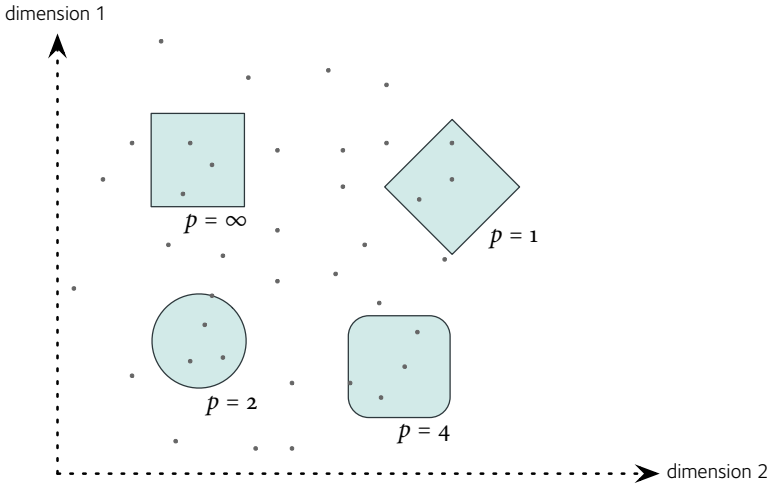


Figure 3.6 Visualisation of the Minkowski distances: The illustration depicts some of the most important Minkowski distances when evaluated on the unit circle in two dimensions.

For $p = 1$, the *City-block (Manhattan) distance* is defined as

$$\delta_{L_1}(\boldsymbol{d}, \boldsymbol{q}) = \sum_{j=1}^{dim} |d_j - q_j| \tag{3.28}$$

The *Euclidean distance*, with $p = 2$, is given as

$$\delta_{L_2}(\boldsymbol{d}, \boldsymbol{q}) = \sqrt{\sum_{j=1}^{dim} (d_j - q_j)^2} \tag{3.29}$$

While the City-block distance sums up the length of lines parallel to the coordinate axes, the Euclidean distance denotes the length of the line between two points. Figure 3.6 visualises the shape of some of the most important Minkowski distances when evaluated on the unit circle in two dimensions.

The Minkowski distances are widely used in the context of vector-space retrieval. However, it should be noted that depending on the context and the use case, other distance measures are also frequently used. For example, for comparing the visual similarity, [LCW02] have shown that the Minkowski distance measure is not very effective in modelling perceptual similarity and have proposed an adjusted distance metric. Similarly, in other use cases, for instance, the Mahalanobis distance, which accounts for correlations between dimensions, might be better suited (see, e.g.,[JHS07], for a more in-depth discussion on distance measures for image retrieval).

High-dimensional spaces The application of high-dimensional spaces, as they are used in vector-space retrieval, requires special caution since high-dimensional spaces do not match the geometrical intuition humans generally have from the real world, possibly resulting in an unexpected behaviour of the retrieval system.

The authors of [BGR+99], for example, have shown that under certain broad conditions, as dimensionality increases, the distance to the closest element approaches the distance to the farthest element, i.e.,

$$\lim_{dim \to \infty} \frac{\max_i(\delta(\boldsymbol{q}, \boldsymbol{d}_i)) - \min_i(\delta(\boldsymbol{q}, \boldsymbol{d}_i))}{\max_i(\delta(\boldsymbol{q}, \boldsymbol{d}_i))} \to 0 \tag{3.30}$$

Hence, the contrast in distances becomes non-existent and the elements more and more similar indicating a poor discrimination between the data points in the collection. Figure 3.7(a) illustrates this fact by plotting for 100'000 randomly generated points the maximum distance over the minimum distance from a query point to all points in the collection. It shows that the distances become more and more equal and the distinction between the elements more and more difficult. This is not to say that similarity queries in high-dimensional spaces are never meaningful; however, care should be taken. As shown in [AHK01], this effect is particularly also dependent on the metric used and the authors propose to use fractional distance metrics, i.e., with $0 < p < 1$, which may help to reduce these effects to some extent.

A further effect resulting from the increase in dimensionality, is found in the following setting [WSB98]: We consider the data space being $\mathbb{U} = [0,1]^{dim}$ with data points uniformly distributed, and a hyper-cube of length λ as shown in Figure 3.7(b) in all dimensions. The probability for a point to lie within the hyper-cube given by λ is, hence,

$$\Pr^{dim}[\text{“in hyper-cube given by } \lambda\text{”}] = \lambda^{dim} \tag{3.31}$$

Consider a $dim = 100$ dimensional space: For a hyper-cube of length 0.95 in all dimensions, in a data space with values in $[0,1]^{dim}$, still only a very small fraction, i.e., $\Pr^{dim=100}[\lambda = 0.95] = 0.59\%$ of the data, would be selected.

Similar observations can also be made when using a sphere instead of a hyper-cube [Weboo, p. 32]: Consider a hyper-sphere in a dim-dimensional space around a centre point $p = (0.3, 0.3, \ldots, 0.3)$ with radius 0.7, i.e., the smallest possible sphere with centre p touching the surfaces of a data space defined on $\mathbb{U} = [0,1]^{dim}$. Figure 3.7(c) illustrates this setting. For low dimensions, the centre point of such a space $c = (0.5, 0.5, \ldots, 0.5)$ will – as per our intuition – lie within the sphere. The Euclidean distance between p and c is generally given as

$$\delta_{L_2}\left(\begin{pmatrix} 0.5 \\ 0.5 \\ \vdots \\ 0.5 \end{pmatrix}, \begin{pmatrix} 0.3 \\ 0.3 \\ \vdots \\ 0.3 \end{pmatrix}\right) = \sqrt{\sum_{j=1}^{dim}(0.5 - 0.3)^2} = 0.2\sqrt{dim} \tag{3.32}$$

Hence, if $\bar{\delta} < 0.7$ the sphere contains the centre point, otherwise the centre lies outside the sphere. Given that the distance grows with the number of dimensions *dim*, for dimensionalities above $(\frac{0.7}{0.2})^2 = 12.25$ the centre point will no longer lie within the sphere.

We refer to [WSB98; BGR+99; HAK00; AHK01] for a more in-depth discussion of the peculiarities of high-dimensional spaces and their effects. We note that the effects of high-dimensional spaces have particular consequences for index structures, as we will show again in Section 4.5.

3.3 Retrieval Applications

In the following, we present various retrieval applications for different types of multimedia in the context of the vector-space retrieval model.

3.3.1 Text Retrieval

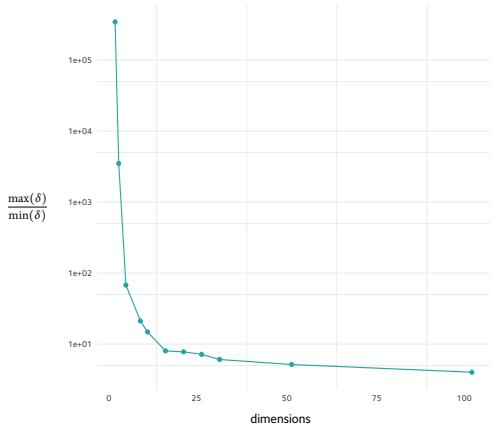
For text-retrieval applications, the representation of a textual document results in a vector $\boldsymbol{u}_i = (u_{i,1}, u_{i,2}, \ldots, u_{i,dim})$ for which each dimension $u_{i,j}$ associates to a specific (possibly normalised) term and, for instance, denotes the frequency of a term appearing within the document (*term frequency*). In Figure 3.8, we give an exemplary visualisation of three documents mapped to a vectorial representation.

Obviously a great part of text retrieval considers stop-word removal and normalisation of words (e.g., transforming words into their singular form, as done in Figure 3.8), which we, however, skip here for reasons of simplification.

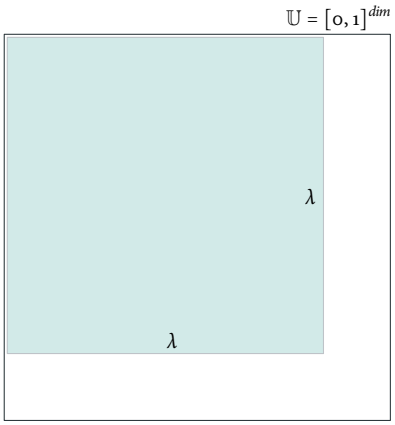
In the text-retrieval setting, a vector will be very sparse and contain mostly zero valued entries (for all the words not appearing in the text). Nevertheless, working with textual documents has the key advantage that the semantic parts can easily be identified as they are given by the words. Hence, the transformation from a textual document to a data vector is comparably an easy task to achieve.

For querying the document collection, the given query is mapped into the same space as the documents: The query vector $\boldsymbol{q} = (q_1, q_2, \ldots, q_{dim})$, hence, associates with each component the same terms as the documents. For comparing two feature vectors, the Minkowski distances introduced previously can be used. Alternatively, classical information retrieval literature also often suggests the use of the cosine measure, which measures the angle between two vectors (rather than the length). The cosine measure $\varsigma_{\cos}$ yields a similarity measure and is defined as

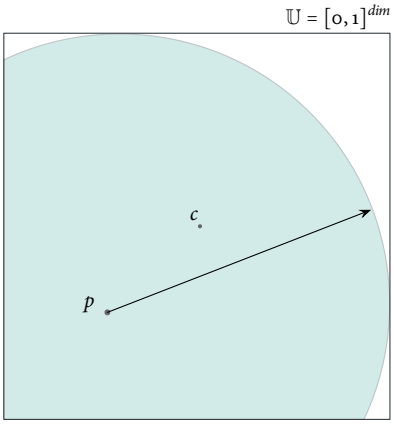
$$\varsigma_{\cos}(\boldsymbol{d}, \boldsymbol{q}) = \frac{\sum_{k=1}^{dim} d_k \times q_k}{\sqrt{\sum_{k=1}^{dim} (d_k)^2} \sqrt{\sum_{k=1}^{dim} (q_k)^2}} \tag{3.33}$$



(a) Contrast plotted against the number of dimensions for a collection of 100'000 uniformly distributed data points (based on [BGR+99]).



(b) Hyper-cube in a 2-dimensional space of length λ (based on [WSB98]).



(c) Hyper-sphere in a 2-dimensional space, with p denoting the centre point of the hyper-sphere and c denoting the centre of the space (based on [Web00, p. 32]).

Figure 3.7 Visualisation of peculiarities of high-dimensional spaces.

with $\boldsymbol{d} = \left(d_1, d_2, \cdots, d_{dim}\right)$ and $\boldsymbol{q} = \left(q_1, q_2, \cdots, q_{dim}\right)$. Hence, if two vectors are orthogonal, the similarity measure yields zero; on the other hand, if the angle is zero, meaning the same terms appear (even though not in the same frequency), the cosine measure yields one. We visualise the cosine measure in a three-dimensional vector space in Figure 3.9.

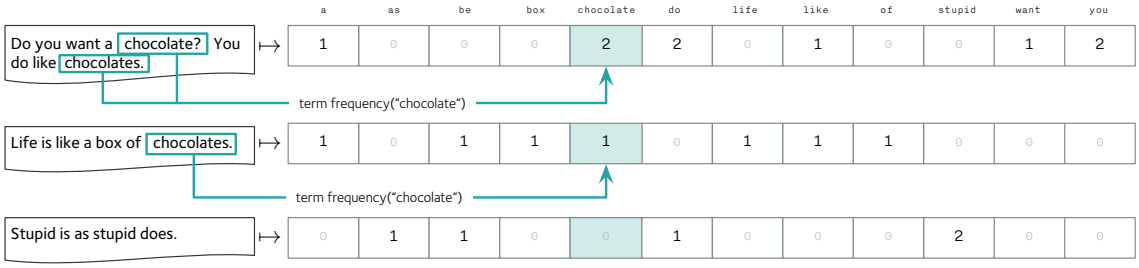


Figure 3.8 Visualisation of a simplistic text-retrieval approach: This illustration displays three documents mapped to a document representation given as the term frequency for each appearing term. We have highlighted the word "chocolate".

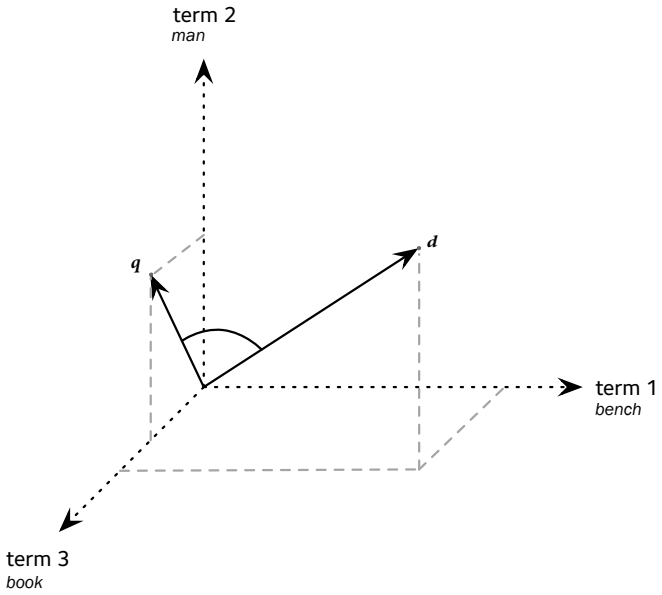


Figure 3.9 Example of the cosine measure applied in a three-dimensional vector space: Each dimension considers one term; the cosine measure computes the angle between the two feature vectors as a measure of similarity.

3.3.2 Image Retrieval

In the image and video context, the extracted visual features (descriptions) are said to capture specific, intrinsic visual properties of an image or a video [DJL+08]. However, in comparison to textual features, the approach significantly differs as the semantically meaningful parts are not as easy identifiable and extractable. For visual multimedia objects, [Eak96] distinguishes three levels of features that can be extracted: Primitive, logical and abstract features, which we will discuss in the following. In Figure 3.10, we give an exemplary visualisation of a transformation using a primitive feature; the example uses a simple global colour feature computed by the median colour of the image.

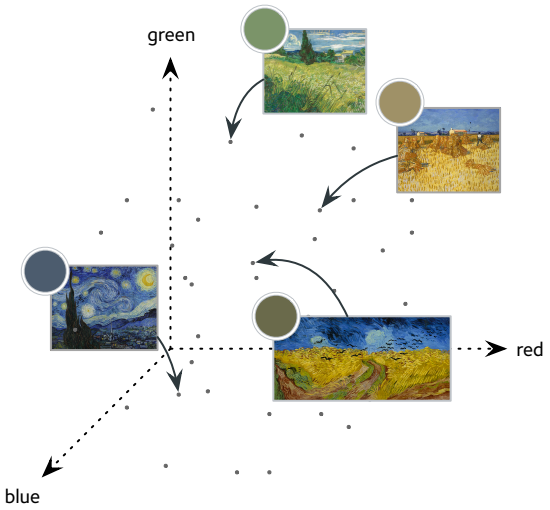


Figure 3.10 Simplified visualisation of a vector-space for images: The illustration depicts the embedding of a picture into the vectorial feature space based on the median colour, as an example of a possible primitive feature.[E]

Level 1: Primitive features Primitive (low-level) feature extractors, such as colour, texture, shape, and spatial descriptors extract low-level information from a visual document. Primitive features are either working globally on the image, or locally by using some form of partitioning (by splitting an image into regions) to obtain more selectivity. A variety of primitive features and the corresponding extractors are standardised by the MPEG-7 standard (e.g., [MSS02; Kos02]). More complex features include SIFT (scale-invariant feature transform) [Low99], the improved version SURF (speeded-up robust features) [BTV06], and the DAISY descriptor [TLF10]. We refer to [DKN08; DJL+08] for an overview and an evaluation of well-established features for image and video retrieval.

Level 2: Derived features Derived, logical features involve some degree of logical inference about the identity of the objects depicted, e.g., by detecting environments [ZLX+14], or objects [KL16; KSH12]. A comparably recent method for the extraction of derived features involves the use of convolutional neural networks, such as CaffeNet [JSD+14], a deep learning framework based on AlexNet [KSH12]. Depending on the network layer used, the entries of a logical feature vector may denote the probability for the presence of a certain class of concepts in the image. Derived features are usually very large as the number of dimensions depends on the number of detected classes (e.g., for AlexNet [KSH12], depending on the layer used, 1000 classes up to 4096 classes), but also (by ignoring very small probability values) very sparse. In [SDM15], the authors use the logical features similar to primitive features for comparison and compute a distance measure between a query vector and a vector of probabilities for certain concepts present in the multimedia document.

$\begin{pmatrix} 29.82 \\ 6.86 \\ 17.37 \\ \vdots \end{pmatrix}$

average colour in CIELab colour space per partition

$\begin{pmatrix} 4649 \\ 2780 \\ 4813 \\ 4743 \end{pmatrix}$

edge distribution per partition

primitive features

```
clothing (0.63558)
consumer goods (0.57482)
garment (0.56654)
...
park bench (0.06584)
```

$\begin{pmatrix} \vdots \\ 0.63558 \\ \vdots \\ 0.06584 \\ \vdots \end{pmatrix}$

CaffeNet

derived features

```
anger (0.00054)
contempt (0.01169)
disgust (0.00173)
fear (0.00000)
happiness (0.11952)
neutral (0.86232)
sadness (0.00277)
surprise (0.00140)
```

$\begin{pmatrix} 0.00054 \\ 0.01169 \\ 0.00173 \\ 0.0 \\ 0.11952 \\ 0.86232 \\ 0.00277 \\ 0.00140 \end{pmatrix}$

Emotion Recognition for MS Cognitive Services

abstract features

Figure 3.11 Examples of extracted features for visual documents: We visualise the results of the extraction and the resulting feature vector which is ultimately stored. For the derived features, we show the probabilities for the top four classes extracted using CaffeNet [JSD+14]. The abstract features have been extracted using the Microsoft Azure Emotion API [BZF+16].[F]

Level 3: Abstract features Abstract attributes involve a significant amount of high-level reasoning about meanings and purposes, for instance, by detecting emotional or symbolic significance, and are, hence, comparably the hardest to extract and detect. The detection of such features based on convolutional neural networks for emotion detection is currently being explored (e.g.,[BZF+16; KPB+16]).

We give an example of the three feature levels in Figure 3.11. For the primitive features, we have partitioned the image into four quadrants and computed the average colour in the CIELab colour space; similarly, the feature vector based on the edge distribution is constructed by counting the number of edge pixels per quadrant. The derived features are extracted using CaffeeNet[2] [JSD+14], a deep convolutional neural network trained on concepts; the abstract features are extracted using the Microsoft Azure Emotion API[3] [BZF+16], a deep convolutional neural network trained on emotions.

The vitrivr/iMotion retrieval system introduced in Section 2.5.3 includes a large set of features. In Table 3.1, we list a small sample of selected features present in the system and the corresponding data size.

[2] http://demo.caffe.berkeleyvision.org/

[3] https://azure.microsoft.com/en-us/services/cognitive-services/emotion/

Table 3.1 Selection of features used in vitrivr/iMotion (based on [RGS14; RGT+16a]).

Feature	Level	Dimensions	Description
global median colour	1	3	aggregation over all pixels and frames within a segment to a single median colour
colour histogram	1	15	relative occurrence of colours using a fuzzy colour histogram
directional motion histograms	1	200	regional histograms of quantised motion
partitioned edge image	1	256	ratio of edge vs non-edge pixels per regional partition
FC7 of VGG16 convolutional network	2	4096	contains data from the last fully connected layer of a model trained on the MS COCO train2014 data [LMB+14] and describing 80 classes, e.g., person, dog, bottle, etc.

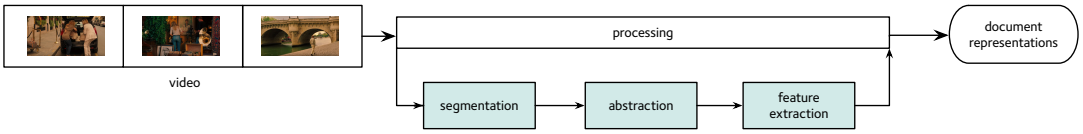


Figure 3.12 Video processing stages: The video is first segmented into segments which are then abstracted into segment representatives. The image features are extracted from the selected abstraction.[G]

3.3.3 Video Retrieval

Compared to image documents, video data is composed of the additional time dimension. We distinguish, hence, features which are pertaining to video only and require the time dimension (e.g., motion features), from features which can be extracted from the single stills as it is done using the image features introduced above.

For extracting image features, the feature transformation proceeds in multiple stages as shown in Figure 3.12: A segmentation step, which involves detecting boundaries of coherent segments as determined by semantics or by editing points, is employed to group frames together. This allows to reduce the number of images to process, as it avoids considering every frame. The detected segments are generally said to be visually or semantically similar to each other. For the detection of segments, various strategies may be used, for instance, the computation of the portion that changes between two frames. In the next step, a representation for each segment is computed (by generating a representative, possibly artificial, image) or selected (by choosing a representative key frame), respectively. The goal of the abstraction step is to select an image which includes as much information and is as representative as possible for the whole segment. The representation is then used as an input to one of the image feature extractors introduced previously.

For the features specific for videos, for example, trajectories over multiple frames can be analysed and processed and transformed into a feature vector (e.g., [RGT+17a]). Similarly, descriptions on the auditory levels can be extracted (e.g., [Gas17]).

3.4 Related Work in Multimedia Retrieval

In 1968, Gerard Salton, one of the pioneers of the research area of information retrieval, defined the field as being "concerned with the structure, analysis, organisation, storage, searching, and retrieval of information" [Sal68, p. V]. Since then, the field has split into the field of information retrieval researching nearly exclusively textual methods, and the field of multimedia retrieval focusing on multimedia data such as image, video and audio data. In the following, we present related work from the area of multimedia retrieval with the focus on image and video data and give an overview of recent developments.

3.4.1 Image Retrieval

In the early days of multimedia retrieval, the focus lay particularly on searching by using primitive, low-level features in image databases.

The IBM QBIC system, one of the most prominent examples of retrieval systems, allowed the user to search using a sketch or an example image based on colour, shape and texture features in databases of image or video data [NBE+93; FSN+95]. Similarly, early work in [KKO+92; KT92] presents the QVE (query by visual example) application which takes an outline (edge) sketch of a painting for performing a search. Photobook [PPS93; PPS96] comprises a set of tools for browsing and searching images and videos: The authors introduce a distinction between "stuff" for searching based on manual textual descriptions and "things" to search using appearance and shape. [JFS95] presents a demo application for image retrieval based on sketches or sample images that makes use of wavelet coefficients for the comparison (the applications imgSeek[4] and retrievr[5] use the same principles for their search in image data). Chabot [OS95] allows to perform content-based queries, which are, however, specified textually, e.g., by searching for images with "mostly red colour" or "some purple" and then compare the textual information to a colour histogram. Furthermore, the system allows to perform concept queries, e.g., to search for a sunset; however, the annotation of the concept has to be performed manually beforehand. MARS (Multimedia Analysis and Retrieval System) [RHM97] uses texture features, i.e., contrast and inverse difference moment (IDM), in a vector-space model for performing searches. The authors use the learnings from text-retrieval systems (e.g., term frequency,

[4] https://sourceforge.net/projects/imgseek/
[5] http://labs.systemone.at/retrievr/

normalisations, etc.) and apply these to image retrieval. PICASSSO supports searching on shapes [dP96], colours [dMP+98] and spatial relationships [dP97] using sketches and exemplary images. Blobworld [CTB+99] is also a sketch-based retrieval system for images. It segments images into regions which are supposed to represent (parts of) objects and in this way increases the retrieval performance. VisualSeek [SC96] is a hybrid retrieval system that integrates feature-based image retrieval with spatial query methods. With VisualSeek, a user can specify the composition of an image by defining spatial relationships between regions of the image. PicHunter [CMO+96; CMM+00] focuses on the feedback part of the retrieval and uses a Bayesian approach for modelling the user intent to navigate from one image to the next to answer a query. For a given query image, the system suggests to the user which images they might be interested next to find the target image. PicSOM [LKO02] uses self-organising maps with the visual content descriptors provided by MPEG-7. This organisation of the data allows the user to query the collection, but also to browse through visually similar images. As one of the newer systems, MindFinder [CWW+10] allows to use very rough sketches to search in millions of images; the system is supposed to scale well. The authors make use of a raw curve-based algorithm for the matching; in addition, users may use colour information and also explicitly tag objects. Finally, LiRe is one of the most widely used multimedia retrieval systems nowadays. It comes with a great variety of low-level image features, in particular also the MPEG-7 descriptors, for performing content-based image retrieval [LM13]. LiRe makes use of Lucene as underlying system for storing the extracted descriptions.

3.4.2 Video Retrieval

Video retrieval has grown out of the field of image retrieval by considering video as a collection of images out of which a few representative images are extracted and stored (cf. QBIC [FSN+95]). VideoQ [CCM+98; CCM+97], on the other hand, allows to search – together with colour, texture and shape features – also based on the specification of spatio-temporal information. Similarly, JACOB [ALM96; LA96] extracts from the optical flow-field a set of motion features that can be used for querying in combination with colour and texture information. CueVideo [PSA+98] provides techniques to easily (e.g., speech-based) annotate video segments and search based on these annotations. The MUVIS [GKC+02; Che04] is a large video retrieval system that supports also content-based retrieval based on colour, shape, texture, objects layout, edge direction, etc.; it comes with a feedback scheme to enhance the retrieval quality. In [CMQ09], the authors develop a system that takes storyboard-like sketches depicting both, objects and their movement, and search in this way in a collection of videos. In [BLG16], the authors extend the aforementioned LiRe system for image retrieval to also support video retrieval.

3.4.3 Current Trends

Convolutional neural networks (e.g., [KL16; KSH12]) have spurred research in the area of image recognition and, hence, the results see more and more integration in multimedia retrieval systems in the form of derived features. Moreover, the advent of large evaluation campaigns such as TRECVID [SOK06] and the Video Browser Showdown/Video Search Showcase [SAB+14; CSB+17] has led to a new wave of video retrieval systems. These, more modern systems, integrate larger varieties of features, modalities and, in particular, also semantic concepts, to give the user a wide set of possibilities to search in multimedia documents.

[LLN+13], for instance, uses a classifier to classify video segments into a small set of categories such as music, entertainment, indoor, outdoor, etc. Similarly, [SGG+13], implements concept detectors for people, vehicles, landscapes and on-screen text. On the other hand, [MAA+15] detects a set of 346 high-level concepts (e.g., water, aircraft).

This trend is complemented by new efforts for better user interfaces for either specifying queries or browse results: SIRET [LBS14], for instance, allows the user to sketch a query by only using a small number of coloured dots. For result display, [BHM15] constructs a graph out of clustered scenes to give the user a better overview. [HWH15], on the other hand, optimises the user interface for easily and quickly browsing a collection of videos on a tablet by making use of gestures and in this way enable new interaction modes.

To be able to satisfy the user query intent based on various modalities, more recent systems integrate large varieties of query modalities including query-by-sketch, but also querying using concepts or based on on-screen text or speech [RGT+17b; RGG+18; LBS14].

4

A hundred years from now, I'm quite sure, database systems will still be based on Codd's relational foundation.

— Chris Date

Foundations of Database Systems

In the following chapter, we focus on the foundations of database systems. As noted in [Özs99], there is more to providing a database system for multimedia data, than just supporting similarity matching and related indexing techniques. In this chapter, we elaborate on the different aspects involved in providing a database system: We start by introducing the general architecture of a database and focus then on the data model. We expound on the query model, discuss means for formulating a query and present the steps involved in processing one. We then consider distributed database systems. Finally, we detail the physical aspects of the data storage and present a large set of index structures for high-dimensional vector data.

In general, we address the database foundations from a relational point of view (based on [Cod70; Cod90]), as it has been the predominant approach for databases in the past decades, and only briefly point – where appropriate – to other models. It should be noted that the ideas of relational databases are not merely reflected in the data model, but also in the architecture and the processing of queries.

4.1 Database Architecture

While database systems may vary greatly in their internals, to a great extent, relational databases follow a similar architectural pattern. In the following, we present a data model architecture of traditional, relational databases which involves various levels of abstraction for modelling the data to store, and a component architecture which presents the components involved in a database system.

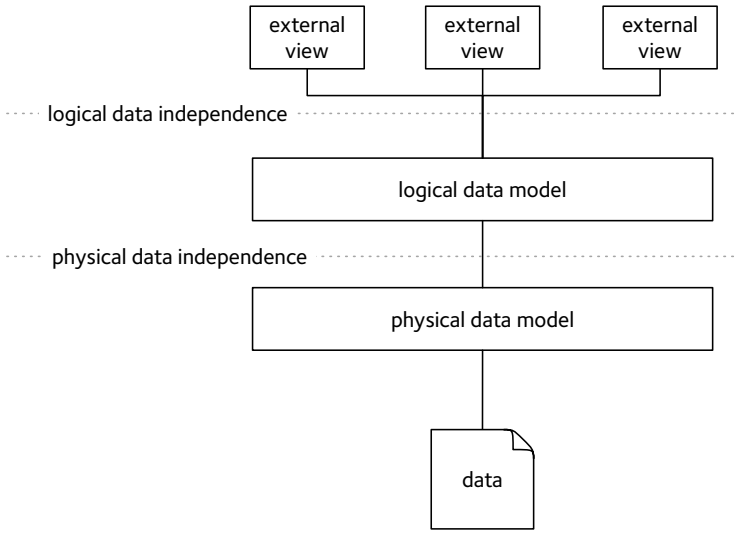


Figure 4.1 Three-level data model architecture of a database system: The separation into multiple levels of abstraction results in favourable independence properties between the levels.

4.1.1 Data Model Architecture

The separation of the data model into the external views, the logical and the physical data model forms the main abstraction of a database and reflects the key advantage of a database system over a simplistic file-based storage using the means provided by an operating system. We distinguish the following three levels[1] (see Figure 4.1):

external views External views are created for the single applications describing the parts of the database that a particular client to the database is interested in, while hiding at the same time all the other parts.

logical data model The logical model describes the structure of a database on a logical level, i.e., on the level of tuples, relations, etc.

physical data model The physical model describes the structure of the database on the basis of the data storage used (e.g., disk pages) and the access paths available for the database to query the data.

While the data in reality only resides on the physical level (i.e., on the level below the physical data model), the abstraction and separation into multiple levels of abstraction results in favourable independence properties which ensure that changes in the lower levels of the data model architecture do not require any adaptions of the schemas at the next higher level. This abstraction hierarchy seamlessly introduces independence properties

[1] The three-level view we use in the following should not be confused with the ANSI/X3/SPARC three-level architectural framework [TK78] which also distinguishes three levels of schemas: The external level with a number of external views, the conceptual level, and the internal level describing the database schema. Although closely related, the ANSI/X3/SPARC architecture concentrates not on the internals of a database system, but rather on the interplay of real-world concepts and the database.

between the levels which form a significant contribution of database systems. The following independence properties can be identified (based on [Cod90, pp. 345]):

logical data independence The logical data independence denotes the property to change the logical schema without the need to change any external view.

physical data independence The physical data independence describes the property to change the physical model, i.e., the storage representation, and access methods (e.g., for reasons of performance), without the need to change the logical model.

As noted previously, both independence properties ultimately reflect the superiority of a database system over a simple file-based data organisation which generally does not convey any guarantees in this respect and does not allow on its own the abstraction of data depending on the level of consideration.

4.1.2 Component Architecture

From a system's point of view, in the following, we introduce the components required to building a database. The following high-level modules of a database can generally be identified (based on [HSH07, pp. 144]):

- client communications manager,
- process manager,
- data manager,
- query processor,
- transactional storage manager,
- a number of shared components.

These modules are depicted together with more specific components in Figure 4.2 and discussed in more detail in the following.

Client communications manager The client communications manager is responsible for the communication between the database system and a client over a network, for instance, via an Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC) protocol. This manager is responsible for establishing a connection, remembering the caller state and responding to queries with query results or with error messages.

Process manager The process manager assumes the role of coordinator. It takes care of incoming requests to the client communications manager by calling the corresponding components and assigning processes/workers to handle the incoming query. This manager sets up and frees control structures for the task execution; it is responsible for admission control.

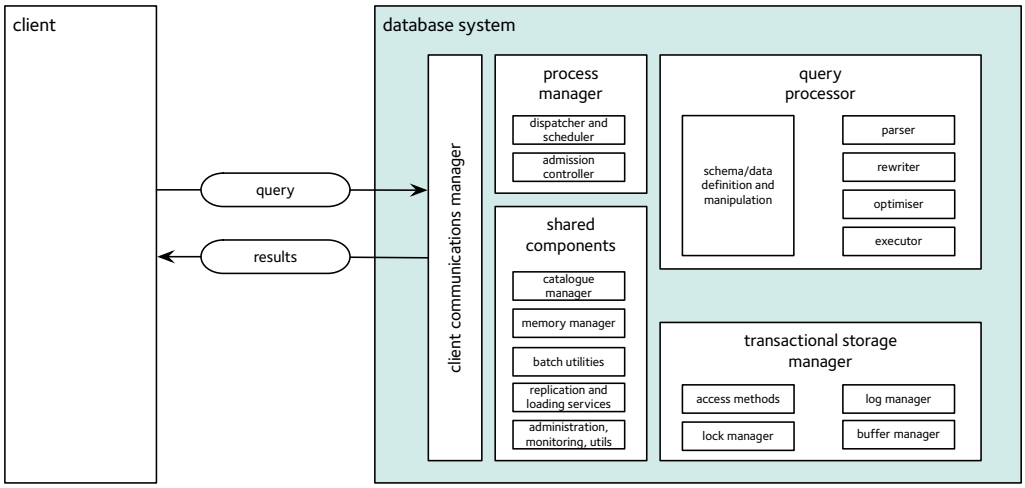


Figure 4.2 Main components of a database architecture (based on [HSH07, p. 144]).

Query processor The query processor is dedicated to processing and monitoring the execution of queries within the database system. It is composed of a query parser, a rewriter, a query optimiser and an executor. Moreover, it provides methods for the schema and data definition, their manipulation, and for the creation of indexes. The query processor parses user queries to query plans which are optimised and ultimately compiled to a sequence of operators handled by the query executor for execution.

Transactional storage manager The storage manager is responsible for the physical storage of any kind of data by providing the system with algorithms and data structures for the organisation and the access to the data. It ensures transactional guarantees by providing means for locking and versioning, and for managing conflicts in data access or data manipulation operations. To increase the system efficiency, a buffer manager is responsible for fetching and caching data from disk into main memory.

Shared components There exists a group of components whose functionality is shared throughout all components in the system. The shared components include a catalogue manager storing metadata and information on relations, indexes, integrity, authorisation, data domains, etc. Furthermore, a memory manager, batch utilities, tools for replication and loading, and further tools for administrative, monitoring or other purposes are available.

Example 4.1 Summary of available components (based on [HSH07, pp. 144]).

To summarise the available components and detail their interplay, in the following, we consider a query entering the system and passing through the modules discussed above.

1. A query enters the client communications manager which was invoked by a client establishing a connection (e.g., over a network), for example via an ODBC or JDBC connectivity protocol.

2. A thread of computation is assigned by the database system to the query. The orchestrating process manager ensures that the data and control outputs are connected through the communications manager to the client. Moreover, the process manager is also responsible for ensuring admission control and may deny the query execution based on the admission protocol.

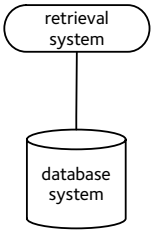
3. Given the admittance to the system, the query processor will start with the processing and execution of the query. For this purpose, the text-based query is parsed into an internal query plan, optimised and executed. The set of involved operators (e.g., projection, selection, join, etc.) is invoked by the query processor to respond to the query.

4. For executing the operators, a number of accesses to the data are possibly necessary. The transactional storage manager is primarily dedicated to the access to the storage and allows to fetch data in a transactional manner through the acquisition of locks and by maintaining a log. The buffer manager is used to cache items from the storage in memory.

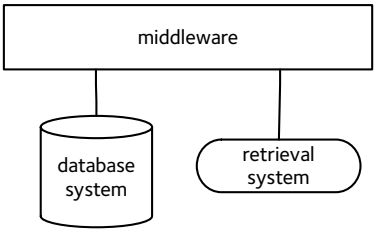
5. The data fetched by the storage manager is returned to the query processor which performs based on the query the computation of results for the client.

6. At the end of the query execution, the transaction is completed and locks are released, the results are returned to the client and the worker thread is closed together with the connection.

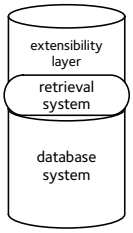
For supporting multimedia data within a database system, [WLL+15] distinguishes four architectural approaches (as summarised in Figure 4.3) emerging in terms of the integrative architecture:



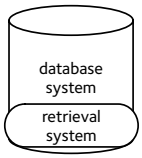
(a) Retrieval functionality on top of a database system.



(b) Middleware layer on top of a retrieval and a database system.



(c) Loose coupling of a retrieval and a database system via the database extensibility layer.



(d) Tight coupling of a retrieval and a database system by integrating the retrieval functionality into the database engine.

Figure 4.3 Classification of the integration of retrieval and database systems.

retrieval functionality on top of a database system This architecture puts retrieval capabilities into a separate software layer while the standard, built-in functions of a database are used. The retrieval layer translates the queries into standard database operations.

middleware layer on top of a database and a retrieval system This architecture uses both kinds of systems and employs a middleware layer which forwards data and queries to the corresponding sub-system.

retrieval supported by an extensibility layer (loose coupling) In this architecture, the functionality for retrieval is added through the database extensibility layer, e.g., by means of user-defined types (UDT) and functions (UDF).

retrieval supported by the database engine (tight coupling) In this architecture, the retrieval capabilities are added into the database engine and database kernel by modifying the internals of the database.

In the following, we concentrate on the the tight coupling approach and consider extensions which are added on the level of the database internals.

4.2 Relational Data Model

In the following, we present the essence of the relational data model. We follow [Gar09, p. 17] and consider three aspects of a data model:

structure of the data The data model specifies data structures and logical constructors to define the logical model of the data.

operations on the data The data model contains the definition of possible operations which can be executed on the data, for instance, operations to pose queries and to modify existing data.

constraints on the data The data model includes the definition of possible constraints which can be applied on the data.

After introducing the relational model, we also briefly present relaxations of the relational model. We continue presenting extensions to the relational data model for supporting multimedia data.

4.2.1 Structure of the Data

Today, the (either purely or object-)relational model [Cod70; Cod90] is the most widely adopted data model in conventional database systems. It is based on the notion of an m-ary mathematical relation over m data domains with data organised in m-tuples which are in turn arranged into relations.

A relation is named by a *relation symbol* (R) and has a fixed schema

$$\text{SCH}(R) = \{a_1, a_2, a_3, \ldots, a_m\} \tag{4.1}$$

where $a_1, \ldots, a_m$ denotes an ordered list of *attributes* belonging to the relation R. An attribute a_i describes the name and the data domain involved in the relation. The Cartesian product of the data domains of the single attributes constitutes the set of possible tuples

$$t \in \text{DOM}(a_1) \times \text{DOM}(a_2) \times \cdots \times \text{DOM}(a_m) \tag{4.2}$$

where $\text{DOM}(a_j)$ denotes the data domain of the attribute a_j.

A *data domain* $\text{DOM}(\cdot)$ denotes a simple or complex data domain (e.g., by user-defined types). Generally, a domain is thought of being atomic and, therefore, the values of the domain are indivisible with respect to the formal relational model. We subsume the available data domains in $\mathbb{D}$.

A *relation* is defined by both its schema $\text{SCH}(\cdot)$ and the extent composed of a finite set of tuples, $\text{VAL}(\cdot)$, over the defined schema. The extent is a mathematical relation (of degree m) on the data domains, which is in essence a sub-set of the Cartesian product of the data domains defining the relation R, i.e.,

$$\text{VAL}(R) \subseteq \text{DOM}(a_1) \times \text{DOM}(a_2) \times \cdots \times \text{DOM}(a_m) \tag{4.3}$$

Relations are characterised by the following properties (based on [EN11, pp. 63]):

ordering Given the mathematical set definition of a relation, a relation has no order (i.e., it is not a list, but a set). A tuple, on the other hand, is an ordered list of m values from the data domain of each attribute.

multi-valued attributes Given that the domains are atomic, multi-valued attributes are not allowed, but must be represented in relational form.[2]

duplicates Relations generally do not have any duplicate entries [Cod90, p. 373].[3]

4.2.2 Operations on the Data

The relational model gives rise to a set of important operators and operations. In particular, these include:

- monadic operators, including the projection operator ($\pi_A(\cdot)$), which results in a relation with all (except the de-duplicated) tuples, but only a sub-set of all attributes of a relation, and selection[4] ($\sigma_\varphi(\cdot)$), which results in a sub-set of the relation that satisfies the specified predicate φ;
- binary operators requiring full schema-compatibility, such as union ($\cup$), intersection ($\cap$), difference ($-$) operations, which require in both operands (relations) the corresponding attributes;
- binary operators requiring (partial or) no schema-compatibility, such as the Cartesian product ($\times$) and various forms of join ($\bowtie$) operations, which do not require full schema equality;
- manipulation operations resulting in changes to the extent (or possibly also the schema) of a relation (e.g., insert, update, delete).

In the following, we briefly introduce the most important standard operations found in the relational data model. We refer to [Cod70; Cod90] for more details.

Projection $\pi_A(\cdot)$ The projection $\pi_A(\cdot)$ is defined as

$$\begin{aligned} \textsc{sch}(\pi_A(R)) &:= A \\ \textsc{val}(\pi_A(R)) &:= \{t \mid \exists t' \in \textsc{val}(R) : t.A = t'.A\} \end{aligned} \tag{4.4}$$

[2] The definition of atomicity of a data domain is a controversial topic in the research community and the concept has been sometimes relaxed to introduce some relativity. [Cod90, p. 6], for example, considers atomicity of the data with respect to the property that the data cannot be decomposed into smaller pieces by the database system ("excluding certain special functions" [Cod90, p. 6]). In particular in the context of multimedia retrieval, see, for example, [SDP+07; VB98, p. 30; Gia13, pp. 15].

[3] We contrast this with the implementation of SQL, which does not consider duplicate removal. [Cod90, pp. 372] presents a more in-depth discussion on this topic. In the following, we assume that relations are in general free of duplicates.

[4] In the original publication [Cod70], the selection operation was termed *restriction*.

where $A := \{a_1, \ldots, a_j\}$ denotes a sub-set of projection attributes from the original relation ($A \subseteq \text{SCH}(R)$) with attributes to be selected from the schema of R.

Selection $\sigma_\varphi(\cdot)$ The selection $\sigma_\varphi(\cdot)$ is defined as

$$\begin{aligned} \text{SCH}(\sigma_\varphi(R)) &:= \text{SCH}(R) \\ \text{VAL}(\sigma_\varphi(R)) &:= \{t \mid \forall t \in \text{VAL}(R) \ \wedge \ t \vDash \varphi\} \end{aligned} \tag{4.5}$$

where $t \vDash \varphi$ denotes the satisfaction of the filtering predicate φ by t (i.e., $\varphi(t)$ yields true). The predicate φ is constructed by a binary operation $\{<, \leq, =, \neq, \geq, >\}$ and possibly logical connectives (for conjunctions, disjunctions, negations, etc.).

Set operations $\cup, \cap, -$ The set operations union $\cup$, intersection $\cap$ and difference $-$ are defined as

$$\begin{aligned} \text{SCH}(R_1 \otimes R_2) &:= \text{SCH}(R_1) = \text{SCH}(R_2) \\ \text{VAL}(R_1 \otimes R_2) &:= \{t \mid \forall t \in (\text{VAL}(R_1) \otimes \text{VAL}(R_2))\} \end{aligned} \tag{4.6}$$

where we use $\otimes := \{\cup, \cap, -\}$ to denote one of the set operators.

Cartesian product $\times$ In the following, we provide a definition of the Cartesian product $\times$. We say $A_{R_i} := \text{SCH}(R_i) = \{a_{R_i,1}, \ldots, a_{R_i,m}\}$ and A'_{R_i} denotes the schema A_{R_i} where the attributes have been renamed by prefixing the name of the relation, i.e., $A'_{R_i} = \{R_i.a_{R_i,1}, \ldots, R_i.a_{R_i,m}\}$. Then,

$$\begin{aligned} \text{SCH}(R_1 \times R_2) &:= A'_{R_1} \cup A'_{R_2} \\ \text{VAL}(R_1 \times R_2) &:= \{t \mid \exists t_1 \in \text{VAL}(R_1), \exists t_2 \in \text{VAL}(R_2) : t.A'_{R_1} = t_1.A_{R_1} \wedge t.A'_{R_2} = t_2.A_{R_2}\} \end{aligned} \tag{4.7}$$

Natural join $\bowtie$ Using the definition provided for the Cartesian product, the natural join $\bowtie$ is defined as

$$\begin{aligned} \text{SCH}(R_1 \bowtie R_2) &:= \text{SCH}(R_1) \cup \text{SCH}(R_2) \\ \text{VAL}(R_1 \bowtie R_2) &:= \{t \mid \exists t_1 \in \text{VAL}(R_1), \exists t_2 \in \text{VAL}(R_2) : t.A'_{R_1} = t_1.A_{R_1} \wedge t.A'_{R_2} = t_2.A_{R_2} \\ &\qquad \wedge t_1.X = t_2.X\} \end{aligned} \tag{4.8}$$

with $X := \text{SCH}(R_1) \cap \text{SCH}(R_2) \neq \varnothing$ being the attributes appearing in both relations.

4.2.3 Constraints on the Data

With the concept of keys, the relational model provides a means to distinguish tuples from each other: A set of one or more attributes that in conjunction allow to uniquely identify a tuple in a relation may form a key. The *primary key* refers to a candidate key that is chosen as the principal means for identifying a tuple. Moreover, the relational model comes with integrity constraints which enforce the correctness of data.

[Cod90, pp. 246] distinguishes constraints related

- to the data domain, for instance, defining the specific data type for an attribute;
- to single columns, for example, defining a maximum value for an attribute;
- to full entities, for instance, requiring that the primary key may not be missing;
- referential constraints, for example, necessitating that a foreign key must exist from the same domain as the primary key;
- user-defined constraints.

4.2.4 Relaxations of the Relational Model

In the evolution of data models, in the past decade, the strict relational model has seen a relaxation in many ways: For example, the introduction of a NoSQL (Not only SQL) data model is generally motivated by the support for semi-structured data, and by the attempt to overcome the limitations of scaling to large datasets given by the classical relational model. While there is no clear definition of a NoSQL data model, in general, implementations hint at the fact that this class of models (and the related concepts) is different with respect to the following properties:

- From the database perspective, the data model is said to be schema-less; the data structures are often being managed by the application using the data.
- The supported operations are in many ways limited. For example, selection operations may only be performable on a limited number of attributes (e.g., only on the key), join operations are not supported, etc.
- Integrity constraints are – if at all – available only in a limited fashion.
- The NoSQL model no longer strictly requires the Atomicity, Consistency, Isolation, Durability (ACID) properties of transactions, but relaxes these properties – for the sake of performance – to what is usually referred to as BASE (basically available, soft state, eventually consistent).

The call for relaxing the strict relational model has its foundation in the idea that there is no "one-size-fits-all model" [SC05] that is able to satisfy the great variety of use cases to store data nowadays. It is argued that while for structured data a relational database

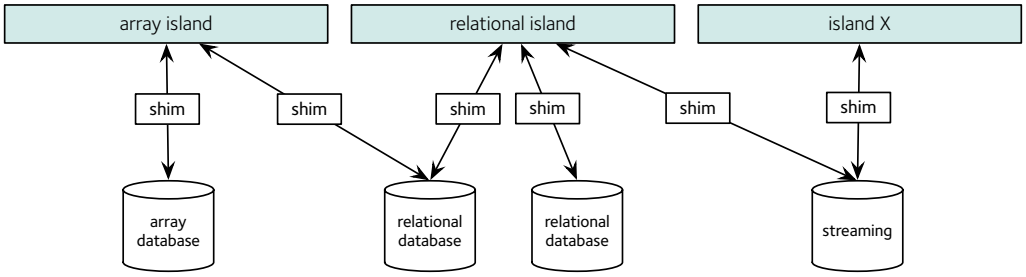


Figure 4.4 High-level architecture of a polystore: The illustration depicts multiple islands of information which subsume multiple data management systems using the same data model (based on [DZE+15]).

system may be the best (and possibly also the most efficient) option, for other types of data a specialised data management system may be the better solution.

Under the premise that a traditional approach with only one data model would be ill advised, in [Sto15], the authors introduce the term *polystore* for a data management system subsuming multiple data and storage models within one system. Such a system is said to be able to store disparate data within one store by transparently subsuming the functionality of multiple storage systems. Similar approaches can already be found in early multimedia systems (e.g., [CHS+95]) which make use of federation.[5] In [DZE+15], the authors introduce the notion of an island of information which subsumes multiple data management systems using the same data model. A shim for mapping the island's language to the native language is available. Their system BigDAWG supports querying over multiple islands of information and comes with optimisation techniques for optimising queries spanning multiple islands. In Figure 4.4, we visualise the high-level architecture of a polystore.

4.2.5 Multimedia-Specific Extensions

Given the complexity of multimedia objects, there have been various attempts in extending existing data models to be specifically suited for multimedia data. In the following, we introduce prominent extensions for supporting multimedia data and retrieval operations.

Rank-relational algebra In the following, we consider how to model similarity-based queries in a relational algebra. Consider a scoring function $\mathcal{S}(sp_1, sp_2, \ldots, sp_s)$ (we use $\tilde{\mathcal{S}}$ to denote the evaluation of the function) which ranks tuples based on a number of similarity predicates $SP = \{sp_1, \ldots, sp_s\}$ where sp_j may denote, for instance, the ranking based on a certain feature. In [LCI+05], the authors present a *rank-relational algebra* to capture ranking as a first-class construct: A rank-aware relation is a relation $\tilde{R}_{SP}$ with respect to a relation

[5] To distinguish polystores from the concept of federated databases that support similar functionality, the authors in [DZE+15] note that polystores come with sometimes very different data models rather than one data model shared by all the databases.

R and a monotonic scoring function $\mathcal{S}(sp_1, sp_2, \ldots, sp_s)$; we define $\mathcal{SP} \subseteq \{sp_1, sp_2, \ldots, sp_s\}$. Such a relation has the following properties:

- the score for a tuple t, $\bar{\mathcal{S}}_{\mathcal{SP}}[t]$, is given by the score under $\mathcal{S}(sp_1, sp_2, \ldots, sp_s)$; this results in an implicit attribute $\bar{\mathcal{S}}$ denoting the score;
- the ordering relationship $\prec_{R_{\mathcal{SP}}}$ is defined by the ranking of the scores, i.e., $\forall t_1, t_2 \in \tilde{R}_{\mathcal{SP}} : t_1 \prec_{R_{\mathcal{SP}}} t_2$ iff $\bar{\mathcal{S}}_{\mathcal{SP}}[t_1] < \bar{\mathcal{S}}_{\mathcal{SP}}[t_2]$.

Rank-relations are, hence, scored and ordered "relations". Following the definition of rank-aware relations, the authors of [LCI+05] introduce a ranking operator ρ_ψ under a ranking predicate ψ, i.e.,

$$\begin{aligned} t \in \rho_\psi(\tilde{R}_{\mathcal{SP}}) \quad &\Leftrightarrow \quad t \in \tilde{R}_{\mathcal{SP}} \\ t_1 \prec_{\rho_\psi(\tilde{R}_{\mathcal{SP}})} t_2 \quad &\Leftrightarrow \quad \bar{\mathcal{S}}_{\mathcal{SP}\cup\{\psi\}}[t_1] < \bar{\mathcal{S}}_{\mathcal{SP}\cup\{\psi\}}[t_2] \end{aligned} \tag{4.9}$$

Furthermore, in [LCI+05], the authors extend the traditional relational operations (selection, union, intersection, difference, join) and present a number of equivalence laws allowing to perform algebraic optimisations on similarity-based queries.

Fuzzy-logic algebra Rather than working on the ranks, in [Fag99; CMP+00], the authors employ a fuzzy approach and rely on the basic concepts of fuzzy set theory. The authors consider a membership function μ_t, for which

$$\mu_t : R \to [0, 1] \tag{4.10}$$

denotes the grade of membership of a tuple to a relation R. We use $\bar{\mu}_t$ to denote the evaluation of the function μ_t. Similarly, for an attribute a fuzzy domain is defined for which the membership function μ_a,

$$\mu_a : \mathbb{D}_a \to [0, 1] \tag{4.11}$$

denotes a score for a specific value of $\mathbb{D}_a$ of an attribute $a \in A$.

To be compatible with the traditional, relational algebra which is said to be based on "crisp sets", the grade of membership for the tuples in the relation is set to $\bar{\mu}_t = 1$, whereas for tuples not belonging to the relation, the membership value is $\bar{\mu}_t = 0$. In this setting, a similarity measure can be understood as introducing a grade of membership to a relation. The intuition behind fuzzy relations is that the membership score introduces a notion of tuple imprecision allowing to express how much a tuple fits the concept expressed by the relation. With that, a relation can take ordinary Boolean predicates, but also similarity predicates which make the result ultimately a fuzzy relation. While a Boolean predicate would evaluate to either zero or one, a similarity operators may result in values in $[0, 1]$.

In this context, [CMP+00] introduces a new *top* operator which can be applied in a fuzzy setting and which allows to retrieve the first κ tuples of a relation according to a ranking criterion $\zeta(\cdot)$ (possibly a distance function). The authors define[6] the top operator

$$\eta_\zeta^\kappa(R) = \{t \mid \forall t \in \text{VAL}(R) \wedge |\eta_\zeta^\kappa(R)| = \kappa \wedge \nexists t' \in \text{VAL}(R) \wedge t' \notin \eta_\zeta^\kappa(R) \wedge \zeta(t') < \zeta(t)\} \quad (4.12)$$

i.e., the top operator will select κ tuples of a relation R for which there does not exist any "better" element according to the ranking criterion $\zeta(\cdot)$.

The definition of relation membership paves the way for using fuzzy set theory for operations involving fuzzy relations. In the following, we consider in particular fuzzy-union and fuzzy-intersect operations which combine tuples from multiple fuzzy relations. For both operations, the combining function $\xi(\cdot)$, which combines the membership values from multiple relations to one membership value, can be generalised to

$$\xi : [0,1]^m \mapsto [0,1] \quad (4.13)$$

combining m fuzzy sets to one coherent (scored) fuzzy set. For a fuzzy aggregation operation, the following properties should be satisfied:

- $\xi(0,\ldots,0) = 0$ and $\xi(1,\ldots,1) = 1$ (boundary condition);
- $\xi(\bar{\alpha}_1,\ldots,\bar{\alpha}_m) \leq \xi(\bar{\beta}_1,\ldots,\bar{\beta}_m) \Rightarrow \bar{\alpha}_i \leq \bar{\beta}_i \forall i$ (monotonicity);
- $\xi(\cdot)$ is a continuous function (commutativity);
- $\xi(\bar{\alpha}_1,\ldots,\bar{\alpha}_m) = \xi(\bar{\alpha}_{p(1)},\ldots,\bar{\alpha}_{p(m)}$ where $p(\cdot)$ is a permutation function (symmetry in all arguments);
- $\xi(\bar{\alpha},\ldots,\bar{\alpha}) = \bar{\alpha}$ (idempotency);

where we use $\bar{\alpha}, \bar{\beta} \in [0,1]$ to denote a membership value. From fuzzy theory, it is established that the functions for meaningful fuzzy-union correspond to the functions called triangular co-norms (t-conorms), while fuzzy-intersect is solved by triangular norms (t-norms). For the union operation on fuzzy sets, the following functions are commonly used in fuzzy theory (with $\bar{\mu}_1, \bar{\mu}_2$ denoting membership values):

$$\text{standard union: } \xi_{\text{union}}(\bar{\mu}_1, \bar{\mu}_2) = \max(\bar{\mu}_1, \bar{\mu}_2) \quad (4.14)$$

$$\text{algebraic union: } \xi_{\text{union}}(\bar{\mu}_1, \bar{\mu}_2) = \bar{\mu}_1 + \bar{\mu}_2 - \bar{\mu}_1\bar{\mu}_2 \quad (4.15)$$

Similarly, for the intersect operation on fuzzy sets, the following functions are often used:

$$\text{standard intersect: } \xi_{\text{intersect}}(\bar{\mu}_1, \bar{\mu}_2) = \min(\bar{\mu}_1, \bar{\mu}_2) \quad (4.16)$$

$$\text{algebraic intersect: } \xi_{\text{intersect}}(\bar{\mu}_1, \bar{\mu}_2) = \bar{\mu}_2\bar{\mu}_2 \quad (4.17)$$

For both, the union and the intersect operation, further combining functions from the corresponding class can be used with reasonable results (cf. [Fag99]). [Fag99; CMP+00] further consider the except operation which we skip at this point.

[6] Note the similarity of the definition of the top operator to the definition of a *κNN* search we have provided in Equation 3.14.

Similarity join As for the operations introduced in multimedia-specific data models, the similarity join has seen some use, for example, for constructing κNN graphs which are directed graphs for which each data point is represented as a node that connects to the nodes of its κ nearest neighbours [JDJ17]. The similarity join may also be used to join tuples for which the distance is below a threshold ε. Following [BK03], we distinguish the following three similarity join operations:

range distance join The range distance join selects all items for which the distance between the specified attribute is below a given threshold ε. The range distance join can be used for clustering similar items by specifying a maximum distance.

κNN **distance join** The κNN distance join returns those κ closest pairs of the two joined relations for which the distance is minimal. This kind of join can be used, for example, to find the κ multimedia documents in a database that are most similar to each other (compared to all other comparisons).

κNN **join** The κNN join selects for each element in the database the "closest" κ elements.

We give a visualisation of the three methods in Figure 4.5.

Generalised icon The approaches introduced so far for multimedia data have concentrated on the querying aspect. In the following, we consider the data domain for storing multimedia data. [Cha96], for example, introduces the notion of a generalised icon defined as

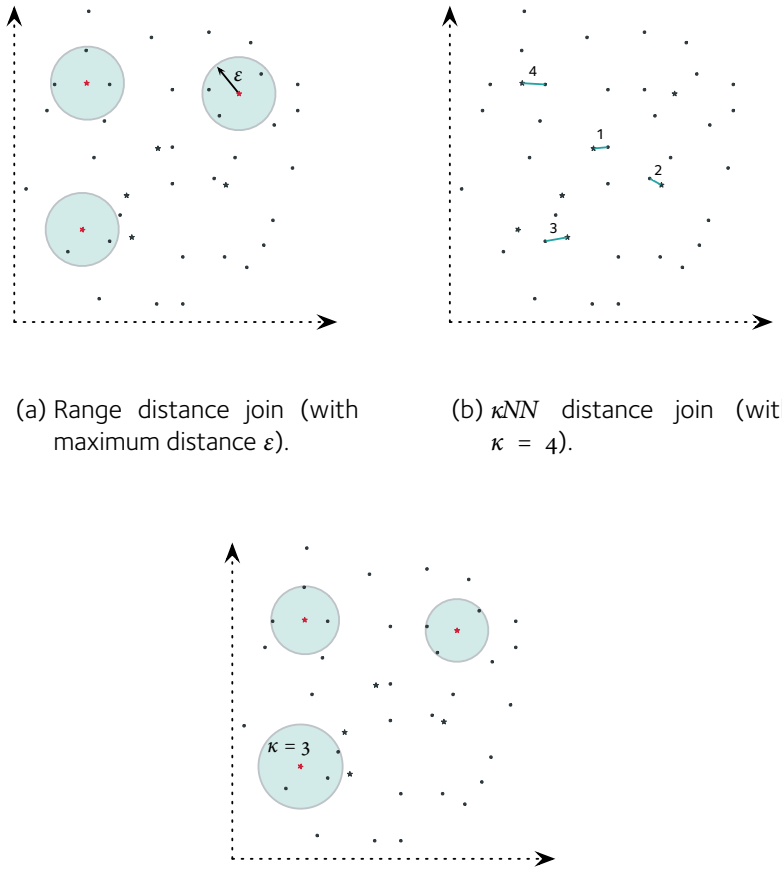
$$gi := (gi_{\text{logical}}, gi_{\text{physical}}) \tag{4.18}$$

in which gi_{logical} denotes the logical part (i.e., a label, semantic representation, feature point, etc.) and gi_{physical} the corresponding physical part being the multimedia document.

In the model introduced in [Cha96], the physical part is input to a feature transformation function which extracts features stored in the logical part of the icon. Consequently, an update to the physical part triggers the feature transformation function. In [Gia13], we have extended the notion of generalised icon to be a m-tuple in which the logical part is no longer singular, i.e.,

$$gi := (gi_{\text{logical},1}, gi_{\text{logical},2}, \ldots, gi_{\text{logical},m-1}, gi_{\text{physical}}) \tag{4.19}$$

allowing to store multiple extracted features within a generalised icon. The notion of generalised icon models the relationship between the multimedia document and the content metadata belonging to the document.



(a) Range distance join (with maximum distance ε).

(b) κNN distance join (with $\kappa = 4$).

(c) κNN join (with $\kappa = 3$).

Figure 4.5 **Similarity join operations:** We use various symbols for the data points assumed to come from different relations. (based on [BK03]).

Relationships [Nar96] notes that a data model for multimedia data should be able to capture basic data types, but also be able to identify spatial and temporal relationships between multimedia documents. The authors claim the necessity of part-of, is-a and similar-to relationships for capturing the notion of complex objects. Moreover, the authors argue that it should be possible for multimedia documents to be (fuzzy) members of "similarity classes" of similar documents.

Likewise, in [Gro97], the authors see the need for being able to model relationships between multimedia documents and real-world objects, in particular, the need for appearing-in relationships. The authors propose to store real-world objects being represented in multimedia documents as "semcons" which the authors refer to as iconic data with semantics. More concrete, a semcon is a part of a multimedia document representing a real-world object. Such a semcon stands in an appears-in relationship with a multimedia document, if the real-world object modelled as semcon appears in the document.

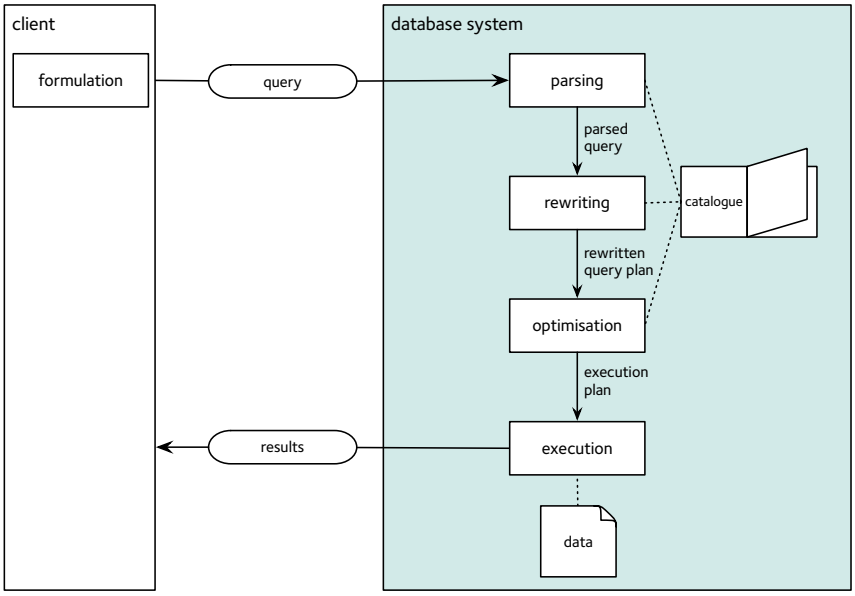


Figure 4.6 Overview of query formulation and processing in a database system: The steps include the formulation of a query, the parsing of the query, rewriting the query, optimising the query execution and, finally, executing the query (based on [Kos00; Lan10]).

4.3 Query Formulation and Processing

In this section, we consider the formulation and processing of a query in a database system. Querying a database is initiated with the scaffolding of a query in a manner understandable by the database, i.e.,

1. query formulation.

The query is sent to the database system where it is further processed following

2. query parsing,
3. query rewriting,
4. query optimisation,
5. query execution.

The processing of the query ultimately results in a result set $\mathcal{R}$ satisfying the query. More formally,

$$(q, R) \to \mathcal{R} \tag{4.20}$$

Figure 4.6 illustrates the steps involved in formulating and processing a query, which we will detail in the following.

4.3.1 Query Formulation

In the *query formulation*, the user expresses their information need and query intent by scaffolding a query that is understandable by the database system. Compared to the procedural relational algebra we have used so far when introducing the relational data model, the formulation of a query in traditional database systems follows a declarative approach. The tuple relational calculus has become a common means to declaratively formulate queries on a theoretical basis; it has formed the foundations for the SQL standard, which is usually applied for querying structured, textual data in conventional databases.

As noted in [Fuh12], an expressive, declarative query language for retrieval purposes is still missing. Using standard SQL, the authors in [Bla88; GBS04], for example, map retrieval queries to the standard database language. A major critique of this approach is, however, the fact that the generated SQL statements are often very complicated, not well-readable and, hence, only difficult to optimise.

For multimedia data, the ISO SQL/MM standard 13249-5 [ISO13249-5:2003; ME01] standardises the storage and manipulation support for multimedia data types based on user defined types (UDT) and functions (UDF) [BRT+09]. However, as noted in [BRT+09], the standard does not define how exactly to query these data types. It does, nevertheless, allow users to specify operations as ranking functions; moreover, it allows to mix traditional filtering predicates with ranking predicates.

Example 4.2 ISO SQL/MM standard 13249-5 for multimedia data.

We consider Alice querying for all films directed by "Woody Allen" and for which the colour histogram matches her query image. In this example, we adopt the ISO SQL/MM standard 13249-5 [ISO13249-5:2003] for querying multimedia data. The scores resulting from the comparison lie between $[0, 100]$, where zero denotes full equality and the value 100 denotes maximum possible dissimilarity. In this example, we assume that the representative images are stored together with the segments in the attribute `stills` and the query image is stored in the variable `q`. A simple database query with $\varepsilon = 5$ would then be formulated as:

```
SELECT *
FROM films f NATURAL JOIN segments s
WHERE f.director = 'Woody Allen' AND
      SI_FindClrHstgr(q).SI_Score(s.stills) < 5;
```

In this SQL statement, the function `SI_FindClrHstgr` will extract a colour histogram from the given multimedia document stored in the query image varibale. The subsequent `SI_Score` function determines the score compared to the given content.

For the lack of more general query mechanisms in the SQL multimedia standard, in other approaches, for example in [ABS04; BBZ12a; BRT+05], the SQL language is extended to support similarity retrieval and ranking. Likewise, in [Gia13; GAS14a], we have extended the SQL language to support retrieval functionalities, including the distance computation, searching for nearest neighbours, normalising and combining distances, etc.

Example 4.3 SQL extensions for multimedia (based on [Gia13; GAS14a]).

We consider again Alice querying for all films directed by "Woody Allen" and for which the colour feature matches her query vector. We assume that the query vector is stored in the variable `q`. In [Gia13; GAS14a], we adapt SQL to support κNN queries and formulate a similarity-based query for $\kappa = 10$ as follows:

```
SELECT *
FROM films f NATURAL JOIN segments s
WHERE f.director = 'Woody Allen'
USING DISTANCE MINKOWSKI(1) (s.feature_colour, q)
ORDER USING DISTANCE
LIMIT 10;
```

In this statement, we query using the Minkowski distance with $p = 1$ on the features stored in the attribute `feature_colour`. The results are ordered by distance and limited to $\kappa = 10$ elements.

Parallel to developments in creating new textual-based languages (or extensions) for multimedia retrieval, various approaches have been taken to create a visual language to formulate queries. Not specific to multimedia, in the Delaunay system, for instance, the authors provide visualisation options for object-oriented databases [CAL+97]. In KNIME [BCD+09], it is possible to scaffold queries in terms of workflows. Specific to multimedia retrieval, in [OÖI01; OÖL+01], the authors present a visual counterpart to their multimedia object query language which provides an easier way to express queries as in their textual-based language. In [AGS+13], we have presented a flow-based programming approach to formulating a query, allowing a user to flexibly specify data scanning operations and combine multiple result sets by connecting the corresponding building blocks on a canvas.

We refer to [CS10, pp.93] for a comprehensive overview of multimedia query languages used in research systems. [LC08a], on the other hand, summarises query languages which have been introduced specifically for video retrieval.

4.3.2 Query Parsing and Rewriting

Starting from a query given in textual (or visual) form, the query parser has the task to transform the statement into a parsed, abstract query tree which can be understood and processed by the internals of the data management system. In particular, the tasks of the query parser include checking the syntactical correctness of the statement, resolving names and references, possibly also expressions and data types and translating the query into an internal format which can further be processed. At this step, also certain checks have to be performed, for instance, on the admission of the database user to the data required, on the usage of variables, on the compatibility of the result sets combined by operations which require schema-equivalence, etc.

In the subsequent rewriting phase, the syntax tree is simplified by removing redundancies, folding expressions, removing unnecessary parts and pushing down filters in the query expressions.

4.3.3 Query Optimisation

The optimisation step transforms the internal query into an executable query plan. The goal of this step is to find a query plan which is optimal in the execution time (i.e., the execution time is minimal). Generally, two optimisation levels are distinguished:

algebraic optimisation In the algebraic optimisation, equivalence rules of relational algebra for rewriting the query tree are applied.

non-algebraic optimisation In the non-algebraic optimisation, transformations on physical operators and their ordering, e.g., the choice of an index scan over a table scan, the use of materialisations or pipelining, are applied.

For choosing an appropriate, cost-efficient query plan, in the following, we consider two approaches to the non-algebraic optimisation: The *cost-based planning* which estimates the costs based on a variety of heuristics, including the estimated size of the result set, the number of pages to load, the selectivity of operators, etc. The *empirical cost-modelling* approach, on the other hand, considers the costs based on earlier measurements of the query execution.

4.3.3.1 Cost-based Planning

The classical *cost-based planning* approach found in traditional databases computes a scalar cost value $c \in \mathbb{R}$ and attaches the cost to a query plan. This scalar cost is then used for choosing the most efficient query plan to execute. The costs are predicted based on database statistics and a variety of heuristics.

Newer approaches, for instance, consider multiple (λ) objectives and construct a cost vector $\boldsymbol{c} \in \mathbb{R}^{\lambda}$ whose optimum is the Pareto front [PY01]. The parametric query optimisation (e.g., [CG94; HS02]), on the other hand, attaches a cost function $c : \mathbb{R}^{\lambda} \mapsto \mathbb{R}$ to the plans. This results in a set of candidate plans which are optimal in one of the dimensions of the parameter space, but whose values of the parameters are not known at optimisation time, but can be adjusted by the user at query time.

Following the cost-based line of research, [BBK+97; Böh00; SAA+10] have developed a cost model for similarity searches in high-dimensional spaces for a specific set of scanning methods. As noted in [BBK+97], constructing a cost function for index-based searching in high-dimensional spaces is, however, a difficult endeavour. It requires to take into consideration all aspects of every single scanning method. In particular, it is difficult to consider within the cost model the various heuristics that are applied, for instance, also at index construction time [Böh00]. In the following, we summarise based on [Böh00] a few of the factors determining the efficiency of similarity scans:

- data set, in particular
  - the dimensionality of the data,
  - the collection size,
  - the data distribution from which the points are taken and possibly the correlation of dimensions;
- distance function used in the query;
- index structure, in particular
  - shape of the page regions (e.g., a rectangle, a sphere, composed page region),
  - layout of page on secondary storage (e.g., clustered layout, etc.);
- query processing algorithm and with that also the query type (e.g., κNN search), possibly also the ability of parallelisation of the algorithm.

The cost-based optimisation has multiple drawbacks: First, the model easily grows very complex. For instance, as noted by the author, the cost model introduced in [Böh00] assumes idealised index structures which do not consider any deterioration of the index (e.g., due to heavy overlaps in page regions of the R-tree). This and other effects are not only very hard to quantify, but they also add an additional level of complexity to the cost function. Moreover, while the factors determined by [Böh00] are very broad, they do not consider user preferences with respect to the quality of the results from the scan.

4.3.3.2 Empirical Cost-Modelling

Following a different approach, [SLM+01] *empirically* estimates the cost of a query by considering previous query executions. The idea of this approach is that while quantifying

the read cost with accuracy is difficult, it might be simpler and more accurate to estimate the costs experimentally [AI16; BBK01], e.g., by considering previous queries for predicting the query execution cost of future queries under certain query plans [CR94; SLM+01]. In [SLM+01], the authors use a feedback loop which adjusts the cost estimates (e.g., the selectivity) based on the true costs of an executed query. Similarly, [HG14] estimates the costs of query execution using a machine learning approach trained on previously executed queries: The authors map the features of a SPARQL query to a high-dimensional space using the frequencies of all the SPARQL operators as features and perform a regression. Likewise, in MongoDB [Ste11], the system experiments with several query plans in parallel and records the most efficient plan. Auto-tuning approaches that go even further and create new indexes based on the queries or dynamically adjust the physical design are also suggested in research literature (cf. [CN07; AI16]).

4.3.4 Query Execution

In the last step of the query processing, after a fully specified query plan has been selected, it is executed for retrieving the results to a query. The data is read from the storage, processed, results are computed and returned to the client.

[BBZ12a] notes that in the given context of multimedia retrieval, it is advisable for users to be able to express preferences with respect to the execution of a query. As a means for specifying the execution, for example, [BCR09] identify the use of query hints for query optimisation.

In the following, we detail the similarity searches from an executional perspective. Moreover, we consider the progressive execution of a query, a technique which is particularly interesting for long-running queries as found in multimedia retrieval systems.

4.3.4.1 Execution of Similarity Searches

In its most simplest form, a query is executed by performing a data scan and comparing the query to every tuple from the relation. In the following, we detail the execution of an εNN and a κNN query in such a setting.

ε nearest neighbour (εNN) query In Algorithm 4.1, we present an algorithm for answering εNN queries in a data scan. For εNN queries, the processing of a query is comparably straightforward. In its essence, the algorithm will scan the tuples of relation R, compute the distance between query and the attribute value of every tuple, and determine based on the score whether to add the element to the result set or not.

Algorithm 4.1 Execution of an εNN query.

Input: query $\boldsymbol{q}$, relation R, distance function δ, maximum distance ε
Output: result set $\mathcal{R}$

```
1: for all t ∈ R do
2:     δ̄ ← δ(t, q)
3:     if δ̄ < ε then
4:         R ← R ∪ {t}
5:     end if
6: end for
7: return R
```

κ nearest neighbour (κNN) query In Algorithm 4.2, we present an algorithm for answering κNN queries. Compared to εNN queries, the algorithm is more complex and requires additional data structures. The algorithm starts off by initialising a sorted list of candidates, $\mathcal{P}$, for instance, a priority queue whose elements are sorted (in ascending order), which is considered having a maximum capacity κ. The distance between the query and the attribute value of each tuple is computed and the element is added to the queue. While the queue has not reached its full capacity of κ elements, elements are added to it; in the case the length of the queue overruns the maximum capacity, the element with the highest distance is removed.

Algorithm 4.2 Execution of a κNN query.

Input: query $\boldsymbol{q}$, relation R, distance function δ, maximum number of elements κ
Output: result set $\mathcal{R}$

```
1: priority queue P ← ∅
2: for all t ∈ R do
3:     δ̄ ← δ(t, q)
4:     P.add(δ̄, t)
5:     if P.length > κ then
6:         P.dropLast()
7:     end if
8: end for
9: R ← P
10: return R
```

Rank aggregation-based scanning and Group testing Rank aggregation-based scanning [FKS03] makes use of a small number of independent "voters" which rank the tuples of a relation independently based on the similarity to the query. Consider, for instance, the use of multiple index structures being able to answer the same query, or even contacting multiple separate systems responding with a ranked list of elements. The authors combine

the different rankings using median rank aggregation according to which the resulting score is the median of the ranks received from each single voter.

To some extent similar, in [SFJ14], the authors use group testing for computing a similarity score. Group testing was originally developed to test blood samples for infections without having to examine each sample. The idea of applying group testing in multimedia retrieval is to compute a group distance to multiple elements of a group and in this way be able to approximate the similarity measures of the individual tuples.

4.3.4.2 Progressive Execution

It must be noted that multimedia retrieval queries are commonly longer running than general database queries. Various authors (e.g., [HS03; BGS08; KG05]) have already noted the usefulness of ranking algorithms that compute results progressively and report them as early as possible offering an updating result set whose precision increases with time. Moreover, as noted in [BGS08], for many applications, it is more interesting to have a first approximate answer set early rather than waiting for a long time for the exact results.

Progressive querying can be implemented, e.g., by partitioning the data and performing the retrieval on sub-sets of the data, or on varying granularities of the data (cf. [FTA+06]). The parts of the data are then read after each other and processed. The resulting answers are added to the final result set and returned early to the user.

Starting from the notion of an atomic query operation $\mathfrak{op}$, generally, two algorithms for progressively executing a query are conceivable [BGS08]. First, we present a filtering algorithm which removes in every step more and more results from the final result set (progressive filtering) as shown in Algorithm 4.3. The algorithm will apply the query operations with the goal to remove more and more elements from the final result set.

Algorithm 4.3 Progressive filtering (based on [BGS08]).

Input: query q, relation R, operations to apply $\mathfrak{op}_1, \ldots, \mathfrak{op}_k$
Output: intermediate candidate sets $\mathcal{A}^j$ (for all j), result set $\mathcal{R}$

1: $\mathcal{A}^0 := R$
2: step 1: $\mathcal{A}^1 := \mathfrak{op}_1(\mathcal{A}^0)$
⋮
3: step i: $\mathcal{A}^i := \mathfrak{op}_i(\mathcal{A}^{i-1})$
⋮
4: $\mathcal{R} := \mathcal{A}^k$

In the bottom-up algorithm (see Algorithm 4.4), instead, results are added to the final result set depending on the filtered results from the query operation, requiring the fusion of the single result sets to a new result set.

Algorithm 4.4 Bottom up progressive querying (based on [BGS08]).

Input: query q, relation R, operations to apply $\mathrm{op}_1, \ldots, \mathrm{op}_k$
Output: intermediate candidate sets $\mathcal{A}^j$ (for all j), result set $\mathcal{R}$

1: $\mathcal{A}^0 := \{\}$
2: step 1: $\mathcal{A}^1 := \mathcal{A}^0 \cup \mathrm{op}_1(R)$
⋮
3: step i: $\mathcal{A}^i := \mathcal{A}^{i-1} \cup \mathrm{op}_i(R)$
⋮
4: $\mathcal{R} := \mathcal{A}^k$

Progressive querying may be combined, for instance, with early termination strategies. Such a strategy may be employed to stop the progressive query process earlier than reaching the final stage of the query plan. Such a termination strategy may be, for example, based on an error bound accepted by the user and estimated from the results retrieved (cf. [ML14]), or on a minimal number of results retrieved (e.g., [CK97] suggests the use of a `STOP AFTER` clause which will stop a database query after a certain number of results are being retrieved), or on a maximum time (cf. [AMP+13]).

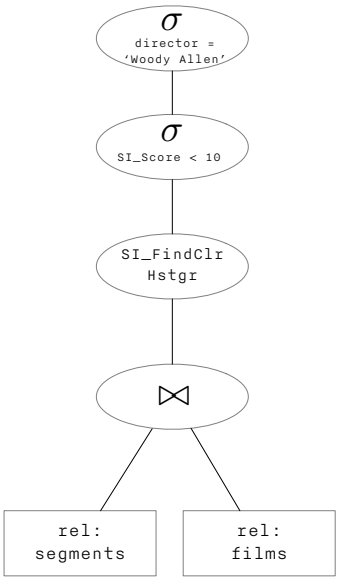
While in this section we have only considered the progressive execution of query plans which are fully specified at the moment of starting a query, the authors of [KG05] consider a recreational approach. The authors propose – for small and medium-sized databases – the dynamic creation of new query paths as time passes by. In such a setting, the database system would determine the path to answer a query by probing for new query paths.

Example 4.4 Summary of query processing.

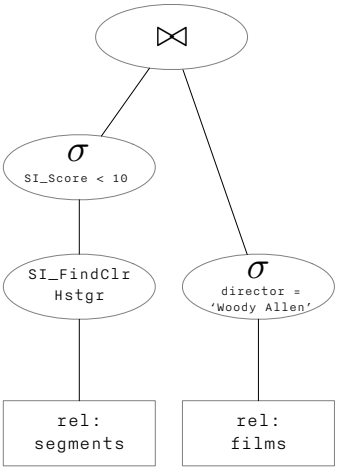
To give a summary of the query processing in a database, we consider again the example query proposed previously as formulated following the ISO SQL/MM standard 13249-5 [ISO13249-5:2003]:

```
SELECT *
FROM films m NATURAL JOIN segments s
WHERE m.director = 'Woody Allen' AND
      SI_FindClrHstgr(queryImage).SI_Score(s.stills) < 10;
```

In the parsing step, the above query is parsed to an abstract syntax tree, possibly looking as the following exemplary (and simplified) tree.



In the rewriting and the optimisation phase, the filtering predicates are pushed down, possibly leading to a query tree as follows.



The optimisation step further optimises the execution, by deciding which branch of the tree to evaluate first (i.e., the branch coming from the relation `segments` or the branch from the relation `films`), what algorithm to use for joining the results, etc. Finally, the query plan is executed and the results are returned to the user.

4.4 Storage Management and Access

On the lowest level of a database system, a relation of a database is stored on a persistent storage. Given that the data structures (and possibly also the storage medium) may vary within the physical storage, a logical tuple has to be identifiable. Hence, for the mapping of a logical to a physical tuple, a function

$$R \rightarrow \text{TID} \tag{4.21}$$

which maps a tuple $t \in R$ to a tuple identifier $\text{TID} \in \text{TID}$ is necessary. In the following, we will consider the problem of storage management in more detail.

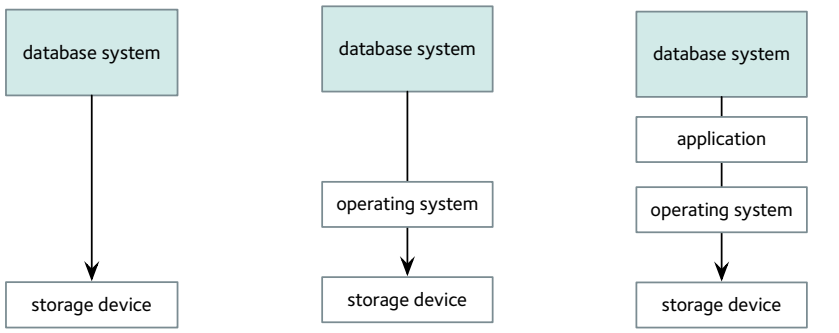
The interaction with the storage manager of a database system involves persistently storing and updating, and retrieving data. The storage manager provides an interface with transactional support to access the stored data from the query processor of the database system. It supports data structures for storing data and indexes. The storage layer comes with support for buffer management to cache data read from disk, transaction management and recovery to ensure transactional properties, which we, however, skip at this point of this thesis as they are secondary to multimedia data management systems.

4.4.1 Local Storage

Conventional databases traditionally read data in form of disk pages, which are usually very small in size (typically 8 - 128 KBs) and only hold very few records. The main reason for the small size of pages is a consequence of the memory hierarchy and the fact that the devices with the smallest capacity offer the fastest access speed.

Upon request, pages are moved from disk into memory where they reside and wait for being processed by the layers further up in the storage hierarchy. It must be noted that the gap between the disk and memory is the biggest one in terms of time. Given a query with a Boolean predicate, a sequential scan (without using any index structure) will have to read through all data pages and, hence, load all pages from disk into memory. On the other hand, in an index scan, both the index pages (depending on the structure of the index, for instance, in a tree-based index structure, not all index pages have to be considered) and the (selected) data pages storing the candidate tuples have to be read.

Generally, the local storage mechanisms can be categorised in whether a database interacts with the disk drivers in a raw access mode, whether it uses file system facilities of the operating system or whether it accesses the file system through an application (as illustrated in Figure 4.7). In the raw mode, a database system can place blocks on the storage medium – circumventing the operating system – in a way that considers the access patterns used in the database system. For example, for queries requiring large amounts of data,



(a) Raw mode. (b) Access through operating system. (c) Access through application.

Figure 4.7 Local storage mechanisms.

reserving a contiguous block of data allows to access the data sequentially. At the other end of the spectrum, in the application-driven mode, the database system persists the data through an application which may decide on its own how to place the data, how to perform caching, etc.

From the localised perspective, also an in-memory approach may be considered. In-memory databases exploit main memory for the data storage and eliminate in this way the disk I/O bottleneck [ZCO+15]. Strictly speaking, in-memory data management is not part of the storage model as the persisting operation is not durable; nevertheless, the approach has a very valid use. With the growing amounts of memory available in modern systems, it becomes more and more interesting to keep large parts of the database in memory. Having parts of the data in memory is a frequently applied technique in retrieval applications which keep, for instance, the index in memory to be able to quickly respond to queries, while storing the documents on a persistent storage (cf. [BP98]).

4.4.2 Distributed Storage

From a distributed storage perspective, we distinguish following [WSZ17b] block-based, object-based and file-based storage models.

In block-based storage, the distributed storage medium is regarded by the database as being a block device (e.g., a hard disk), i.e., the storage is organised in blocks of fixed size. On the basis of a unique identifier each block may be accessed. Block-based storage is a low-level paradigm for higher level storages providing file-based or object-based access.

The object-based storage model exposes data in terms of objects together with their metadata, identified by globally unique object identifiers. The namespace is normally flat; an object server stores an object to location mapping, while a metadata server is responsible for storing the metadata.

The level of abstraction in file-based storage models are files organised in hierarchical structures. Distributed file systems, such as the Hadoop Distributed File System (HDFS), are typically orchestrated by name nodes which take over incoming queries, manage where files are stored in the data nodes and store the available metadata. Compared to conventional database data files (as noted in Section 4.4.1), the files stored on distributed file systems are comparably large (typically 64 MB up to 1 GB) and hold a large number of tuples [Elt17]. The reasons for this discrepancy are manifold: First, modern computers come with a large amount of memory, allowing to move large files up the memory hierarchy. Moreover, given that every file in a distributed file system is represented in the memory of the name nodes, a large number of files would result in large memory consumption on the name nodes only for the management of the available files. Furthermore, distributed file systems are often built with the purpose of storing very large data sets and providing a streaming access to user applications [SKR+10]. Reading small files would result in many seeks from varying data nodes and, thus, in inefficient access to the data. Finally, when using the map/reduce paradigm, the map operation will generate one task per block. For many small files, a large number of tasks has to be created which results in a large overhead of bookkeeping and orchestration. Hence, in distributed file systems, the access is moved from a few tuples as in traditional database systems to a comparably large number of tuples which are read and moved into memory. Obviously, this has implications of the memory hierarchy, also in terms of the use of index structures.

4.4.3 Polystores and Adaptive Storage

As mentioned in Section 4.2.4, new approaches in data models consider the combination of multiple data models, depending on the data at hand, and break the paradigm of "one-size-fits-all" [SC05]. The choice of logical data model also influences the physical data model. The concept of polystores, as discussed in [Sto15] and presented in Section 4.2.4, obviously, also has to support varying physical data models for storing the available data.

On the other hand, in [IKM07], the authors make use of the incoming queries, to adapt the physical data storage to improve the query efficiency by cracking the data blocks into smaller pieces. The adjustment of the physical space happens dynamically and is query-driven. In the H_2O data store [AIA14], the system supports various storage layouts (i.e., row-major, column-major, grouping of columns) and decides dynamically for an incoming query which design to use and how to adapt the physical storage. Similarly, OctopusDB [DJ11] introduces the notion of storage views being secondary, alternative physical data representations which are used based on the workload.

4.5 Index Structures for High-Dimensional Data

The execution of a query, in particular, the computation of a distance measure for each tuple of the relation, can result in a computationally expensive task. Consider, for instance, the most straightforward way of executing a similarity query on a relation with n elements by performing an exhaustive distance computation for each tuple. The computation of an element-wise distance, e.g., the Manhattan or Euclidean distance, requires $\mathcal{O}(n \times dim)$ distance calculations meaning that the distance computation, in general, is dependent on both the collection size n and the dimensionality dim of the data. Given that the user is, however, mostly only interested in a small fraction of the full relation, e.g., the top κ results, with $\kappa \ll n$, it seems that the full computation of distances for every tuple of the relation is a time-intensive wasting of resources at query time. Particularly for large collections, scanning a relation sequentially and computing the distance for each tuple, is a computationally too heavy task to be feasible to be performed.

From a physical data model perspective, classical database systems make use of *index structures* as auxiliary access structures to improve the efficiency of data lookups. Instead of sequentially processing every single tuple from a relation, indexes allow to efficiently search within the indexed data space and provide means to early prune elements from the final result set, such that only the candidate set of remaining elements that has not been filtered out in the index scan has to be processed further. With indexes, a database system trades off both, computation at off-line time and storage, to lower the on-line query latency.

The process of creating an index can generally be considered a function where a tuple is mapped to an index value, i.e.,

$$I : R \rightarrow \text{TID} \times \mathcal{I} \tag{4.22}$$

with I denoting an index composed by index values $i \in \mathcal{I}$, e.g., a hash value, a bit string, etc. that is used at query time when scanning the index to have some concise information about the indexed attribute. TID denotes the tuple identifier space which allows to relate a tuple from the logical data model to the physical data model. At query time, the index is employed for retrieving a candidate set, $\mathcal{A}_{\text{TID}}$, which is further processed to retrieve the final result set. Formally,

$$(q, I_R) \mapsto \mathcal{A}_{\text{TID}} \tag{4.23}$$

maps a query and an index to a candidate set of tuple identifiers. This filtering phase is generally followed by a refinement phase in which the selected items of the data collection are retrieved and further processed, i.e.,

$$(q, \mathcal{A}_{\text{TID}}, R) \mapsto \mathcal{R} \tag{4.24}$$

Figure 4.8 illustrates the processing of a query with the aid of an index structure. Instead of sequentially reading the full data, in the first step the index is queried; the resulting

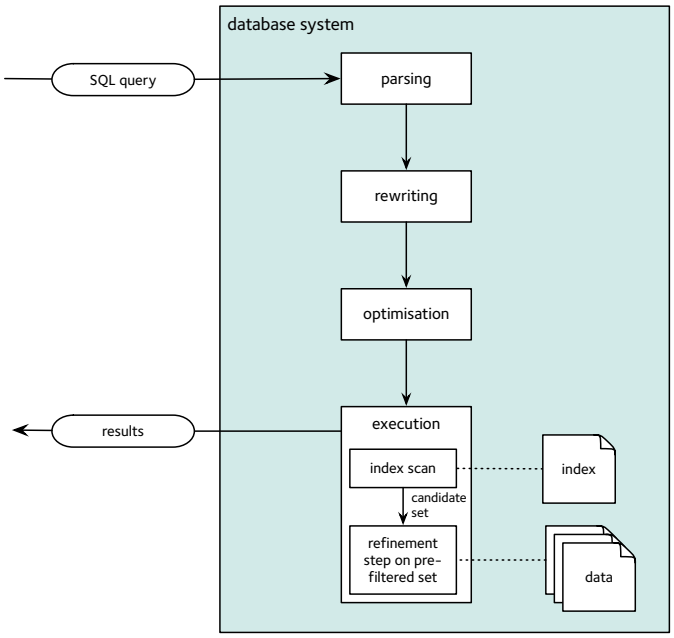


Figure 4.8 Index-based query processing: The illustration visualises how a query is processed using an index scan with a subsequent filtered data scan which refines the results.

candidate set is then further processed by accessing (parts of) the data and refining the results.

To efficiently determine, for instance, the nearest neighbours in a similarity search, an index structure overcomes the intensive computation by selecting candidates from the indexed data space that match the query best and early prunes elements from the candidate set. This avoids the full distance computation for every data point in the collection. While the set of selected, candidate results is larger than the final result set, as it also contains *false hits* which show up in the result set, but are not relevant, it is much smaller than the collection size. To retrieve the final result set, for all the data points in the candidate set $\mathcal{A}_{\text{TID}}$, the data is accessed in a filtered data scan and refined by computing the actual distance to the query. Hence, the set containing false positives, as generated in the first scanning phase, is cleaned after-the-fact in a refinement step.

While it is generally a property of an index scan to produce false hits, normally, index structures do not miss any relevant result, referred to as *false dismissals*, in the scan. In the last decade, the concept of *approximate index structures* has received more and more attention (cf. [IM98; FTA+06; CP02]). Approximate similarity searches trade high efficiency at the expense of some acceptable imprecision [AGS14]. Rather than returning all relevant results with a probability of 1.0, approximate indexes only retrieve nearest neighbours with a high probability which is not necessarily exactly 1.0. As a consequence, an element "lost" in the first index scanning stage, will not be able to be rescued in the subsequent

refinement phase scanning the data. False dismissals may be acceptable in the context of a full retrieval application as similarity searches are often part of a larger application which by itself contains other approximations (e.g., by the feature transformation) and hence, the loss in precision – compared to the gain in retrieval performance which may result from the approximate scan – may be negligible [ML14].

For conventional database systems, B$^+$-trees have become the de facto data structure for indexes. B$^+$-trees are self-balanced, m-ary trees (with m often rather large). They can be applied to structured data on which a total order exists with good performance ($\mathcal{O}(\log_m n)$, with m denoting the branching factor).

In the following, we consider high-dimensional feature vector data representing the contents of a multimedia document. Such data has, compared to textual or numerical data, generally not order; instead, its ordering is dependent on the query. This section presents various index structures for κNN similarity scans in high-dimensional vector spaces. For notational simplicity, we will use the notation introduced previously where $\mathcal{D}$ denotes the collection of feature vectors, and $\boldsymbol{d} \in \mathcal{D}$ is a feature vector stored in the collection. Similarly, $\boldsymbol{q}$ denotes a query vector. Despite the different notation used in the following, obviously, the index structures still follow the relational setting introduced in this chapter.

We will mainly focus on the properties of the index structures at on-line time and neglect the off-line creation time. Nevertheless, it should be mentioned that depending on the use case, it is important to also optimise for the time between data insertion and the data being available for retrieval. [ZIP16], for example, consider this problem by dynamically generating fine-grained indexes based on the queries posed to the system, while by default only providing a coarse-grained, quickly creatable index.

4.5.1 Hierarchical Indexes

In the 1980s and 90s, the field of databases has seen a large body of research on tree-based indexing methods for high-dimensional vector data. In general, these first approaches solved the problem of indexing for high-dimensional data by partitioning the data space and clustering the elements to a hierarchical tree structure. In this setting, data points which are "close" to each other in the feature space are supposed to be "close" to each other on the physical storage as well; similarly, data objects which are "further away" in the feature space are supposed to be "further away" in the storage space, as shown in Figure 4.9.

4.5.1.1 Space-Filling Curves, Pyramid Technique

As discussed previously, for using B$^+$-trees a total ordering of the data is necessary, which is, however, not an inherent property of feature data from $\mathbb{R}^{dim}$. Nevertheless, such an

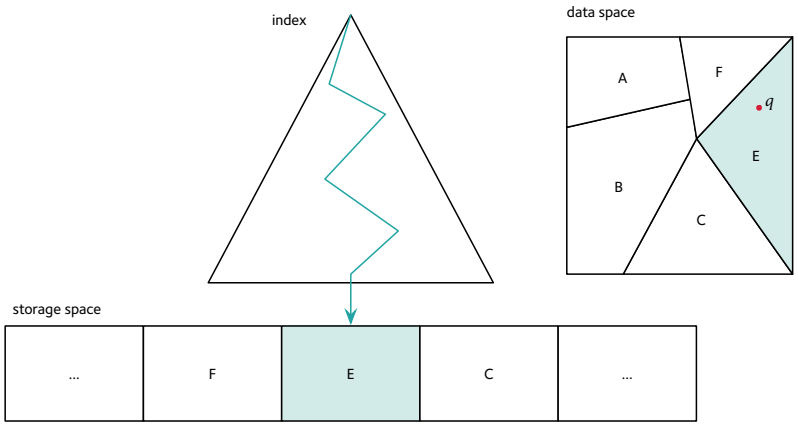


Figure 4.9 Index-based retrieval using hierarchical indexes: The storage space matches to some extent the layout of the data space (based on [CS10, p. 236]).

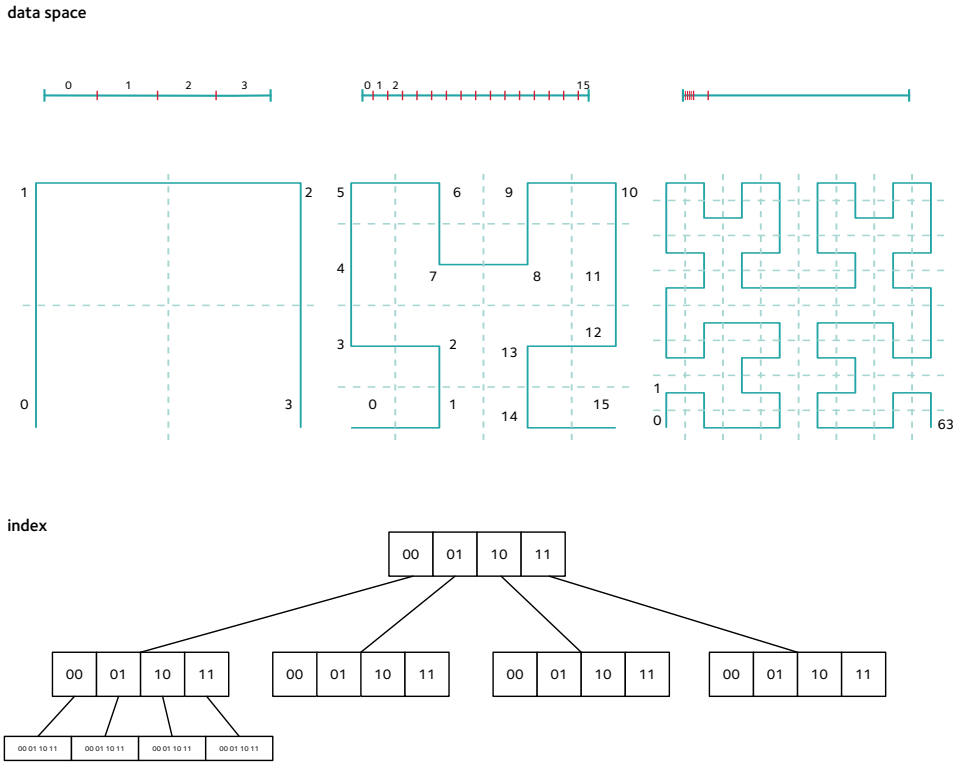


Figure 4.10 Space-filling Hilbert curve and the corresponding index: The illustration depicts the Hilbert curve in two dimensions and the corresponding mapping to a tree-based structure (based on [LK00]).

ordering can be produced, for instance, using space-filling curves (cf. [LKoo]). Space-filling curves, such as the Hilbert curve, map data points from a dim-dimensional space into a 1-dimensional space, i.e., $\mathbb{R}^{dim} \rightarrow \mathbb{R}$, by considering that points "near" in a dim-dimensional space are also supposed to be "near" in the mapped representation. In Figure 4.10, we show a visualisation of a Hilbert curve for $dim = 2$ and the resulting hierarchy, which can be mapped onto a tree structure.

Similarly, in the Pyramid technique [BBK98], a one-dimensional embedding of a high-dimensional space is computed which is then indexed using a B^+-tree, as the embedding creates an ordering in the data.

Although both methods create computationally favourable index structures for querying, as they both only encompass one dimension, the computation of space-filling curves for a large number of dimensions is not a trivial task. Moreover, for skewed, correlated or clustered points, space-filling curves do not provide an efficient solution [BBK+97].

4.5.1.2 Quadtrees, Octrees, k-d-Trees

Quadtrees, octrees and k-d-trees also belong to the family of hierarchical index structures. Their functioning is similar in that they are constructed by recursively partitioning the data space, resulting naturally in a tree structure.

Quadtrees [FB74], for instance, partition two-dimensional spaces into four quadrants. Analogously, octrees [Mea80] split three-dimensional spaces. The generalisation of quadtrees and octrees creates a family of structures that splits the data space into 2^{dim} hyper-cubical regions perpendicular to one of the axes.

k-d trees [Ben75], on the other hand, divide the data space into two sub-spaces in alternation along one of the axes, finally resulting in a multi-dimensional binary search tree. The regions of the k-d tree are, hence, not necessarily cubical, but still rectangular.

4.5.1.3 R-Tree Family

R-trees [Gut84] are height-balanced trees similar to B^+-trees and form a multi-dimensional generalisation of the B^+-tree. Contrary to the quadtrees, octrees and k-d-trees, the R-tree does not partition the data space, but only the points of the sub-tree. The nodes in an R-tree store pointers to their child nodes and a *dim*-dimensional rectangular bounding box covering all nodes from the lower hierarchy level. In Figure 4.11, we visualise a two-dimensional R-tree with the bounding boxes and the corresponding tree structure.

As noted in [BKK01], the major problem of R-tree-based index structures comes from overlapping bounding boxes, which increases with a growing number of dimensions, and which degenerates the use of the tree-structure to a sequential search [BKK01]. For splitting the inner nodes (i.e., the bounding boxes) of the R-tree as a result of a new insertion, [Gut84] proposes multiple heuristics with various costs, including an exhaustive algorithm, a quadratic-cost algorithm and a linear cost algorithm. As a variation to the originally proposed R-tree, R^*-trees, for instance, implement a better splitting strategy which minimises coverage [BKS+90]. R^+-trees, on the other hand, avoid overlapping regions completely and allow that objects might be inserted in multiple leaves [SRF87]. X-trees extend the R^*-tree heuristics to completely avoid overlaps [BKK01]. The SS-tree [WJ96] and similarly the

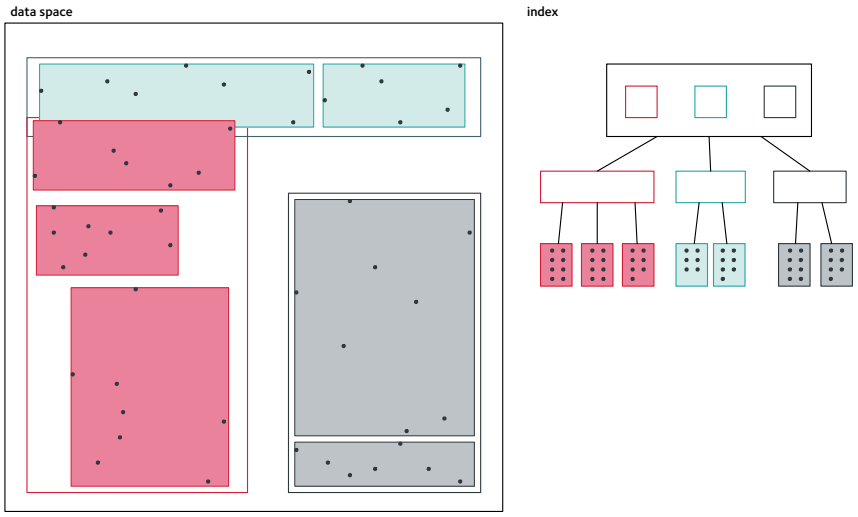


Figure 4.11 **R-tree index:** The illustration depicts an R-tree in a two-dimensional space. The minimal bounding box of the red rectangle has increased as to cut through the greenish rectangle, resulting in overlapping regions (based on [Wik17]).

TV-tree [LJF94] use spherical regions, rather than rectangular splits. In [KS97], the authors present the SR-tree, which combines the spherical regions of the SS-tree and a splitting strategy from the R* tree.

4.5.1.4 Curse of Dimensionality

We have presented some peculiarities of high-dimensional spaces in Section 3.2.3 already. The effects of high-dimensional spaces have also consequences on index structures. The *curse of dimensionality* [Bel61, pp. 94] describes the problem of exponential increase in volume when adding one additional dimension to a Euclidean space. With respect to indexes, this behaviour may result in exponentially increasing query costs with increasing dimensionality. It was shown empirically (cf. [BKK01; KS97]) that with an increasing number of dimensions, the performance of tree-based index structures is bound to degrade – in the best case – to a sequential data scan [SHJ+05].

Considering, for instance, the most simple partitioning scheme, we split the data space in each dimension into two halves (as this is done, for instance, in k-d-trees). The number of partitions grows with 2^{dim}. For instance, for dimensionality $dim = 100$, around 10^{30} partitions are existent. Even for large collections, most of the partitions will, however, be empty or very sparsely populated. Moreover, simply the task of addressing each partition would lead to exorbitant space requirements.

As a consequence of the curse of dimensionality, [WSB98] claims that for an increasing number of dimensions conventional data and space partitioning structures are outperformed by a sequential scan (already for more than 10 dimensions). Moreover, there is no

organisation of high-dimensional vector spaces based on partitioning or clustering which does not ultimately degenerate to a sequential scan for a dimensionality above a certain threshold. Hence, any tree-based structure will not be more efficient than a brute-force exhaustive distance calculation with the additional disadvantage of having to maintain an auxiliary data structure. In the following, we therefore present index structures which circumvent the anomalies of high-dimensional spaces, for instance, by avoiding clustering and partitioning at all or performing it only in low dimensionalities.

4.5.2 Cluster Pruning

Cluster Pruning (CP) [CPR+07] and extended Cluster Pruning (eCP) [GJA10] use vector quantisation to create an approximate clustering of the data collection. At indexing time, a set of random representatives are selected from the data points to which the data points are assigned to. At query time, only those partitions are considered, for which the representative of the cluster is "closest" to the query, while all the other clusters are pruned. Table 4.1 gives a notational summary for CP indexes detailled in the following.

Table 4.1 Notational summary for CP indexes.

α	number of representatives to a assign a feature point to (default $\alpha = 1$)
β	number of clusters to read at query time (default $\beta = 1$)
Γ	clusters $(\Gamma_1, \ldots, \Gamma_v)$
Λ	number of levels in multi-level clustering
v	number of representatives (default $v = \sqrt{n}$)
λ	leaders $(\lambda_1, \ldots, \lambda_v)$
ρ	representatives $(\rho_1, \ldots, \rho_v)$

4.5.2.1 Index Structure

To construct a CP index, at off-line time, *v representatives* ($\{\rho_1, \ldots, \rho_v\} \subset \mathcal{D}$) are selected randomly from the collection. Each data point in the collection is then quantised to the closest representative and, hence, becomes a *follower*, i.e., there exists a quantiser

$$\mathfrak{q} : \mathcal{D} \to \{\rho_1, \ldots, \rho_v\} \tag{4.25}$$

that assigns each data point to one representative.

In [CPR+07], the authors suggest randomly selecting $v = \sqrt{n}$ representatives as it minimises the number of distance calculations at query time $\mathcal{O}(v + \frac{n}{v})$. Assuming a uniform assignment of point to representatives, each cluster will, therefore, be assigned $\sqrt{n}$ data points on average. In the case of not uniformly distributed data, however, the clusters may be assigned a varying number of data points; as a consequence, certain clusters may be nearly empty, while others may contain a large number of elements. Following the observations made in [SHJ+05], the authors of [GJA10] argue that too small clusters only contribute little to the result quality, but still require an I/O operation to be read. On the other hand, too large clusters result at query time in expensive I/O operations and great CPU costs for computing the distances. As a consequence, the authors suggest to choose a larger number of cluster representatives for the quantisation, but eliminate at the end of the cluster creation process the corresponding number of smallest and largest cluster leaders and re-cluster the data points to the remaining representatives.

To improve the retrieval quality by increasing the chances for a true near neighbour to be reported, in [CPR+07], the authors suggest to apply a technique known as *soft-assignment* [PCI+08]: Rather than assigning a data point only to the closest representative, the data point is assigned to the $\alpha > 1$ closest representatives. Increasing α also increases the size of clusters. A cluster will, therefore, contain on average $\alpha\sqrt{n}$ data points; therefore, the attempt to increase the result quality will at the same time reduce the efficiency of reading and processing the index.

After the assignment of all points to a representative, for each cluster j a representative *leader* λ_j is appointed which is stored together with the cluster. The authors of [CPR+07] suggest three strategies for doing so:

- the leader λ_j corresponds to the representative ρ_j chosen in the beginning;
- the leader is the centroid of the cluster points (note that in this case $\lambda_j \in \mathbb{U}$, but not necessarily $\lambda_j \in \mathcal{D}$, as $\mathcal{D} \subset \mathbb{U}$);
- the leader is the medoid of the cluster (i.e., a data point of the data collection which is closest to the centroid), hence, $\lambda_j \in \mathcal{D}$.

Ultimately, the index stored on disk consists of two parts:

- a list of v leaders $\{\lambda_1, \ldots, \lambda_v\} \in \mathbb{U}$;
- for each leader $\lambda_j (j \in \{1, \ldots, v\})$, the data points assigned to it.

The clustering strategy presented for CP forms a method of vector quantisation that creates a partitioning of the data collection into clusters which are identified by a leading data point. However, as the selection of representatives is performed randomly, the index construction is non-deterministic.

In Figure 4.12, we illustrate the CP indexing in a simple case. Algorithm 4.5 summarises the basic steps for constructing the CP index.

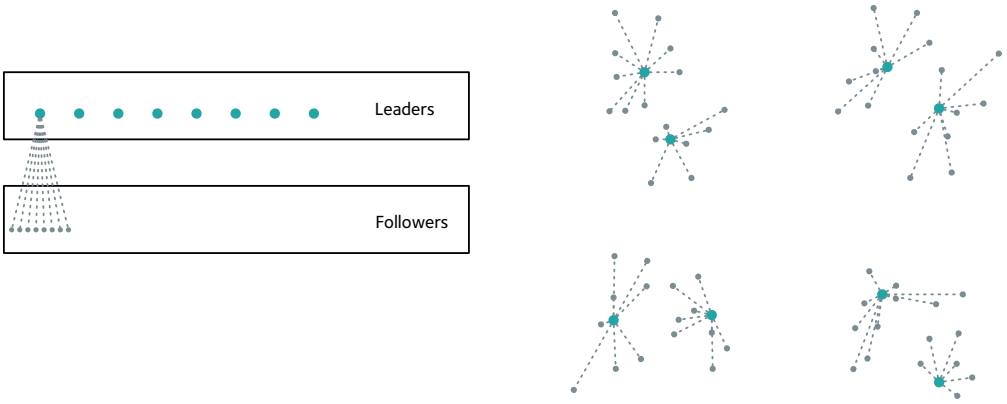


Figure 4.12 Visualisation of the CP index: The illustration depicts the index with a collection of $n = 64$ elements; we choose $\sqrt{n} = 8$ leaders.

Algorithm 4.5 Indexing algorithm for CP (based on [CPR+07]).

Input: set of n data points $\mathcal{D}$, number of clusters v
Output: clusters $\Gamma_1, \ldots, \Gamma_v$ with the corresponding leaders $\lambda_1, \ldots, \lambda_v$ and assigned data points $\boldsymbol{d} \in \mathcal{D}$

1: randomly choose $v = \sqrt{n}$ representatives $\rho_i, \ldots, \rho_v$
2: for all $\boldsymbol{d} \in \mathcal{D}$ do
3: assign data point $\boldsymbol{d}$ to the closest cluster $\Gamma_1, \ldots, \Gamma_v$
(possibly use soft-assignment to assign a data point to the closest α clusters)
4: end for
5: for all $\Gamma_j \in \{\Gamma_1, \ldots, \Gamma_v\}$ do
6: choose a leader λ_j by a custom strategy
7: end for

Multi-level clustering As the cluster size directly affects the retrieval time, in [CPR+07], the authors propose a variation to CP which aims at reducing the cluster size by applying a multi-level clustering strategy. Multi-level clustering recursively clusters the set of leaders and ultimately creates a tree structure which contains in the leafs the data points and in the inner nodes the leaders for the corresponding level. This extension reduces the number of distances to be computed – possibly at the expense of retrieval quality – to $\mathcal{O}((\Lambda + 1)n^{\frac{1}{\Lambda+1}})$ where Λ denotes the number of levels. Figure 4.13 visualises a two-level clustering based on the previous example shown in Figure 4.12.

The results in [CPR+07] show a deterioration of the result quality for increasing Λ. Instead, the only improvement noticeable, is in the index generation time which decreases with multi-level clustering.

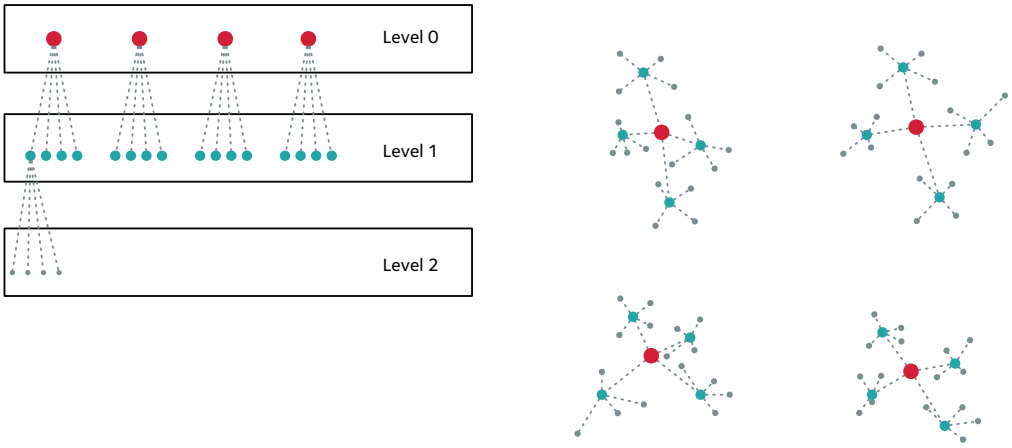


Figure 4.13 **2-level CP index:** The illustration depicts $n = 64$ data points. At the top level, $v_0 = 64^{\frac{1}{3}} = 4$ representatives are present; on the following level $v_1 = 64^{\frac{2}{3}} = 16$. Each representative subsumes $64^{\frac{1}{3}} = 4$ elements.

4.5.2.2 Search Algorithm

At on-line, query time, the query is initially only compared to the leaders $\{\lambda_1, \ldots, \lambda_v\}$ and the distance between the query and the leaders is computed, i.e., $\delta(\boldsymbol{q}, \lambda_j)$ with $j \in [1, v]$. Following this, the query is compared only amongst the data points in the cluster that pertain to the closest leader, pruning all other data points which do not belong to it. At query time, the number of distance calculations to be performed is, assuming a uniform data distribution,

$$\mathcal{O}(v + \frac{n}{v}) = \mathcal{O}(\sqrt{n} + \sqrt{n}) = \mathcal{O}(2\sqrt{n}) \tag{4.26}$$

i.e., the distance to each cluster leader λ_j ($j \in [1, v]$) and, furthermore, to all data points in the cluster has to be computed.

To improve the retrieval quality, in [CPR+07], the authors suggest to not only read the one closest cluster to the query, but to retrieve the closest β clusters and compare the contents of these clusters to the query at hand. Hence, depending on β,

$$\mathcal{O}(v + \beta\frac{n}{v}) = \mathcal{O}((1 + \beta)\sqrt{n}) \tag{4.27}$$

distance computations have to be executed. Setting the parameter β is difficult, as the quality of the results is not predictable in advance. Obviously, as β increases, the retrieval time increases as well.

In Algorithm 4.6, we summarise the search algorithm for CP.

Algorithm 4.6 Search algorithm for CP (based on [CPR+07]).

Input: clusters $\Gamma_1, \ldots, \Gamma_l$ with the corresponding leaders $\lambda_1, \ldots, \lambda_v$, query vector $\boldsymbol{q}$, number of clusters to read at query time β
Output: candidate set $\mathcal{A}_{\text{TID}}$

1: for all $\lambda_j \in \{\lambda_1, \ldots, \lambda_v\}$ do
2: compute the distance from the query to the leader, i.e., $\delta(\boldsymbol{q}, \lambda_j)$
3: end for
4: add all elements of the clusters pertaining to the closest β leaders to the candidate set $\mathcal{A}_{\text{TID}}$

4.5.3 Locality-Sensitive Hashing

The main idea of Locality-Sensitive Hashing (LSH) [IM98; GIM99] is to produce a hash for all data points such that for points "close" to each other the probability of a collision in the hash is much higher than for points which are "far" apart. At query time, the hashes of the indexed collection are compared to the hashed query and only the buckets to which the query is hashed to are retrieved and considered further as candidate results. Table 4.2 gives a notational summary for LSH indexes which we will introduce in more detail in the following.

Table 4.2 Notational summary for LSH indexes.

α	number of hash functions used by a hash combining function
β	number of combining hash functions γ_j to use
μ	modulo value possibly applied on the combining hash function
γ	hash combining function $(\gamma_1, \ldots, \gamma_\beta)$
η	hash function with $\eta \in \mathcal{H}$ $(\eta_1, \ldots, \eta_\alpha)$
$\mathcal{H}$	family of hash functions
H	hash tables $(H_1, \ldots, H_\beta)$

4.5.3.1 Index Structure

For constructing the LSH index structure, each data point is hashed to a bucket. Although every kind of projection operation could theoretically be used, intuitively, for achieving a high precision and retrieval quality, if two data points are "close" in the feature space, the probability that they collide in their hash should be high; vice-versa if two vectors are "far" apart, the probability that they collide in their hash should be small.[7]

[7] This hashing strategy is therefore very different from the conventional application of hashing, which rather tries to avoid collisions even for very similar items. In here, the hashing function tries to maximise the collision probability of close items and minimise the collision probability of distant items.

[JRU14] describes three properties for families of hashing functions which are effective for the task at hand:

- The probability for close pairs being candidate pairs should be higher than for pairs which are more far apart.
- The functions must be statistically independent. More precisely, it should be possible to estimate the probability that two (or more) functions give a certain response, by the product of their independent probabilities.
- The evaluation of the functions must be efficient.

Formally, using a hash function η, a *dim*-dimensional vector is projected onto the space of integers, as denoted in

$$\eta : \mathbb{R}^{dim} \rightarrow \mathbb{Z} \tag{4.28}$$

A family $\mathcal{H}$ of hash functions is called locality-sensitive or more specifically $(\bar{\delta}_1, \bar{\delta}_2, p_1, p_2)$-sensitive, if for any two data points $\boldsymbol{u}_i, \boldsymbol{u}_j \in \mathbb{R}^{dim}$

$$\text{if } \| \boldsymbol{u}_i - \boldsymbol{u}_j \| \leq \bar{\delta}_1 \text{ then } \Pr_{\mathcal{H}}[\eta(\boldsymbol{u}_i) = \eta(\boldsymbol{u}_j)] \geq p_1$$
$$\text{if } \| \boldsymbol{u}_i - \boldsymbol{u}_j \| \geq \bar{\delta}_2 \text{ then } \Pr_{\mathcal{H}}[\eta(\boldsymbol{u}_i) = \eta(\boldsymbol{u}_j)] \leq p_2$$

where η is chosen from $\mathcal{H}$ and, in order to be useful, the following inequalities have to be satisfied

$$p_1 > p_2 \quad \text{and} \quad \bar{\delta}_1 < \bar{\delta}_2 \tag{4.29}$$

Differently said, for close data points, the probability p_1 is said to be high if $\boldsymbol{u}_i$ and $\boldsymbol{u}_j$ map to the same bucket; similarly, for data points which are distant from each other, the probability to collide in the same bucket should be small. Figure 4.14 visualises the desired behaviour of a $(\bar{\delta}_1, \bar{\delta}_2, p_1, p_2)$-sensitive function.

Amplification Generally, a family $\mathcal{H}$ cannot be used "as is" for retrieval, as the gap between p_1 and p_2 might be only very small [AI08]. Instead, we seek strategies to amplify the gap between both probabilities. In particular, the goal is to increase the probability p_1 (for colliding if two data points are similar) and lower the probability p_2 (for being hashed to the same bucket even if the two data points are dissimilar). A strategy for achieving this is by concatenating hash functions [GIM99], as we will discuss in the following. We will consider the AND and the OR amplification as two approaches to amplify the gap between the probabilities and concatenate multiple hash functions.

For the *AND amplification*, we use a function $\dot{\eta}_{\text{AND}}$ which is constructed by picking without replacement α hash functions from $\mathcal{H}$, i.e., $\{\eta_1, \eta_2, ..., \eta_\alpha\}$, for a fixed α. Using the AND amplification, a new family of functions $\dot{\mathcal{H}}_{\text{AND}}$ is constructed for which the following must hold

$$\dot{\eta}_{\text{AND}}(\boldsymbol{u}_i) = \dot{\eta}_{\text{AND}}(\boldsymbol{u}_j) \Leftrightarrow \eta_k(\boldsymbol{u}_i) = \eta_k(\boldsymbol{u}_j) \qquad \forall 1 \leq k \leq \alpha \tag{4.30}$$

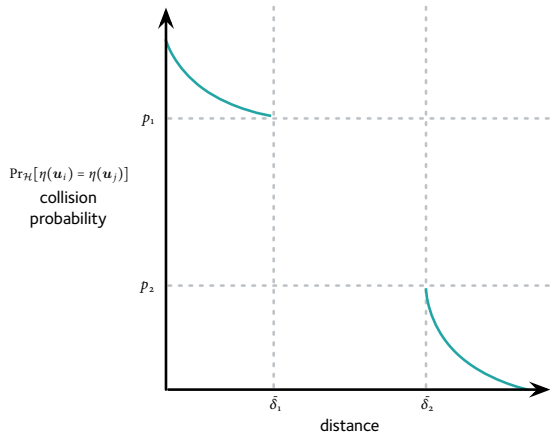


Figure 4.14 Behaviour of a $(\bar{\delta}_1, \bar{\delta}_2, p_1, p_2)$-sensitive function: The illustration compares the probability of collision, i.e., $\Pr_{\mathcal{H}}[\eta(\boldsymbol{u}_i) = \eta(\boldsymbol{u}_j)]$, and the distance (based on [JRU14, p. 100]).

Based on the statistical independence of the functions, $\dot{\mathcal{H}}_{\text{AND}}$ forms a family of $(\bar{\delta}_1, \bar{\delta}_2, (p_1)^\alpha, (p_2)^\alpha)$-sensitive functions. Note that the AND amplification lowers the probabilities of collisions; choosing α reasonably can decrease the value of p_2 to almost zero. Moreover, it amplifies the difference in probabilities between close and far points.

For the *OR amplification*, we construct in similar way a function $\dot{\eta}_{\text{OR}}$ by picking without replacement β hash functions from $\mathcal{H}$, i.e., $\eta_1, \eta_2, ..., \eta_\beta$, for a fixed β. Using the OR amplification, a new family of functions $\dot{\mathcal{H}}_{\text{OR}}$ is constructed for which the following must hold

$$\dot{\eta}_{\text{OR}}(\boldsymbol{u}_i) = \dot{\eta}_{\text{OR}}(\boldsymbol{u}_j) \Leftrightarrow \eta_k(\boldsymbol{u}_i) = \eta_k(\boldsymbol{u}_j) \qquad \text{for some } k \in \{1, ..., \beta\} \tag{4.31}$$

Based on the statistical independence of the functions, $\dot{\mathcal{H}}_{\text{OR}}$ is a family of $(\bar{\delta}_1, \bar{\delta}_2, 1-(1-p_1)^\beta, 1-(1-p_2)^\beta)$-sensitive functions. Note that the OR amplification boosts the probability of collision; choosing β reasonably, can increase the value of p_1 to almost one. However, it should be noted that it boosts the probability more for close points than for points which are further away from each other and, hence, also amplifies the gap in collision probability between close and far points.

In Figure 4.15, we visualise the influence of the parameters α and β. The steepness of the S-curve reflects how effectively false positives and false negatives can be avoided amongst the candidate pairs [JRU14, p. 90]. Obviously, the steeper the curve, the better the discriminative power of the hash function.

Using both the AND and the OR amplification, in Algorithm 4.7, we summarise the construction phase of an LSH index. We first choose β independent hash functions (OR amplification) $\gamma_1, \ldots, \gamma_\beta$ which are constructed by combining (AND amplification) the hash functions, i.e., γ_i is constructed by $\eta_{i,1}, \eta_{i,2}, \ldots, \eta_{i,\alpha}$. As a result of the amplification, the construction algorithm results in a very large number of buckets with most of them

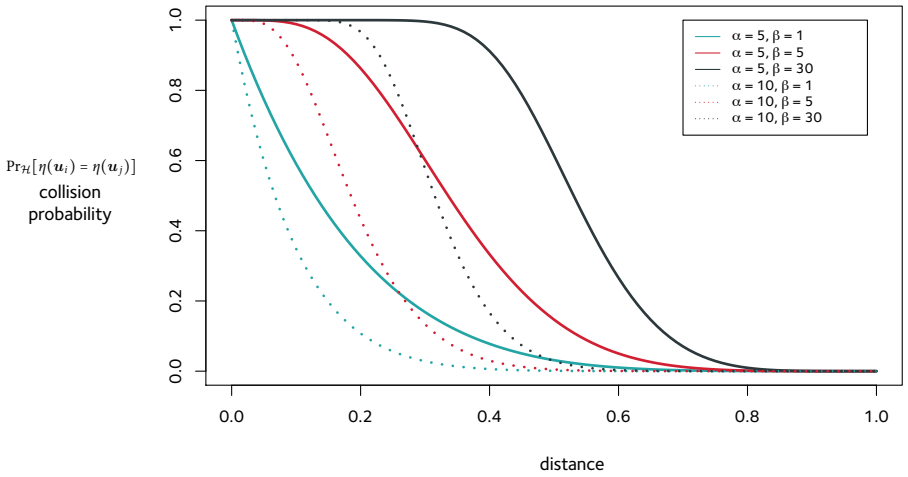


Figure 4.15 Visualisation of the AND and the OR amplification: The illustration depicts the relationship between collision probability and distance for varying parameters α of the AND amplification, and β of the OR amplification (based on [AI08]).

being empty. Hence, in implementing Algorithm 4.7, we retain only non-empty buckets by again hashing the values of $\gamma_i(\boldsymbol{d})$ using standard hashing to a smaller number (μ) of buckets.

The index stored on disk, hence, consists of the following parts:

- β combining hash functions $\gamma_i(\cdot)$ (with $i \in [1, \beta]$) consisting of α hash functions $\eta_j(\cdot)$ (with $j \in [1, \alpha]$), and the corresponding β independent hash tables $H_1, \ldots, H_\beta$;
- for each hash table, the data points assigned to each specific bucket;

Algorithm 4.7 Indexing algorithm for LSH (based on based on [GIM99; AI08]).

Input: set of n data points $\mathcal{D}$, number of combining hash functions and hash tables β, number of simple hash functions α, family of Hash functions $\mathcal{H}$

Output: hash tables $H_1, \ldots, H_\beta$ with assigned data points $\mathcal{D}$, and hash functions $\gamma_1, \ldots, \gamma_\beta$

1: Generate β hash functions $\gamma_i(\cdot)$ and the corresponding hash tables H_i such that

$$\gamma_i = (\eta_{i,1}, \eta_{i,2}, \ldots, \eta_{i,\alpha})$$

where $\eta_{i,1}, \ldots, \eta_{i,\alpha}$ are chosen randomly from the family of hash functions $\mathcal{H}$

2: for all $\boldsymbol{d} \in \mathcal{D}$ do

3: store $\boldsymbol{d}$ in bucket $\gamma_i(\boldsymbol{d})$ (possibly modulo μ) of hash table H_i for $i \in [1, \beta]$

4: end for

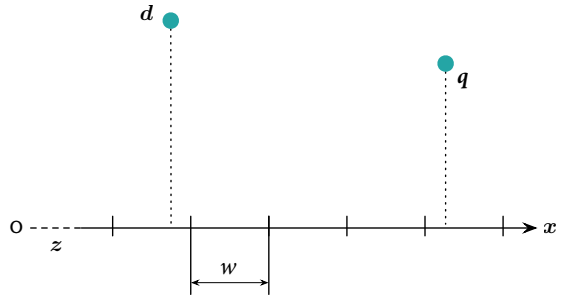


Figure 4.16 Hashing function for Minkowski distances: The variable z shifts the projection given by x; w chops the line into segments of equal width.

4.5.3.2 Hash Families

In the following, we present a few commonly used locality-sensitive hash function families $\mathcal{H}$, which are well known in research literature.

Hamming distance Assuming binary data points, i.e., $\mathbb{U} = \{0,1\}^{dim}$, [IM98] suggest to use for the Hamming distance a family of functions $\mathcal{H}$ which contains all projections of the input point on one of the coordinates, i.e.,

$$\eta_j(\boldsymbol{u}) = u_k \tag{4.32}$$

with $k \in [1, dim]$. More precisely, the hash function returns at random a dimension of the document $\boldsymbol{d}$. Note, however, that applying the same function η_j will always have to return the same dimension. This family is locality-sensitive as discussed in [GIM99].

Minkwoski distance [DII+04] presents for Minkowski distances $\delta_{L_p}(\cdot)$ a generic family of hash functions based on p-stable distributions. For the hash function, we pick a random projection $\boldsymbol{x}_j \in \mathbb{R}^{dim}$ with entries from a p-stable distribution, chop the line into equi-width segments (w_j) and shift by a random value $\boldsymbol{z}_j \in [0, w_j)$. Formally, the hashing function is given as

$$\eta_j(\boldsymbol{u}) = \lfloor \frac{\boldsymbol{x}_j \boldsymbol{u} + \boldsymbol{z}_j}{w_j} \rfloor \tag{4.33}$$

which we visualise in Figure 4.16.

The Cauchy distribution, i.e., $c(x) = \frac{1}{\pi}\frac{1}{1+x^2}$, is a 1-stable distribution, which can be applied for Manhattan distances. For $p = 2$, the Gaussian distribution, i.e., $\gamma(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$, can be used for setting the entries of $\boldsymbol{x}_j$.

For Euclidean spaces, there exists a large body of research on further, suitable hash functions (e.g., [TT07] suggests a spherical partitioning).

4.5.3.3 Search Algorithm

At query time, the query is hashed using the combining hash functions $\gamma_1, \ldots, \gamma_\beta$. The data points from the hash tables are pooled and the points from the bucket $\gamma_j(\boldsymbol{q})$ in the i-th hash table H_i (with $i \in [1, \beta]$) are added to the candidate set $\mathcal{A}_{\text{TID}}$.

To increase the number of collected candidates (depending also on the average number of elements per bucket), following the ideas of [LJW+07; Pan05], the query can additionally be randomly perturbated, i.e., $\boldsymbol{q} + \boldsymbol{\rho}$ with offset $\boldsymbol{\rho}$. The additional query points generated in this way can be used for scanning the index and the buckets retrieved for the extra query points are also retrieved.

In Algorithm 4.8, we summarise the search algorithm of LSH.

Algorithm 4.8 Search algorithm for LSH (based on [GIM99; AI08]).

Input: hash tables $H_1, \ldots, H_\beta$ and hash functions $\gamma_1, \ldots, \gamma_\beta$, query vector $\boldsymbol{q}$
Output: candidate set $\mathcal{A}_{\text{TID}}$

1: for all $i \in [1, \beta]$ do
2: add all points found in the bucket $\gamma_i(\boldsymbol{q})$ of hash table H_i to the candidate set $\mathcal{A}_{\text{TID}}$
3: end for

To analyse the running time complexity of LSH, two parts have to be considered [AI15]: the time to hash the query to the corresponding buckets and the time to retrieve the data points stored in the buckets to read. In the following equation, the first term, T_c, denotes the time needed to perform the projection and compute the hashes, while the second term, T_g, marks the time necessary to load the points from the necessary bucket. We use v to denote the number of collisions which can be estimated to β times the average number of points per bucket (depending also on the hash function chosen). The running time complexity of LSH, hence, sums up to

$$\mathcal{O}(T_c + T_g) = \mathcal{O}(dim \times \alpha \times \beta) + \mathcal{O}(v) = \mathcal{O}(dim \times \alpha \times \beta) + \mathcal{O}(\beta \times \text{avg points per bucket}) \tag{4.34}$$

where the average number of points per bucket is in the average case $\frac{n}{\mu}$. We refer to [AI15] for more details on the guarantees of the hashing functions related to the Minkowski distances.

4.5.4 Metric Inverted-File

The basic idea underlying this index structure which is largely based on [CFN08], and later improved and named Metric Inverted File (MI-File) in [AS08; AGS14], comes from the assumption that two very similar data points share also a very similar view of their surroundings. Hence, in the MI-File index every data point is represented – under the assumption that the triangle inequality holds in the indexed feature space – by the distance

to a number of selected reference points. For comparing two data points, hence, the ordering of the closest reference points is compared and the permutation of the closest points is used as a measure to approximate the real distance. Table 4.3 gives a notational summary for MI-File indexes which we will detail in the following.

Table 4.3 Notational summary for MI-File indexes.

κ_ι	number of reference points to store in index
κ_σ	number of reference points to consider at query time
v	number of reference points to use (default $v = 2\sqrt{n}$))
$\boldsymbol{\omega}_j$	permutation of reference points for $\boldsymbol{d}_j$
ξ_j	position of reference point $\boldsymbol{\rho}_j$ in order list of distances
$\boldsymbol{\rho}$	reference point $(\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_v)$

4.5.4.1 Index Structure

For constructing the MI-File index, first, a set of reference points $\{\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_v\} \subset \mathcal{D}$ is randomly chosen. The authors of [CNB+01] report that the random selection of reference points is as good as a more complex selection of reference points, e.g., based on the data distribution.

A data point $\boldsymbol{d} \in \mathcal{D}$ is then projected to a new vector which describes the distance to the reference points, i.e.,

$$\mathcal{D} \to (\xi_1, \ldots, \xi_v) \tag{4.35}$$

where ξ_i indicates the position of the reference point $\boldsymbol{\rho}_i$ in the ordered list of distances from $\boldsymbol{d}$ to the reference points. For example, $\xi_j = 8$ means that the reference point $\boldsymbol{\rho}_j$ ranks eighth in the list of ordered distances to all reference points $\boldsymbol{\rho}_i$ for $i = \{1, \ldots, v\}$.

Ultimately, a permutation vector $\boldsymbol{\omega}$ stores for a data point $\boldsymbol{d}$ a list of identifiers to the closest references sorted by the distance. Rather than storing all v reference points for each data point, to decrease the amount of data stored, [AS08; AGS14] suggest to only keep the κ_ι closest points.

Algorithm 4.9 summarises the creation algorithm for the MI index, while in Figure 4.17, we visualise the creation of the MI index.

In [AS08], the authors theoretically show that the number of reference points should at least be $v \geq 2\sqrt{n}$ to generate a unique ordering for each element in the collection. However, the authors empirically determine that the number of reference points can generally be chosen lower while still providing a high accuracy.

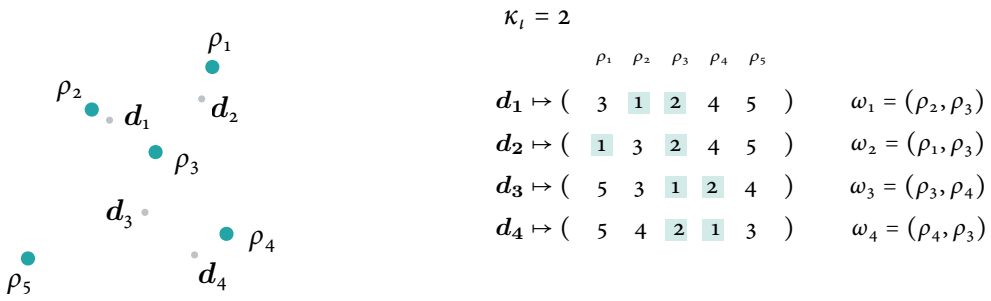


Figure 4.17 **Visualisation of the MI-File index:** The illustration depicts the MI-File index for $\kappa_\iota = 2$, i.e., only the reference points highlighted are ultimately stored (based on [AGS14]).

Algorithm 4.9 **Indexing algorithm for an MI-File index** (based on [CFN08; AS08; AGS14]).

Input: set of n data points $\mathcal{D}$, number of reference points ν, number of reference points to store κ_ι
Output: reference points $\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_\nu$, permutation $\boldsymbol{\omega}$ per element $\boldsymbol{d} \in \mathcal{D}$

1: randomly select ν data points from $\mathcal{D}$ and consider these to be reference points, i.e., $\{\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_\nu\} \subset \mathcal{D}$
2: for all $\boldsymbol{d} \in \mathcal{D}$ do
3: compute the distance of $\boldsymbol{d}$ to each reference point $\{\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_\nu\}$ and store a sorted list (by distance) of identifiers to the reference points ($\boldsymbol{\omega}$)
4: end for

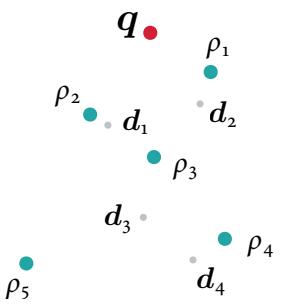
To summarise, the index stored on disk, ultimately consist of:

- reference points $\boldsymbol{\rho}_1, \ldots, \boldsymbol{\rho}_\nu$;
- the data points represented by a list of identifiers to the reference points, sorted by the distance to the corresponding reference point.

4.5.4.2 Search Algorithm

At query time, the query is similarly projected into the index space and represented as an ordering of reference points (see Figure 4.18). An approximative distance can then be computed by comparing the orderings to the reference points of both the query and each data point. For this, various methods of rank correlation are presented in [CFN08], such as Spearman's Rho, Kendall's Tau and Spearman's Footrule. The authors of [AS08] have not found any significant difference between the use of different rank correlation methods. We exemplary present the Spearman Footrule distance (SFD) defined as

$$\delta_{\text{SFD}}(\boldsymbol{d}_1, \boldsymbol{d}_2) = \sum_{i=1}^{\nu} |\boldsymbol{\omega}_1(i) - \boldsymbol{\omega}_2(i)| \tag{4.36}$$



$\kappa_l = 2$

$\kappa_\sigma = 2$

$\omega_1 = (\rho_2, \rho_3)$ $\quad \delta_{\text{SFD}}(\omega_q, \omega_1) = 1 + (\kappa_l + 1) = 4$

$\omega_2 = (\rho_1, \rho_3)$ $\quad \delta_{\text{SFD}}(\omega_q, \omega_2) = 0 + (\kappa_l + 1) = 3$

$\omega_3 = (\rho_3, \rho_4)$ $\quad \delta_{\text{SFD}}(\omega_q, \omega_3) = (\kappa_l + 1) + (\kappa_l + 1) = 6$

$\omega_4 = (\rho_4, \rho_3)$ $\quad \delta_{\text{SFD}}(\omega_q, \omega_4) = (\kappa_l + 1) + (\kappa_l + 1) = 6$

$$q \mapsto \begin{pmatrix} \rho_1 & \rho_2 & \rho_3 & \rho_4 & \rho_5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$$

$\omega_q = (\rho_1, \rho_2)$

Figure 4.18 **Querying an MI-File index:** Following the example introduced above, here we visualise querying a MI-File index for $\kappa_\sigma = 2$, i.e., only the highlighted reference points are ultimately considered (based on [AGS14]).

where $\boldsymbol{\omega}_j$ denotes the ordering of the reference points for a data point $\boldsymbol{d}_j$, and with $\boldsymbol{\omega}_j(i)$ we access the rank of the reference point $\boldsymbol{\rho}_i$ of data point j.

Following the observation of [AS08] that increasing the number of reference points does not necessarily improve the effectiveness of retrieval, the authors suggest to limit the number of reference points considered at query time to a fixed number κ_σ (with $\kappa_\sigma \leq \kappa_l$). Hence, the list of reference points of the query is limited to only consider the closest κ_σ points when comparing two lists of orderings.

The fact that at index creation time, we have limited the number of reference points to κ_l and at query time we limit the reference points of the query to κ_σ can result in the situation where a reference point appears in the query, but not in the data point. For this, the Spearman Footrule distance is adjusted such that for a missing reference point the maximum rank distance $(\kappa_l + 1)$ is assumed, i.e.,

$$\delta'_{\text{SFD}}(\boldsymbol{d}_1, \boldsymbol{d}_2) = \sum_{i=1}^{\nu} \begin{cases} |\boldsymbol{\omega}_1(i) - \boldsymbol{\omega}_2(i)| & \text{if } \boldsymbol{\omega}_1 \text{ and } \boldsymbol{\omega}_2 \text{ contain a ranking for } \boldsymbol{\rho}_i \\ \kappa_l + 1 & \text{otherwise} \end{cases} \tag{4.37}$$

The search algorithm presented here has a running time of $\mathcal{O}(n\kappa_\sigma)$ as for each data point in the collection the κ_σ reference points have to be considered. To keep the nearest neighbours, a priority queue of fixed capacity can be used (see Algorithm 4.2).

Inverted index [AS08; AGS14] propose using an inverted index grouped by reference point to store the data points of the collection and increase the efficiency at retrieval time by only having to consider the data points appearing in the list of the reference point. As a consequence, they are able to lower the computational complexity to $\mathcal{O}(\kappa_\sigma \frac{\kappa_l\, n}{\nu})$ as for κ_σ reference points considered in the retrieval a posting list of average length $\frac{\kappa_l\, n}{\nu}$ has to be read. However, it can be expected that for small numbers of reference points or for large collections, respectively, the list will always contain a large portion of the collection.

4.5.5 Product Quantisation

Product Quantisation (PQ) [JDS11] divides the feature space into low-dimensional sub-spaces which are quantised separately. Data points are then represented by the quantisation indexes for each sub-space. With the produced code, an approximate distance can be computed at query time and used for pruning elements with large distances. Table 4.4 gives a notational summary for PQ indexes.

Table 4.4 Notational summary for PQ indexes.

γ_j	number of clusters per split j
μ	number of splits (with dim being a multiple of μ)
$\mathfrak{q}$	quantiser
$\boldsymbol{v}_j$	sub-vector j of full vector $(\boldsymbol{v}_1, \ldots, \boldsymbol{v}_\mu)$
s	signature

4.5.5.1 Index Structure

For constructing the PQ index, first, the data points are split into μ distinct sub-vectors $\boldsymbol{v}_j$, dividing the feature space $\mathcal{D} \subset \mathbb{R}^{dim}$ into μ disjoint sub-spaces as follows

$$\boldsymbol{d} = \left(\underbrace{d_1, \ldots, d_k}_{\boldsymbol{v}_1}, \ldots, \underbrace{d_{dim-k+1}, \ldots, d_{dim}}_{\boldsymbol{v}_\mu} \right) \tag{4.38}$$

such that the dimensionality dim is a multiple of μ and where $k = \frac{dim}{\mu}$. The splits of the data point $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_\mu$ are then quantised using low-complexity quantisers $\mathfrak{q}_1, \ldots, \mathfrak{q}_\mu$ which quantise a data point to

$$\boldsymbol{d} \mapsto \big(\mathfrak{q}_1(\boldsymbol{v}_1), \ldots, \mathfrak{q}_\mu(\boldsymbol{v}_\mu) \big) \tag{4.39}$$

i.e., a data point is represented by the indexes of the clusters the sub-vectors fall into.

To create the sub-quantisers, at off-line, indexing time, a clustering scheme is learned based on the data at hand using a K-means clustering algorithm. The index, then, stores the identifier to a centroid for each sub-quantiser, which denotes the cluster the sub-vector is quantised to. The index stored on disk, ultimately, stores two elements:

- a list of clusters for each split $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_\mu$;
- for each data point a list of identifiers denoting into which cluster the splits of each data point falls into;

Figure 4.19 visualises the PQ index construction phase. In Algorithm 4.10, we summarise the construction algorithm for PQ indexes.

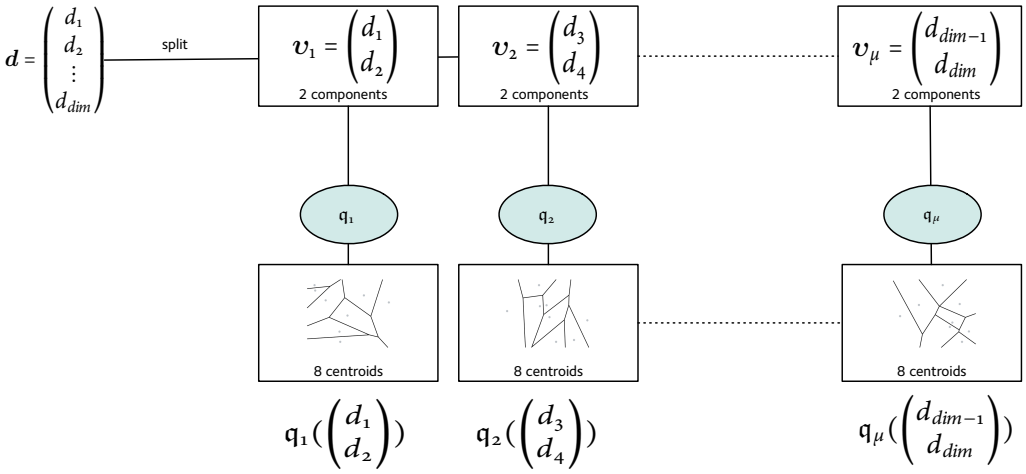


Figure 4.19 Visualisation of the construction phase of the PQ index (based on [Jég12]).

Algorithm 4.10 Indexing algorithm for PQ (based on [JDS11]).

Input: set of n data points $\mathcal{D}$, number of sub-vectors μ, number of clusters per split γ_i ($\forall i \in \{1, \ldots, \mu\}$)
Output: sub-quantisers $\mathfrak{q}_1, \ldots, \mathfrak{q}_\mu$ and cluster indexes for each element $\boldsymbol{d} \in \mathcal{D}$

1: for all $\boldsymbol{d} \in \mathcal{D}$ do
2: split the data point $\boldsymbol{d}$ into μ sub-vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_\mu$ of length $\frac{dim}{\mu}$
3: for all $\boldsymbol{v}_j \in \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_\mu\}$ do
4: quantise split $\boldsymbol{v}_j$ using the j-th sub-quantiser $\mathfrak{q}_j$ producing at most γ_j cells
5: end for
6: end for

4.5.5.2 Search Algorithm

At on-line, query time, for the PQ index, using the list of cluster centre, we can compute approximative distances for each data point. These approximative distances can be used together with a priority queue of fixed capacity κ (see Algorithm 4.2) to early prune results from the candidate list.

The authors of [JDS11] suggest two methods for computing the distance: In the *symmetric* distance computation both the query vector and the database vectors are represented by their respective centroids, i.e.,

$$\delta(\boldsymbol{u}_i, \boldsymbol{u}_j) \approx \delta(\mathfrak{q}(\boldsymbol{u}_i), \mathfrak{q}(\boldsymbol{u}_j)) \tag{4.40}$$

where $\mathfrak{q}(\cdot)$ marks the quantisation yielding the cluster centre to which the sub-vector is quantised to. In the *asymmetric* case, on the other hand, the sub-vector is not encoded, i.e.,

$$\delta(\boldsymbol{u}_i, \boldsymbol{u}_j) \approx \delta(\boldsymbol{u}_i, \mathfrak{q}(\boldsymbol{u}_j)) \tag{4.41}$$

As the asymmetric distance computation shows less distortion without loss in retrieval performance, [JDS11] suggests to favour the asymmetric computation over the symmetric distance computation. To increase the retrieval efficiency, rather than computing the distance for each data point, the distance is read from a lookup table that is associated with each sub-quantiser. The search algorithm presented here has a time complexity $\mathcal{O}(n)$ as every element in the collection has to be scanned; however, it should be noted that by appropriately coding the list of cluster indexes, only small signatures have to be loaded.

Non-exhaustive PQ The authors of [JDS11] also present a non-exhaustive variation of the PQ index which avoids scanning the whole collection and whose complexity is lower than $\mathcal{O}(n)$. The variation makes use of a coarse quantiser $\tilde{\mathrm{q}}(\cdot)$ first, which in the beginning quantises a data point, e.g., using a coarse K-means clustering. The residual vector, which results from the difference of the data point to the centroid of the cluster it is projected to, i.e., $\boldsymbol{\rho}(\boldsymbol{d}) = \boldsymbol{d} - \tilde{\mathrm{q}}(\boldsymbol{d})$, is then indexed using the PQ index presented above.

4.5.6 Spectral Hashing

Spectral Hashing (SH) [WTF08; WFT12] belongs, similar to LSH, to the family of hash-based indexing methods, but falls into the category of "learning to hash", i.e., the hash functions are generated based on the data. The idea of the SH index is to find a hash function such that similar items are mapped to similar hash codes. Hence, small distances in the feature space should result in small Hamming distances between the codes. Table 4.5 gives a notational summary for SH indexes which we will detail in the following.

Table 4.5 Notational summary for SH indexes.

A	affinity/similarity matrix
Φ_j	j-th Laplacian eigenfunction
γ	distance value for which two elements are considered similar
I	identity matrix
λ_j	eigenvalues to the corresponding j-th Laplacian eigenfunction
v	length of signature/binary vector
$\min_{\mathbb{U}}, \max_{\mathbb{U}}$	minimum and maximum value of data space
s	signature of length v
$\boldsymbol{y}$	binary vector of length v
Y	matrix of binary vectors, i.e., $Y = [\boldsymbol{y}_1\ \boldsymbol{y}_2\ \ldots\ \boldsymbol{y}_n]$

4.5.6.1 Index Structure

Rather than using random projections as it is done with LSH, for the SH index, the authors apply a learning step for learning the representations in the index space. According to [WTF08], a code is said to be "good" if the number of bits to code the full dataset is small, similar items are mapped to a similar code and the code can be computed efficiently. More formally, the authors of [WTF08] require

- each bit of the codeword to have a 50% chance of being one or zero,
- the bits of the codeword being independent of each other,
- the codes to minimise the average Hamming distance between similar points in the feature space.

These requirements result ultimately in the following optimisation problem with corresponding constraints:

$$\begin{aligned} &\text{minimise} && \sum_{ij} A_{ij} \|\boldsymbol{y}_i - \boldsymbol{y}_j\|_2^2 \\ &\text{subject to} && \text{i)} \quad \boldsymbol{y}_i \in \{-1, 1\}^v \\ &&& \text{ii)} \quad \sum_i \boldsymbol{y}_i = 0 \\ &&& \text{iii)} \quad \frac{1}{n} \sum_i \boldsymbol{y}_i \boldsymbol{y}_i^T = I \end{aligned} \tag{4.42}$$

where $A_{ij} = \exp \frac{-\|\boldsymbol{u}_i - \boldsymbol{u}_j\|^2}{\gamma^2}$ denotes the similarity/affinity matrix (expressed by a Gaussian function with γ defining the distance value for which two data points are considered similar) in the feature space and v marks the length of the signature. While constraint i) denotes the fact that the contents of $\boldsymbol{y}_i$ are binary, constraint ii) models each position of the codeword to have a 50% chance of being 1 or -1. Constraint iii) models the independence of the single bits. The objective function in Equation 4.42 can be relaxed and rewritten to

$$\begin{aligned} &\text{minimise} && \operatorname{trace}(Y^T(\Delta - A)Y) \\ &\text{subject to} && \text{i)} \quad Y(i,j) \in \{-1, 1\} \\ &&& \text{ii)} \quad Y^T \mathbf{1} = 0 \\ &&& \text{iii)} \quad Y^T Y = I \end{aligned} \tag{4.43}$$

with $Y = [\boldsymbol{y}_1\ \boldsymbol{y}_2\ \ldots\ \boldsymbol{y}_n]$ being a $n \times b$ matrix, and Δ a diagonal $n \times n$ matrix with $\Delta(i,i) = \sum_j A(i,j)$. Moreover, $\mathbf{1}$ denotes an all-1 vector, and I is the identity matrix.

Removing constraint i), this problem becomes solvable. The solution to the problem is given by v eigenvectors of $\Delta - A$ with minimal eigenvalue. To map a new unseen data point into the embedding learned above, the problem turns into a spectral problem whose solutions to this out-of-sample extension are the first v eigenfunctions of the Laplace-Beltrami operator defined on manifolds. For A given as defined previously, a closed-form solution for these eigenfunction exists.

Figure 4.20 Visualisation of the eigenfunctions for the SH index: The illustration depicts the first four eigenfunctions for a uniform rectangular distribution together with the thresholded equivalent (based on [WTF08]).

Algorithm 4.11 summarises the algorithm for indexing a collection using the SH index. In there, the one-dimensional j-th Laplacian eigenfunction of a uniform distribution on $\mathbb{U} := [\min_{\mathbb{U}}, \max_{\mathbb{U}}]$ is analytically defined as

$$\Phi_j(x) = \sin\left(\frac{\pi}{2} + \frac{j\pi}{\max_{\mathbb{U}} - \min_{\mathbb{U}}} x\right) \tag{4.44}$$

and the corresponding eigenvalue is

$$\lambda_j = 1 - e^{\left(-\frac{\gamma^2}{2}\left|\frac{j\pi}{\max_{\mathbb{U}} - \min_{\mathbb{U}}}\right|^2\right)} \tag{4.45}$$

We visualise in Figure 4.20 the first four eigenfunctions for a uniform rectangular distribution (and their thresholded equivalents).

Algorithm 4.11 Indexing algorithm for the SH index (based on [WTF08]).

Input: set of n data points $\mathcal{D}$, number of bits v
Output: binary code s per element $\boldsymbol{d} \in \mathcal{D}$

1: find the principal components of the data using a PCA
2: project the data collection $\mathcal{D}$ using the PCA to $\mathcal{D}'$
3: compute the single-dimension analytical Laplacian eigenfunctions Φ_j along every dimension for the j smallest eigenvalues λ_j (with $j = [1, v]$)
4: for all $\boldsymbol{d}' \in \mathcal{D}'$ do
5: compute $\Phi_j(\boldsymbol{d}')$ (with $j = [1, v]$)
6: threshold the eigenfunctions at zero to obtain the binary codes
7: end for

To fit a multi-dimensional, uniform distribution to the data, first a Principal Component Analysis (PCA) is performed. Then, the data points are projected using the v eigenfunctions (see Equation 4.44) for the smallest eigenvalues (see Equation 4.45) along each PCA direction. Thresholding the eigenfunctions at zero and concatenating the results of the eigenfunctions finally constructs a binary code which can then be stored in the index.

The index stored on disk, hence, consists of the following parts:

- a projection matrix (from the PCA), to transform a new data point into a multi-dimensional uniform data space;
- a sorted list of eigenfunctions;
- the computed codes for all elements of the collection.

Multi-dimensional eigenfunctions While [WTF08] only applies one-dimensional eigenfunctions, in [WFT12], the authors extend the algorithm to also apply the outer-product eigenfunctions and in this way improve the result quality for high-dimensional data.

4.5.6.2 Search Algorithm

Following the definition of the resulting hash values in SH indexing, the Hamming distance between the produced binary codes is said to approximately reflect the true distance in the feature space. For querying, hence, an approximate distance can be computed, by computing the Hamming distance between the projected query and the value stored in the index, i.e.,

$$\delta_{\text{Hamming}}(\mathfrak{p}(\boldsymbol{q}), \mathfrak{p}(\boldsymbol{d})) \tag{4.46}$$

where $\mathfrak{p}(\cdot)$ denotes the projection into the Hamming space.

The search algorithm has to scan all elements in the collection, i.e., $\mathcal{O}(n)$, and keep the nearest neighbours (e.g., by storing the candidates in priority queue of fixed capacity, see Algorithm 4.2) found in the Hamming space as result candidates.

4.5.7 Vector Approximation-File

The Vector Approximation-File (VA-File) [WSB98] is an index structure with the goal of never yielding false dismissals, but at the same time to be very performant. The idea of VA-File indexing is to compress the data points in a quantisation step by storing only the quantisation indexes of the cells the points falls into and later query these indexes in a sequential manner. The quantisation is constructed such that it allows to compute the upper and lower bounds to the distance, which can ultimately be used for quickly pruning false hits at query time. Table 4.6 gives a notational summary for VA-File indexes.

Table 4.6 Notational summary for VA-File indexes.

$\bar{\delta}^{lBnd}$	lower bound distance value
$\bar{\delta}^{uBnd}$	upper bound distance value
$\bar{\delta}^{uBnd}_{\kappa}$	distance value to the κ-th nearest neighbour
$\mu_{i,j}$	marks for dimension dim_i $(\mu_{i,0}, \ldots, \mu_{i,v})$
$\Gamma_{i,k}$	cells in dimension dim_i $(\Gamma_{i,1}, \ldots, \Gamma_{i,v})$
v	number of cells per dimension

4.5.7.1 Index Structure

To quantise the data points, the feature space is divided into $dim \times v$ cells. For that, each dimension is split separately into v slices which are enumerated sequentially. The slices are created in a first training phase based on various strategies presented below and determined by marks denoting the beginning and the end of a slice $(\mu_{i,0}, \ldots, \mu_{i,v})$.

For dimension $i \in \{1, \ldots, dim\}$, for example, a point falls into a cell $\Gamma_{i,k}$, if it lies between the lower $(\mu_{i,k})$ and the upper mark $(\mu_{i,k+1})$, i.e.,

$$\Gamma_i(d_i) = k \Leftrightarrow \mu_{i,k} \leq d_i \leq \mu_{i,k+1} \tag{4.47}$$

The indexes of the cells a point falls into are ultimately the value stored in the index (see Figure 4.21). With this, a data point is ultimately quantised by approximating it by the cell it is surrounded by. By reasonably choosing the number of cells and appropriately coding the list of cell indexes, only small signatures have to be stored. The VA-File index stored on disk, ultimately, stores two elements:

- the marks for each dimension separately;
- for each data point the cell indexes for each dimensions.

In the following, we present two strategies for generating the marks for each dimension.

Equi-sized cells strategy Assuming a uniform distribution of the data along each dimension, the equi-sized cells strategy evenly distributes the partitioning points. The marks are computed by first determining the minimum and the maximum value for each dimension and equally subdividing the feature space along each dimension. The k-th mark in dimension i is, hence, given as

$$\mu_{i,k} = \mu_{i,0} + k \times \frac{\mu_{i,v} - \mu_{i,0}}{v} \tag{4.48}$$

where $\mu_{i,0}$ denotes the minimum and $\mu_{i,v}$ denotes the maximum value for a dimension i and v denotes the number of slices to assign for dimension i.

This strategy assumes a uniform distribution of the feature data to produce equally spaced marks. However, if the data is not equally distributed, the number of elements per

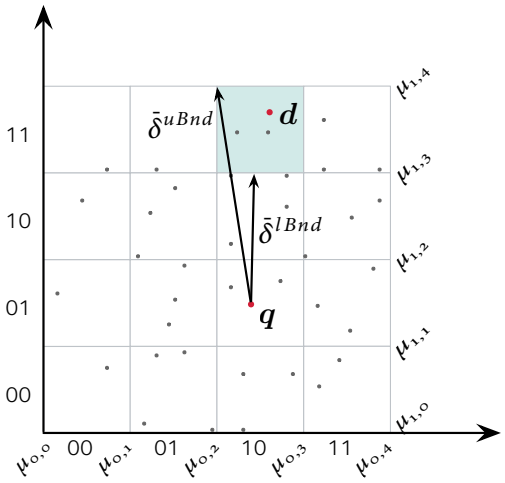


Figure 4.21 Distance computation in the VA-File index (based on [Web00, p. 84]).

cell may vary largely and lead to a poor pre-filtering in the first phase of the search and a deteriorated behaviour of the index structure at on-line time.

Equi-populated cells strategy The equi-populated cells strategy, on the other hand, tries to partition the data space such that each slice is equally populated. It follows the ideas of quantile hashing [KS87] and constructs the marks by stochastically approximating the underlying distribution. The advantage of this strategy lies in the fact that the probability of a point being in a cell γ is similar for each cell, i.e.,

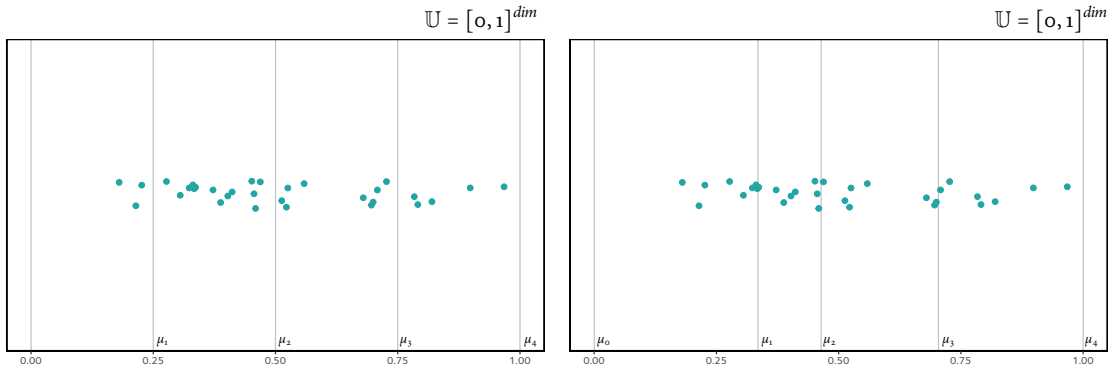
$$\Pr[\text{“in cell } \gamma\text{”}] = \prod_{i=1}^{dim} \frac{1}{\nu} = \nu^{-dim} \tag{4.49}$$

Moreover, the probability of two points sharing a cell γ is

$$\Pr[\text{“two points share cell } \gamma\text{”}] = 1 - (1 - \frac{1}{\nu^{dim}})^{n-1} \approx \frac{n}{\nu^{dim}} \tag{4.50}$$

This results in an optimal distribution of the probability over the whole data space and, thus, to a much more effective early pruning.

Figure 4.22 visualises both mark generation strategies. It can be clearly seen that for a non-uniform data distribution the equi-populated cells strategy produces cells with an approximately similar number of elements, while the equi-sized cells strategy positions the cells at the same distance from each other, disregarding the underlying data distribution.



(a) Equi-sized cells strategy.

(b) Equi-populated cells strategy.

Figure 4.22 Strategies for generating the marks for the VA-File index.

4.5.7.2 Search Algorithm

The generated VA-File signature allows to early determine upper and lower bounds for the distance with respect to a given query. In Figure 4.21, we visualise the computation of the upper and lower distance bounds in a two-dimensional space.

The lower bound distance for dimension i, given a query point $\boldsymbol{q}$, for a Minkowski distance $\delta_{L_p}(\cdot,\cdot)$, can be computed by

$$\delta^{lBnd}_{L_p,i}(\boldsymbol{q}) = \begin{cases} (\mu_{i,k} - q_i)^p & \text{for } q_i < \mu_{i,k} \\ (q_i - \mu_{i,k+1})^p & \text{for } q_i > \mu_{i,k+1} \\ 0 & \text{for } q_i \in \Gamma_{i,k} \end{cases} \tag{4.51}$$

Similarly, the upper bound distance is defined as

$$\delta^{uBnd}_{L_p,i}(\boldsymbol{q}) = \begin{cases} (\mu_{i,k+1} - q_i)^p & \text{for } q_i \leq \frac{\mu_{i,k}+\mu_{i,k+1}}{2} \\ (q_i - \mu_{i,k})^p & \text{otherwise} \end{cases} \tag{4.52}$$

The total lower bound and the upper bound to the distance of a data point $\boldsymbol{d}$ are ultimately given by summing up for each dimension the distance of the query point to the cell the data point falls into, i.e.,

$$\bar{\delta}^{uBnd}_{L_p} = \sqrt[p]{\sum_{i=1}^{dim} \bar{\delta}^{uBnd}_{L_p,i}} \qquad \bar{\delta}^{lBnd}_{L_p} = \sqrt[p]{\sum_{i=1}^{dim} \bar{\delta}^{lBnd}_{L_p,i}} \tag{4.53}$$

where $\bar{\delta}^{lBnd}_{L_p,i}$ and $\bar{\delta}^{uBnd}_{L_p,i}$ denote the evaluated lower distance bound function and upper distance bound function, respectively.

To increase the retrieval efficiency, we can avoid recomputing distances multiple times at search time (especially for large n). Instead, the distances from the query vector to each

cell can be precomputed once and, hence, the calculation of the final distance becomes a lookup and a sum over the single dimensions.

Using the approximated distance given by the upper and lower distance bounds, the search algorithm decides whether to prune a data point or whether to add it to the candidate set $\mathcal{A}_{\text{TID}}$. A point is added to the list of candidates if

- κ elements have not yet been found,
- κ elements have been found, but the lower distance bound of the new data point is lower than the upper distance bound of any of the elements added so far to the candidate set.

Hence, a priority queue of fixed capacity can be used (see Algorithm 4.2). Following this principle, the VA-File index will always return all true positives and have no false dismissals.

Formally, a data point is in the candidate set $\mathcal{A}_{\text{TID}}$ if its lower bound is smaller than the distance of the κ-th (sorted) nearest neighbour as denoted in

$$\delta_{L_p}^{lBnd}(\boldsymbol{d}, \boldsymbol{q}) \leq \bar{\delta}_{\kappa}^{uBnd} \Rightarrow \boldsymbol{d} \in \mathcal{A}_{\text{TID}} \tag{4.54}$$

where we use $\bar{\delta}_{\kappa}$ to denote the distance to the κ-th nearest neighbour (for the corresponding document $\boldsymbol{d}_{\kappa}$ to the distance $\bar{\delta}_{\kappa}$, it is true that $\boldsymbol{d}_{\kappa} \in \kappa NN(\boldsymbol{q})$ and $\boldsymbol{d}_{\kappa} \notin [\kappa - 1]NN(\boldsymbol{q})$).

The search algorithm has a computational complexity of $\mathcal{O}(n)$, given that all points have to be scanned. However, it should be noted that by appropriately coding the list of cell indexes, only small signatures need to be loaded. In Algorithm 4.12, we summarise the search algorithm for VA-File.

Algorithm 4.12 Search algorithm for VA-File index (based on [Web00, pp. 97]).

Input: index values $(\text{TID}, \ddot{\imath}) \in I_R$, marks $\mu_{i,j}\ \forall i, j$, number of elements to retrieve κ of the κNN search, query vector $\boldsymbol{q}$
Output: candidate set $\mathcal{A}_{\text{TID}}$

```
1:  priority queue P ← ∅
2:  A_TID ← ∅
3:  for all (TID, ï) ∈ I_R do
4:      δ̄^lBnd_{L_p,i} ← lower distance bound from q using the cell indexes ï
5:      if P.length ≤ κ or δ̄^lBnd_{L_p,i} ≤ max(P) then
6:          A_TID ← A_TID ∪ {TID}
7:          δ̄^uBnd_{L_p,i} ← upper distance bound from q using the cell indexes ï
8:          P ← P ∪ {δ̄^uBnd_{L_p,i}}
9:          if P.length > κ then
10:             P.dropLast()
11:         end if
12:     end if
13: end for
```

4.5.7.3 VA⁺-File

[FTA+06] suggests a number of improvements to the original VA-File index introduced previously, particularly when the data is highly correlated or clustered. The authors argue that for constructing an appropriate VA-File index,

- the data should be transformed into a more suitable domain;
- the available space for storing cell indexes should not be distributed uniformly among the dimensions, but dimensions with a higher information value should split the dimensions in more cells;
- a scalar quantiser should be designed separately for each dimension without making any assumption on the uniformity of data, but instead make use of data statistics.

In this way, the goal is to increase the number of elements pruned during the index scan to, consequently, also lower the number of elements that have to be read in the subsequent data scan. To achieve this, the VA⁺-File [FTA+06] involves three steps presented in the following.

First, the authors propose to apply a PCA to de-correlate the dimensions of the data (see Figure 4.23(b)). While the PCA step can also be used for reducing the dimensionality of the data, it is not necessary to do so.

Second, rather than assigning the same number of bits and, hence, also the same number of marks to every dimension, it is especially favourable in the decorrelated data domain to assign to each dimension a number of bits which reflects the information available in the dimension. Hence, the goal is to minimise the number of necessary bits to reach a maximum accuracy. For this, let σ_i^2 denote the variance of a dimension i, and β_i the number of bits assigned to represent the space available for storing the cell indexes for dimension i (and β being the total length of the signature). Following the principle that if $\sigma_i^2 \geq 4^m \sigma_j^2$, then it is more beneficial to have $\beta_i \geq \beta_j + m$, Algorithm 4.13 introduces a non-uniform bit allocation algorithm as presented by the authors. While the algorithm has not yet distributed all bits to the existing dimensions, it will select the dimension with the highest variance and assign one more bit to it. For a number of bits β_i, the dimension i can be split into 2^{β_i} cells. Figure 4.23(c) visualises the effects of the non-uniform bit allocation.

Finally, the authors suggest to use a separate quantiser for each dimension that is based on the statics of the data available. As the maximum number of cells per dimension is known from the assigned number of bits per dimension, using the K-means algorithm, a quantiser for each dimension can be learned based on the data (see Figure 4.23(d)). To get from the clusters to the VA-File marks, we take the mean between two cluster centres.

Algorithm 4.13 Non-uniform bit allocation for VA$^+$-File index (based on [FTA+06]).

Input: variance σ_i^2 for every dimension $i \in \{1, \ldots, dim\}$
Output: bit allocation schema β_i $(i \in \{1, \ldots, dim\})$

1: $e_i = \sigma_i^2 \quad \forall i \in \{1, \ldots, dim\}$
2: $b_i = 0 \quad \forall i \in \{1, \ldots, dim\}$
3: $m = 0$
4: while $m < \beta$ do
5: $\quad j = \text{argmax}_i \, e_i$
6: $\quad \beta_j \leftarrow \beta_j + 1$
7: $\quad e_j \leftarrow \frac{e_j}{4}$
8: $\quad m \leftarrow m + 1$
9: end while

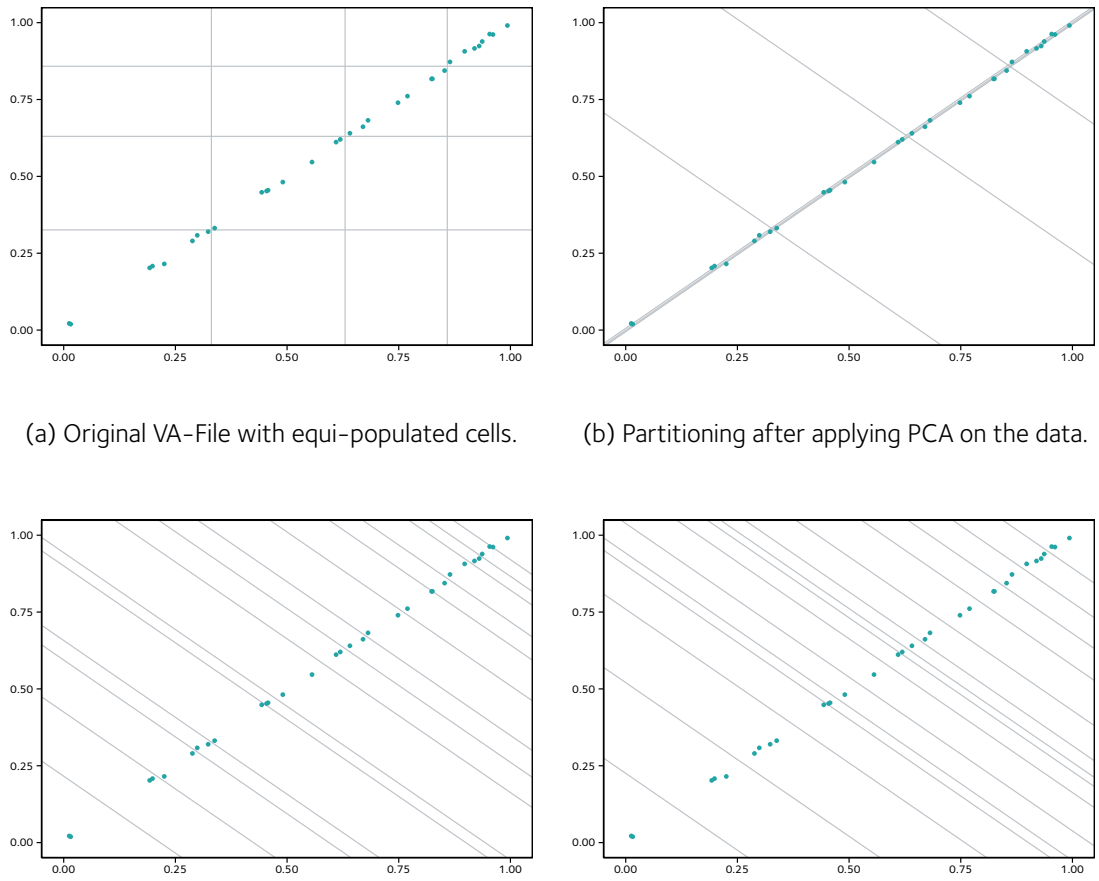
With these three adjustments to the original VA-File algorithm, the VA$^+$ indexing scheme is supposed to work better for unknown data distributions. In Algorithm 4.14, we summarise the construction of a VA$^+$-File index. Figure 4.24 visualises the four main steps of the algorithm.

Algorithm 4.14 Indexing algorithm for VA$^+$-File index (based on [FTA+06]).

Input: set of n data points $\mathcal{D}$, number of bits β
Output: binary code s per element $\boldsymbol{d} \in \mathcal{D}$

1: $\mathcal{D}' \leftarrow$ project the data collection $\mathcal{D}$ using a PCA
2: define a bit assignment schema based on the variance for every dimension
3: use a K-means algorithm for generating clusters and define marks by taking the mean between two cluster centres
4: for all $\boldsymbol{d}' \in \mathcal{D}'$ do
5: $\quad$ for $j \leftarrow 1 \ldots dim$ do
6: $\quad\quad$ determine the cell the value of d'_j falls into and store index in signature s
7: $\quad$ end for
8: end for

The VA$^+$-File index is queried similarly to the original VA-File index, with the difference that the query has first to be transformed into the new data domain using the PCA. The subsequent steps then equally involve the computation of distance bounds and the scanning of all elements of the collection (i.e., $\mathcal{O}(n)$). While the worst-time search time complexity of VA$^+$-File is the same as for VA-File, it can be expected that VA$^+$-File performs better for non-uniformly distributed data.



(a) Original VA-File with equi-populated cells.

(b) Partitioning after applying PCA on the data.

(c) Non-uniform bit distribution between dimensions.

(d) Partitioning based on K-means algorithm.

Figure 4.23 Construction of the VA$^+$-File index: The illustration visualises the necessary steps in creating a VA$^+$-File index. For better comparability of the visualisations, we use the inverse PCA transformation on the VA-File (based on [FTA+06]).

4.5.8 Classification of Index Structures for High-Dimensional Data

In the following, we categorise the high-dimensional index structures introduced previously according to a number of classes presented in this section. The summary of the categorisation is given in Table 4.8. We refer to [AKM+16; PC09; JZS+16] for other classifications presented here. Other categorisations not considered in this section include, for example, the distributability of the index structure, the suitability of the index structures for certain distance functions, index structures which introduce a hierarchy into the data, etc.

Data dependence Data dependent indexes are said to be generated based on the distribution of the underlying data. Such indexes often involve a training phase in which a training set is used to adapt the index creation parameters.

For instance, for the indexes presented in this section, the CP and MI-File indexes select representatives and leaders from the data. Similarly, SH uses training data to determine the eigenfunctions to be used. The PQ index generates the K-means clustering based on the data, and VA-File establishes the marks using a training set. LSH, on the other hand, is said to be agnostic to the data as the hash functions are not selected based on the data distribution, but are random projections.

Signature generation This category classifies the index structures based on the operations involved in creating the index and the signatures stored in the index. We distinguish indexes that perform a data quantisation from indexes which perform a projection into a new space (e.g., by embedding a data point into a binary space).

For instance, the CP index quantises the vectors to selected representatives. PQ performs a quantisation on the vector splits and, hence, performs the quantisation on a lower dimensionality. VA-File performs a scalar quantisation per dimension. LSH and SH on the other hand, perform a projection to the hash space, while MI-File projects the vectors to a new space defined by the distances to the selected reference points.

Approximation type The approximation type [PC09; AGS14] describes how the approximation reduces the cost of a similarity search. Indexes in the category of space transformation solve the problem on a new approximate space in which executing the query is a computationally less expensive task. For example, dimensionality reductions or quantising techniques can be regarded as falling into this class. On the other hand, the class of pruning techniques contains indexes which reduce the number of objects to be compared by avoiding looking at certain regions of the data space. This class involves, for example, tree-based organisations or hierarchical decompositions of the space.

For instance, CP will prune very many data points by not considering the data points assigned to other clusters. LSH will prune all points which are not in one of the buckets to which the query is hashed to. MI-File will perform a projection to a new space, but it does not allow to prune regions from the results. Similarly, PQ, SH and VA-File also do not allow to aggressively prune the space.

Exhaustive search Indexes requiring an exhaustive search are index structures which have to scan over each element in the collection to determine the set of result candidates. Indexes that do not fall into this class are able to partition the indexed data and, at query time, prune results from the candidate set without having to access all data points.

For instance, the VA-File index requires reading every element in the index, as for each element a distance bound has to be computed which is used to determine whether it falls into the candidate set. Similarly, the SH index produces a distance approximation using the signature stored in the index; however, the signature does not allow to partition the data and early prune elements, but allows to compute an approximate distance. The PQ index, in its default implementation, also requires an exhaustive search, though, note that [JDS11] also proposes a non-exhaustive variation of the PQ index using a coarse quantiser. The MI-File index as presented in [CNB+01] is also based on an exhaustive search; the authors [AS08; AGS14], on the other hand, propose a non-exhaustive variation of the MI-File index using an inverted index, in which only the data points appearing with the closest representatives are considered.

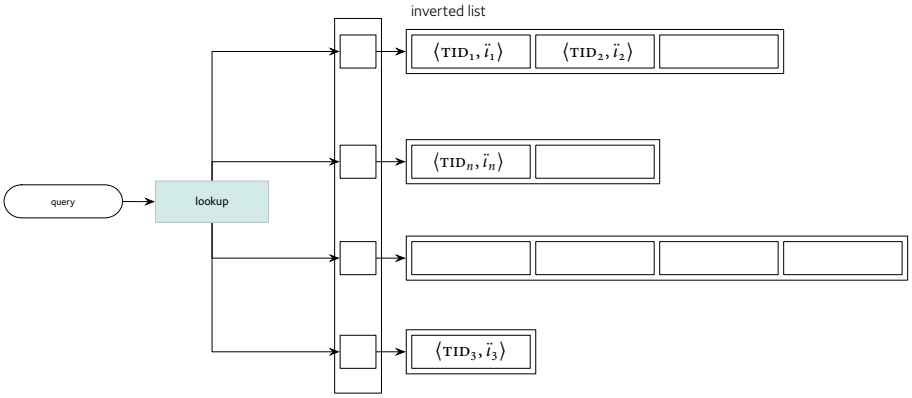
Lookup vs. ranking In this category, we distinguish indexes that perform lookup operations and indexes which return a ranking in the candidate set. The former index structures increase the retrieval performance by reducing the number of distance computations to perform. The index structures that return a ranking, on the other hand, perform an exhaustive search as they compare the query with each data point in the index space. However, the goal of these index structures is to be able to evaluate the approximate distance in a very efficient way.[8]

This classification can further be separated into index structures which preserve the distance and are, therefore, able to reconstruct some distance information from the value stored in the index, and indexes which are not.

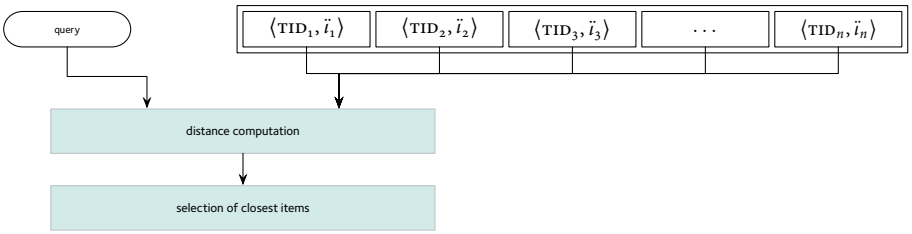
For instance, based on the value stored in the LSH index, no distance estimation can be given for LSH as the hash code is only used for a lookup. For the SH index, instead, the Hamming distance between two signatures denoting the code similarity, should reflect the true distance; however, the index does not provide any distance information for the final distance. Similarly, MI-File allows a ranking based on the rank correlation distance, but the resulting distance is not in the bounds of the true distance. CP and PQ, on the other hand, allow to estimate the distance based on the cluster leader. Finally, VA-File produces a ranking based on distance bounds which reflect the information of the true distance.

Incremental updates The categorisation on incremental updates marks indexes which can be updated without loss in retrieval quality, compared to indexes which necessitate rebuilding the full index to accommodate index updates. This becomes more and more

[8] Note, moreover, that indexes which are able to return a meaningful distance could potentially even avoid a subsequent data scan, as the candidate set already has an ordering of elements based on the distance retrieved from the index.



(a) Lookup of index items.



(b) Raking of index items.

Figure 4.24 Visualisation of the lookup and the ranking strategy used with indexes (based on [JZS+16]).

important, as noted in [ZIP16], since the gap between the time the data is available and the time when it can be accessed increases dramatically with the collection size.

For example, for LSH index updates are easily possible given the fact that the index is independent of the data distribution. Similarly, for CP and MI-File which choose the leader points or the reference points, respectively, at index creation time, updating the index is possible. For PQ, SH and VA-File, on the other hand, the distribution of the data influences the index creation and, hence, would need adoption at insertion time. While small number of updates will not perceivably deteriorate the index quality, too many insertions in the index will require a new generation of the index.

Quality guarantees This class categorises index structures by the quality guarantees given. We distinguish classes which provide no guarantees, classes which provide probabilistic guarantees and indexes which are exact. While the former two classes may generate false dismissals by missing relevant results in the index scanning phase, and are, hence, approximate indexes, the latter denotes indexes which do not miss any relevant results (but, of course, returns also false positives).

For instance, CP cannot give any guarantees that the quantisation to the closest leader may not lead to missing candidate results. Similarly, MI-File and SH cannot give any guarantees on the results yielded during the index scan. Both, LSH (for certain hash functions) and PQ, provide some probabilistic guarantees. VA-File, on the other hand, falls into the category of exact index structures.

In Table 4.7, we summarise the storage and search complexity for the indexes presented. It must, however, be noted that the complexity only form one side of the coin and only present part of the truth: Consider, for example, a VA-File index. While the complexity for scanning the index is $\mathcal{O}(n)$ and, hence, equal to the full data scan, by appropriately coding the stored information, the data scanned by the VA-File is much smaller than a full data scan. Hence, only comparing the complexities is not enough; for a true comparison of the index structures, a qualitative evaluation (see Chapter 7) is necessary.

Table 4.8 gives an overview of the classification of the index structures presented in this chapter.

Table 4.7 Comparison of storage and query complexity of indexes for high-dimensional data (in the average case): Note that the variables belong to the notations introduced for the index structure and the same symbol may, hence, differ in semantics within this table. For the storage complexity we use $\texttt{size}(\cdot)$ to denote the size of the function argument.

index structure	storage complexity	query complexity
CP	$\mathcal{O}(\nu\ \texttt{size}(\text{ref point}) + n\ \texttt{size}(\text{ref id}))$	$\mathcal{O}((1+\beta)\alpha\frac{n}{\nu})$
LSH	$\mathcal{O}(\alpha\ \beta\ \texttt{size}(\text{hash function})) + \mathcal{O}(n\ \beta\ \texttt{size}(\text{bucket id}))$	$\mathcal{O}(dim\ \alpha\beta) + \mathcal{O}(\beta\ \times \frac{n}{\mu})$
MI	$\mathcal{O}(\nu\ \texttt{size}(\text{ref point})) + \mathcal{O}(n\ \kappa_\iota\ \texttt{size}(\text{ref id}))$	$\mathcal{O}(n\kappa_\sigma)$
PQ	$\mathcal{O}(\mu\ \gamma\ \texttt{size}(\text{cluster point})) + \mathcal{O}(n\ \mu\ \texttt{size}(\text{cluster id}))$	$\mathcal{O}(n\ \mu)$
SH	$\mathcal{O}(\texttt{size}(\text{PCA matrix})) + \mathcal{O}(\texttt{size}(\text{eigenfunctions})) + \mathcal{O}(n\ \nu)$	$\mathcal{O}(dim) + \mathcal{O}(n)$
VA	$\mathcal{O}(dim\ \nu) + \mathcal{O}(n\ dim\ \texttt{size}(\text{cell index}))$	$\mathcal{O}(dim\ \nu) + \mathcal{O}(n)$

Table 4.8 Comparison of indexes for high-dimensional data.

index structure	data dependence	signature generation	approximation type	exhaustive search	lookup vs. ranking	incremental updates	quality guarantees
CP	yes through selection of representatives	vector quantisation of full vector	pruning	no only relevant clusters are read	lookup by leader distance related to true distance	yes	no
LSH	no	projection based on hash functions	pruning	no only relevant buckets are read	lookup	yes	probabilistic guarantees
MI	yes through selection of representatives	projection to sorted vector of references	space transformation	yes non-exhaustive MI introduced in [AS08]	ranking by rank correlation not related to true distance	yes	no
PQ	yes through creation of clustering	vector quantisation at lower dimensionalities	space transformation	yes non-exhaustive PQ introduced in [JDS11]	ranking by sum of distance to cluster centroids related to true distance	limited	probabilistic guarantees
SH	yes through learning of eigenfunctions	projection based on thresholded eigenfunctions	space transformation	yes	ranking by Hamming distance not related to true distance	limited	no
VA	yes through definition of marks	scalar quantisation per dimension	space transformation	yes	ranking through lower and upper distance bounds related to true distance	limited	exact

4.6 Distribution

The efforts in reducing the computational complexity for retrieval purposes constitute only one side of the coin in improving the system efficiency. To further lower the latency, in particular at query time, and to scale to large collection sizes, it is imperative to also address the distribution of both the work and data. In the following, we discuss various aspects of distribution in a database system. We first detail the architectural aspects of a distributed system and expound on the distribution from a work and from a data perspective.

4.6.1 Architecture of a Distributed System

In this section, we consider the architecture of a distributed system from three angles and discuss the system, the data model and the component architecture.

4.6.1.1 System Architecture

In terms of the system architecture, literature generally distinguishes master/worker and Peer-to-Peer (P2P) architectures. In a master/worker architecture, a master takes over the role of orchestrating many workers in their tasks. Hence, the architecture is said to be master-oriented in that clients initiate requests which are processed by workers under the supervision of the master. On the other hand, in P2P networks, all peers feature the same stack of functionality without necessitating a centralised coordinator; as a consequence, all peers play an equal role, and are both master and worker at the same time. A request posed by a client can be answered by any of the peers in the network, and the coordinator role is dependent on the request started within the P2P network. Representatives of P2P networks include, for instance, networks based on distributed hash tables such as CHORD [SMK+01]. In Figure 4.25, we contrast both a master/worker and a P2P architecture.

Compared to text-retrieval systems, in multimedia retrieval, P2P systems have only seen a limited adoption. The reason for this is that the semantical parts used for querying can more easily be distinguished in text retrieval (cf. [MBN+06]). As a consequence, a major stumbling block becomes the fact that there is no obvious method to cluster similar items together (see also Section 4.5.1.4). In the multimedia context, [MH03], for example, presents a system which allows to perform similarity searches over a P2P network. The authors employ global clustering to put similar data items to the same peer within the network. At query time, queries are forwarded to the corresponding peer storing the cluster with the similar items to the given query. Likewise, [AGL+07] employs a P2P network for multimedia content searches and presents a routing algorithm which avoids flooding the network when searching for similar items. In [BFL+10] the authors use an index structure

(a) Master/worker architecture.

(b) P2P architecture.

Figure 4.25 System architecture of distributed systems.

based on space-filling curves and combine it with the Chord P2P protocol for answering queries posed by a client.

Master/worker systems, on the other hand, are more common in the multimedia retrieval context. For instance, the retrieval system presented in [Web97; WBS00] follows a master/worker approach in that a coordinator is necessary to orchestrate the available nodes for performing the query. The system employs a distributed VA-File index which is queried on all sites in parallel. DISIMA [OÖL+97] follows also a master/worker architecture with a central broker based on CORBA. Similarly, in [Vri99], a centralised master is the main point of querying which in turn initiates sub-queries at the workers.

4.6.1.2 Data Model Architecture

[Cod90, pp. 345] lists distribution independence, according to which the available data may be distributed (transparently) without any necessary changes to the application even in the case of re-distribution of the data, as an important independence property of a database system. As a consequence, the user should be presented with one consistent and unified logical structure of the data residing across all sites. This external view is derived from the *global logical data model* which subsumes the local conceptual data models of the participating sites. The global conceptual data model provides distribution, partitioning and location transparency; as a consequence, the operational details of the network, the partitioning and the placement of the data are hidden from the external view. The *local logical data model* defines the logical data organisation at each site and hides the storage internals, while the physical data model defines the local, physical organisation, which is again not visible at the logical level.

The right-hand side of Figure 4.26 visualises the data model architecture of a distributed database as discussed above. The illustration extends the schema architecture as given by the three-level data model architecture (see Section 4.1.1).

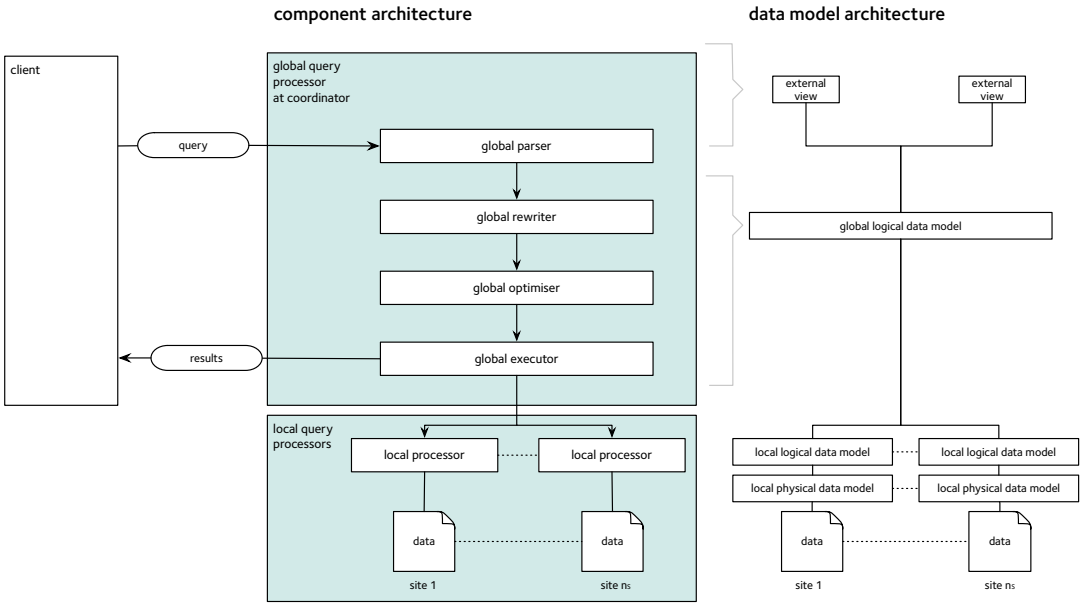


Figure 4.26 Component and data model architecture of a distributed database (based on [EN11, pp. 889; ÖV11, p. 34]).

4.6.1.3 Component Architecture

Similar to the data model architecture, the component architecture distinguishes both global and local components. The global components are generally responsible for the communication with a client, the reception and processing of requests (e.g., performing global-level analyses and optimisations, providing an execution monitor for running a query, etc.). On the other hand, the local processors perform local query optimisations and ultimately execute the localised part of the query.

Extending the architecture presented in Figure 4.6, on the left-hand side of Figure 4.26, the architecture of a distributed database is depicted which adopts the following components: At the coordinator site[9] and similar to the local processor, the global query parser parses the incoming queries which are optimised at the global level by the global query rewriter and optimiser. The global query executor takes over the responsibility for executing a query and monitoring the execution at global scale. Ultimately, the local query processors hide away the internals of the individual sites, providing the global view with a consistent logical local view.

4.6.2 Work Distribution

We detail in this section the distribution of work and, first, expound on distributed query processing. We then consider distributed data processing for large-scale applications.

[9] In a P2P system, the coordinator site is considered the site contacted by the client which initiates the query within the system.

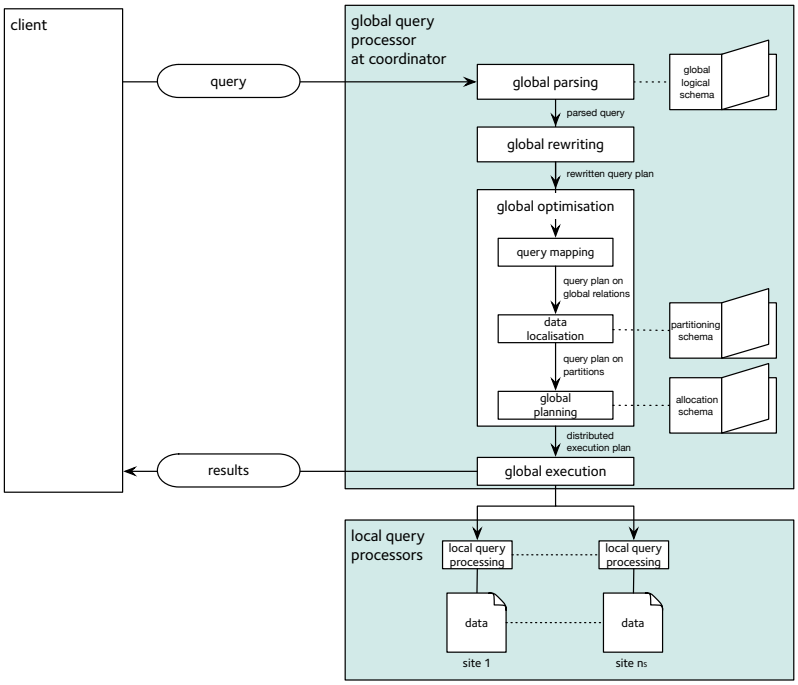


Figure 4.27 Distributed query processing (based on [ÖV11, p. 216]).

4.6.2.1 Distributed Query Processing

In the following, we extend the local query processing model we have introduced in Section 4.3 and consider processing a query in a distributed manner, as depicted in Figure 4.27.

Figure 4.27 distinguishes the processing of a query on a global level at a coordinator site and the distributed processing within multiple local processors (based on [ÖV11, pp. 215; EN11, pp. 902]).

1. A query enters the system at a coordinator which parses and rewrites the query, similar to the query processing discussed in Section 4.3.
2. For the optimisation and planning, three steps are involved: At the first query mapping layer, which is agnostic to the distribution of relations and only refers to the global logical data model, the query execution is optimised from a logical perspective similar as in a non-distributed database (however, the optimisation may consider certain distribution costs).
3. In the next step, the query is localised by considering the partitioning information of the involved relations (as stored in the partitioning schema in the catalogue). Hence, for each relation it has to be considered which attributes reside within which partition. In this step, the query is decomposed into sub-queries for each partition.
4. The query is then optimised physically: While the previous optimisation considered the logical optimisation, and the previous step produced a query for each fragment, the order of execution of the sub-queries has not yet been determined and the allocation of

partitions to sites (as stored in the allocation schema in the catalogue) and the resulting communication costs, have not yet been considered. The global optimisation step results in a distributed execution plan.

5. The distributed execution plan is executed under the orchestration of the global executor which invokes the queries at the local sites.
6. At the local sites, the sub-queries are executed involving techniques similar to those discussed in Section 4.3, including local parsing and local optimisation.
7. Finally, at the coordinator site, the results from the single sites are merged to one coherent result set.

We will consider partitioning and the allocation of the partitions to sites in Section 4.6.3 in more detail.

4.6.2.2 Distributed Data Processing

Map/reduce [DG08] is a programming model for distributed computing and nowadays the predominant technique for large-scale processing of data. The model divides work into chunks that can be assigned to a large number of separate workers under the orchestration of a coordinator.

For this, the model asks for the specification of two functions, a *map* and a *reduce* function, which are, following [DG08], formally defined as

$$\begin{aligned} \text{map} &: (id_1, v_1) \rightarrow \text{list}(id_2, v_2) \\ \text{reduce} &: (id_2, \text{list}(v_2)) \rightarrow \text{list}(v_2) \end{aligned} \tag{4.55}$$

where id_1 and id_2 denote keys to the values v_1 and v_2, respectively. Hence, the map function takes as input a key/value pair and emits a set of key/value pairs. These sets are shuffled and grouped by key; for each key all values are fed into a reduce function which produces a final output. Figure 4.28 summarises the processing of map/reduce jobs.

The focus of map/reduce-based processing is generally on background tasks, given that map/reduce tasks are often heavy-weight processing tasks and, hence, not suited for interactive processing. In multimedia retrieval, map/reduce has been applied, for example, for the feature transformation (e.g., [GAJ+17; MSG+13a; SMG+13; WYL+10; LCW+14]), *κNN* joins (e.g., [LSC+12; ZLJ12; YIS12; CJN+11]) and for batched retrieval (e.g., [MSG+13a; GAJ+17; CJN+11]). For index processing, [YL13] applies map/reduce with a locality-sensitive hashing index; [AH14] uses map/reduce with a k-d-tree. In [Ams14], the author proposes the distribution of the (extended) Cluster Pruning (DeCP) index to be built and searched using the map/reduce paradigm.

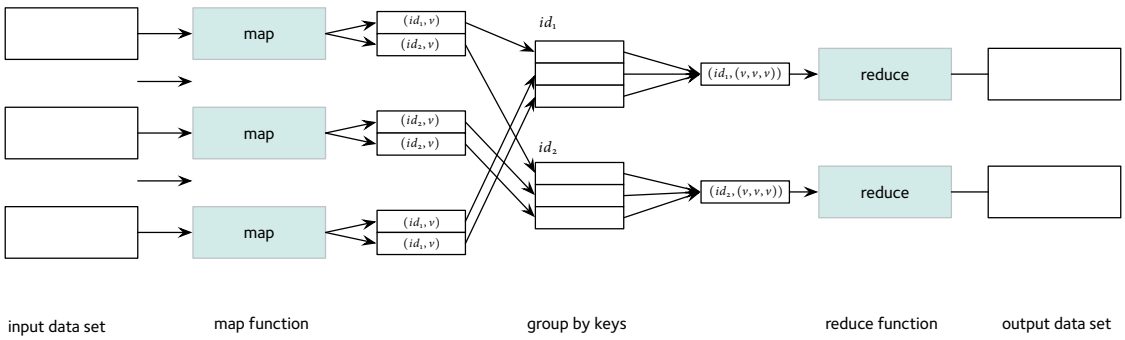


Figure 4.28 Overview of map/reduce processing (based on [ÖV11, p. 759]).

Table 4.9 Common operations of functional data processing: The operations noted here are a sub-set of the available operations in Apache Spark.

Operation	Description
`.map`	returns a data set containing the result of applying a function to each element
`.flatMap`	returns from a set of collections a flattened data set containing the elements as results of applying a function
`.reduce`	returns the aggregations of elements in the data set
`.filter`	returns a data set that only contains elements which satisfy a given predicate
`.join`	returns the results of joining two data sets; various join operations are available, including the inner, outer, full, semi, anti and natural join
`.groupBy`	returns a clustering of elements into groups defined by a mapping function

The map/reduce model, as well as the most widely used implementation of it found in the Hadoop stack, does not consider data locality in the processing of the data. This strongly impacts performance for the worse [JSX+10]. Other implementations building on the ideas of map/reduce try to overcome this issue by considering data locality and performing the work as close as possible to the data (e.g., [ZCF+10]).

Functional data processing [WSZ17a] can be considered as an evolution to the simple map/reduce model by supporting more data processing primitives inspired by functional programming. The model, hence, provides a declarative style of programming with functions as first-level citizens of the language. In Table 4.9, we give an exemplary list of common operations which can often be found in functional data processing.

4.6.3 Data Distribution

4.6.3.1 Data Partitioning

To be able to handle the vast amounts of data, database systems have to fragment their data and distribute it over multiple sites. In a shared-nothing architecture, compared to a shared-disk architecture, spreading data over multiple sites is done by first partitioning a relation into disjoint sub-sets and then allocating the sub-sets to the available sites. The resulting partitioning schema is recorded in the (global) catalogue defining the rules for fragmenting a relation. The definition must ensure that the whole database can ultimately be reconstructed using the inverse operations. In the following, we discuss three partitioning strategies, i.e., horizontal, vertical and hybrid partitioning.

Horizontal partitioning Horizontally partitioning a relation results in a row-based decomposition of a relation and, hence, the assignment of a tuple as a whole to one partition. We define a horizontal partitioning function as a function

$$p^{\text{hp}} : R \to P \tag{4.56}$$

i.e., a function mapping a tuple t from the relation R to a partition ($P := \{p^{\text{hp}}_{R,1}, \ldots, p^{\text{hp}}_{R,n_P}\}$, denoting the set of available partitions and where n_P is the number of partitions defined by the user). The partition $p^{\text{hp}}_{R,i}$ of a relation R, created from a horizontal partitioning scheme, is then given by

$$p^{\text{hp}}_{R,i} = \sigma_{\varphi_i}(R) \tag{4.57}$$

where $\sigma_{\varphi_i}(\cdot)$ denotes a selection operation over relation R and φ_i marks a filtering predicate given through the partitioning scheme. The filtering predicate may relate to the property of an attribute yielding a primary horizontal partitioning; in more complex cases, the predicate may be derived otherwise (e.g., by another relation), resulting in a derived horizontal partitioning [ÖV11, pp. 81]. A horizontally partitioned relation can be reconstructed using a union operation of the single partitions, i.e.,

$$R = \bigcup_{\forall i} p^{\text{hp}}_{R,i} \tag{4.58}$$

[HSH07, p. 168] distinguishes four partitioning schemes for horizontal partitioning:

round-robin horizontal partitioning Round-robin horizontal partitioning assigns one tuple after each other in a round-robin fashion to the available partitions, i.e., the tuple t_i is assigned to the partition $i \mod n_P$. This partitioning scheme ensures uniform data distribution.

hash-based horizontal partitioning Hash-based horizontal partitioning partitions tuples based on the result of a hash function applied on an attribute.

range-based horizontal partitioning Range-based horizontal partitioning assigns tuples to partitions based on value intervals of an attribute.

hybrid horizontal partitioning Hybrid horizontal partitioning combines both range- and hash-based partitioning.

Vertical partitioning Vertical partitioning, on the other hand, results in partitions containing only a sub-set of the attributes of the relation together with a tuple identifier (TID) which is necessary to be able to reconstruct the relation. Hence, the relation is said to be decomposed by columns. In the context of vertical partitioning, a partition is given by

$$p_{R,i}^{\text{vp}} = \pi_{A_i \cup \{\text{TID}\}}(R) \tag{4.59}$$

where $A_i \subseteq \text{SCH}(R)$ denotes a sub-set of the attributes of the relation and TID a tuple identifier. A vertically partitioned relation is reconstructed by means of a join operation over the tuple identifier, i.e.,

$$R = p_{R,1}^{\text{vp}} \bowtie_{\text{TID}} \cdots \bowtie_{\text{TID}} p_{R,n_P}^{\text{vp}} \tag{4.60}$$

Hybrid partitioning Hybrid partitioning combines both horizontal and vertical partitioning, possibly recursively repeated. The partitions are constructed as a sequence of selection/projection operations, e.g.,

$$p_{R,i}^{\text{hybp}} = \pi_{A_i \cup \{\text{TID}\}}(\sigma_{\varphi_i}(R)) \tag{4.61}$$

4.6.3.2 Partition Allocation and Source Selection

In the following, we consider the partition allocation problem. More precisely, the question is which partition to physically store at which site. Formally, allocating a partition to a site is a surjective function

$$P \to S \tag{4.62}$$

mapping a partition $p_{R,i} \in P$ to a site $s \in S$.

The choice of an allocation scheme can have a great effect on the performance of a system and result in great gains [HSH07, p. 168; KTH+12]. This is because an allocation scheme allows to potentially skip partitions at query time by only selecting a small number of partitions to process the query at and avoiding having to query all available sites. Considering, for example, an algorithm which allows to select a sub-set of λ partitions,

$$\mathcal{O}(n_S + \frac{\lambda}{n_S}\frac{n}{\lambda}) = \mathcal{O}(n_S + \frac{n}{n_S}) \tag{4.63}$$

elements have to be processed, where n_S denotes the number of sites to contact, rather than $\mathcal{O}(n)$ elements. In more detail, for each site, it has to be determined whether it will be contacted and, for the selected number of partitions λ, the elements per site are processed (assuming a uniform distribution of data and partitions).

In text-retrieval systems, the strategy of creating topically clustered partitions (topical shards) has become the standard way of processing queries against large document collections [KTH+12]. Rather than randomly distributing elements over all partitions, organising the collection in a sophisticated way may allow to only query a sub-set of the whole collection. This can be done while still returning results at the same quality as when querying the whole collection, but at a much lower computational effort for the whole system. A partition selection algorithm is, hence, supposed to maximise the search accuracy by contacting the most relevant partitions, and at the same time minimise the search costs by only searching as few partitions as possible [KTH+12]. This generally results in two questions being which partitions to involve in the query processing (or how to create a ranking of the partitions) and how many partitions to contact ultimately. The first question is generally addressed by a central index. Obviously, centrally storing the full database is not feasible, therefore, only an index of samples (centralised sample index) is stored [SC03]. Particularly in P2P systems, each site compresses as much as possible the data it contributes to the full system in synopses, e.g., using hash sketches (cf. [MBN+06]) or Bloom filters, which can be used to choose the peers that are likely to contribute to the results (cf. [SFK+14; EMH+06]). On the other hand, research on the second question is more limited as argued in [KTH+12].

For multimedia data, there is no obvious partitioning scheme that can be generally applied to the feature data and that allows to prune nodes at query time from the retrieval

without possibly losing result elements (see Section 4.5.1.4). In the multimedia retrieval context, [BFL+10] applies a space-filling curve for creating clusters and maps the clusters to a Chord P2P network; at query time, only the most promising peers are contacted for answering the query at hand. Similarly, [MH03] use a global clustering algorithm and communicate the centroids of each cluster to all peers in a P2P network. Based on the cluster centroid, a query is forwarded to a selected number of peers. However, both approaches do not give any guarantees on finding all elements that would answer a query. The only alternative to this approach is to consider all partitions, i.e., a query has to be processed by all sites and the sub-results of each site have finally to be merged. Similarly, in [Web97; WBS00], the authors shard a VA-File index to multiple sites which are queried in parallel by multiple workers.

Part III

Database Support for Multimedia Retrieval

Simple can be harder than complex: You have to work hard to get your thinking clean to make it simple.

— Steve Jobs

5

Modelling a Multimedia Data Management System

In the following, we describe the ingredients to model and construct a multimedia data management system which we base on the concepts from both multimedia retrieval and database systems. We first introduce an architecture model, describing the architectural aspects of a multimedia data management system. Then, we present a generic data model for a multimedia database and the operations pertinent to the model. We expound on a unified query model for searching in both the multimedia content and the corresponding metadata from a logical and executional perspective. We further consider the distribution and physical storage of the data in a distribution and storage model, respectively.

5.1 Architecture Model

From a data model perspective, our architectural model follows the classical database separation providing an abstraction into a physical and logical model. From a component perspective, in Figure 5.1, we present a component architecture model which follows the classical architecture of a database system as introduced in Section 4.1 as well, however, with a greater focus and a number of adjustments specific for multimedia data management. Our model tightly couples retrieval and database functionality within one system making the functionalities of the corresponding parts first class citizens within our multimedia data management system. The changes made to the architecture model are visible, in particular, in the following components:

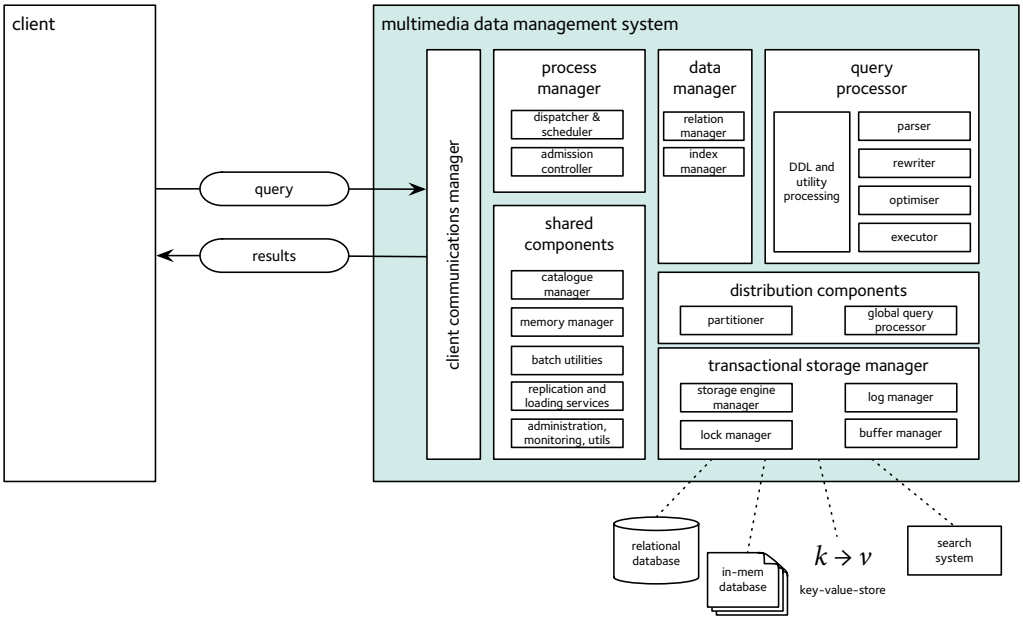


Figure 5.1 Main components of a multimedia database management system (based on Figure 4.2).

Data manager The data manager is responsible for the management of the data on a logical level. While this component is to some extent also present in the previously introduced architecture model, in our model, it is made more explicit. The data manager manages both the schema and the extent of the relations. We have extended the supported data domains to also be able to store dense and sparse vector data as a means to handle unstructured data from which a set of features has been extracted. Moreover, the data manager provides various index structures to index vectorial data and increase the retrieval performance from an efficiency perspective.

Query processor The query processor, responsible for processing the incoming client queries, has been extended to not only support Boolean queries, but also to be able to execute similarity queries. For this, adjustments are necessary to the logical composition of a query and to the query processing pipeline, with a special focus on the optimisability and execution of such queries.

Transactional storage manager The storage manager, managing the data access and manipulation on various storage engines, has been adjusted to provide a wider variety of storage mechanisms to support the variability of the data to manage and store. Instead of only supporting file-based storage mechanisms which read data from a disk, the storage manager gives an interface for accessing any sort of storage mechanism, including also storage engines which have a different underlying data model.

Distribution components Within our model, the distribution of data receives a greater focus than in traditional database architectures. The distribution of queries to multiple sites, the partitioning of data, etc. have been internalised into our model while providing distribution independence meaning that any change to the distribution model will not require changes to the logical data model.

We will give more details on the adaptions made to the available components and newly introduced parts in the remainder of this chapter.

5.2 Data Model

In the following section, we provide a formal data model which is founded on

- the *vector-space retrieval model* [SWY75] for organising and retrieving unstructured information (as introduced in Section 3.2.3),
- the *relational data model* [Cod70] providing the structural formalisms for structured data (see Section 4.2).

Orthogonal to the structuring properties of the relational data model, for unstructured information such as multimedia data, we assume data from the vector-space model which allows to model an (unstructured) multimedia document and structure it based on the given framework. At the intersection of the relational and the vector-space model, we combine the structural aspects of the relational model with the handling of unstructured information as given by the vector-space model. For the definition of our data model, we propose to extend the relational foundations by selected aspects of the vector-space model, in particular the representation of unstructured information as multi-dimensional, real-valued vectors and the computation of a distance measure or ranking based on the comparison of vectorial data points.

5.2.1 Structure of the Data

To be able to handle unstructured information, we consider a feature transformation function, as discussed in Section 3.2.3, i.e.,

$$f : \mathcal{O} \rightarrow \mathbb{R}^{dim} \tag{5.1}$$

translating a multimedia document $o \in \mathcal{O}$ into a real-valued vector $\boldsymbol{d} \in \mathcal{D} \subseteq \mathbb{R}^{dim}$. On the other hand, using the relational data model, structured data (e.g., logical metadata) can be handled and stored within a relation composed of a schema $\textsc{sch}(R)$ and an extent $\textsc{val}(R)$ to a relation R. For being able to handle the vector data representing the multimedia

documents, we extend the relational data model to support real-valued data domains, i.e., the set of domains $\mathbb{D}$ is extended by

$$\mathbb{D} \cup \{\mathbb{R}^{dim}\} \text{ with } dim \in \mathbb{Z}_+ \tag{5.2}$$

We must point to the fact that the relational data model in its original form is based on simple, primitive data domains only. It can be argued that the domain for multi-dimensional data points ($\mathbb{R}^{dim}$) violates this premise and the introduction of a non-first normal form (NF²) algebra might be necessary. However, following [Cod90, p. 6], we consider in our model atomicity of the data with respect to the property that the data cannot be decomposed into smaller pieces by the data management system *except by certain special functions* which can decompose the data. Similar to a string being composed of single characters, we consider the multi-dimensional real-valued vectors being composed of real numbers which are, however, not decomposable. In our case, we consider the distance functions being such special functions, which can access the single parts of the vectorial data points; otherwise, the data is considered atomic. Hence, we argue that the first normal form is not violated.

Particularly in text-retrieval systems, the high-dimensional vectors are only sparsely populated and, hence, only a small number of elements of the vector has values other than zero. For high-dimensional, sparse data points, it might be more appropriate to handle such data differently than densely populated vector data. A sparse data point $\check{\boldsymbol{d}} \in \mathcal{D}$ is generally described as

- $dim := \check{\boldsymbol{d}}_{\text{size}} \in \mathbb{Z}$ denotes the total size of the data point,
- $\check{\boldsymbol{d}}_{\text{values}} \in \mathbb{R}^{\check{dim}}$ storing all non-zero values,
- $\check{\boldsymbol{d}}_{\text{index}} \in \mathbb{Z}^{\check{dim}}$ storing the indexes for the corresponding non-zero values,

where the number of non-zero elements $\check{dim} < dim$. For each value of $\check{\boldsymbol{d}}_{\text{values}}$ the corresponding index in $\check{\boldsymbol{d}}_{\text{index}}$ denotes the position of the value within the full vector. Differently said,

$$\check{\boldsymbol{d}} \in \mathbb{R}^{dim}$$

$$\check{d}_i := \begin{cases} 0 & \text{if } i \notin \check{\boldsymbol{d}}_{\text{index}} \\ \check{d}_{\text{values},i} & \text{if } i \in \check{\boldsymbol{d}}_{\text{index}} \end{cases} \qquad \forall\, i \in \{1, \ldots, dim\} \tag{5.3}$$

with $\check{d}_i$ denoting the position i within the full data point $\check{\boldsymbol{d}}$. The set of data domains equally supports the domain of sparse vectors given as a triplet, i.e.,

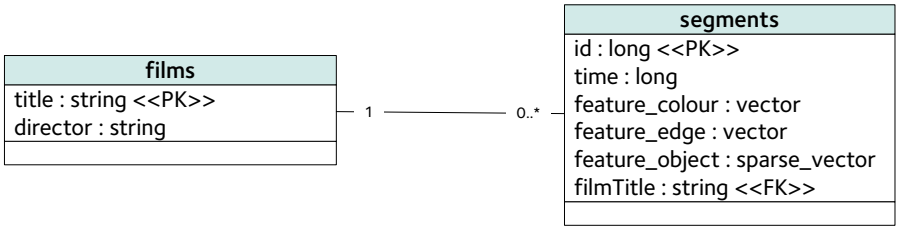
$$\mathbb{D} \cup \{(\mathbb{Z}, \mathbb{R}^{\check{dim}}, \mathbb{Z}^{\check{dim}})\} \text{ with } \check{dim} \in \mathbb{Z}_+ \tag{5.4}$$

As with dense vectors, also with sparse vectors, we do not consider the first normal form violated.

Example 5.1 Structure of the data.

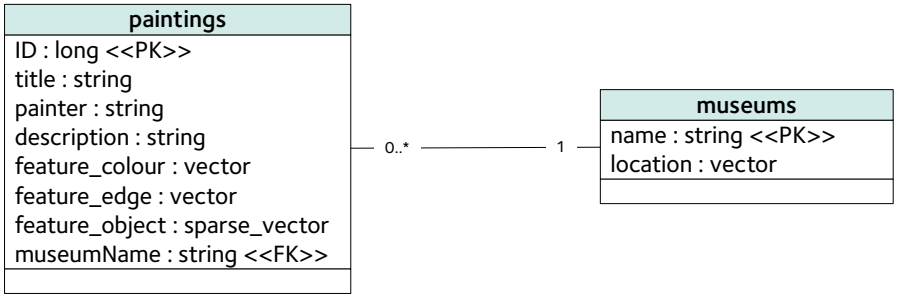
We consider the scenarios introduced in Section 2.5 and model the data in the following logical models.

In the first, film scenario, we store a `films` relation with the title of the film and the director. The `segments` relation, stores for every segment its timing, and a number of features extracted from the segment. The following illustration gives an overview of the logical model; we use `PK` to denote primary keys and `FK` to mark a foreign key.



We have chosen the data type `vector` for both the extracted colour features and edge features. On the other hand, for the object features, given that these are often comparably long features with each element denoting the probability for the existence of an object and with many entries being zero, we have chosen to use the sparse vector data type.

In the second, art scenario, we assume the following two relations: A relation `paintings`, stores the title, the painter and a description together with some content metadata of the painting; the relation `museums` has attributes for the name of the museum and its location (given as a GPS coordinate). Each painting is entered with the information at which museum the painting is exposed at.



As previously, the low-level visual features are stored as vectors, while for the object features we use the `sparse vector` data type. The `location` in the relation `museums` is stored in vectorial form as well.

5.2.2 Operations on the Data

Our data model supports the same operations as given by the conventional relational model; in particular, it provides

- monadic operators, e.g., projection ($\pi_A(\cdot)$) and selection ($\sigma_\varphi(\cdot)$);
- binary operators requiring full schema-compatibility, e.g., union ($\cup$), intersection ($\cap$), difference ($-$) operations;
- binary operators requiring (partial or) no schema-compatibility, e.g., Cartesian product ($\times$) and various forms of join ($\bowtie$);
- manipulation operations resulting in changes to the extent of a relation (e.g., insert, update, delete).

In the following, we introduce a similarity operator as a first-class operator in our model to be able to execute ranking queries.[1] Moreover, we extend the binary operators union and intersection such that they can be employed on relations for which a distance measure has been computed.

5.2.2.1 Similarity Operator

For the definition of a similarity operator, we start by introducing an operator $\tau_{\delta(\cdot,\cdot),a,\boldsymbol{q}}(R)$ which performs a similarity query under a distance $\delta(\cdot,\cdot)$ applied on an attribute a of relation R and compared to a query vector $\boldsymbol{q}$. In the following, we give a generic definition for a similarity operator.

Definition 5.1 Similarity operator (based on [LCI+05]).

We define a similarity operator in relational algebra under

- a distance function ($\delta(\cdot,\cdot)$),
- applied on an attribute a,
- compared to a query vector $\boldsymbol{q}$,

denoted as $\tau_{\delta(\cdot,\cdot),a,\boldsymbol{q}}(R)$, as

$$\begin{aligned} \textsc{sch}(\tau_{\delta(\cdot,\cdot),a,\boldsymbol{q}}(R)) &:= \textsc{sch}(R) \cup \{\bar{\delta}\} \\ \textsc{val}(\tau_{\delta(\cdot,\cdot),a,\boldsymbol{q}}(R)) &:= \{t \in \mathcal{R} \mid \mathcal{R} = \textsc{val}(R) \ \wedge \ t.\bar{\delta} = \delta(\boldsymbol{q}, t.a)\} \end{aligned} \tag{5.5}$$

[1] While in here, for simplicity reasons, we assume the use of a distance measure, similar definitions could also be established for similarity measures. Furthermore, in the following, we use the terms *similarity-based relation*, *similarity operator* and *similarity predicate* despite using distance measures rather than similarities. The chosen nomenclature subsumes both distance and similarity functions.

where $\bar{\delta}$ denotes an implicit distance attribute of a relation which introduces an ordering on a relation R. We refer to a relation which can be ordered based on a distance attribute being a *similarity-based relation* $\tilde{R}$, defined as

$$\text{SCH}(\tilde{R}) = \text{SCH}(R) \cup \{\bar{\delta}\} \tag{5.6}$$

By the similarity operator an ordering relationship $\prec_{\bar{\delta}}$ is defined on the tuples, i.e.,

$$\forall t_1, t_2 \in \tilde{R} : t_1 \prec_{\bar{\delta}} t_2 \text{ iff } t_1.\bar{\delta} < t_2.\bar{\delta} \tag{5.7}$$

In our model, the computation of a distance measure creates an *implicit attribute* on the relation, denoted here as $\bar{\delta}$ (more precisely, $t.\bar{\delta}$), which introduces an ordering relationship and can be used for sorting the relation by distance. We denote the distance attribute being implicit, as the attribute is only partially available within the relation. For example, within binary operations (e.g., union, intersection, except operations, the Cartesian product and the join operation), the distance attribute will be ignored and dropped in the results, i.e.,

$$\tilde{R}_1 \otimes \tilde{R}_2 = R_1 \otimes R_2 \tag{5.8}$$

where we use $\otimes := \{\cup, \cap, -\}$ to denote one of the classical set operators. In Section 5.2.2.5, we will extend the union and intersection operations to consider the distance information as well.

The similarity operation satisfies the *idempotence* property, i.e.,

$$\tau_{\delta(\cdot,\cdot),a,\boldsymbol{q}}(\tau_{\delta(\cdot,\cdot),a,\boldsymbol{q}}(R)) = \tau_{\delta(\cdot,\cdot),a,\boldsymbol{q}}(R) \tag{5.9}$$

For the similarity operator, the *commutativity* of the operation with other operators holds, i.e.,

$$\sigma_{\varphi}(\tau_{\delta(\cdot,\cdot),a,\boldsymbol{q}}(R)) = \tau_{\delta(\cdot,\cdot),a,\boldsymbol{q}}(\sigma_{\varphi}(R)) \tag{5.10}$$

Similarly, *associativity* and *distributivity* are given.

While the definitions provided so far are applicable, a similarity query will almost never be required to return all ranked data objects of an entire relation [LJW+06]. In the following, we therefore introduce two variations of the similarity operator which are more practical, as they limit the number of results returned to the user. We extend the similarity operator by a limiting predicate for executing both ε nearest neighbour and κ nearest neighbour searches.

5.2.2.2 Similarity Operator with *εNN* Predicate

ε nearest neighbour queries are queries which limit the number of results retrieved from a similarity search by a maximum distance ε. We define an *εNN* search in relational algebra, denoted as $\tau^{\varepsilon NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\varepsilon}(R)$, as follows.

Definition 5.2 Similarity operator with *εNN* predicate.

For a maximum distance ε, a given distance function $\delta(\cdot,\cdot)$ applied on an attribute a and compared to a query vector $\boldsymbol{q}$, the *εNN* operation $\tau^{\varepsilon NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\varepsilon}(R)$ is given as

$$\begin{aligned} \text{SCH}(\tau^{\varepsilon NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\varepsilon}(R)) &:= \text{SCH}(R) \cup \{\bar{\delta}\} \\ \text{VAL}(\tau^{\varepsilon NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\varepsilon}(R)) &:= \{t \mid \forall t \in \text{VAL}(R) \ \wedge \ t.\bar{\delta} = \delta(\boldsymbol{q}, t.a) \ \wedge \ t.\bar{\delta} < \varepsilon\} \end{aligned} \tag{5.11}$$

For the *εNN* operator, the same properties as noted without the limiting factor and the properties of the selection operator $\sigma_\varphi(\cdot)$ hold. Hence, the composition of multiple *εNN* operations is commutative and associative. Moreover, in combination with the other operators of the relational algebra, distributivity holds. The operator satisfies the idempotence property.

Example 5.2 Similarity operator with *εNN* predicate.

Bob is looking for all museums which are within ε = 1 km walking distance from his current location. Formally, he employs a similarity search for which the ranking is given through the distance computation from his current location to each museum stored in the database. We make use of a special distance function δ_{walking} which compares the attribute `location` (of the relation `museums`) with the current location and returns the walking distance between both locations. Formally, such a query can be expressed as

$$\tau^{\varepsilon NN}_{\delta_{\text{walking}(\cdot,\cdot)},\texttt{location},\text{📍},\varepsilon=1\text{ km}}(\texttt{museums})$$

where we use 📍 to denote Bob's current location in vectorial form.

5.2.2.3 Similarity Operator with *κNN* Predicate

We now introduce a *κNN* operator which retrieves the "best" κ elements according to a similarity criterion given by a distance function $\delta(\cdot,\cdot)$. For *κNN* searches, in the following, we introduce a nearest neighbour operation $\tau^{\kappa NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa}(R)$.

Definition 5.3 Similarity operator with *κNN* predicate.

We define a *κNN* search in relational algebra, for a given maximum number of results to return (κ), a given distance function ($\delta(\cdot,\cdot)$) applied on an attribute a and compared to a query vector $\boldsymbol{q}$, denoted as $\tau_{\kappa,\delta(\cdot,\cdot),a,\boldsymbol{q}}(R)$, as

$$\begin{aligned}
\textsc{sch}(\tau^{\kappa NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa}(R)) &:= \textsc{sch}(R) \cup \{\bar{\delta}\} \\
\textsc{val}(\tau^{\kappa NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa}(R)) &:= \{t \mid \forall t \in \mathcal{R} \subseteq \textsc{val}(R) \ \wedge\ |\mathcal{R}| = \kappa \ \wedge\ t.\bar{\delta} = \delta(\boldsymbol{q}, t.a) \wedge \\
&\qquad \nexists t' \in (\textsc{val}(R) - \mathcal{R}) : \delta(\boldsymbol{q}, t'.a) < t.\bar{\delta}\}
\end{aligned} \tag{5.12}$$

More formally, a *κNN* search is defined as (based on [FST+11])

$$\tau^{\kappa NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa}(R) := \{t_1, t_2, \ldots, t_\kappa\} \tag{5.13}$$

where

$$\begin{aligned}
t_1 &:= \{t \mid \forall t \in \textsc{val}(R) \wedge t.\bar{\delta} = \delta(\boldsymbol{q}, t.a) \ \wedge\ \nexists t' \in \textsc{val}(R) : \delta(\boldsymbol{q}, t'.a) < \delta(\boldsymbol{q}, t.a) \wedge t' \neq t\} \\
t_2 &:= \{t \mid \forall t \in \textsc{val}(R) - \{t_1\} \wedge t.\bar{\delta} = \delta(\boldsymbol{q}, t.a) \ \wedge\ \nexists t' \in \textsc{val}(R) - \{t_1\} : \\
&\qquad \delta(\boldsymbol{q}, t'.a) < \delta(\boldsymbol{q}, t.a) \wedge t' \neq t\} \\
&\vdots \\
t_\kappa &:= \{t \mid \forall t \in \textsc{val}(R) - \{t_1, \ldots, t_{\kappa-1}\} \wedge t.\bar{\delta} = \delta(\boldsymbol{q}, t.a) \ \wedge\ \nexists t' \in \textsc{val}(R) - \{t_1, \ldots, t_{\kappa-1}\} : \\
&\qquad \delta(\boldsymbol{q}, t'.a) < \delta(\boldsymbol{q}, t.a) \wedge t' \neq t\}
\end{aligned}$$

Note that in the case of ties, any set of κ elements which satisfies the conditions is accepted.

The *κNN* operation, $\tau^{\kappa NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa}(R)$, satisfies the properties discussed in the following: The *conjunction* of *κNN* operations can be rewritten as

$$\tau^{\kappa NN}_{(\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa_1)\wedge(\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa_2)}(R) = \tau^{\kappa NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\min(\kappa_1,\kappa_2)}(R) \tag{5.14}$$

Similarly, a *disjunction* of *κNN* operations can be rewritten as

$$\tau^{\kappa NN}_{(\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa_1)\vee(\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa_2)}(R) = \tau^{\kappa NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\max(\kappa_1,\kappa_2)}(R) \tag{5.15}$$

The operation satisfies the *idempotence property*. Hence, repeating the operation multiple times has exactly the same result as applying it once, i.e,

$$\tau^{\kappa NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa}(\tau^{\kappa NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa}(R)) = \tau^{\kappa NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa}(R) \tag{5.16}$$

The *commutativity* of the operation (e.g., with a selection operation) is not given. Instead, given a selection operation $\sigma_{\varphi}(\cdot)$ with a predicate φ,

$$\sigma_{\varphi}(\tau_{\kappa,\delta(\cdot,\cdot),a,\boldsymbol{q}}(R)) \subseteq \tau_{\kappa,\delta(\cdot,\cdot),a,\boldsymbol{q}}(\sigma_{\varphi}(R)) \tag{5.17}$$

i.e., a selection applied on a similarity search will possibly return fewer elements than vice-versa. Similarly, the operation is non-associative.

Example 5.3 Similarity operator with *κNN* predicate.

Bob is looking for the $\kappa = 10$ closest museums from his current location. As in Example 5.2, we use again the special distance function δ_{walking} returning the walking distance between two locations. Formally, such a query can be expressed as

$$\tau^{\kappa NN}_{\delta_{\text{walking}},\texttt{location},\texttt{📍},\kappa=10}(\texttt{museums}) \tag{5.18}$$

5.2.2.4 Combination of *εNN* and *κNN* Predicates in the Similarity Operator

This section presents similarity predicates which combine both *εNN* and *κNN* limiting predicates for the same query. More precisely, the distance measure, the attribute the measure is applied on, and the query vector are the same for both operations. We will discuss more complex examples which combine varying similarity queries with different predicates in Section 5.3.1.

Conjunction of *εNN* and *κNN* queries In the following, we consider the conjunction of *εNN* and *κNN* queries, i.e.,

$$\tau^{\kappa NN \wedge \varepsilon NN}_{(\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa_1)\wedge(\delta(\cdot,\cdot),a,\boldsymbol{q},\varepsilon_2)}(R) \tag{5.19}$$

In Figure 5.2, we visualise the semantics of the conjunction of *εNN* and *κNN* queries. The query point for all queries in the visualisation is the same. The continuous line represents the *εNN* predicate, while the dotted line shows the answer of the κ nearest neighbours predicate. The highlighted part corresponds to the final answer set of the complex query. In the first case, the answer is restricted by the *κNN* condition, while in the second case, it is restricted by the *εNN* condition. The last visualisation depicts the case where the ε-radius is equal to the radius of the κ-th nearest neighbour.

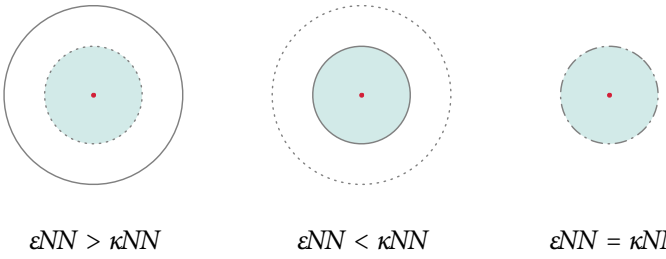


Figure 5.2 Conjunction of *εNN* and *κNN* queries: The continuous line represents the *εNN* predicate, the dotted line represents the *κNN* predicate (based on [AVT+04]).

The conjunction of *εNN* and *κNN* queries results in an intersection of the results, i.e.,

$$\tau^{\kappa NN \wedge \varepsilon NN}_{(\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa_1)\wedge(\delta(\cdot,\cdot),a,\boldsymbol{q},\varepsilon_2)}(R) = \tau_{\delta(\cdot,\cdot),a,\boldsymbol{q}}(\tau^{\kappa NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa_1}(R) \cap \tau^{\varepsilon NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\varepsilon_2}(R)) \tag{5.20}$$

Note that, as per our definition, the distance attribute is an implicit attribute which will be dropped in the conventional binary operations, e.g., union, intersection, etc. Hence, in Equation 5.20, an additional distance computing step is added.

Example 5.4 Conjunction of *εNN* and *κNN* queries.

Bob is looking for the names and the distance of the $\kappa_1 = 10$ closest museums which are within $\varepsilon_2 = 1$ km from his current location. We can express such a query as

$$\pi_{\texttt{name},\texttt{distance}}(\tau^{\kappa NN \wedge \varepsilon NN}_{(\delta_{\text{walking}},\texttt{location},\text{📍},\kappa_1=10)\wedge(\delta_{\text{walking}},\texttt{location},\text{📍},\varepsilon_2=1\text{ km})}(\texttt{museums}))$$

where 📍 stands for Bob's current location in vectorial form and we use a special distance measure δ_{walking} measuring the walking distance. Bob can retrieve the desired results by executing both queries separately and intersecting the results, i.e.,

$$\begin{aligned}&\pi_{\texttt{name},\texttt{distance}}(\tau^{\kappa NN \wedge \varepsilon NN}_{(\delta_{\text{walking}},\texttt{location},\text{📍},\kappa_1=10)\wedge(\delta_{\text{walking}},\texttt{location},\text{📍},\varepsilon_2=1\text{ km})}(\texttt{museums}))\\ &= \pi_{\texttt{name},\texttt{distance}}(\tau_{\delta_{\text{walking}},\texttt{location},\text{📍}}(\\ &\quad \tau^{\kappa NN}_{\delta_{\text{walking}},\texttt{location},\text{📍},\kappa_1=10}(\texttt{museums}) \cap \tau^{\varepsilon NN}_{\delta_{\text{walking}},\texttt{location},\text{📍},\varepsilon_2=1\text{ km}}(\texttt{museums})))\end{aligned}$$

Disjunction of *εNN* and *κNN* queries In the following, we consider the disjunction of *εNN* and *κNN* queries. Similar to above, we denote such a query as

$$\tau^{\kappa NN \vee \varepsilon NN}_{(\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa_1)\vee(\delta(\cdot,\cdot),a,\boldsymbol{q},\varepsilon_2)}(R) \tag{5.21}$$

In Figure 5.3, we visualise the semantics of the disjunction of *εNN* and *κNN* queries. As previously, the query point for both queries is the same. The continuous line represents again the *εNN* predicate, while the dotted line shows the answer of the *κ* nearest neighbours predicate; the coloured part corresponds to the final answer set of the complex query. The

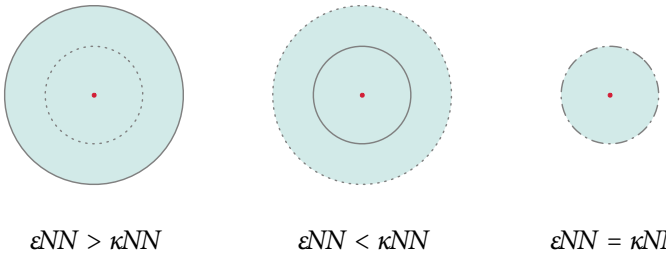


Figure 5.3 Disjunction of *εNN* and *κNN* queries: The continuous line represents the *εNN* predicate, the dotted line represents the *κNN* predicate (based on [AVT+04]).

first case shows the situation where the *εNN* query selects more elements than the *κNN* query predicate. In the second case, the *κNN* query yields more elements than the *εNN* query. In the last case, both the radius of the *εNN* query and the radius of the κ-th nearest neighbour are equal.

The disjunction of *εNN* and *κNN* queries results in a union of the results, i.e.,

$$\tau^{\kappa NN \vee \varepsilon NN}_{(\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa_1)\vee(\delta(\cdot,\cdot),a,\boldsymbol{q},\varepsilon_2)}(R) = \tau_{\delta(\cdot,\cdot),a,\boldsymbol{q}}(\tau^{\kappa NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\kappa_1}(R) \cup \tau^{\varepsilon NN}_{\delta(\cdot,\cdot),a,\boldsymbol{q},\varepsilon_2}(R)) \tag{5.22}$$

Again, the distance attribute is an implicit attribute which will be dropped in the conventional binary operations, e.g., union, intersection, etc. Hence, in Equation 5.22, an additional distance computing step is added.

Example 5.5 Disjunction of *εNN* and *κNN* queries.

Bob is looking for the names and distances of all museums within $\varepsilon_2 = 1$ km from his current location, but wants to retrieve at least $\kappa_1 = 10$ museums. The following expression models the query

$$\begin{aligned}
&\pi_{\texttt{name},\texttt{distance}}(\tau^{\kappa NN \wedge \varepsilon NN}_{(\delta_{\text{walking}},\texttt{location},\text{📍},\kappa_1=10)\vee(\delta_{\text{walking}},\texttt{location},\text{📍},\varepsilon_2=1\text{ km})}(\texttt{museums})) \\
&= \pi_{\texttt{name},\texttt{distance}}(\tau_{\delta_{\text{walking}},\texttt{location},\text{📍}}(\\
&\quad \tau^{\kappa NN}_{\delta_{\text{walking}},\texttt{location},\text{📍},\kappa_1=10}(\texttt{museums}) \cup \tau^{\varepsilon NN}_{\delta_{\text{walking}},\texttt{location},\text{📍},\varepsilon_2=1\text{ km}}(\texttt{museums})))
\end{aligned}$$

where 📍 denotes the location of Bob in vectorial form. We use again a special distance measure δ_{walking} to compute the walking distance.

5.2.2.5 Set Operations on Similarity-based Relations

Performing set operations on a relation for which a distance has been computed is only partially useful. The distance attribute, which is an implicit attribute, should, in general, not be considered for the conventional binary operations, as shown already in Equation 5.8,

$$\tilde{R}_1 \otimes \tilde{R}_2 = R_1 \otimes R_2 \tag{5.23}$$

where $\tilde{R}$ denotes a similarity-based relation and we use $\otimes := \{\cup, \cap, -\}$ to denote one of the classical set operators.

For practical use, the distance value should be carried over the operation and adjusted depending on whether the tuple is contained in both sets and on the distance measure assigned to the tuple in each of the operands. In the following, we introduce *similarity-based set operations*. More precisely, we introduce a similarity-based union operation under a function $\dot{\xi}_{\cup}$, $\tilde{\cup}_{\dot{\xi}_{\cup}}$ (for simplicity, in the following, we write $\tilde{\cup}_{\dot{\xi}}$), and a similarity-based intersect operation under a function $\dot{\xi}_{\cap}$, $\tilde{\cap}_{\dot{\xi}_{\cap}}$ (for simplicity, in the following, we write $\tilde{\cap}_{\dot{\xi}}$). We consider the function $\dot{\xi}$ being a distance combining function, i.e.,

$$\dot{\xi} : \mathbb{R}_0^+ \times \mathbb{R}_0^+ \to \mathbb{R}_0^+ \tag{5.24}$$

For two conventional relations, the result of the similarity-based operations will be equal to the conventional operations, i.e.,

$$R_1 \ \tilde{\otimes}_{\dot{\xi}} \ R_2 = R_1 \ \otimes \ R_2 \tag{5.25}$$

where we use $\otimes := \{\cup, \cap\}$ and $\tilde{\otimes}_{\dot{\xi}}$ denotes the same operations with the similarity-based extensions. On the other hand, for a similarity-based operation between a conventional relation and a similarity-based relation, we note

$$\begin{aligned} \textsc{sch}(\tilde{R}_1 \ \tilde{\otimes}_{\dot{\xi}} \ R_2) &:= \textsc{sch}(R_1) \cup \{\bar{\delta}\} = \textsc{sch}(\tilde{R}_1) \\ \textsc{val}(\tilde{R}_1 \ \tilde{\otimes}_{\dot{\xi}} \ R_2) &:= \{t \mid \forall t \in (\tilde{R}_1 \bowtie_{\textsc{sch}(R_1)} R_2) \wedge t.\bar{\delta} = \dot{\xi}(t.\bar{\delta}_1, \max_{\bar{\delta}}(\tilde{R}_1))\} \end{aligned} \tag{5.26}$$

where $\tilde{\otimes}_{\dot{\xi}}$ denotes a similarity-based binary operator applied between two relations using a function $\dot{\xi}$; we use $t.\bar{\delta}_1$ to mean the distance measure coming from relation $\tilde{R}_1$. In essence, this denotes a join operation on all attributes except the distance. For the missing distance values of R_2, the maximum distance value of $\tilde{R}_1$ is applied.

Similarly, for two relations of which both have a distance information

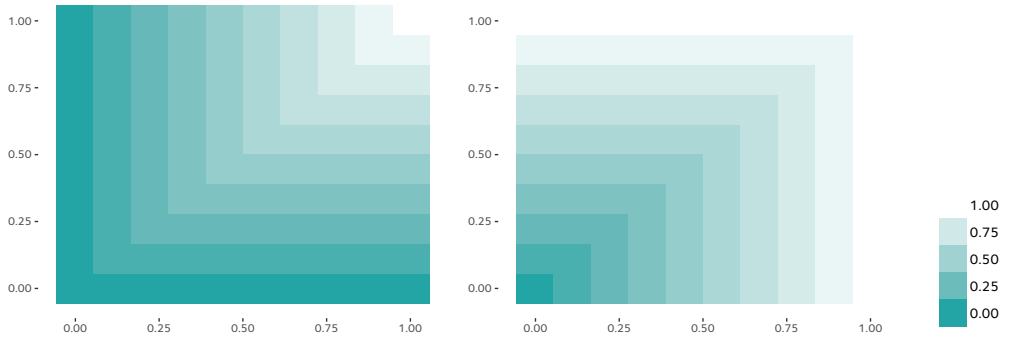
$$\begin{aligned} \textsc{sch}(\tilde{R}_1 \ \tilde{\otimes}_{\dot{\xi}} \ \tilde{R}_2) &:= \textsc{sch}(R_1) \cup \{\bar{\delta}\} = \textsc{sch}(\tilde{R}_1) \\ \textsc{val}(\tilde{R}_1 \ \tilde{\otimes}_{\dot{\xi}} \ \tilde{R}_2) &:= \{t \mid \forall t \in (\tilde{R}_1 \bowtie_{\textsc{sch}(R_1)} \tilde{R}_2) \wedge t.\bar{\delta} = \dot{\xi}(t.\bar{\delta}_1, t.\bar{\delta}_2)\} \end{aligned}$$

where we use again $t.\bar{\delta}_1$ and $t.\bar{\delta}_2$ to mean the distance measure coming from relation $\tilde{R}_1$ and $\tilde{R}_2$, respectively.

We have presented combining functions in Section 3.2.2 and Section 4.2.5.[2] Based on these functions, we define for similarity-based intersect operations $\tilde{\cap}_{\dot{\xi}}$ the distance combining function

$$\dot{\xi}_{\cap}(\bar{\delta}_1, \bar{\delta}_2) = \max(\bar{\delta}_1, \bar{\delta}_2) \tag{5.27}$$

[2] Note that in Section 3.2.2 the definition uses distances, while in Section 4.2.5 the definition is based on similarities.



(a) Similarity-based union. (b) Similarity-based intersect.

Figure 5.4 Standard distance combining functions: The visualisation plots the results of a similarity-based aggregation operation for two distance values in $[0,1]$ (based on [ORC+98]).

Likewise, for similarity-based union operations $\tilde{\cup}_{\dot{\xi}}$ the distance combining function is defined as

$$\dot{\xi}_{\cup}(\bar{\delta}_1, \bar{\delta}_2) = \min(\bar{\delta}_1, \bar{\delta}_2) \tag{5.28}$$

Further distance combining functions may be applied for combining distance values with equally sensible results (consider, for instance, the class of t-norms [KY95, pp. 82] and t-conorms [KY95, pp. 74]). In Figure 5.4, we visualise the results of the similarity-based union and intersect operations using the functions presented in Equation 5.27 and Equation 5.28.

5.2.2.6 Similarity Predicate

For reasons of readability, we summarise both similarity predicates *εNN* and *κNN* queries, and define $\tau_{\delta(\cdot,\cdot)a,\boldsymbol{q},\vartheta}(\cdot)$ with $\delta(\cdot,\cdot)$ denoting a distance function which is executed on $\boldsymbol{q}$ and a, with ϑ being a limiting predicate limiting the number of retrieved elements in the similarity search, i.e.,

$$\tau_{\delta(\cdot,\cdot),a,\boldsymbol{q},\vartheta}(R) := \{t \mid \forall t \in \mathcal{R} \subseteq \text{VAL}(R), t.\delta = \delta(\boldsymbol{q}, t.a) \wedge t \vDash \vartheta\} \tag{5.29}$$

where $t \vDash \vartheta$ denotes the satisfaction of ϑ by t (i.e., $\vartheta(t)$ yields true). Summarising the similarity operations introduced so far, Definition 5.4 provides the definition of a similarity predicate.

Definition 5.4 Similarity predicate.

A similarity operation returns a list of results ordered according to the similarity predicate ψ. We define the similarity predicate ψ as

$$\psi := \langle \delta(\cdot,\cdot), a, \boldsymbol{q}, \vartheta \rangle \tag{5.30}$$

where

$\delta(\cdot,\cdot)$ denotes the distance function to use for computing a distance value,

a represents the attribute involved in the computation of the distance,

$\boldsymbol{q}$ is the query vector used in the distance function,

ϑ stands for a limiting predicate which limits the cardinality of the result set by a user preference given, for instance, as a maximum distance (εNN search) or as a maximum cardinality (κNN search) of the results.

In the following, we note $\tau_{\psi}(\cdot)$ and subsume the parts of the similarity predicate in the variable ψ.

5.2.3 Design Decisions in the Data Model

Feature transformation In our data model, we refrain from making the feature transformation function part of the extended relational model (cf. [OÖL+01; GMR+09; PMJ02]). Instead, our model considers the feature transformation being part of the user application rather than of the data management system system. With this, we separate the pragmatic aspects of an application which are included in a retrieval engine and the data management aspects (as proposed by [Fuh12], see also Section 1.2 and Section 2.3).

Multimedia object As a consequence, our model does not particularly address the multimedia object and its storage (cf. [WNM+95; GMR+09; Gia13]), as this would also require ensuring consistency guarantees in regards of changes in the multimedia object and the extracted features (consider, for example, the notion of generalised icon introduced in Section 4.2.5). Hence, our model only considers the multimedia document in its already processed, transformed and extracted form, i.e., as a multi-dimensional, real-valued data point. For storing this data, we have extended the available data domains to also support real-valued vectors. We refer to [Spr14, pp. 88] for a brief discussion on the advantages and disadvantages of storing a multimedia object within a digital library.

Similarity join While it would generally be possible to extend our data model to also support similarity joins (as presented in Section 4.2.5), in this thesis, we have refrained from defining this operation. The use of the similarity join is comparably limited in multimedia retrieval applications and, hence, the definition is left for future work.

5.3 Query Model

Following the definition of the data model, in this section, we provide a query model for searching in both structured, relational data and unstructured data which need similarity operations for sensible retrieval. We base our query model on the requirements for a multimedia query language as studied by [BBZ12b; BBZ12a]. Amongst others, the authors identify the following requirements for a multimedia query language:

- support for various query types, including attribute-based queries, κNN queries, range queries, etc., by means of single- and multi-query objects;
- support for multiple information sources and complex queries that combine multiple query modes, for instance, similarity-based retrieval in the content metadata together with Boolean retrieval in the logical metadata;
- support for user preferences, e.g., with regard to the execution of a query, the retrieval quality, etc.

Within the framework of these criteria, we formalise in the following our query model allowing to scaffold a query. For this, we distinguish the parts pertaining to the

- logical query model (Section 5.3.1),
- executional query model (Section 5.3.2).

We close this section by a set of model queries which apply our query model and present the expressibility given by our model.

5.3.1 Logical Query Model

By making use of the operators introduced so far, we have an algebra that is powerful enough to employ similarity queries. We define a basic query from the logical perspective in Definition 5.5.

Definition 5.5 Logical query model $\check{\mathrm{Q}}$.

We define a basic query $\check{\mathrm{Q}}$ as a quadruple

$$\check{\mathrm{Q}} := \langle A, R, \varphi, \psi \rangle$$

where

A denotes a set of projection attributes ($A \subseteq \text{SCH}(R)$) to use for performing a projection π_A on a relation,

R stands for a relation or a – more complex – relational expression on which the (Boolean and similarity) predicates are evaluated,

φ indicates a Boolean, filtering predicate whose result is given by the evaluation of the predicate on the extent; more precisely, φ is applied in a selection operation ($\sigma_\varphi(\cdot)$), with $\forall t \in \text{VAL}(R) : t \vDash \varphi$,

ψ signifies a similarity predicate that introduces an ordering in the resulting relation based on the evaluation of a distance function, i.e., $\tau_\psi(\cdot)$.

Our logical query model follows largely the projection/selection structure known from the formulation of queries traditional databases.

The *projection clause* is given by the projection attributes A. The data underlying the query is represented in the *source relation* which may also be given as complex expression combining multiple relations. Given that evaluating the query model results in a relation, logical queries can be combined to achieve more complex semantics. The Boolean predicate φ defines the *selection clause*. Following our earlier approach in [Gia13], we separate the clause for performing a similarity retrieval from the Boolean predicate. The *similarity predicate* ψ has been specified already in Definition 5.4.

The query model allows the specification of a query in a declarative way. As a consequence, we are required to define the execution order of its parts. In the following, we consider the semantics of our query model and will particularly focus on mixed queries, that is to say on queries combining both filtering and similarity predicates. While the filtering predicate selects from the relation the set of tuples which strictly satisfy the condition specified by the propositional formula, the similarity predicate allows to compute a distance score for each tuple in the result set and re-order the results. However, as noted previously, a similarity query will almost never be required to return all ranked data objects of the entire collection. Instead, for both efficiency and usability reasons, the similarity search will reduce the number of elements retrieved by a maximum distance (εNN query) or a maximum number of elements (κNN query). Given that our model formulation allows to specify both filtering and similarity predicates at the same time, the ordering of the operations has to be defined, in particular, since the results differ depending on the chosen semantics, as we will show in the remainder of this section. In the following, we distinguish three semantics when combining a filtering and a similarity predicate in the context of εNN and κNN queries (following [CDM+15, p. 317]).

Combination of an *εNN* query and a Boolean, filtering query The combination of an *εNN* query together with a filtering query corresponds to combining two Boolean queries, i.e., one using the filtering predicate φ, the other one with $\forall t \in R : t.\bar{\delta} \leq \varepsilon$. The results will, hence, include the objects that satisfy both conditions. Formally,

$$\sigma_{\varphi}(\tau_{\psi}(R)) = \tau_{\psi}(\sigma_{\varphi}(R)) \tag{5.31}$$

for φ making use of a limiting predicate based on the maximum distance ε.

Example 5.6 Combination of an *εNN* query and a Boolean, filtering query.

Bob is looking for the names of all museums which are at most $\varepsilon = 1$ km walking distance from his current location and which display paintings by van Gogh. In this example, consider again the `location` information stored in the relation `museums`. The maximum distance ε is then set to 1 km for a distance function computing the walking distance between the users location and the museum. We can express such a query, where we use 📍 to denote Bob's current location, as follows.

Logical query parameters	
A	`name`
R	`museums` ⋈ `paintings`
φ	`painter` = "Vincent van Gogh"
ψ	δ_{walking}(`location`, 📍) ≤ 1km

In relational algebra, such a query may be expressed as

$$\pi_{\text{name}}(\tau^{\varepsilon NN}_{\delta_{\text{walking}},\texttt{location},📍,\varepsilon=1\text{ km}}(\sigma_{\texttt{painter}\text{ ="Vincent van Gogh"}}(\texttt{museums} \bowtie \texttt{paintings})))$$

The order of executing the operation is not relevant as the results will always be equal, i.e.,

$$\tau^{\varepsilon NN}_{\delta_{\text{walking}},\texttt{location},📍,\varepsilon=1\text{ km}}(\sigma_{\texttt{painter}\text{ ="Vincent van Gogh"}}(\texttt{museums} \bowtie \texttt{paintings})) =$$
$$\sigma_{\texttt{painter}\text{ ="Vincent van Gogh"}}(\tau^{\varepsilon NN}_{\delta_{\text{walking}},\texttt{location},📍,\varepsilon=1\text{ km}}(\texttt{museums} \bowtie \texttt{paintings}))$$

Combination of a pre-filter *κNN* query and a Boolean, filtering query For a *κNN* query, one possible combination considers the results of the similarity query as an *a-priori* condition (pre-filter *κNN*). In this setting, the *κNN* search is executed first and the results are filtered by the Boolean predicate. This may yield result sets with less than κ result elements, and may even produce empty results if the κ best elements do not satisfy the filtering conditions. Possibly instead of only retrieving κ elements in the first place, $\kappa' > \kappa$ can be retrieved, deriving κ' based on the selectivity of the *κNN* search [CGM04] to ensure

that still κ elements are retrieved after applying the Boolean filter. Formally,

$$\sigma_{\varphi}(\tau_{\psi}(R)) \tag{5.32}$$

i.e., the similarity query is executed before the selection operation has been applied.

Combination of a post-filter *κNN* query and a Boolean, filtering query Considering the results of the similarity query as an *a-posteriori* condition, the Boolean predicate is executed first and the κ most similar elements out of the elements that satisfy the filtering condition are returned to the user (post-filter *κNN*) ordered by distance score. The execution of such a query requires evaluating first the filtering query; the resulting relation is then queried using the similarity predicate. Formally,

$$\tau_{\psi}(\sigma_{\varphi}(R)) \tag{5.33}$$

i.e., the similarity query is executed after the selection operation has been applied.

Example 5.7 Conjunction of a *κNN* query and a filtering query.

Bob has taken a photograph of a poster displaying a van Gogh painting and is looking for the $\kappa = 10$ paintings by van Gogh similar (by colour) to the photograph taken (📷). We may express such a query as

Logical query parameters	
A	`title`
R	`paintings`
φ	`painter` = "Vincent van Gogh"
ψ	δ(`feature_colour`, 📷) with κ = 10

In relational algebra, using the pre-filter semantics, the query is expressed as follows.

$$\sigma_{\texttt{painter} = \text{“Vincent van Gogh”}}(\tau^{\kappa NN}_{\delta,\texttt{feature\_colour},📷,\kappa=10}(\texttt{paintings}))$$

On the other hand, using a post-filtering semantics, such a query may be formulated as

$$\tau^{\kappa NN}_{\delta,\texttt{feature\_colour},📷,\kappa=10}(\sigma_{\texttt{painter} = \text{“Vincent van Gogh”}}(\texttt{paintings}))$$

Note that exchanging the $\tau_{\psi}(\cdot)$ and the $\sigma_{\varphi}(\cdot)$ operation will not yield the same results. In the first case, if the κ most similar elements retrieved in the similarity search are not by van Gogh, the result set will be empty. Instead, applying the post-filter semantics will first filter for all elements for which the Boolean predicate (`painter` ="Vincent van Gogh") is true and only then apply the similarity predicate.

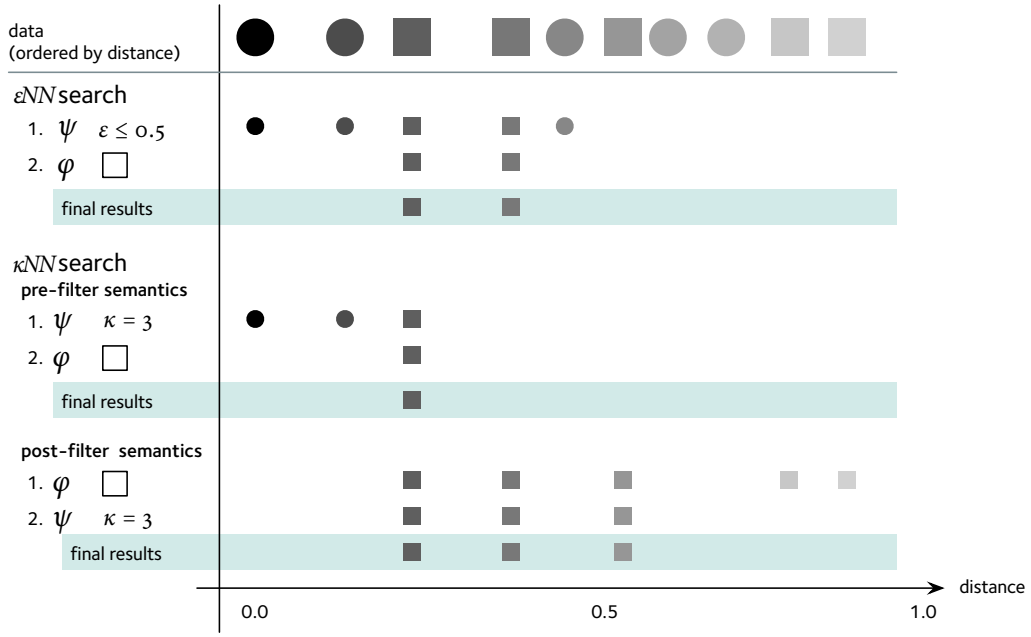


Figure 5.5 Visualisation of the semantics of the combination of a similarity query and a filtering query: The visualisation shows that while for *εNN* queries the order of execution is independent, for *κNN*, a pre-filter and a post-filter semantics has to be distinguished.

In Figure 5.5, we summarise the various combination possibilities: We have simplified the example by considering a collection composed of squares and circles. The first row considers the full data ordered by distance without performing any limitation by distance or cardinality and without applying the Boolean predicate. We first display the result of applying an *εNN* query (here with $\varepsilon \leq 0.5$), and a filtering based on the Boolean predicate denoting that the shape is a square. Note that for the *εNN* query, the order of the operation can be interchanged yielding the same results. For the *κNN* search (here with $\kappa = 3$), we visualise both the pre-filter and the post-filter semantics. As shown in Figure 5.5, the final results of the query (in the highlighted box) depend on the order of the operations.

While for the *εNN* search, as mentioned previously, the results of the execution are independent of the ordering of the operations and, hence, the execution semantics is unambiguous, in the case of the *κNN* search the ordering of the operations is meaningful [LJW+06; CGM04]. In the context of multimedia retrieval, the a-posteriori semantics might be the more sensible one: A user may look for the most similar multimedia documents that satisfy the filtering predicates, but sorted by similarity. For these reasons, our model gives preference to the a-posteriori, post-filtering semantics of the search, as it is the more sensible one in the retrieval context. The precedence of the operations is defined as follows: First the relational expression R is evaluated, the Boolean predicate is applied (φ), a ranking is introduced (ψ) and a projection on the results is executed (A), i.e.,

$$\check{\mathrm{Q}} \mapsto \pi_A(\tau_\psi(\sigma_\varphi(R))) \tag{5.34}$$

is the defined order of execution.[3]

Example 5.8 Conjunction of a post-filter κNN query and an εNN query.

Bob is looking for the names of at most $\kappa = 10$ museums exhibiting paintings similar to the one he has taken a photograph of and which are within $\varepsilon = 1$ km from his current location (📍). Using the post-filter semantics, we can express such a query in relational algebra as

$$\pi_{\texttt{name}}\big(\tau^{\kappa NN}_{\delta_{L_p},\texttt{feature\_colour},📷,\kappa=10}\big(\tau^{\varepsilon NN}_{\delta_{\text{walking}},\texttt{location},📍,\varepsilon=1\text{km}}(\texttt{paintings} \bowtie \texttt{museum})\big)\big)$$

As noted previously, exchanging the order of the κNN and the εNN query may result in an empty result set. Bob is obviously interested in having the results of the εNN museums which exhibit a painting as similar as possible to the photograph taken. Hence, a post-filter semantics is the more sensible choice for combining the κNN and the εNN query.

5.3.2 Executional Query Model

So far, we have only considered the logical perspective of a query. The declarative style of database languages obviously puts more weight on the question of *what* to retrieve rather than the *how* to execute and process a query. [BBZ12a] notes that in the given context of multimedia retrieval, it is advisable for users to be able to express preferences with respect to the execution of the query. Hence, orthogonal to the logical query model, we introduce in the following an executional query model.

The query execution model is applied in the processing of a query following the traditional four stages of query processing:

query parsing The query is parsed and transformed to a query tree which is understandable by the data management system.

query rewriting In our model, the rewriting operation does not only perform transformations which result in equivalent trees to the query trees. Instead, based on executional query model which specifies executional preferences, the resulting query trees and the corresponding results may not yield results equivalent to each other, as, for example, approximate scanning methods may be applied.

[3] While our model prefers the post-filter semantics for κNN queries, it should be noted that a pre-filter semantics can still be achieved by stacking two queries in each other, for instance, by specifying a logical query with a similarity predicate within the relational expression R, and defining the Boolean predicate in the outer query. In this way, the pre-filtering κNN is executed first as part of the relational expression, while the Boolean predicate is evaluated second.

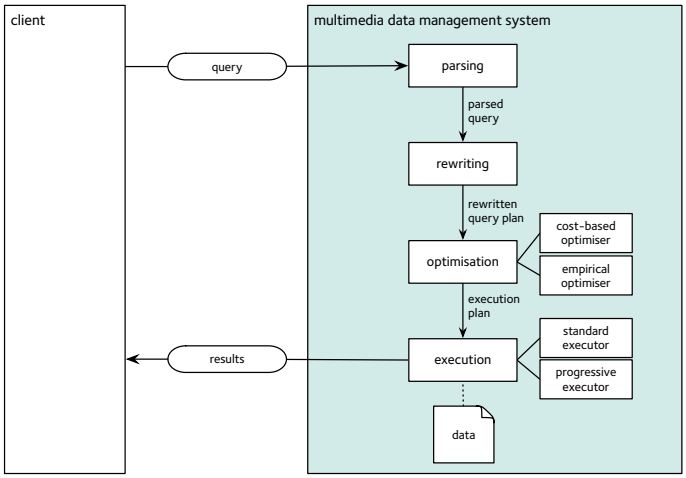


Figure 5.6 Processing model of a query in a multimedia data management system.

query optimisation In the optimisation step, our model supports both a cost-based planner and an empirical planner for optimising the execution of a query.

query execution In the query execution step, two modes of execution are distinguished. The default mode assumes a request/response processing in which the user awaits (without any intermediate results) a response from the database system. The progressive execution mode, on the other hand, attaches an observer to the query processor and returns results of the intermediate query operations in a streaming fashion.

Figure 5.6 gives an overview of the processing of a query within our model.

In the following, we consider how a query is executed and the means to adjust the execution of the similarity predicates. We distinguish

- the scan method (see Section 5.3.2.1),
- the optimisation method (see Section 5.3.2.2),
- the execution method (see Section 5.3.2.3),

as adjustable parts which we will discuss in more detail in the following.

5.3.2.1 Scan Method

In the following, we consider various methods of scanning a relation for answering a similarity predicate. Besides having the data management system decide on the scanning method, as known from traditional database systems, also the following scanning options are available.

Data scan In a data scan, each tuple of a relation is read in sequential order and processed.

Precise index scan A precise index scan uses the available scans which are known to retrieve precise results and not miss any tuple. The only precise index scanning method available within our system is the Vector Approximation-File index (see Section 4.5.7).

Approximate index scan Approximate index structures (see Section 4.5) trade high efficiency at the expense of some acceptable imprecision by missing certain results (false dismissals). The index structures yielding approximate results include the Cluster Pruning index (see Section 4.5.2), the Locality-Sensitive Hashing index (see Section 4.5.3), the Metric Inverted-File index (see Section 4.5.4), the Product Quantisation index (see Section 4.5.5) and the Spectral Hashing index (see Section 4.5.6). In an approximate index scan, the approximate index structures are queried for answering a query.

Stochastic scan Stochastic query execution follows the idea of rank aggregation-based scanning (see Section 4.3.4.1). We combine the number of appearances from different index scans to compute a distance score denoting the probability for the element to be part of the κ nearest neighbours. In particular, we employ this method using multiple index structures which can quickly be queried. We assume that the more often a tuple appears in the candidate list of the various scans, the more likely the tuple is to appear higher in the top κ results. We define the approximate distance as

$$\delta(t.a, \boldsymbol{q}) \approx 1 - \Pr[t \in \mathcal{R}_{\boldsymbol{q}}(R)] = 1 - \frac{1}{|\dot{\mathcal{A}}_{\boldsymbol{q}}|} \sum_{\forall i : \mathcal{A}^i_{\boldsymbol{q}} \in \dot{\mathcal{A}}_q} \begin{cases} 1 & \text{if } t \in \mathcal{A}^i_{\boldsymbol{q}}(R) \\ 0 & \text{if } t \notin \mathcal{A}^i_{\boldsymbol{q}}(R) \end{cases} \tag{5.35}$$

where $\mathcal{R}$ denotes the result set for a query, $\mathcal{A}^i(R)$ denotes the filtered (candidate) tuples; we summarise the candidate results of the single scans in the variable $\dot{\mathcal{A}}$. Note that the number of results to return κ is adjusted to κ' (with $\kappa' > \kappa$) to increase the precision of stochastic scanning.

Stochastic querying may particularly be interesting if the results of the single scans can be retrieved very fast (e.g., by an approximate index scan). Moreover, it allows to lower the number of candidates to be accessed in a data scan, as only κ (or κ') results are returned from the stochastic scan.

Stochastic scanning may be applied in isolation or in combination with a data scan. In isolation, only the values stored in the index are returned. This might be interesting in instances where the index does not only store the indexed value, but also other attributes, and an additional data scan might not be necessary. Together with a subsequent data scan, obviously, the full tuples are returned.

Parallel scan Scanning multiple structures in parallel can be considered a useful method for scanning data only if the intermediate results can be retrieved and processed by the client application as well. In parallel querying, rather than selecting one single scan method, the optimiser may select multiple query paths which are executed in parallel by the system. Parallel scanning obviously hurts the efficiency of the entire system (e.g., as multiple queries are executed at once and as intermediate results have to be collected from distributed sites), but it allows to trade the parallel computation with query response time if used together with the progressive execution. Hence, results from a coarse index structure may be returned first, while more precise scans are returned later. In combination with progressive querying which allows to return intermediate results to the user, a user may decide how long to wait for the results whose quality improves over time.[4] Note that the results of the parallel scan are independent of each other and not combined.

5.3.2.2 Optimisation Method

In our query processing model, we provide both a simplistic cost-based and an empirical query planner. Both optimisation strategies obviously consider the selected scanning methods for selecting specific query paths. Note that our model allows the combination of cost-based and empirical optimisation, i.e., on the level of a similarity-based query an empirical optimisation may be used, while on the level of the full query involving a Boolean predicate, a cost-based optimisation might be applied.

Cost-based optimisation The cost-based planner performs a parametric query optimisation. Hence, while it uses estimations based on the relation and tuple size to estimate a cost for each execution step, it considers the scanning method parameters for selecting an execution path. The costs for scanning are estimated by

$$\widehat{n} \times \widehat{\texttt{size}(t)} \tag{5.36}$$

where $\widehat{n}$ and $\widehat{\texttt{size}(t)}$ are used to denote the estimated size of the relation and the size of a tuple, respectively. For index-based scans, the tuple size is obviously adjusted to the size of the index tuple.

Empirical optimisation The empirical optimisation tracks the execution time and performs experiments for stored queries for every data scanning methods (e.g., for various indexing mechanisms) within the query tree and leverages this knowledge for the selection of the

[4] Obviously, the result quality is supposed to improve in progressive querying. However, measuring the quality is not possible given the fact that the ground truth is not available at query time. Hence, we have manually set empirical confidence values for the single scanning methods, ensuring that a possibly less precise index will not overwrite the results returned by a precise index.

Table 5.1 Selection of additional features used for the empirical query optimiser for index scans.

CP	LSH	MI	PQ	SH	VA
number of leaders	number of hash functions	number of reference points	number of subvectors	number of eigenfunctions	number of marks
	number of hash tables	number of reference points to store	number of clusters per split		
	number of buckets	number of reference points used for querying			

best execution plan with the goal of reducing the query time. Figure 5.7 gives an overview of the optimisation loop.

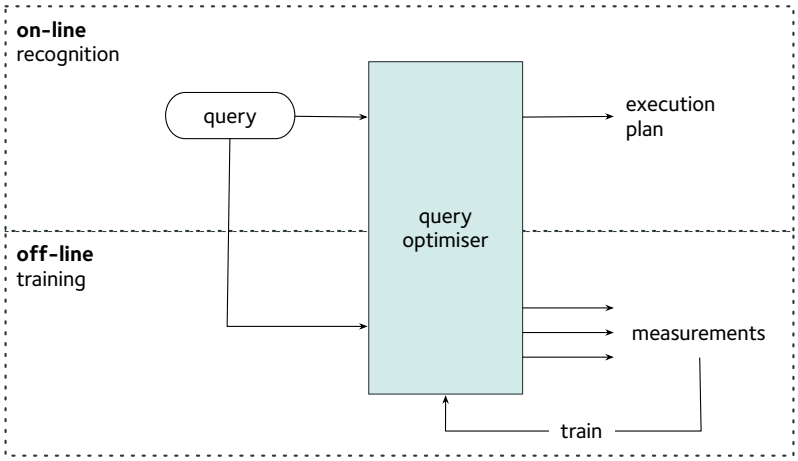


Figure 5.7 Optimisation loop empirical optimiser: The illustration displays the off-line, training phase during which queries are executed at the measured time is used for training the optimiser. At on-line time, the optimiser is used to estimate the execution time for the given scanning method.

In detail, we extract features from a given query and map these to a vector space. The features generally include the following parameters

- relation size,
- dimensionality of the vector data,
- number of elements to retrieve.

Moreover, we add index-specific parameters to the data points, as summarised in Table 5.1.

The extracted features are normalised to $[0, 1]$ and using regression analysis a time-estimating predictor is created. For each of the index structures and for the sequential

scan, a separate regression model is generated. At on-line time, the various estimations of execution times are compared to each other and the query plan with the estimated lowest execution time is chosen.

5.3.2.3 Execution Method

Request/response execution The request/response execution of queries is the general mode known from traditional databases: For a query, a response is returned at the end of the query processing, while the user is awaiting the results; no intermediate results are returned to the user.

Progressive execution The idea of progressive retrieval is to offer to the user an updating result set whose precision increases with time [KG05]. Such an approach is particularly interesting for long-running queries which can often be found in multimedia retrieval systems. With progressive querying, the query returns all intermediate results of the query steps between posing the query and having the final results of the query ready. Algorithm 5.1 presents the algorithm of progressive querying.

Algorithm 5.1 Progressive querying.

Input: relation R, operations to apply $\mathrm{op}_1, \ldots, \mathrm{op}_k$, observer *obsrv*
Output: result set $\mathcal{R}$

1: step 1: $\mathcal{A}^1 := \mathrm{op}_1(R)$
2: *obsrv*.notify($\mathcal{A}^1$)
⋮
3: step i: $\mathcal{A}^i := \mathrm{op}_i(\mathcal{A}^{i-1})$
4: *obsrv*.notify($\mathcal{A}^i$)
⋮
5: $\mathcal{R} := \mathcal{A}^k$
6: return $\mathcal{R}$

In our approach, we consider atomic operations $\mathrm{op}_1, \ldots, \mathrm{op}_k$. An observer which registers on the executed function is responsible for returning sub-results while the query is still running. The definition of atomic operations that can be returned by the system depends on the high-level operation and its implementation.[5]

A stopping criterion may be specified with the progressive execution to early stop the execution as soon as a criterion is satisfied (e.g., the results coming from a precise scan).

[5] In our approach, index structures are executed atomarily; however, depending on the construction of the query tree also the results from single partitions could be returned to the query processor independently.

Table 5.2 Overview of parameters of the logical query model.

Projection attributes	Relational expression	Boolean predicate	Similarity predicate
* (all attributes)	simple relation	no Boolean predicates	no similarity query
set of attributes	relational expression	with Boolean predicates	similarity query

Table 5.3 Overview of parameters of the executional query model.

Scan method	Optimisation method	Execution method
data scan	cost-based optimisation	request/response execution
precise index scan	empirical optimisation	progressive execution
approximate index scan		
stochastic scan		
parallel scan		

5.3.3 Summary and Model Queries

In Table 5.2 and Table 5.3, we summarise the logical and executional query parameters coming from the corresponding model. Obviously, not all combinations are sensible; for example, querying in parallel a relation without returning intermediate results, does not make sense.

In the following, we introduce a number of archetype model queries which can be evaluated using the introduced query model.

Basic κNN query We consider first a basic κNN similarity query. Such a query needs the specification of a similarity predicate. More precisely, it requires the specification of an attribute (a) involved in the computation of a distance; for computing a distance score, the distance function ($\delta(\cdot,\cdot)$) and a query vector ($\boldsymbol{q}$) have to be indicated as well. The desired cardinality of the answer set is given by specifying a limiting predicate to only retrieve a sub-set of κ neighbours out of all elements. Without specifying any executional parameters, the model assumes a simple request/response execution using the cost-based optimiser.

Example 5.9 Basic κNN query.

Alice has taken a photograph of a film running on the television and wants to learn the title of it. She is looking for the title of the $\kappa = 10$ film titles which contain the scene shown on television ordered by similarity to the photograph taken. Her application extracts from the photograph the inherent features and sends a query to the data management system. The query is specified by providing projection attributes (A), a relation (R), a similarity

predicate (ψ) given by the specification of distance ($\delta(\cdot,\cdot)$), the attribute to use for the similarity retrieval (a), the query vector ($\boldsymbol{q}$) and a limiting factor specifying the number of elements to return (κ). In detail, the query is posed as:

Logical query parameters		Executional query parameters	
A	`title`	scan method	(determined by optimiser)
R	`films` ⋈ `segments`	optimisation method	cost-based optimisation
φ	-	execution method	request/response execution
ψ	δ_{L_2}(`feature_colour`, 📷) with $\kappa = 10$		

In here, 📷 denotes the colour features extracted from the photograph in vectorial form. The query returns a list of film names ordered according to the implicit attribute $\bar{\delta}$ denoting the distance between the stored vectors and the given query vector.

Query specifying a Boolean and a similarity predicate In the following, we consider a query specifying both a Boolean and a similarity predicate. By the order of precedence of the operations introduced previously in Equation 5.34, first the Boolean predicate is evaluated on the relational expression; only then the similarity predicate is evaluated.

Example 5.10 Query specifying a Boolean and a similarity predicate.

Alice would like to retrieve a list of $\kappa = 10$ films directed by "Woody Allen" in which the Eiffel Tower appears. Assuming that `feature_object` may detect the Eiffel Tower, hence, a query may be specified as (with 🗼 denoting an object specification of the Eiffel Tower in vectorial form)

Logical query parameters		Executional query parameters	
A	`DISTINCT title`	scan method	(determined by optimiser)
R	`film` ⋈ `segment`	optimisation method	cost-based optimisation
φ	-	execution method	request/response execution
ψ	δ_{L_2}(`feature_object`, 🗼) with $\kappa = 10$		

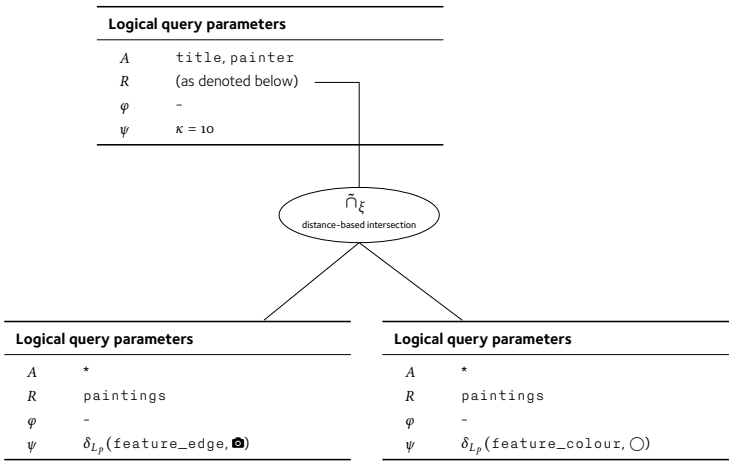
By the precedence rules specified in Equation 5.34, first the Boolean predicate is evaluated, i.e., the data is first filtered by the director "Woody Allen"; in the next step, the similarity query is executed on the remaining data to find the κ multimedia documents whose probability for displaying the Eiffel Tower is highest.

Complex similarity query In what we consider being a complex similarity query, we follow the definition of [BMS+01] noting that such queries are constructed by multiple features and/or multiple query objects (see Section 3.2.2).

Depending on the desired semantics of the query, our query model is able to model such complex queries by making use of similarity-based union and intersection operations within the relational expression given in the logical query model.

Example 5.11 Complex similarity query (based on Example 3.2.2).

Bob wants to find similar paintings of van Gogh's wheat field with a green hue. In his application, he is able to take a photograph and specify an additional colour information. The retrieval application translates the query into a complex query using a distance combining function and by performing a similarity-based intersection operation. For reasons of simplification, in the following, we do not specify any executional parameters. The query can be visualised as (we use 📷 to denote the extracted features of a photograph, and ◯ to mark the specified colour)



By the precedence rules specified, first the relational expression is evaluated and with that each part of the relational expression is solved. Hence, for the same relation `paintings`, the distance is computed for both attributes separately and combined using a distance-based intersection operation. Ultimately, the number of elements is limited to $\kappa = 10$ and a projection is performed.

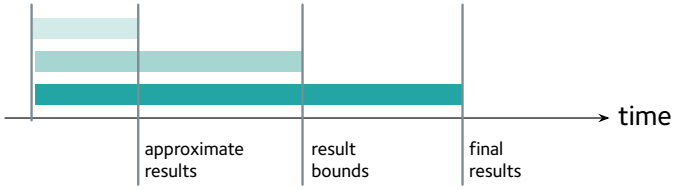
Progressive parallel query With progressive querying, the query returns all intermediate results of the query steps between posing the query and having the final results of the query ready. By combining queries that quickly produce approximate results (e.g., scanning a very coarse index) with queries that take longer to produce correct results (e.g., scanning a very fine-grained index or the full data), we model progressive results. Executing queries in parallel and considering the most up-to-date results as the currently best results, could be useful when scanning the same data on multiple coarseness levels. For this, however, the application must be able to handle first, approximate results, but also result sets which improve over time. As noted previously, such an approach obviously decreases the efficiency of the data management system, but it allows to trade the parallel computation with query response time.

Example 5.12 Progressive parallel query.

The application Alice is using to find paintings similar to the photograph taken allows the user to see first, imprecise results which update while the user has the application opened. The results improve in quality while she waits until she receives the final results. We consider the situation where Alice is looking for the name of the $\kappa = 10$ most similar scenes compared to the photograph taken. The query is posed as follows:

Logical query parameters		Executional query parameters	
A	`title`	scan method	parallel scan
R	`film` $\bowtie$ `segment`	optimisation method	cost-based optimisation
φ	-	execution method	progressive execution
ψ	δ_{L_2}(`feature_colour`, 📷) with $\kappa = 10$		

The results may – exemplified – be returned to the user as visualised in the following illustration, where first approximate results arrive first, followed by result bounds (e.g., by the VA-File index), followed by the final results at which the query is stopped.



5.4 Distribution Model

The distribution model we employ assumes distribution, partitioning and location transparency. Hence, on the level of the logical model, the distribution of the relation to multiple sites is not visible. The user does not formulate a query in terms of specific partitions or sites, but rather phrases a global query that is translated in the localisation step by the distribution model into sub-queries which are run on the individual partitions and at the individual sites. A result merging strategy is used to combine the results from each partition to construct a final, coherent result set.

5.4.1 Distributed Query Processing

5.4.1.1 General Distributed Query Processing

Following the general processing of a query in a distributed system and the ideas of [ML14; TVM+11], at query time, the partitioned entities or index structures are queried separately by the individual local processors responsible for a partition under the central orchestration of a global query processor at the coordinator site. Each worker takes care of filtering out the local nearest neighbours (limited by the predicate ϑ) for a given query and sends the partial result set to the coordinator. In a reduction step, the coordinator merges the results of the local searches to a global, coherent result set based on the partial result sets of nearest neighbours. In Algorithm 5.2, we summarise the algorithm for executing a query in a distributed setting. Note that the algorithm is generally dominated by the local searches and the processing at the individual sites.

Algorithm 5.2 Distributed query processing (based on [ML14]).

Input: relation R distributed over multiple sites $S = \{s_1, \ldots, s_n\}$, query $\check{\mathrm{Q}}$
Output: result set $\mathcal{R}$

1: broadcast query $\check{\mathrm{Q}}$ to all sites $s_1, \ldots, s_n$
2: for all site $s_i \leftarrow s_1, \ldots, s_n$ do
3: $\quad \mathcal{A}^{s_i} \leftarrow$ locally execute query $\check{\mathrm{Q}}$ and retrieve partial result sets
4: end for
5: $\mathcal{R} \leftarrow$ merge the partial result sets to one coherent result set
6: possibly reapply limiting predicates of the similarity predicate

5.4.1.2 Index-based Query Processing

In line with our categorisation of indexes which perform exhaustive searches over the whole index, and indexes which are able to prune large parts of the index (e.g., by only reading a set of buckets), we distinguish two ways of processing an index in a distributed fashion:

distributed pruning Distributed pruning refers to the processing of a query by an execution method which allows to prune elements (e.g., Cluster Pruning, Locality-Sensitive Hashing). In this processing schema, the query is sent to all sites and the buckets which do not satisfy the query are skipped at each site.

distributed ranking Distributed ranking refers to the processing of a query by executing locally a similarity search (exhaustive search) and returning the results satisfying the limiting predicate of the similarity search. The results from the single partitions are then merged and the limiting predicate is applied again. Similar approaches have been used with the VA-File index (cf. [Web97; WBS00; WBS01]).

In Figure 5.8, we visualise both processing strategies. While in distributed pruning only the elements that match the query are returned to the global coordinator, distributed ranking will retrieve a sorted list of (κ) nearest neighbours and merge the results from the single sites to one coherent list.

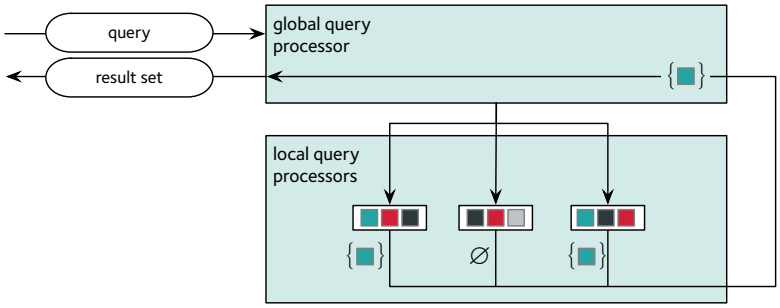
5.4.1.3 Data Transfer for Result Set Merging

In the following, we consider methods to transfer data between the local processors and the coordinator and merge the distributed candidate sets.

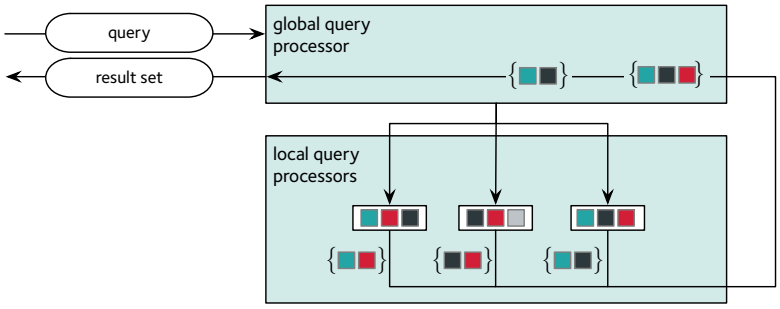
Consider, for instance, an index which is processed by multiple query processors and whose partitioning does not match the partitioning of the underlying data. In this case, the tuple identifiers ($\text{TID} \in \text{TID}$) from the index scan need to be communicated between all sites storing the data which then in turn return the full tuples to the query coordinator.

The straightforward approach would communicate a set of identifiers between all sites. Our distribution model, however, considers also two further approaches.

Given the approximative nature of the query, *Bloom filters* [Blo70] may, for instance, be used. A Bloom filter is a probabilistic data structure which can be used to test the membership of an element in a set. While Bloom filters may yield false positives, they will not yield false negatives. Hence, using Bloom filters to return the set of identifiers may return more elements than truly selected, but it will never miss any result. The advantage of Bloom filters is their small size. Hence, at query time, the results from the index structure at each site may be encoded in a short Bloom filter. The Bloom filters from each site are combined at the coordinator site and sent back to each site which will use the filter for



(a) Distributed pruning strategy.



(b) Distributed ranking strategy.

Figure 5.8 Visualisation of two strategies for distributed index-based query processing.

filtering the local results. Bloom filters have been used, e.g., in the context of text retrieval in P2P networks (e.g., [NYF08]).

On the other hand, consider the situation where a κNN search is performed in a distributed ranking fashion. In this case, each local processor would answer with a set of tuples which represent the local κ nearest neighbours. However, when merging the results, possibly many elements will fall out of the final result set. It seems, hence, a waste of resources to transfer full tuples to the coordinator. Instead, only the necessary information should first be transferred for determining the relevant elements; in the subsequent step, for the remaining items, the full tuple data may be transferred. Such an operation may be denoted as a join operation, i.e.,

$$p_1 \bowtie p_2 = p_1 \bowtie (\pi_{a_1}(p_1) \bowtie p_2) \tag{5.37}$$

where p_1, p_2 denote data partitions.

To reduce the number of tuples transferred to other sites, the *semi-join operation* [Kos00] may be employed. It is defined as

$$p_1 \ltimes p_2 = p_1 \bowtie \pi_{a_1}(p_2) \tag{5.38}$$

resulting in the tuples from p_1 which join with a tuple of p_2, removing the dangling tuples of p_1 [Dat15, pp. 219; Gar09, pp. 1001]. Note that the semi-join operation is not commutative, i.e., $p_1 \ltimes p_2 \neq p_2 \ltimes p_1$.

5.4.2 Data Partitioning Model

In the context of multimedia retrieval, finding a sophisticated partitioning scheme which allows to decide to which partitions to route a query to, becomes a difficult task to solve, i.e.,

$$\check{\mathrm{Q}} \rightarrow \tilde{P} \tag{5.39}$$

where $\tilde{P}$ is a sub-set of P and by whose selection we trade off retrieval time with result quality. Decreasing the size of the sub-set of partitions may still yield a very acceptable result quality while lowering at the same time the retrieval time substantially.

However, the same property which prevents the use of classical index structures such as the lack of order in the feature data, the lack of clusterability, and moreover the lack of meaningful extractable parts in the raw data, makes it similarly difficult to find a partitioning scheme that can be generally applied and which allows to prune partitions at query time from the retrieval without possibly loosing relevant elements. Such a partitioning of the data could not only be interesting for vectorial data, but also for the derived indexes. In particular, we see an application for index structures which normally perform a ranking of the resulting elements and, hence, require an exhaustive search (resulting in a computational complexity of $\mathcal{O}(n)$), such as the VA-File. With these index structures, the partitioning may be used as a coarse quantiser allowing to prune a large number of elements early on (as it is done, e.g., in the non-exhaustive PQ index [JDS11]).

We have noted before that creating clusters is a difficult problem to solve for high-dimensional data. Nevertheless, in [Hel16], we attempt to introduce two partitioning schemes for horizontally partitioning both relations and indexes. The partitioning strategies can be categorised as generating a *derived partitioning*, meaning a partitioning which is not directly based on the attribute data:

partitioning based on Cluster Pruning This partitioning strategy partitions the data based on the ideas of the Cluster Pruning index. A set of leaders are selected from the data and the tuples are assigned to the leader based on a feature attribute. Similarly, in the partitioning strategy, the data is assigned to a partition depending on a set of selected leaders.

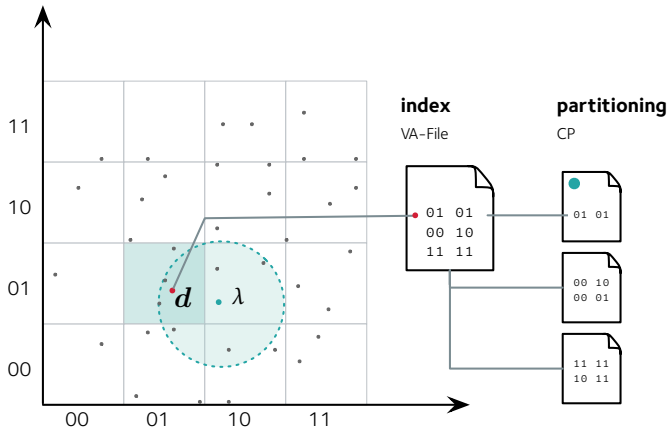


Figure 5.9 Partitioning of a VA-File index based on a CP strategy.

partitioning based on Spectral Hashing This partitioning strategy partitions the data based on the ideas of the Spectral Hashing index. Given that the Hamming distance between two hashes reflects to some extent the true distance in a Euclidean space, we make use of the hash to cluster similar elements into the same partition and separate very different elements into different partitions. A cluster leader for each cluster is chosen.

Figure 5.9 exemplifies the partitioning based on Cluster Pruning by displaying a VA-File index which is partitioned based on the Cluster Pruning strategy: A data point is indexed using the VA-File strategy and partitioned based on the closest leader following the CP indexing mechanisms. At query time, only a selected number of partitions belonging to the closest leaders to the query are read and out of these the κNN are selected using the VA-File querying algorithm.

With both partitioning schemes, a resource representation for each partition is created and a ranking of partitions according to the query is possible. In early evaluations in [Hel16], we have, however, shown that these partitioning schemes do not reliably outperform skipping partitions randomly. This finding is in line with similar findings of [CPR+07] noting that there is a dramatic deterioration of performance with increasing number of clustering levels because informative leaders are lost with more levels.

5.5 Storage Model

In this section, we focus on the storage model which considers the physical aspects of the data and is used to persist the relations from the logical data model. While the storage model in classical database system is mostly comparably simple, as it allows to persist data in a file-based fashion on disk, in the following, we relax the model to store the data from relations and indexes in a large variety of storage media/applications. In our case, the storage model is related to the distribution model as distributing data to multiple sites results in the same challenges as distributing data over multiple storage systems. We extend the physical storage mechanisms to also allow for more high-level approaches, for instance, storing the data in a key-value store or in a relational database. Following the ideas of polystores (see Section 4.4.3), in our storage model, we allow to apply various storage engines depending on the data and the queries at hand.

We consider a storage engine being an adapter between the data processing engine of the database system and the persistence layer. For a storage engine, the definition of (based on the ideas of [DZE+15])

- a mapping of the relational data to the local storage model and vice-versa,
- a mapping of the global operations (e.g., insertion, deletion, retrieval) to the query language of the engine, together with a mapping of data types,

is necessary. With this, a storage engine allows to store the data of a relation on multiple, different storage systems and distribute the data accordingly by means of vertical partitioning. A logical schema requires a mapping to a storage engine; formally, there exists a mapping

$$\text{SCH}(R) \rightarrow SE \tag{5.40}$$

i.e., for each attribute of the schema of a relation $a \in \text{SCH}(R)$, a storage engine $se \in SE$ is defined. The data for a storage engine se_i is given as

$$\pi_{A_{se_i} \cup \{\text{TID}\}}(R) \tag{5.41}$$

where A_{se_i} denotes the set of attributes to be stored on the storage engine se_i and $\text{TID} \in \text{TID}$ denotes a tuple identifier.

Following the definition of vertical partitioning, note the necessity to store an identifier for each tuple at each storage engine to be able to fully reconstruct the relation. The query processing, hence, follows the query processing with vertically partitioned, distributed data. Note in addition that the query processing needs a translation step which translates the localised query at hand to the query language of the storage engine. While the storage schema records the details of each storage engine, a shim translates the query to the query language of the corresponding engine. In Algorithm 5.3, we present the processing of a query in such a context.

Algorithm 5.3 Query processing over multiple storage engines.

Input: relation R split over multiple storage engines $SE = \{se_1, \ldots, se_m\}$, query $\check{Q}$
Output: result set $\mathcal{R}$

1: determine storage engines $SE_{\check{Q}}$ involved in query (possibly all)
2: for all storage engines $se_i \in SE_{\check{Q}}$ do
3: translate query to local query dialect
4: $\mathcal{A}^{se_i} \leftarrow$ evaluate query on storage
5: end for
6: $\mathcal{R} \leftarrow$ merge the partial result sets to one coherent result set by joining the result sets on TID

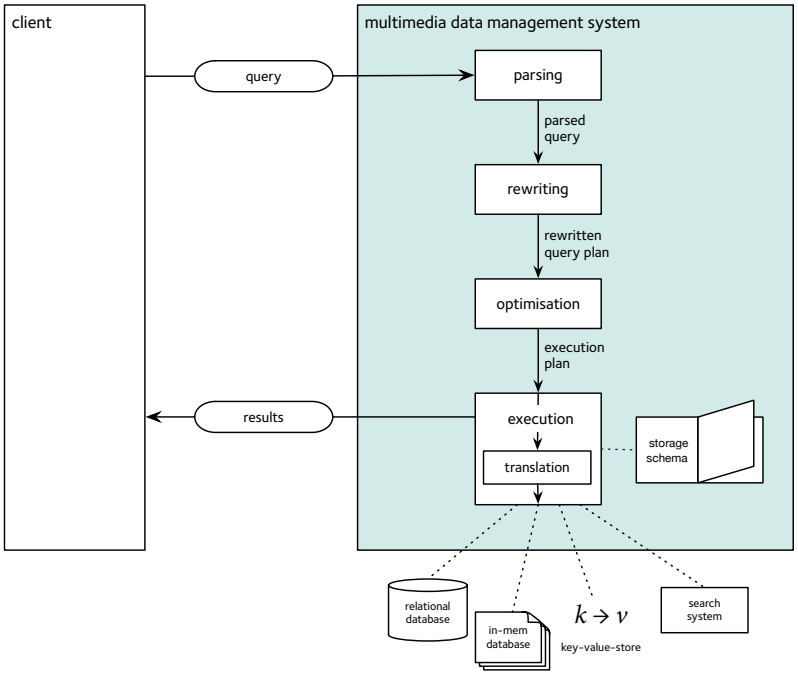


Figure 5.10 Distributed query processing using multiple storage engines.

Figure 5.10 visualises the query processing focusing on the query execution when using multiple storage engines.

Chicken.

— Doug Zongker, *Chicken Chicken Chicken: Chicken Chicken*

6 Implementation

We have prototypically implemented the presented concepts in the software ADAM$_{pro}$ [GS16]. The software has been released[1] under an MIT licence within the vitrivr stack [RGT+16a; RGG+17], which forms together with the retrieval engine Cineast [RGS14] and the vitrivr user interface an open-source multimedia retrieval system. The implementation of ADAM$_{pro}$ provides a fully working data management system for multimedia data.

Our implementation has focused on the retrieval part of the data management system. Hence, we have given great attention to the query modelling and execution including index structures, similarity operations, etc. Components for admission control, replication, locking and logging (for transactions), etc. have been implemented – if at all – in only basic versions. Figure 6.1 presents an overview of the components available in ADAM$_{pro}$.

ADAM$_{pro}$ is implemented mainly in Scala 2.11 using the Apache Spark framework in version 2.2 (July 2017). We give more details on the software stack in the following and present the components implemented in ADAM$_{pro}$. We finally present a web-based client application for our system which allows to visually set up query plans and analyse the query execution within the data management system.

6.1 Software Stack

6.1.1 Apache Spark

Apache Spark[2] [ZCF+10] is a cluster-computing framework for large-scale data processing. The processing is based on the concept of resilient distributed data (RDD) [ZCD+12], an immutable, distributed multi-set which can be operated on in parallel. RDDs allow to execute higher-order programming constructs (e.g., map, reduce, filter, aggregate) on a

[1] https://github.com/vitrivr/ADAMpro/
[2] http://spark.apache.org/

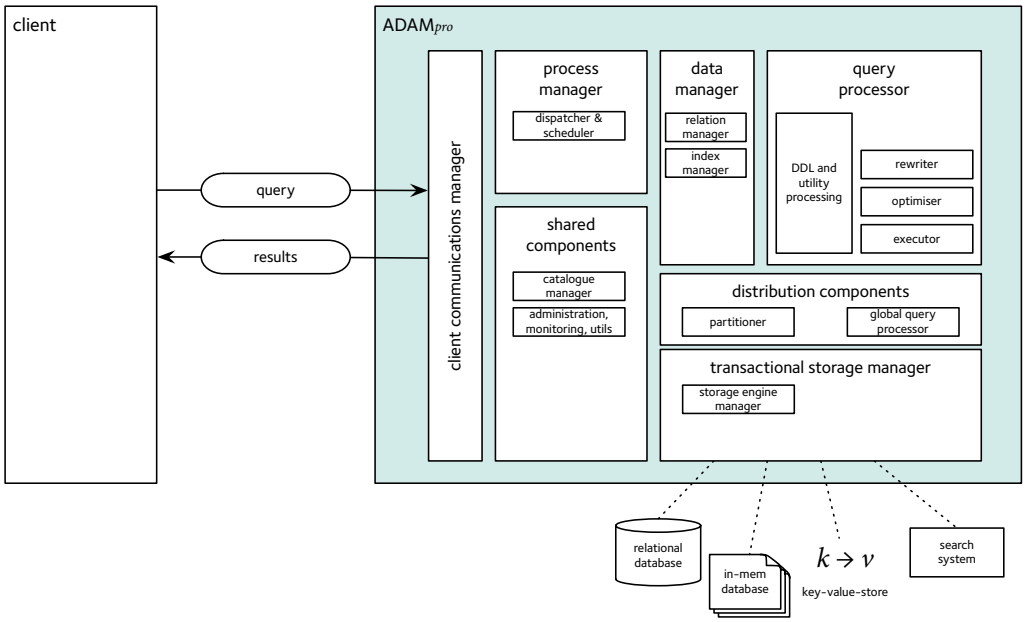


Figure 6.1 Main components of the ADAM$_{pro}$ implementation (based on Figure 5.1).

driver program on the master node (*coordinator*), which subsequently invokes parallel operations on the executors residing on worker nodes (*local processors*).

The Spark stack is composed of the Spark core and further, independent components which can be added ad libitum.

The *Spark core* embraces the basic functionalities of Spark, including the RDD abstraction, the scheduling of tasks, the memory management, the interaction with storage systems, etc. [KKW+15, p. 3]. A high-level deployment view of the components of Spark core is given in Figure 6.2: The illustration depicts the Spark core driver program with the Spark context running on a master node. The worker nodes, which are managed by a cluster manager, run the executors, which are connected to the Spark context and execute the functions invoked on the driver node resulting in localised tasks. To leverage the data locality, the workers may attach, for example, to Hadoop data nodes and execute the tasks at the sites where the data is locally available.

Spark SQL [AGZ+15] introduces on top of the Spark core RDD abstraction the notion of data sets (earlier also named data frames), which provide support for the relational processing of (semi-)structured data. This includes the components of a query processing pipeline, in particular, for parsing, rewriting, optimising and executing a query. The data set API supports all common relational operations (e.g., projection, selection, join, aggregations, etc.). Table 6.1 summarises common operations available in Apache Spark SQL that can be applied on a data set together with a symbolic notation which will be used throughout this chapter.

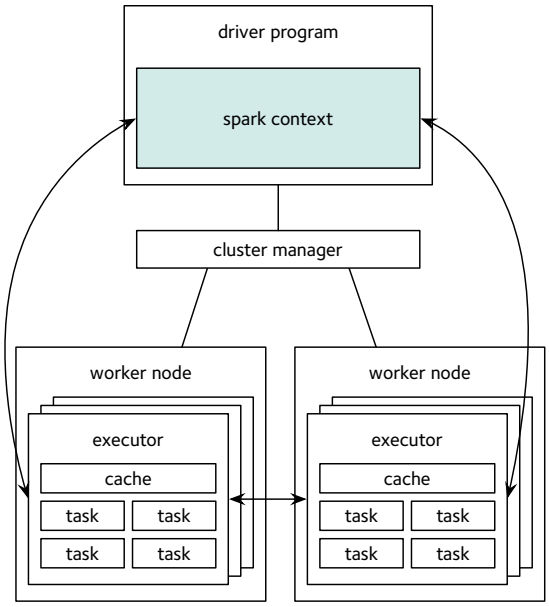


Figure 6.2 Deployment architecture of Apache Spark components: The Spark core driver program runs within the Spark context on a master node; the worker nodes run an executor which execute the functions invoked on the driver (based on [KKW+15, p. 15; Kir16]).

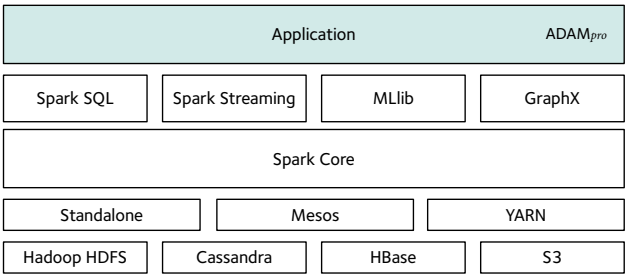


Figure 6.3 Apache Spark software stack: The highlighted box denotes our implementation ADAM$_{pro}$ (based on [Kir16]).

Furthermore, the Spark framework includes a streaming (*Spark Streaming*), a machine learning (*MLlib*) and a graph processing (*GraphX*) library. The framework has connectors to various storage systems (e.g., the Hadoop file system (HDFS), Apache Cassandra, Apache HBase, Amazon S3, etc.). Spark can run as a stand-alone application, or use Apache Mesos or YARN as cluster manager to manage the worker nodes in the cluster.

Figure 6.3 gives an overview of the software stack of Apache Spark. We have highlighted our implementation, ADAM$_{pro}$, within the full stack.

Spark is currently released in version 2.2 (July 2017) under an Apache License 2.0 and is in active development. Various commercial and research applications have been released so far on top of the Apache Spark stack (e.g., for performing genomic analyses [NLF+15] or for processing geographic data [YWS15]); the framework has recently also received increasing interest in the field of multimedia retrieval research (e.g., [GAJ+17]).

Table 6.1 Common data set operations in Apache Spark 2.2: The table depicts the operation and a description, together with a symbolic notation used throughout this chapter (based on [Apa17]).

Operation	Symbol	Description
`.map`		returns a (distributed) data set containing the result of applying a function to each tuple
`.mapPartitions`		returns a (distributed) data set containing the result of applying a function to each partition
`.select`		returns a (distributed) data set on which a projection has been performed
`.filter`		returns a (distributed) data set that only contains elements which satisfy a given Boolean predicate
`.join`		returns the (distributed) result of joining two data sets; various join operations are available, including the inner, outer, full, semi, anti and natural join
`.sort`		returns a new (distributed) data set sorted by the specified expression
`.limit`		returns a new (distributed) data set which is limited in the number of rows
`.collect`		returns a (localised) array that contains all rows of the data set at the master node

6.1.2 Google Protocol Buffers and gRPC

Google Protocol buffers[3] allow the specification of an interface which describes the schema of the data exchanged and specifies services that make use of the data schema defined. Code generators for various programming languages are available to translate the protocol definition into the corresponding code. On the other hand, gRPC[4] offers a remote procedure call framework which makes use of the service definitions in the protocol files to generate client and server stubs that can readily be used for the communication. The gRPC framework supports both request/response and streaming services.

6.2 Components

The structure of the source code of ADAM$_{pro}$ follows to large extents the component view given in Section 5.1 and presented in Figure 6.1. Figure 6.4 gives an overview of the components available in ADAM$_{pro}$ and their relationships in a package diagram. In the following, we briefly discuss the implementation perspective of some of the components.

[3] `https://developers.google.com/protocol-buffers/`
[4] `https://grpc.io/`

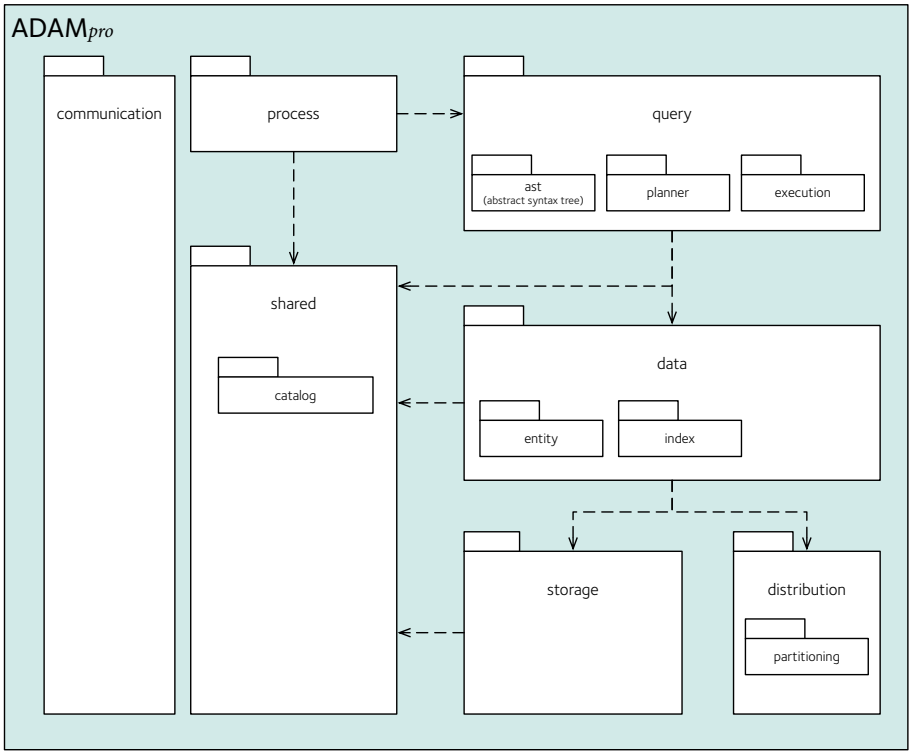


Figure 6.4 Package diagram of ADAM*pro*: The illustration depicts the most important components and their relations between each other.

Client communications manager The client communication manager, responsible for the communication with a client, has been implemented using Google Protocol Buffers and gRPC. Our protocol supports both streaming and request/response queries.

Query processor The query processor acts on queries which have been translated from the communication protocol format to the internal query representation which is used throughout the query rewriting, optimisation and execution phases. The query processing within ADAM*pro* is in alignment with the traditional processing of a query as introduced previously (see Section 5.3.2).

We follow the composite pattern for the implementation of the query expressions and operations (see Figure 6.5), allowing the user to easily compose a query expression. Besides the common relational operations, we also implement operators to incorporate external query systems and, following the ideas of polystores [Sto15], manually query the external stores in their individual query language and translate the resulting tuples into ADAM*pro* tuples for further processing.

The logical query specification follows the query model presented in Section 5.3.1. The executional query model has been implemented by means of executional query hints. The reason for this is that possibly the scanning methods and the optimisation methods cannot be satisfied if, for instance, an index structure is not present or the empirical optimiser has

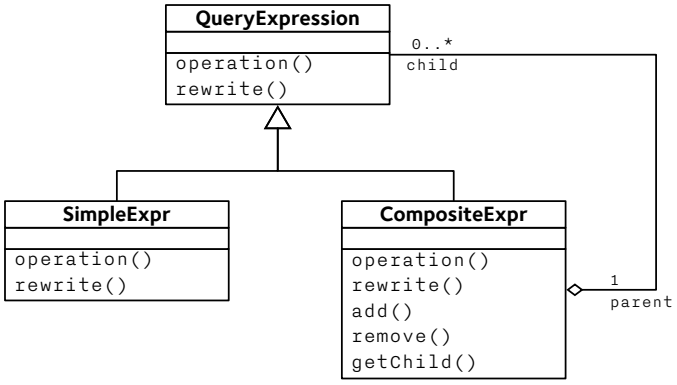


Figure 6.5 Application of composite design pattern for query expressions: The illustration depicts a simplified UML class denoting how a query expression is internally structured.

not yet been trained. For the execution methods available in the model (i.e., request/response or progressive execution) two separate services are available, as the protocol requires specifying the answering mode in advance. Additional to the query hints specifying the scan method and the optimisation method, further query hints may be added to a query denoting which (group of) index structures to use, etc.

To be able to trace queries passing through the query processing pipeline, a tracker is attached to the queries as they move through the system. The tracker can, for instance, be used to dynamically inspect the query processing.

Indexes The implementations of the indexes are based, whenever available, on the code basis given by the authors in the original publication and adjusted to match the Apache Spark framework and the architecture of the ADAM$_{pro}$ system. The implementations follow the algorithms presented in Section 4.5 which are adjusted to support a distributed execution (as described in Section 5.4). We have adopted the existing code for the following indexes:

- Locality-Sensitive Hashing following the implementation of E2LSH [AI15][5],
- Metric-Inverted File[6],
- Product Quantisation[7],
- Spectral Hashing[8],
- Vector Approximation-File[9].

[5] http://www.mit.edu/~andoni/LSH/
[6] https://github.com/giamato/mi-file/ Note that, in comparison to the implementation given by the authors, we have not made use of inverted files for storing the data. Instead, we have adopted the data model given by Apache Spark and store for each data point the reference points in a list.
[7] http://people.rennes.inria.fr/Herve.Jegou/projects/ann.html
[8] http://www.cs.huji.ac.il/~yweiss/SpectralHashing/
[9] The VA-File index has been implemented based on the implementation by the author of [Weboo].

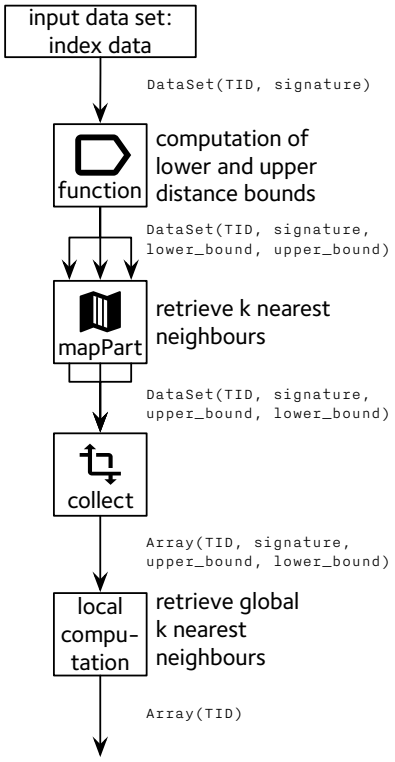


Figure 6.6 Query processing in ADAM$_{pro}$ using a VA-File index.

In the other cases, the index structures have been implemented as closely as possible to the corresponding papers, namely for

- Cluster Pruning,
- VA$^+$-File.

In Appendix A, we list the default parameters set for the index structures implemented.

Exemplary, Figure 6.6 depicts the processing of a query using a VA-File index: The index data is first read and the lower and upper distance bounds are computed based on the marks information for each tuple (`map` operation). Then, for each partition separately, the κ nearest neighbours are retrieved. The partial results are collected at the master node and, using the retrieved distance bounds, the global κ nearest neighbours are determined. The resulting list of tuple identifiers is then possibly redistributed to all available sites and passed on to the next step in the query processing, for instance, the refinement step based on a filtered sequential scan, which computes for the candidate set of tuples the final distance.

Transactional storage manager To be able to handle various types of data, ADAM$_{pro}$ comes with a set of very different storage engines for which a connector has been implemented. While it is possible to query the storage engines directly in the corresponding query language and use the results within ADAM$_{pro}$, the storage engines implement also shims to translate

the query to the dialect used by the engine. The available storage engines include the following:

- Apache Parquet[10] a column-oriented file storage format, stored on the local file system or on a distributed file system, e.g., HDFS or Apache Alluxio[11];
- the distributed wide-column store Apache Cassandra[12];
- Apache Orc[13], a column-oriented file storage format, stored on the local file system or on a distributed file system;
- a Java-port of the local key-value store LevelDB[14];
- PalDb[15], a write-once key-value store;
- PostgreSQL[16] and PostGIS[17], an extension for PostgreSQL that adds support for geographical objects and queries;
- Apache Solr[18], a text-focused information retrieval system running on top of Apache Lucene.

The storage engines implemented in ADAM$_{pro}$ are configurable by means of a configuration file. Furthermore, the storage manager allows to dynamically add and load new storage engines at runtime. Our implementation supports transferring data from one storage engine to another, to allow users to adjust the data storage to their needs.

Example 6.1 Query processing in ADAM$_{pro}$.

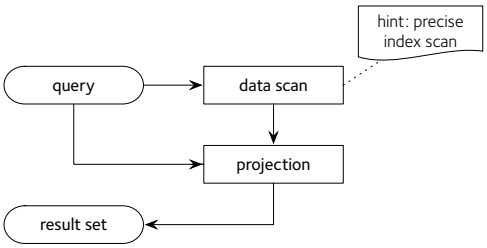
Alice places a similarity-based query in her application using a photograph of a film she is currently watching. The query vector (📷) is generated by her application. She wants to retrieve the ten most similar results to the given query vector. The query is specified as follows as Google Protocol Buffer message:

[10] https://parquet.apache.org/
[11] formerly Apache Tachyon, http://www.alluxio.org/
[12] http://cassandra.apache.org/
[13] https://orc.apache.org/
[14] https://github.com/dain/leveldb/
[15] https://github.com/linkedin/PalDB/
[16] https://www.postgresql.org/
[17] https://www.postgis.net/
[18] http://lucene.apache.org/solr/

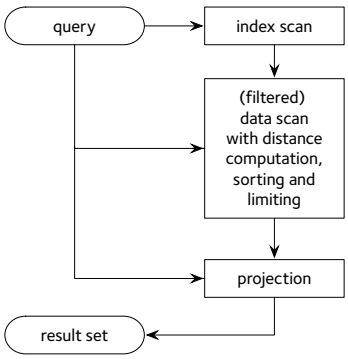
Table 6.2 Exemplary query in ADAM$_{pro}$.

Parameter	Explanation	Example
projection	projection attributes	`title`
relation	name of relation or relational expression	`films` ⋈ `segments`
Boolean predicate	Boolean, filtering predicate to evaluate on the relation	-
similarity predicate	similarity predicate to evaluate on the relation, specifying the details for executing a κNN query, i.e., query vector, distance function, number of elements to retrieve	δ_{L_2} on `feature_colour` with query vector 📷 and κ = 10 as limiting predicate
query execution hints	query execution parameters specifying scanning and optimisation method	precise index-scan, empirical optimisation

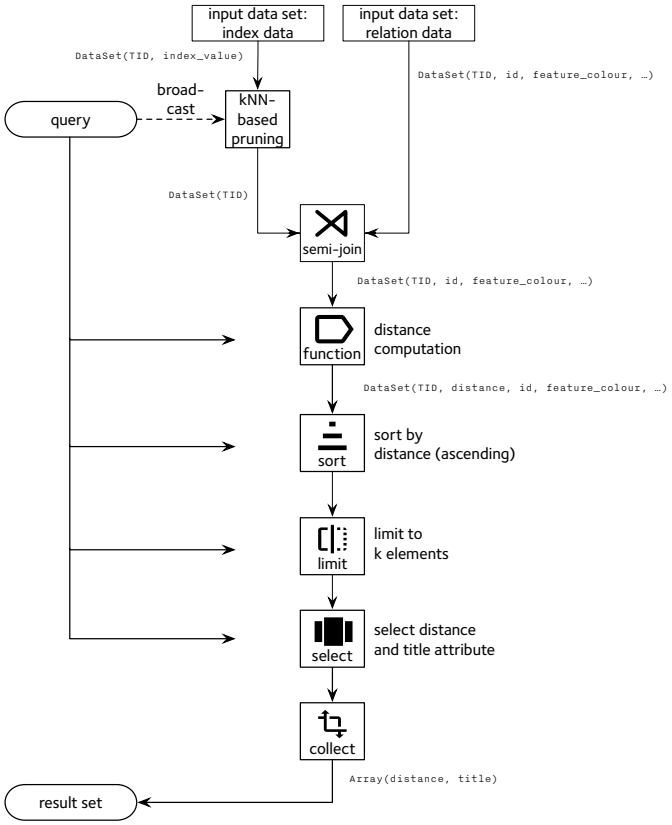
Alice's application formulates the query using a hint-based scan which allows to adjust the query execution based on the given query hints.



In the next step, the query tree is adjusted by the query rewriter based on the query hints. The data scan is resolved into a precise index scan with a subsequent filtered data scan. In the subsequent query optimisation step, the optimiser will decide upon which exact index to use.



Finally, the query plan is translated into a Spark query which contains the explicit operations to use for the similarity-based search within the Spark framework: First, the index returns based on the query a candidate set containing tuple identifiers which may answer the query. This list is semi-joined with the full data set: The tuple identifiers resulting from the index scan are transferred to the data sites and used for filtering the full data. For the remaining data, the final distance from the query to each tuple is computed, the set is sorted by distance and limited to κ elements. Finally, the projection operation is applied and the results are collected at the coordinator to be returned. The specified Spark query is sent to the Spark framework where it is again internally optimised and executed.



6.3 Client Application

To be able to make use of ADAM$_{pro}$, we provide a web-based client which accesses ADAM$_{pro}$ using the previously introduced communication interface. The web client not only allows to create, inspect and delete entities and indexes, but also supports performing queries by graphically setting up, adjusting and running ADAM$_{pro}$ query plans.

Similar to [BCD+09], a user can visually set up a query plan (see Figure 6.7 and Figure 6.8) by dragging the corresponding boxes to a canvas and by connecting these using arrows. Moreover, the available scan methods (including sequential data scan, scan methods

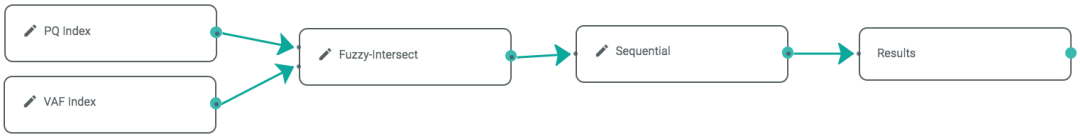


Figure 6.7 Exemplary query scaffolded using the visual query composer of ADAM$_{pro}$: The query is composed of querying two indexes, i.e., a Product Quantisation and a Vector Approximation-File index. The results of the indexes are intersected using a similarity-based intersection. Subsequently, a data scan with the results of the index scans is performed. Finally, the results are returned to the user.

for each of the available index structures, selection of an index structure based on the cost-based or the empirical optimiser, scan methods for external search system such as Apache Solr, etc.) or aggregation methods (e.g., union, intersect, except and the corresponding fuzzy operations) can be added as boxes on the query canvas. Hence, a user can choose, for instance, which index structures to apply in which order, how to combine the results, or which data partitions to consider and which partitions to ignore. Depending on the index structures used, additional options can be passed for scanning. With that, a user is able to fully specify both the logical and the executional query parameters of a query.

The query tree is sent to ADAM$_{pro}$ and executed as specified by the user. After termination, the query plan is displayed together with the query times and the number of retrieved elements per execution step. The user can then modify the query plan by adding further scan types or aggregation boxes in the user interface. By clicking on the boxes, the results of the single boxes are displayed together with provenance information (partition, source, etc.) in the lower bottom of the screen. By this, a user can track how a result tuple has come to appear in the final results, for example, by which scan method or from which partition.

Moreover, in the progressive view (see Figure 6.9), the user has the possibility to execute progressive queries and trace the query through the system. In this mode, all intermediate results are returned as they are processed by the system. The incoming intermediate results are displayed on a timeline, allowing the user to inspect the time of arrival of a query result. Again, the user has also the possibility to inspect the results returned by the system.

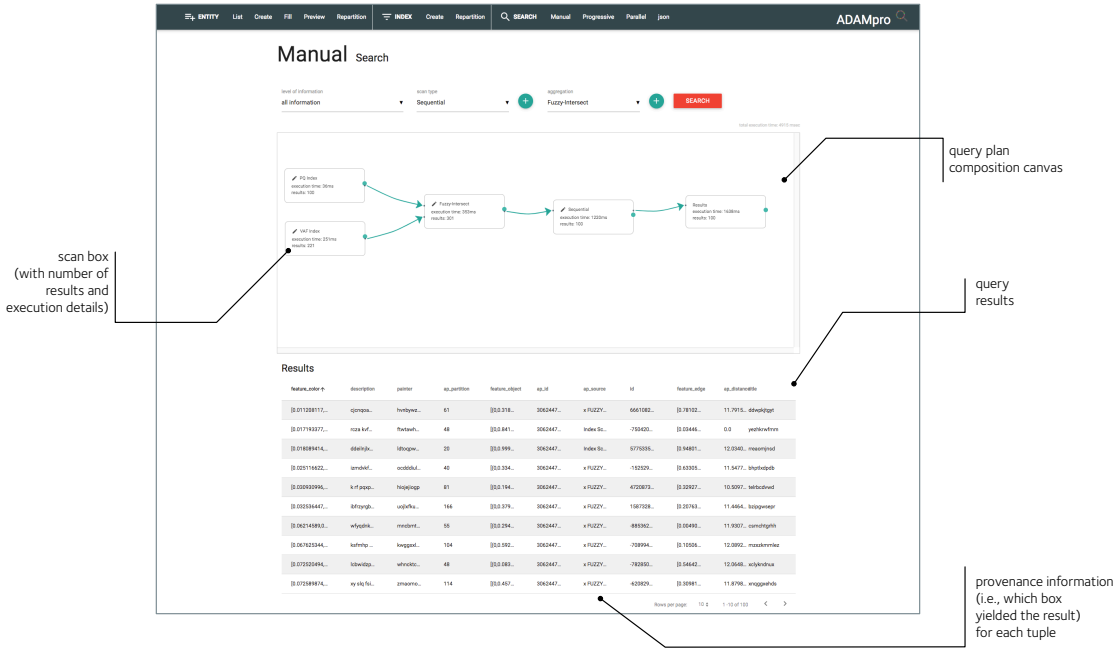


Figure 6.8 **Screenshot of the query composing view in the ADAM$_{pro}$ client:** The view allows the user to manually scaffold a query plan and retrieve and inspect the results. With the incoming results, also the provenance of the tuple (i.e., which scan box provided the result) can be retrieved. A user clicking on a scan box may retrieve the intermediary results.

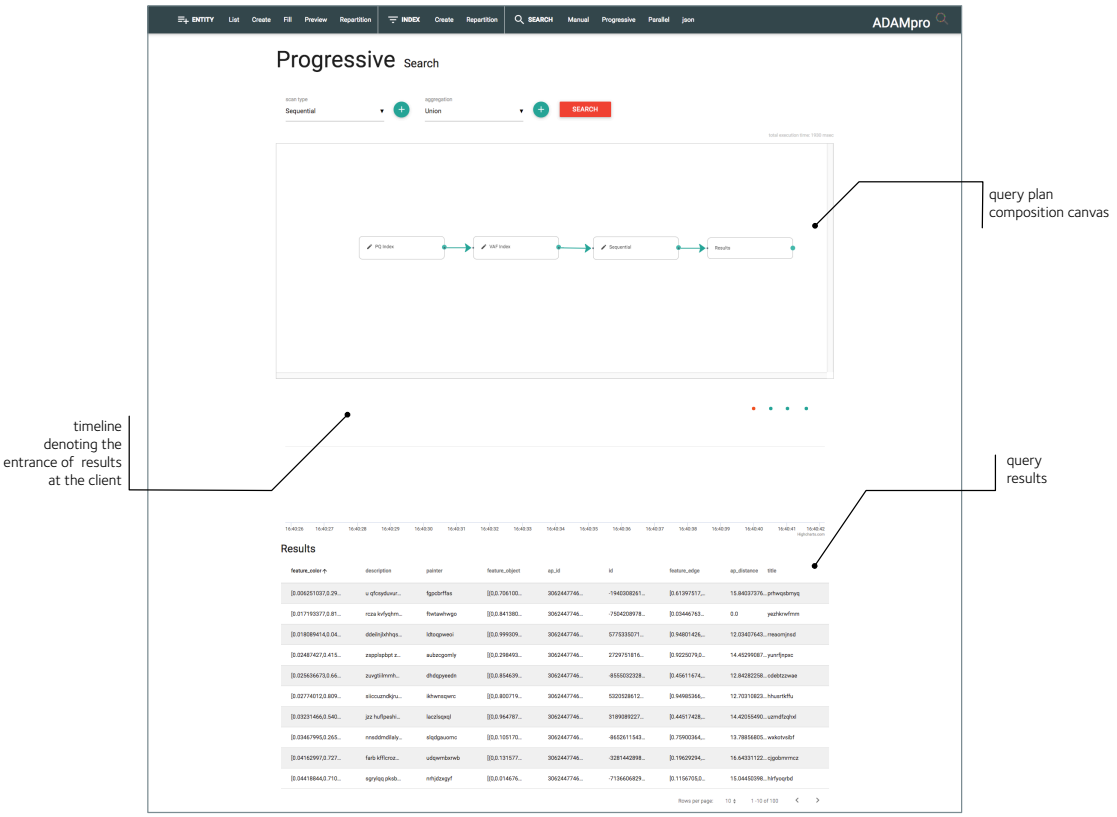


Figure 6.9 Screenshot of the progressive search view in the ADAM$_{pro}$ client: The view allows the user to retrieve results progressively and to see on a timeline the moment in time of a result set arriving at the client.

Part IV

Discussion

With four parameters I can fit an elephant, and with five I can make him wiggle his trunk.

— John von Neumann

7

Evaluation

In this chapter, we present the evaluation results of our implementation within the multimedia data management system ADAM$_{pro}$ which is based on the previously introduced concepts.

From a *qualitative* perspective, ADAM$_{pro}$ has been used successfully within the iMotion/vitrivr stack at multiple instances of the Video Search Showcase/Video Browser Showdown[1] competition ([RGT+17b; RGG+18]). The system presented in [RGT+17b] won the Video Search Showcase 2017 competition thanks to the efficacy and efficiency of the full stack. The competition uses as a basis for evaluating the participating systems the IACC.3 video collection drawn from the Internet Archive and available under a Creative Commons license. It hosts approximately 4'600 videos of a size of 144 GB in total and a running time of 600 hours. The pre-processed videos resulting in approximately 300'000 tuples per extracted feature have been inserted into ADAM$_{pro}$ and used during the competition for performing both similarity and Boolean queries with the goal of finding – on the basis of a query video snippet or a textual description – a specific scene from the video collection.

To give a *quantitative* perspective to the evaluation of ADAM$_{pro}$, in the following, we present the results of an experimental evaluation. We assess the quality of the contributions by presenting the retrieval times and qualities

- for different dimensionalities of the feature data and for varying collection sizes,
- for real-world data,
- for queries mixing both similarity and Boolean predicates,
- for the available scanning and executional methods, i.e., for stochastic and parallel scanning, and for the empirical optimiser,
- for the use of different storage engines and of distribution.

[1] `http://www.videobrowsershowdown.org/`

We will first introduce the setup and the measures used for the quantitative evaluation which serve as a basis to assess the system. In the following, the results of the evaluated parameters are presented. We will close the chapter by a summary and discussion of the results. To facilitate the readability of the parameters used in the evaluation, note that we abbreviate the collection sizes by using K to denote thousands and M to denote millions.

7.1 Preliminaries of the Evaluation

7.1.1 General Setup

We evaluate ADAM$_{pro}$ on an Intel Xeon CPU E5-2630 v4 20 core machine with 128 GB RAM (out of which 50 GB are allocated to Apache Spark and ADAM$_{pro}$). The system attaches to two hard disks (HDD) and three solid-state disks (SSD) in a RAID configuration at level zero. The evaluation machine runs Ubuntu 16.04.3 as the underlying operating system; Docker in version 17.05 is used for the deployment of our software creating lightweight virtual containers in which our software is run. While this setup may have a slight influence on the evaluation and the absolute results, in relative terms the deployment using Docker should, however, not affect the retrieval times. Our evaluation environment has been released on Docker Hub using the tag `2.1-Eval`; the corresponding code has been made public on Github[2].

ADAM$_{pro}$ is generally started as an Apache Spark application in stand-alone mode with 20 workers on a single machine (except for the distributed evaluation discussed in Section 7.2.7). A shared folder is specified for storing the data used by the workers to ensure that the measurements do not consider latencies by any underlying application persisting the data, such as HDFS. In Appendix B, we detail the commands and parameters used for setting up the evaluation environment.

We create relations generally composed of an identifier of type `LONG` and a vector of a fixed length (of type `VECTOR`). The data used is randomly generated; for the vector data, we generate uniformly distributed, random numbers. All index structures have been generated with the default parameters set in ADAM$_{pro}$ (see Appendix A). For storing the index structures, we make use of a file-based storage engine using the Apache Parquet format which can be fully read into memory. The data of the relation is stored using Apache Cassandra 3.11, which is also deployed within a Docker container.

While we report our results for varying parameters of dimensionality and collection size, in the general case, we will often display the results for a fixed dimensionality of $dim = 100$ and fixed collection size of $n = 10M$. We run for each evaluation ten queries to minimise

[2] `https://github.com/vitrivr/ADAMpro/releases/tag/v.2.1-Eval`

outlier errors; three runs are executed before the measurements are started, as the first queries will initiate the system and allocate the memory to be used by ADAM$_{pro}$ and, hence, will take longer in the execution.

We have performed all evaluations using the UNIBAS Chronos system (cf. [Vog15]), an evaluation manager which allows to specify evaluation tasks run against the system at hand and to centrally collect the measurements of the evaluation. The specification of an evaluation task is given in XML; a customised evaluation evaluation agent receives the task specification, executes the evaluation job as specified and records the desired metrics. The parameters set for the evaluation using UNIBAS Chronos are specified in more detail in Appendix B.

7.1.2 Performance Metrics

In the evaluation, we employ two metrics to quantitatively assess our approach; the measurements include

- a time-related metric for measuring the time for performing a retrieval from the moment a query is submitted to the system until the results are returned to the client;
- a quality metric for measuring the quality of the results, in particular, in light of approximate index structures and scans employed in similarity-based retrieval tasks.

For the measurement of the retrieval time, we set a maximum retrieval time of 1'000 seconds; after this time, the query is cancelled and not considered in the quality assessment. As noted in [BGR+99], often sequential data scans outperform with respect to the retrieval time special scanning techniques which make use of index structures. Hence, we compare the run times of the various scanning methods to sequential scans as well, and consider the timing for executing a sequential scan as baseline.

While the definition of a time measure is straightforward, the definition of quality metric bears greater ambiguities and leaves room for interpretation. A quality measure in the given context generally is required (based on [WMZ10])

- to *handle non-conjointness* meaning that it should be able to handle incomplete lists of rankings rather than considering the ranking of the full collection to assess the quality of the retrieval. In our experiments we consider indefinite rankings in which the head and, hence, only a small fraction of the entire list, is seen;
- to *weight high ranks more heavily than low rank* as generally high ranks are considered to be of more importance than lower ranks. To motivate this property, consider, for instance, that users often tend to consider particularly (or even only) the first result page of a search engine, rather than all available result pages.

Common measures for comparing sorted lists, such as Kendall's τ, Spearman's ρ or Spearman's footrule, generally do not support non-conjointness. In some cases, for missing elements of a κNN retrieval a value of $\kappa + 1$ is assumed. However, as described in [WMZ10], this is unsatisfactory. The authors note that the concept of correlation as found in Kendall's τ and Spearman's ρ is not useful when applied to indefinite rankings, particularly as not the correlation is interesting to the user, but the interest is rather in the degree of departure from agreement.

In the following, we present three different quality metrics which we employ in our evaluation.

Competitive recall at κ We denote the intersection between a result set $\mathcal{R}$ (e.g., from an approximate index scan) and a given ground truth gt (from a sequential data scan) to a particular query as $\mathcal{R} \cap gt$. We extend this definition to

$$\mathcal{R}_\kappa \cap gt_\kappa \tag{7.1}$$

meaning the intersection at depth κ. The *overlap at depth* κ is, then, given as the cardinality of the intersection, i.e.,

$$X_\kappa := |\mathcal{R}_\kappa \cap gt_\kappa| \tag{7.2}$$

The *competitive recall at* κ (or agreement at κ) [CPR+07] denotes the cardinality of the intersection normalised by the number of elements to retrieve κ, i.e.,

$$CR_\kappa(\mathcal{R}, gt) := \frac{|\mathcal{R}_\kappa \cap gt_\kappa|}{\kappa} \tag{7.3}$$

Note that competitive recall at κ penalises errors in the result set independent of the positioning within the list. The metric, therefore, only considers whether en element appears within the set of κ elements or not, but not its positioning. Hence, an element missing in the first rank will result in the same penalisation as an error at rank κ.

Average overlap To overcome the drawbacks of competitive recall at κ, the *average overlap* measure weights high ranks more heavily than low ranks by considering each position of the ranking separately. Hence, the metric yields different results depending on whether an element is wrongly positioned within the result set or not; moreover, an error in the first ranks will have a greater impact on the score than an error made at position κ.

The average overlap is defined as

$$AO_\kappa(\mathcal{R}, gt) := \frac{1}{\kappa} \sum_{r=1}^{\kappa} \frac{|\mathcal{R}_r \cap gt_r|}{r} = \frac{1}{\kappa} \sum_{r=1}^{\kappa} CR_r(\mathcal{R}, gt) \tag{7.4}$$

Rank-biased overlap Starting from the definitions introduced above, we consider an evaluation measure which is weighted and indefinite [WMZ10]

$$SIM(\mathcal{R}, gt, \boldsymbol{w}) := \sum_{r=1}^{\infty} w_r CR_r(\mathcal{R}, gt) \tag{7.5}$$

with $\boldsymbol{w}$ being a vector of weights and w_r the weight at position r. Hence, we note that $0 \geq SIM \geq \sum_r w_r$. For a convergent $\boldsymbol{w}$, the expression converges as well. Setting the elements of $\boldsymbol{w}$ to $(1-p)p^{r-1}$, with p being a free parameter, results in a geometric sequence.

The *rank-biased overlap* (*RBO*) is defined in [WMZ10] as

$$RBO^{\infty}(\mathcal{R}, gt, p) := (1-p) \sum_{r=1}^{\infty} p^{r-1} CR_r(\mathcal{R}, gt) \tag{7.6}$$

with $p \in [0, 1)$. For $p = 0$, only the top-ranked item is considered and the evaluation of the *RBO* yields either zero or one. With p getting close to 1.0, the weights become flat and the evaluation arbitrarily deep. The *RBO* measure falls in the range $[0, 1]$ with 0.0 denoting two fully disjoint sets, while 1.0 denotes two identically ordered lists.

The definition of the *RBO* metric given in Equation 7.6 obviously requires an infinite ranking (or at least of all available elements of the collection). In the following, we present a more practical definition of the metric which extrapolates from the visible list to a measure which assumes that the agreement up to κ is continued indefinitely [WMZ10]. [WMZ10] defines the extrapolated RBO^{ext}, with σ denoting the minimum cardinality (i.e., the depth at which one of the two lists has no more elements) and λ the maximum cardinality (i.e., the maximum depth at which the elements are unseen in both lists) of the two compared list,

$$RBO^{\text{ext}}(\mathcal{R}, gt, p) := \frac{1-p}{p} \left(\sum_{i=1}^{\lambda} \frac{X_i}{i} p^i + \sum_{i=\sigma+1}^{\lambda} \frac{X_\sigma (i-\sigma)}{i\sigma} p^i \right) + \left(\frac{X_\lambda - X_\sigma}{\lambda} + \frac{X_\sigma}{\sigma} \right) p^\lambda \tag{7.7}$$

where X_κ denotes the overlap at depth κ as introduced in Equation 7.2. Obviously, given that λ denotes the maximum cardinality of the two lists compared, the maximum value of X_λ is σ. We refer to [WMZ10] for more details on the *RBO* measure. In the following, for reasons of readability, we will use *RBO* to mean the extrapolated rank-biased overlap as presented in Equation 7.7.

7.2 Results of the Quantitative Evaluation

7.2.1 Evaluation of the Effect of Collection Size in Similarity Queries

In the following, we analyse the retrieval time and quality for simplistic nearest neighbour ($\kappa = 100$) queries posed on a relation storing feature vectors of dimensionality $dim = 100$. While the data of the relation is persisted using Apache Cassandra as storage engine, the index structures are stored in Apache Parquet files.

Figure 7.1 displays a plot of the query time for the implemented index structures and the sequential scan in comparison for an increasing collection size. While it is obvious that the retrieval time increases with the size of the collection, we want to highlight other important findings of this evaluation.

First, note that the increase in retrieval time is not always linearly dependent on the collection size. While the linearity is particularly visible in the sequential data scan, note that the retrieval times of the PQ, SH, VA-File and VA$^+$-File indexes imply a sub-linear increase in the times. On the other hand, consider the results for the CP index: Especially for a collection size of 5 million, the spread of the box plot denoting the first and third quartile shows very well the symptomatic behaviour of the index (note similarly for a collection size of 10 million the outlier with a retrieval time of below 20 seconds) which may result – depending on the (random) choice of representatives and the number of assigned points to representative – to very varying results in retrieval time. A different, more favourable, selection of representatives at creation time of the index may have yielded very different retrieval times. Hence, the prediction of retrieval time for a CP index is very difficult and much dependent on the index creation and the queries.

Figure 7.1 also shows that certain index structures, namely LSH and MI-File, at least in their implemented forms, are slower than a sequential data scan. This observation has also been made in [BGR+99]. For LSH, this behaviour is also confirmed in [JDS11], where the authors note that the standard implementation of LSH may consume more memory than the original vectors. Considering the fact that the work is not calculus-driven but rather data-driven [Ams14, p. 82], loading and scanning the LSH signatures may take more time than simply scanning the full data. For the MI-File index, we suppose that the suggestions made in [AS08] to use an inverted file for storing the data are crucial in achieving low retrieval times. As our implementation, however, does not employ inverted files, the index is not able to compete against the sequential scan.

In general, the retrieval times show that with the PQ index retrieval times below 5 seconds are achievable for up to 50 million elements. For precise results (using a Vector Approximation-File index), the collection size may be slightly below 10 million elements. Note that in comparison to the IACC.3 collection used in [RGT+17b; RGG+18] of 600 hours

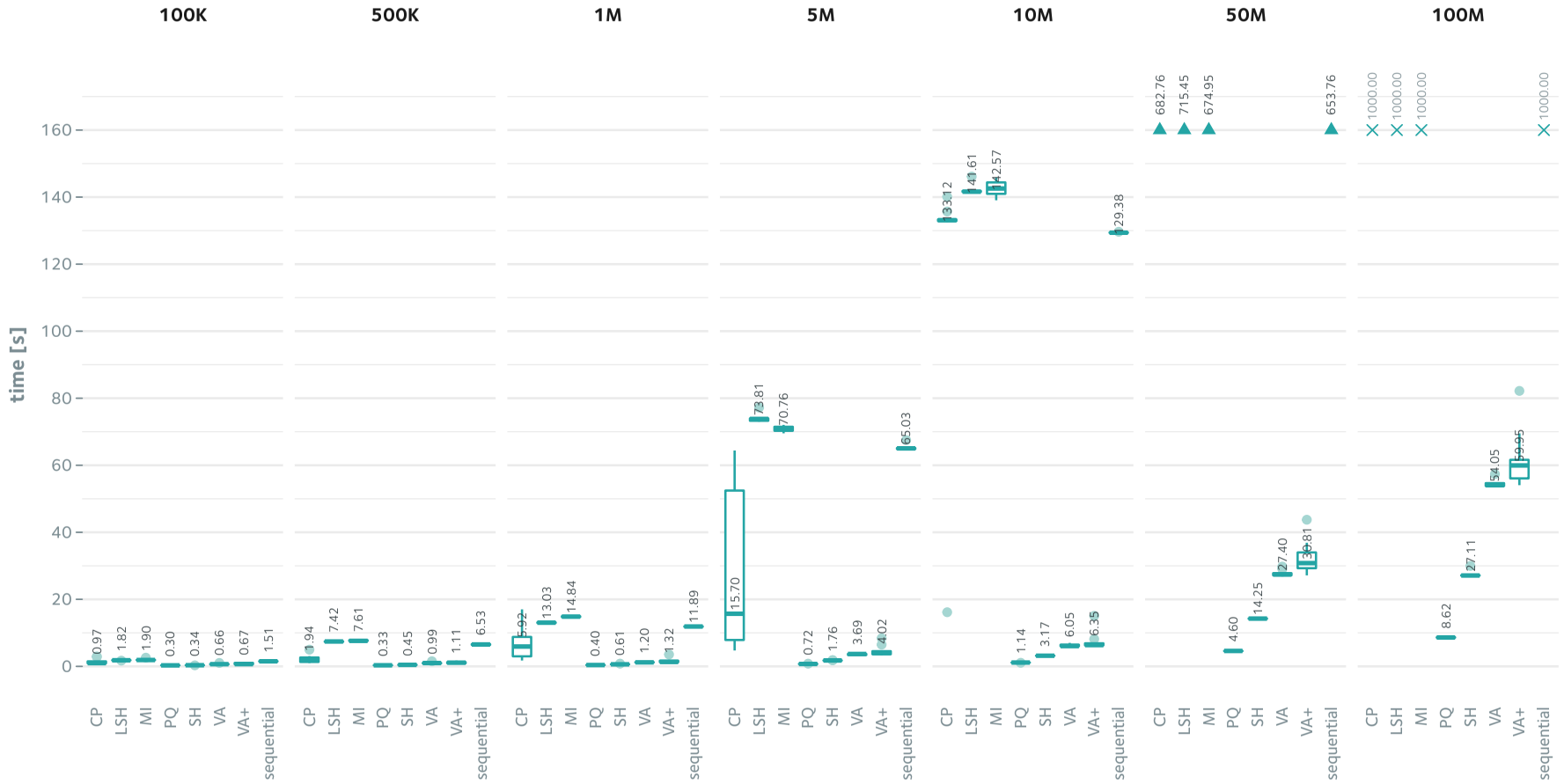


Figure 7.1 **Box plot of query time for existing scan methods at varying collection sizes:** The plot shows an increase in retrieval time (in seconds) for increasing collection sizes (for a fixed dimensionality of $dim = 100$). The median is printed within the plot. Note that queries are stopped after 1000 seconds (denoted by ×). We only display an excerpt of the plot within the boundaries $[0, 150]$; for retrieval times outside the boundaries (denoted by ▲), we only print the median value.

which results in 300'000 tuples, we may extrapolate that 10 million tuples corresponds to roughly 20'000 hours of video. The possibly largest collection available for research purposes and currently explored in the research community consists of about 8'100 hours of video [Mul16].

In Figure 7.2, we display the quality of the retrieval given by the various metrics introduced previously. The values in the plot are given by the median of the measured scores. For illustrative purposes, we additionally display the mean value ($\diamond$) between the quality metrics displayed. The quality measures show for certain index structures to some extent a rather large spread. This means that an index may possibly yield good results in the first positions, however, comparably bad results towards κ elements (or vice-versa).

Generally, for the precise VA-File and the sequential scan the quality is obviously always at 1.0 for all quality metrics. While the retrieval quality for the PQ index generally drops heavily for increasing collection sizes, for the SH index, a decline in quality which is, however, comparably less drastic can be observed as well. This can be explained by the fact that for both index structures the signature size has been fixed at the beginning and not adjusted depending on the collection size.

For LSH, the quality stays constant if not considering the *RBO* metric at low ranks: LSH may return acceptable results in the big picture while possibly missing out on a few elements of the result list in the lower ranks. Consider, for an explanation, Figure 7.3 displaying the distribution of retrieval qualities for n = 10M elements over the single runs, using the various quality metrics introduced previously. Particularly for the *RBO* metric at low p values, i.e., with a focus on the lower ranks, the spread is comparably large. In comparison to the *RBO* value for larger p values, this hints at the fact that in certain cases the top results were missed, while over the whole set of κ elements returned to the user, the quality was on average similar. The implication of this behaviour might of course be that the index structure does not always yield predictable result qualities; on the other hand, it might also be a side effect of the quality measure, in particular when using the *RBO* metric at low p values, which proves not to be stable enough.

Nevertheless, note that low quality measures do not generally make an index structure useless. Depending on the use case, the quality might still be acceptable to the user. Moreover, it is not to say that, hence, for large collections none of the index structures except VA-File is useful. Note that the results presented in Figure 7.2 show the quality for randomly generated queries which are not necessarily part of the collection. For a one-to-one match (when performing query-by-example and considering very similar elements to the one at hand), the quality may be much higher than reported here. This effect can be noticed when querying for elements which truly exist in the database, for which the competitive recall measure at $\kappa = 1$ yields for all index scans a score of $CR = 1.0$.

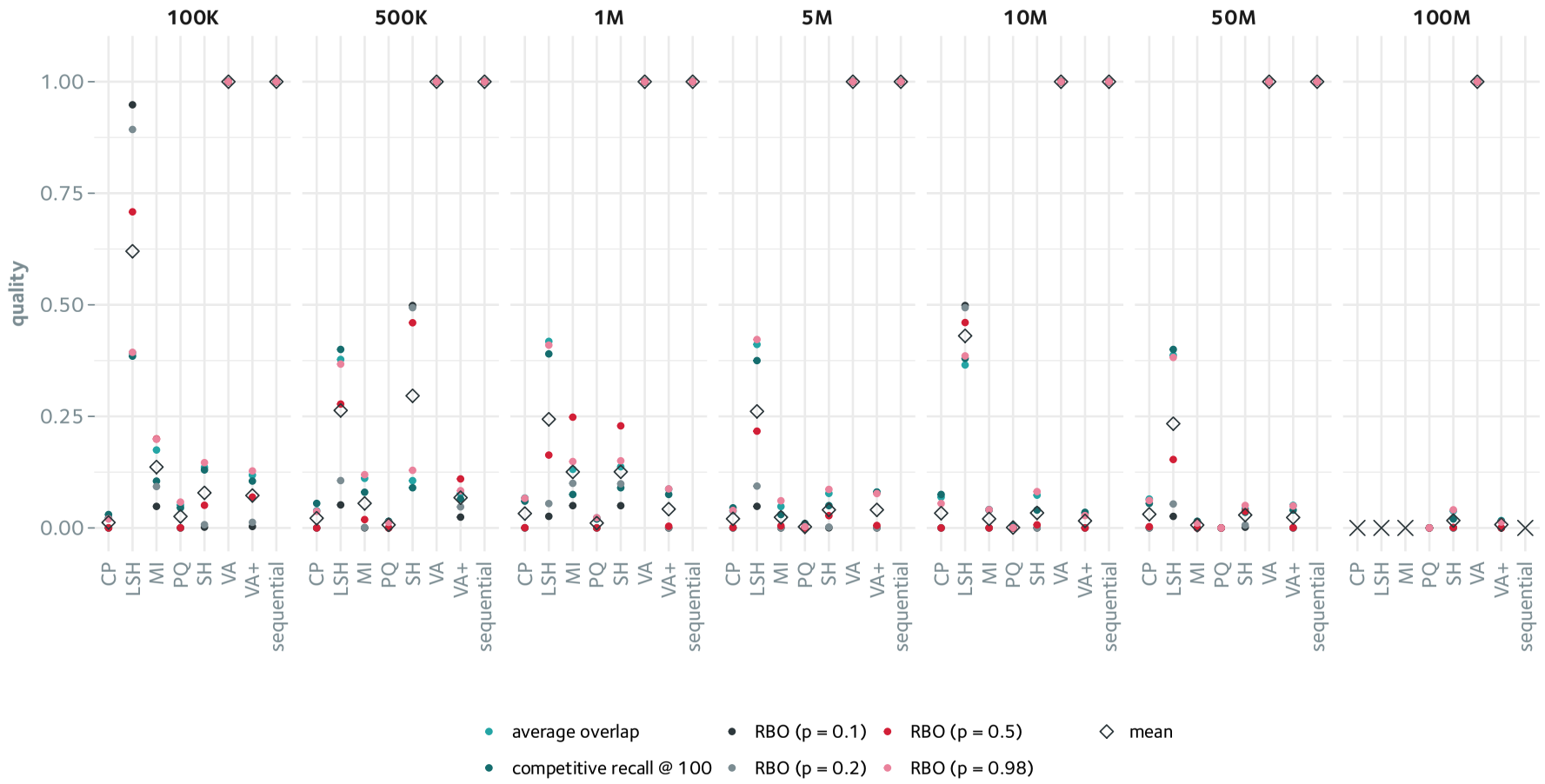


Figure 7.2 **Plot of quality for existing scan methods at varying collection sizes:** The plot shows the median of the various quality measures for increasing collection sizes (for a fixed dimensionality of $dim = 100$ dimensions). The average median quality is printed within the plot as ⋄. Note that queries are stopped after 1000 seconds (denoted by ×).

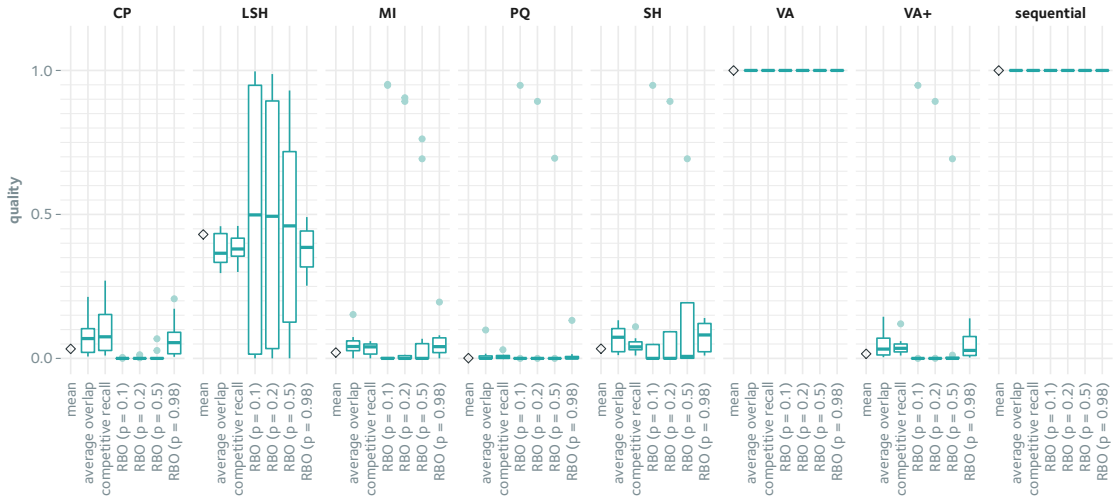


Figure 7.3 Box plot of distribution of quality measures at 10 millions elements: The plot shows the distribution of the quality measures over the single runs for the various scanning methods and the mean (⋄) as plotted in Figure 7.2.

7.2.2 Evaluation of the Effect of Dimensionality in Similarity Queries

The retrieval time for varying dimensionalities in simplistic nearest neighbour ($\kappa = 100$) queries in relations of a fixed collection size of $n = 10$M is considered in Figure 7.4. As previously, the data is stored in Apache Cassandra, while the index structures are stored in Apache Parquet files which can be loaded into memory.

Figure 7.4 hints at which index structures in our implementation use a signature of fixed length and, hence, independent of the dimensionality of the data. More precisely, for the MI-File index and for the PQ index, the time stays constant throughout the various dimensionalities. It goes without saying that, however, while keeping the signature constant, the quality of those index structures drops dependent on the dimensionality as can be seen in Figure 7.5. Considering the qualities for increasing dimensionalities, it would be ill-advised to say that the index structures implemented are not useful at all. For the PQ index, the number of splits and clusters may be increased which may result in (possibly only slightly) higher retrieval times, but also a higher quality in retrieval. Moreover, for (nearly) precisely matching queries, the index structure is still able to yield a precision of 1.0 for finding an existing element of the collection, for instance, in the context of query-by-example.

It is worth mentioning certain peculiarities in the results: First, interestingly, for the VA$^+$-File index at a dimensionality of 10 the query does not stop within the running limit. This behaviour might be an indicator for the fact that, given the uniform distribution of the data and the adjustments made to it (for instance, via a PCA), the VA$^+$-File index will yield at this low dimensionality such a large group of candidate elements from the scan that a large portion of the underlying data has to be accessed for computing the nearest

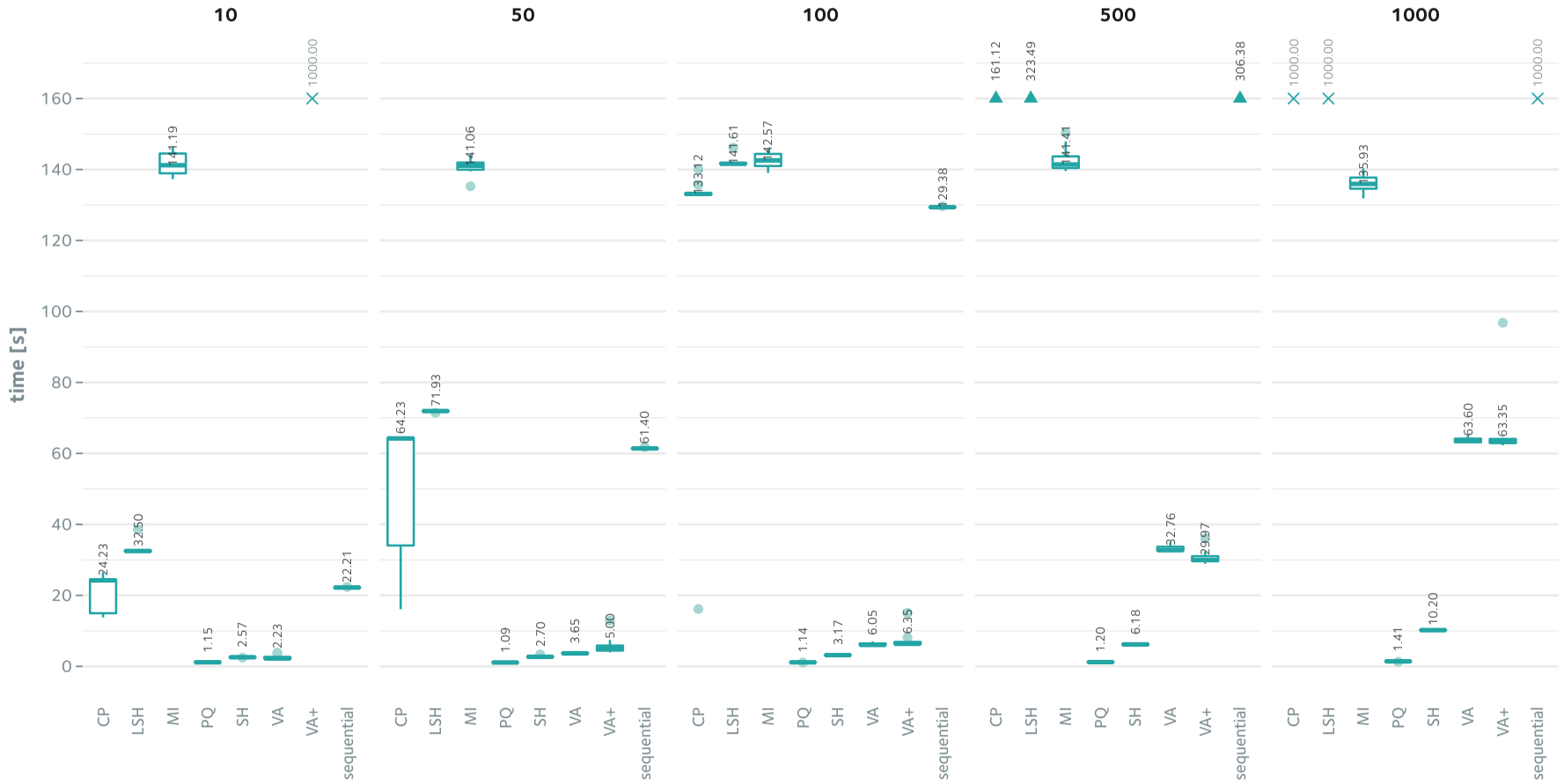


Figure 7.4 **Box plot of query time for existing scan methods at varying dimensionalities:** The plot shows an increase in retrieval time (in seconds) for increasing dimensionalities (for a fixed collection size of $n = 10\text{M}$). The median is printed within the plot. Note that queries are stopped after 1000 seconds (denoted by ×). We only display an excerpt of the plot within the boundaries $[0, 150]$; for retrieval times outside the boundaries (denoted by ▲), we only print the median value.

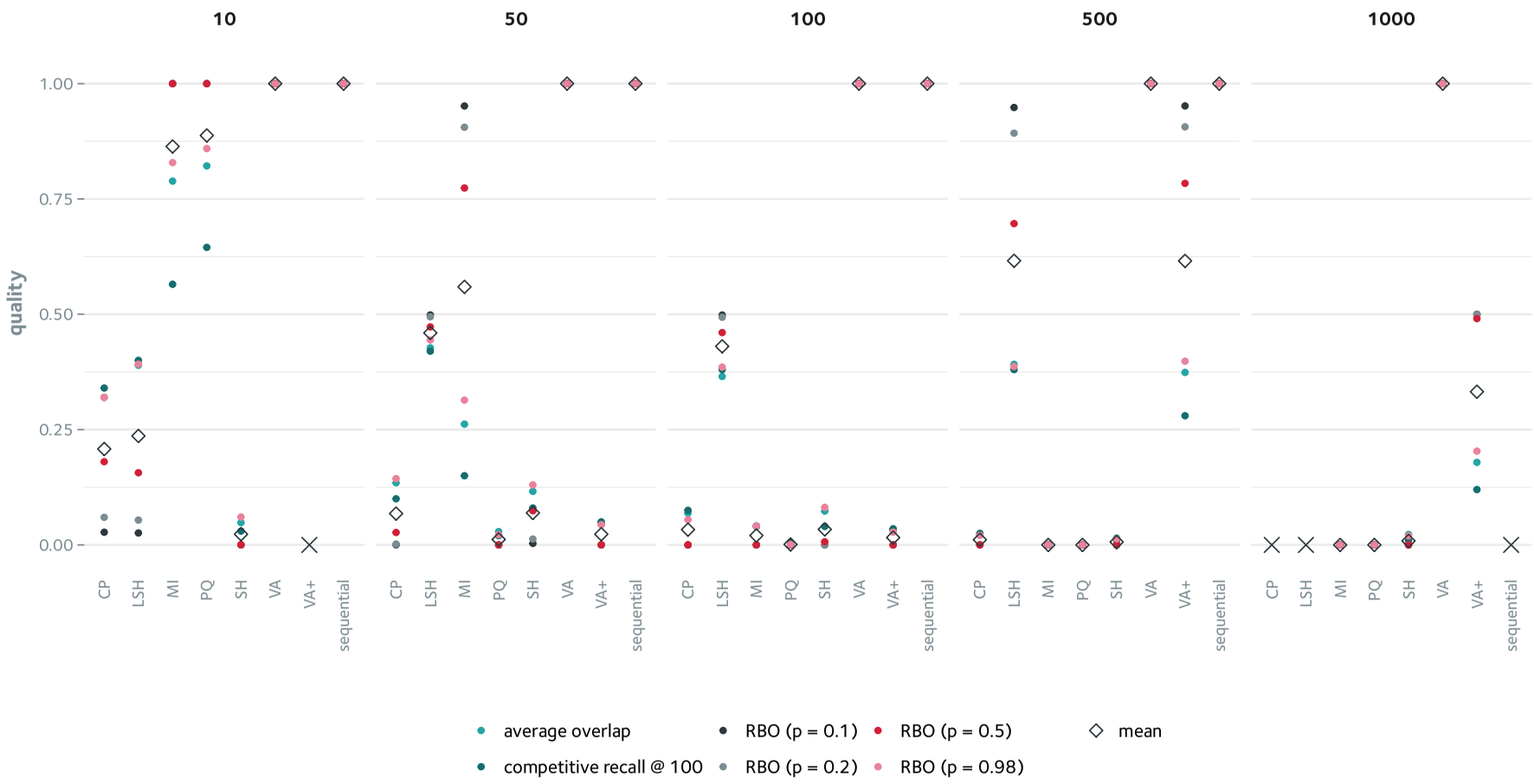


Figure 7.5 **Plot of quality for existing scan methods at varying dimensionalities:** The plot shows the median of the various quality measures for increasing dimensionalities (for a fixed collection size of $n = 10\text{M}$). The average median quality is printed within the plot as $\diamond$. Note that queries are stopped after 1000 seconds (denoted by $\times$).

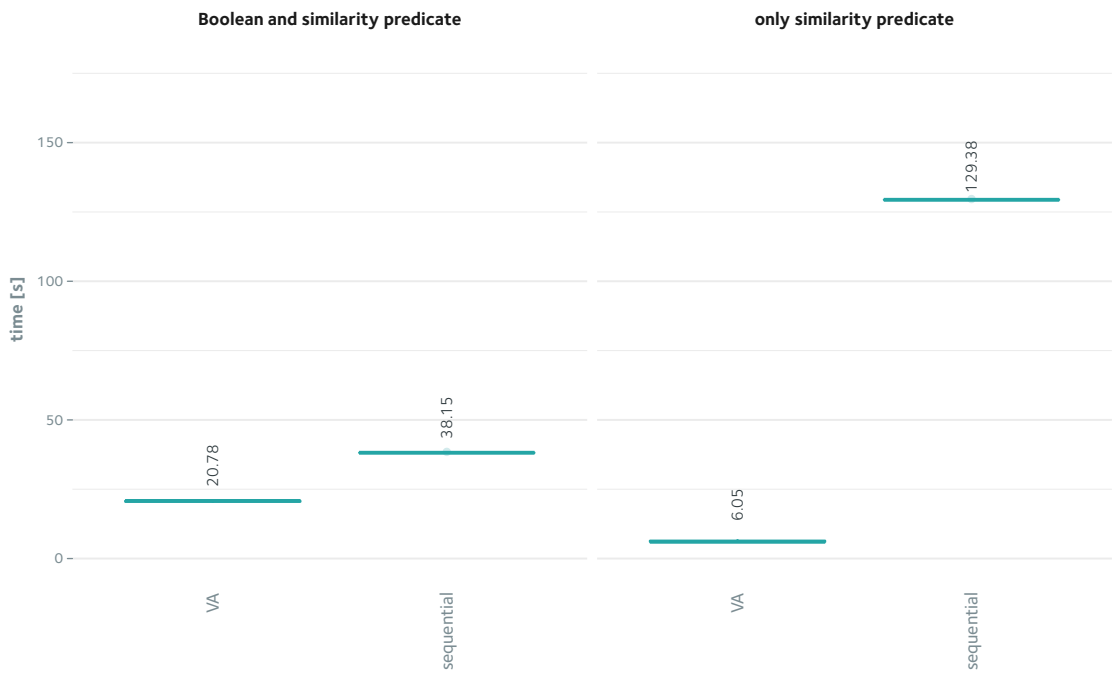


Figure 7.6 Box plot of query time for queries combining Boolean and similarity predicates in comparison to queries which use only a similarity predicate: For illustrating purposes, we display the query time (in seconds) only for a VA-File index structure and a sequential scan. The plot shows that for sequential scans, adding a Boolean predicate heavily reduces the retrieval time as only a sub-set of the data has to be considered. On the other hand, for a VA-File index, the retrieval time increases as the filtering is not able to skip so many elements that the Boolean predicate pays off. Instead, the index scan becomes more complex as for each tuple in the index the Boolean predicate needs to be evaluated.

neighbours. Second, we again point to the behaviour described previously for the CP index which is visible also in Figure 7.4: For a dimensionality of 50, depending on the query, the retrieval time varies greatly. The same is true for dimensionality 500 which is not printed in the plot as it lies outside of the boundaries of the figure. As noted previously, the reason for this behaviour lies in the random selection of leaders which might be not be optimal.

With respect to Figure 7.5, we can note again that the quality generally decreases with increasing dimensionality for the CP, MI-File, PQ and SH index. With LSH, we note an improvement in the results with increasing dimensionality. A reason for this might be that the increasing number of dimensions increases the probability for an element being found through the OR amplification which results in larger lists of candidates and, hence, explains the (partially super-linear) increased retrieval time. Obviously, for the VA-File index and for the sequential data scan the quality is at 1.0. For the VA$^+$-File index, while for low dimensions its power is not visible and the quality is low, for a dimensionality above 500, the quality increases again to comparably high values.

7.2.3 Evaluation using the YFCC100M Data in Similarity Queries

While the content used in our evaluation so far encompassed randomly generated data, in this section we present results of running queries on the YFCC100M [TES+16] collection composed of 100 million (out of which 98'052'262 images) images and videos. The collection is made available together with a set of low-level features, some of which have been computed using the LIRE package [LC08b]. We use the scalable colour feature of dimensionality 64 which denotes a colour histogram in the HSV colour space encoded by a Haar transform.

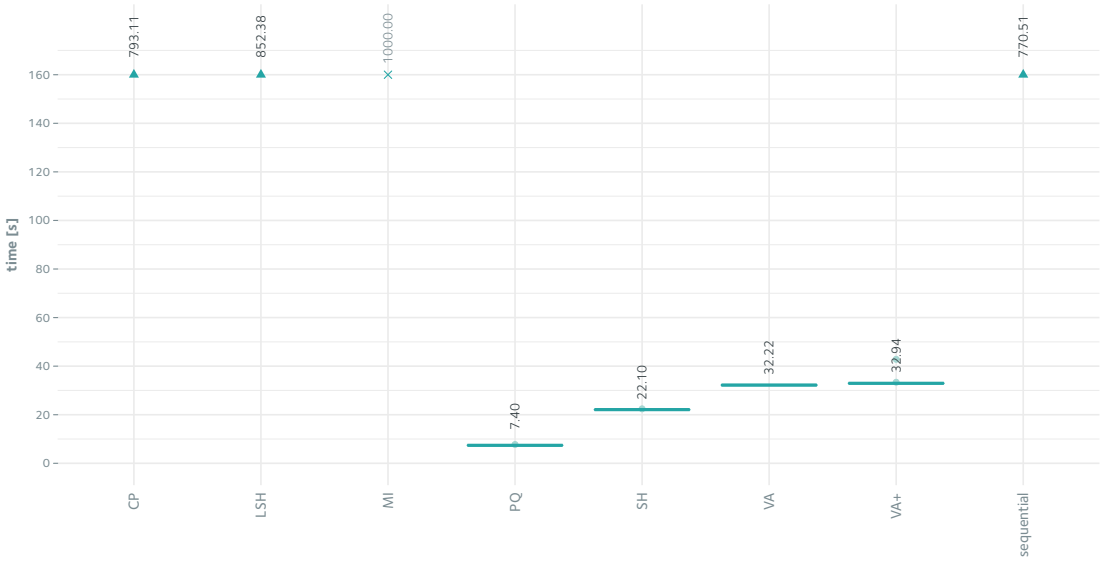
Figure 7.7 presents the results of the evaluation using the YFCC100M data. Obviously, it is difficult to compare Figure 7.7 to the results of Figure 7.1 and Figure 7.4 as both the dimensionality and collection size do not match. We will, hence, refrain from making any statements using the previous evaluations.

Within this evaluation, we note the very low quality for the CP, the PQ and the SH index. As noted previously, for PQ and SH, improvements in the quality may be achieved by adjusting the length of the signature for increasing collection sizes and dimensionalities. Furthermore, an implication imposed by the outcome of the evaluation is that there is no improvement visible for the VA$^+$-File index using real data; instead, the retrieval quality for the VA$^+$-File index is worse than the quality at of the VA-File index at a nearly equal retrieval time for both indexes. A possible explanation for this behaviour might be that the data is not skewed enough to profit from the adjustments made by the VA$^+$-File index. Moreover, the dimensionality of the data used might be too low for the VA$^+$-File index to have an improving effect. Further evaluations would, however, be required to support this statement.

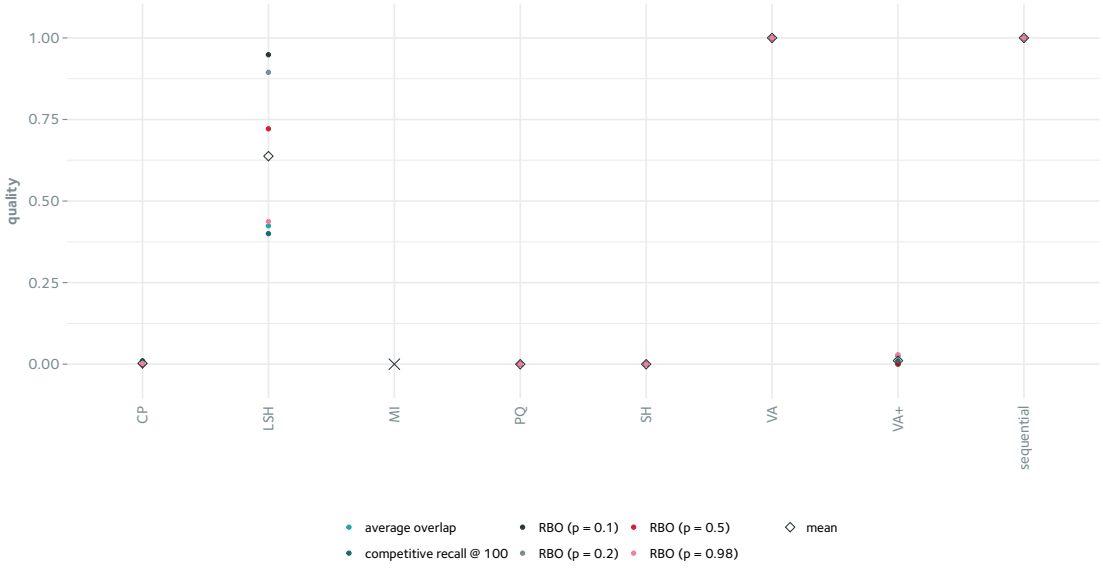
7.2.4 Evaluation of the Effects of Logical Parameters

In the following, we show the results of combining Boolean parameters with similarity queries. For this purpose, we have created a relation of n = 10M uniformly distributed vectors of dimensionality dim = 100. Moreover, the relation stores an attribute of the type integer. The data is based again on randomly generated tuples. For the evaluation, the Boolean predicate selects tuples with an integer value below a randomly selected value.

Figure 7.6 shows the difference in retrieval time for combining a Boolean and a similarity predicate, and for similarity queries only. For illustrative purposes, we only display the retrieval time for the VA-File index and a sequential scan (though similar results are observable throughout). The plot shows that using a Boolean scan, the time for a sequential scan can largely be reduced. Obviously this is due to the fact that the similarity predicate is only evaluated on a much smaller set of tuples. On the other hand, for the VA-File index, surprisingly, the scanning time increases. Foremost is the fact that filtering the index on



(a) Box plot of query time (in seconds) for YFCC100M data.



(b) Plot of quality for YFCC100M data.

Figure 7.7 Plot of query time and retrieval quality for YFCC100M data: The plot shows the retrieval time and the quality for the scalable colour feature of dimensionality 64 for the YFCC100M collection composed of approximately 100 million images.

the results of the Boolean predicate does not seem to pay off for the items which can be skipped in the index scan. Instead, it adds to the work as for every element of the index, the system needs to check whether it should be considered for the similarity predicate or not. Hence, an increase in retrieval time is observable in the plot.

7.2.5 Evaluation of the Use of Executional Parameters in Similarity Queries

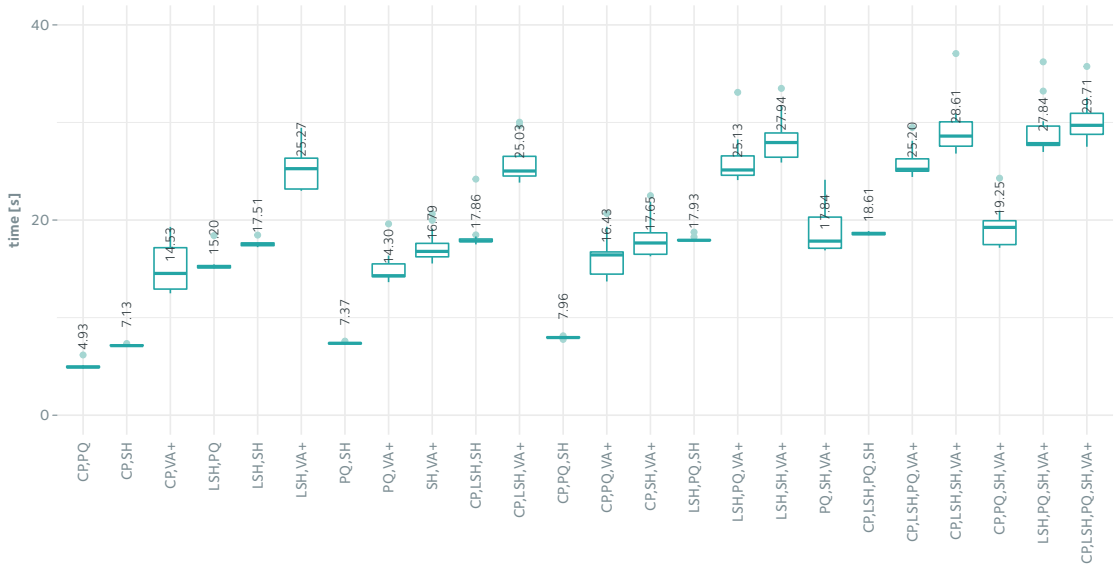
In the following, we present the results of evaluating the available executional parameters. We consider a relation of n = 10M tuples storing only a vector attribute of dimensionality dim = 100. The data for this evaluation have been randomly generated.

Stochastic scanning Figure 7.8 presents the results of stochastic scanning with a subsequent sequential scan on the retrieved items. With stochastic querying, we combine possibly imprecise index structures with the goal of increasing the precision. For reasons of readability, we only display the index combinations for stochastically querying a relation for which the mean time is below the time of sequential scanning, i.e., below 129.38 seconds.

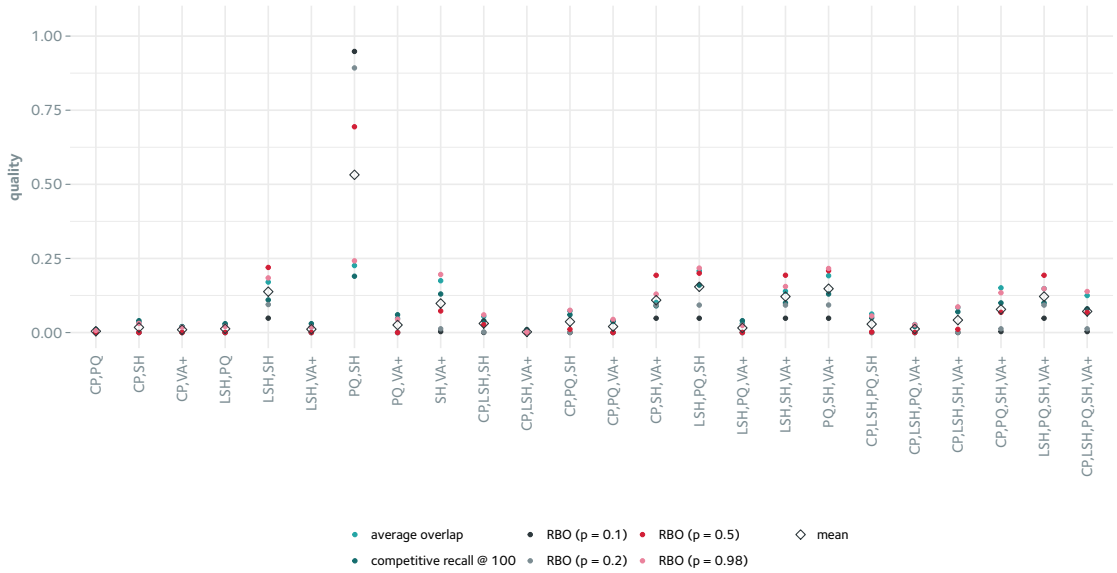
In Figure 7.8, we plot the distribution of retrieval times for a number of combinations of index structures together with a quality plot.

The results show that while our approach is not able to compete against the VA-File index which returns precise results (i.e., with a quality measure of 1.0) within 6.05 seconds on average, the approach may achieve results with a good enough precision in particular for the first ranks (i.e., with a high *RBO* score at a low p value) much faster than a sequential scan. Consider, for instance, the combination of a PQ and a SH index which returns within 7.37 seconds on average results whose *RBO* score for p = 0.1 is at 0.94, and, therefore, close to a precise scan.

More interestingly, the results of the stochastic scan hint at the fact that only scanning an index might be very fast, while it is the access to the data which results in higher retrieval times. Compare, for instance, the scanning time for a CP and a PQ index. At the same collection size, in Figure 7.1, we have shown that scanning a CP index (together with accessing the data), on average takes 133.12 seconds (with an outlier below 20 seconds). Similarly, the plot shows that a LSH index scan (again with accessing the data) takes around 141.61 seconds. However, the combination of LSH and PQ allows to reduce the number of elements for which the tuples of the underlying relation needs to be accessed, ultimately also reducing the query time (to 15.20 seconds for both scans). Considering this further, possibly the use of hybrid index structures which not only store index values and tuple identifiers, but also further data of the relation and, therefore, allow to skip the access to the relation, might be worth considering. First experiments seem to support this hypothesis which should be investigated further in in future research.



(a) Box plot of query time (in seconds) for stochastic scanning.



(b) Plot of quality for stochastic scanning.

Figure 7.8 **Plots of query time and retrieval quality for stochastic scanning:** The plot displays combinations for which the retrieval time using stochastic scanning is below the sequential scanning time at a dimensionality of $dim = 100$ and a collection size of $n = 10\text{M}$.

Parallel scanning and progressive execution With parallel scanning, similar to stochastic scanning, multiple index structures are queried at the same time. However, rather than combining the results of the scans to one single answer, in combination with progressive execution, the results are returned to the user as soon as they are available.

Figure 7.9 exemplarly displays the behaviour of ADAM$_{pro}$ in parallel scanning multiple index structures. The plot has to be read as a timeline which displays the entering of responses to the client. Hence, at time zero, the scanning of all three index structures is started. We use the median time of the scans (the single scans are visible as points in the plot), to illustrate the scanning time. As an overlay to the plot, we print the retrieval quality of the query, which improves as the query goes along.

The plot shows that results are returned to the client on average at 5.01, 8.32 and 9.34 seconds and the retrieval quality increases from 0.0 to 0.035 to 1.0 (for the *RBO* metric at $p = 0.98$). Obviously, with parallel querying, we cannot guarantee that the result quality always improves as during querying the ground truth to the query is not known. To expound on this topic, an index may return after another and take much longer for scanning, but return results whose quality is worse than the previous results. In our implementation, the index structures have a confidence score attached which avoids that precise results are overwritten by an imprecise scan. Within the category of approximate indexes, however, we make no difference between the available structures.

Parallel scanning in combination with progressive querying is mostly useful for large collection sizes where the retrieval time is comparably long anyway. Comparing Figure 7.9 to Figure 7.1 shows that our implementation of parallel and progressive scanning has room for improvement; consider, for instance, the median retrieval time of the PQ index in a standard scan which is at 1.14 seconds, compared to the time at which the PQ index returns in the parallel scan which is at 5.01 seconds. We have analysed this behaviour and have noted that a significant part of it comes from scanning the data in Apache Cassandra for determining the final distance values for the candidates coming from the indexes. Scanning the data is necessary, as the index scan will otherwise only be able to return an internal tuple identifier and distance bounds. Storing a minimal set of required information within the index may reduce the retrieval time – allowing to scan only the index – and be able to return useful information to the user. This idea has been discussed already with stochastic scanning, and first experiments making use of hybrid indexes with a progressive execution hint at the usefulness of such an approach.

Empirical optimisation In the following, we consider the empirical optimiser and present the results of the experimental evaluation in a simplified setting. The empirical optimiser allows to optimise a query execution plan by choosing a specific index structure to use for

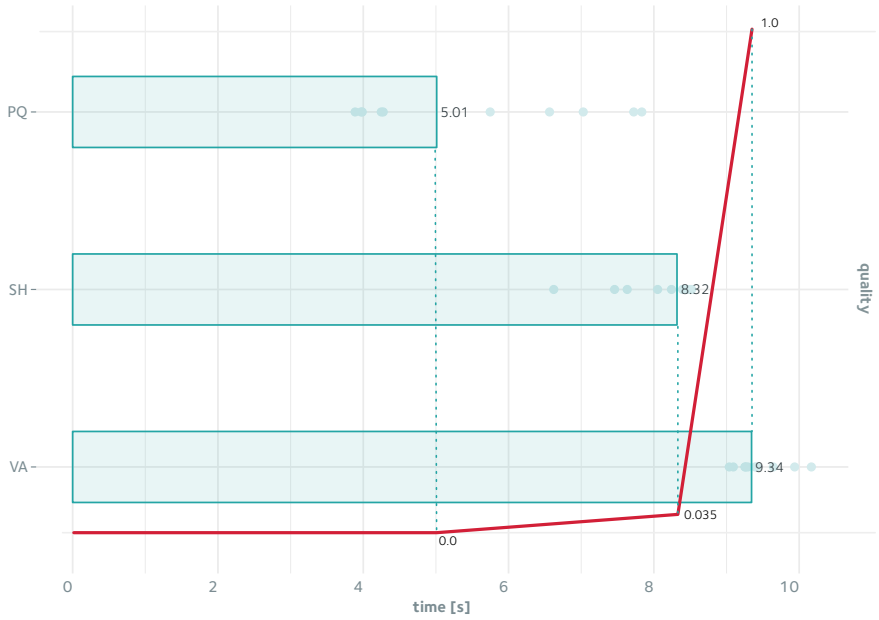


Figure 7.9 Plot of query time and retrieval quality for parallel scanning: The plot displays the retrieval time for combining a PQ, a SH and a VA-File index scan in a progressive manner for a collection of n = 10M at a dimensionality of dim = 100. We consider the median time for the plot, while the single measurements have been added to the plot as well. The red line denotes the median retrieval quality using the *RBO* metric with p = 0.98.

evaluating a similarity predicate based on previously recorded query execution times.

In our setting, the optimiser was trained beforehand on collections of 100K, 500K, 1M, 5M, 10M, 50M, 100M elements for a dimensionality of dim = 100 and for a collection size of 10M for 10, 50, 100, 500, 1000 dimensions. In the training, queries were stopped at 150 seconds and the maximum time was recorded.

Figure 7.10 displays the estimated query time and the difference from the estimated time to the median true execution time. In more detail, a positive value (marked with a green dot) means that the optimiser overestimates the execution time, a negative value and a red dot, on the other hand, denote that the optimiser underestimates the execution time.

There are certain obvious wrong estimations, for instance, for VA$^+$-File for collection sizes below 10M, the estimated retrieval time is always too high. This can possibly be explained by the fact that the training on dimensionality of 10, as shown in Figure 7.4, has comparably strongly influenced the estimated execution time for the VA$^+$-File index, such that the query does not stop within the time boundaries in the estimation. On the other hand, for the other indexes, in particular for the PQ index and the SH index, the estimated retrieval time is only slightly off from the true retrieval time (except for 100M items).

We note that our evaluation obviously only considers a simplistic case in which only one similarity predicate is used within a basic query. The evaluation of complex queries

Figure 7.10 Plot of the time estimation given by the empirical optimiser: The plot displays the time estimations returned by the empirical optimiser; in the background the truly measured retrieval time is shown. Retrieval times and estimations outside the boundaries are denoted by ▲. A positive value and a green dot in the plot mean that the empirical optimiser overestimates the true execution time; a negative value and a red dot denote an underestimation of the retrieval time.

combining multiple similarity predicates has not been considered in here. For such queries, the combination of cost-based and empirical optimisation might prove to be a successful strategy. We leave this, however, for future work.

7.2.6 Evaluation of the Use of Various Storage Engines in Similarity Queries

We consider again a relation with n = 10M randomly generated tuples storing only a vector attribute of dimensionality dim = 100. Figure 7.11 shows the retrieval time for two different storage engines, i.e., for a file-based storage which loads all data into memory. On the other hand, we plot the retrieval time for Apache Cassandra as the underlying storage engine.

As shown in the plot, the retrieval time for sequential scans is much lower using the in-memory storage than using Apache Cassandra. This is explained by the fact that the full data set does not have to be loaded first into memory – passing the gap from secondary storage to main memory – to be processed, but can directly be queried.

On the other hand, for the index structures, the retrieval time is higher when using the in-memory storage. The reason for this observation is that the in-memory storage does not support filtering based on tuple identifiers, but rather requires a full scan of the data to filter out the candidates given by the index. This, in turn, increases the retrieval time, making a sequential scan of the full data faster than an index scan followed by a data scan. This is

ultimately also a consequence of the system implementation using HDFS. Consider, for instance, a VA-File index, which processes a query and returns a list of tuple identifiers of possibly relevant tuples. In the context of a traditional database, the database system would then be able to use the list of tuple identifiers to read only a selected number of database files from disk and only process these files. While the size of database pages in a classical database system is relatively small (a few kilobytes), in HDFS-oriented environments, the files read from disk are for performance reasons of Hadoop generally larger (several megabytes) to overcome the loading coast by mitigating it with the computation-focused costs (see Section 4.4). Hence, there is often no possibility of skipping a file as the probability is very high that every file stores at least one candidate tuple. Hence, as all the files are being read, the gain in performance from using an index comes from skipping certain tuples in the processing of a query and not from not having to read a tuple from disk. However, the index structures presented in Section 4.5 are all founded on the assumption of small database files which are selectively read. Index structures for high-dimensional data which are adapted to the larger Hadoop files are to the best of our knowledge not existent. As a consequence of this fact, in evaluations of multimedia retrieval systems using HDFS – albeit only being limitedly useful – the evaluation tasks are being batched (cf. [SMG+13; MSG+13a; CJN+11]). We leave this topic, therefore, as a matter of future research.

7.2.7 Evaluation of the Distribution Mechanisms

In the following, we consider relations with vectors of dimensionality $dim = 100$ and measure the retrieval time in a fully distributed setting. For this evaluation, we have set up a cluster of four machines (with equal specifications as used so far) on which ADAM_{pro} is running in a physically distributed setting. As underlying file system for storing the indexes, HDFS is used. The machines are connected via a gigabit ethernet connection.

The plot in Figure 7.12 shows in comparison to Figure 7.1 (which has been added to the same figure in brighter colours) that a higher degree of distribution not necessarily improves the retrieval time, in particular for index structures which are not able to expose high data parallelism such as the VA-File. This is also particularly true, as the work performed in similarity-based queries is data-driven, not calculus-driven [Ams14, p. 82]. As noted in [JDJ17], the bottleneck in similarity searches nowadays stems from using structures such as max-heaps/priority queues (see Algorithm 4.2, Algorithm 4.12, etc.) which cannot saturate the high parallelism possible, for example, in GPUs or in highly distributed systems. For instance, in our implementation of the VA-File index, each node will collect its κ nearest neighbours using a priority queue and send the partial result sets to the master node where they are combined. Hence, a higher degree of distribution will result in more candidate points to be merged and possibly more elements to be scanned from the underlying relation.

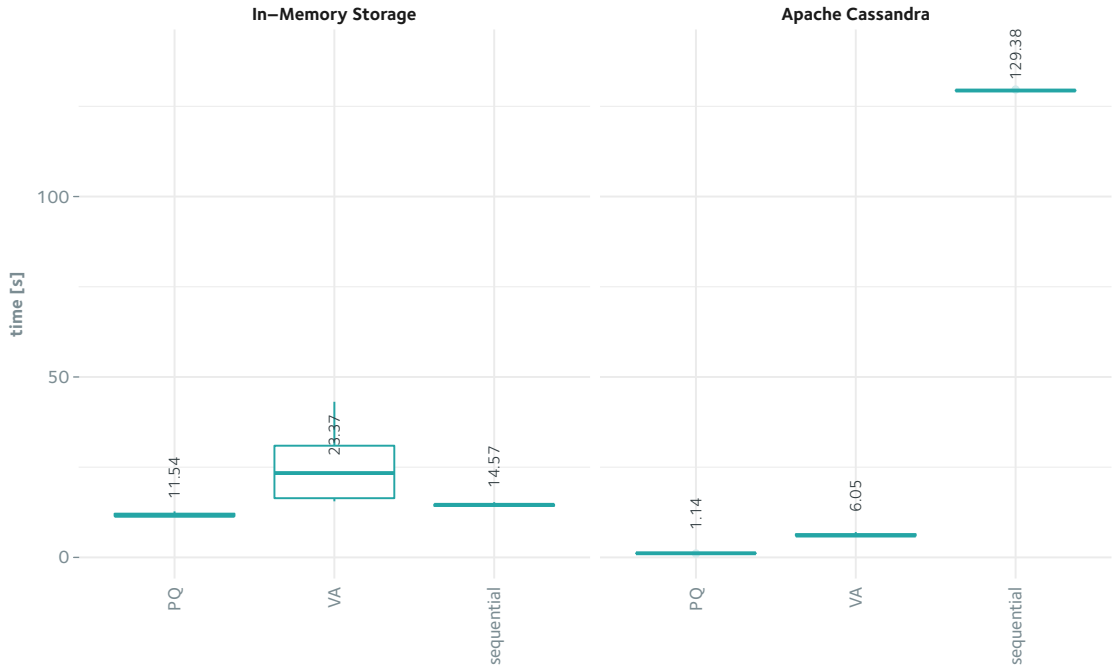


Figure 7.11 Box plot of query time for two different storage engines: For illustrating purposes we display the query time only for a PQ, a VA-File index structure and a sequential scan. The plot shows that the in-memory storage engine is able to heavily improve the sequential scan given that the data is held in memory. However, for the index structures, an increase in retrieval time can be observed, as accessing the underlying data in the in-memory storage does not make use of any key-based accesses.

This explains the increased retrieval times for the VA-File index. Similarly, the same applies for all indexes which we categorised as performing an exhaustive search, i.e., the SH index and the PQ index. While the MI-File index also belongs to this category, some improvements can be detected in the distributed setting in certain cases (namely for 50M and 100M elements). This is possibly due to the fact that the computation of the nearest neighbours with respect to the surroundings for such large collection sizes is computationally bounded and, hence, can profit from the higher degree of distribution. On the other hand, the results for the sequential scan, obviously, improve with the higher degree of distribution to such a degree that the sequential data scan for 100M tuples has a lower retrieval time than by using a VA-File index.

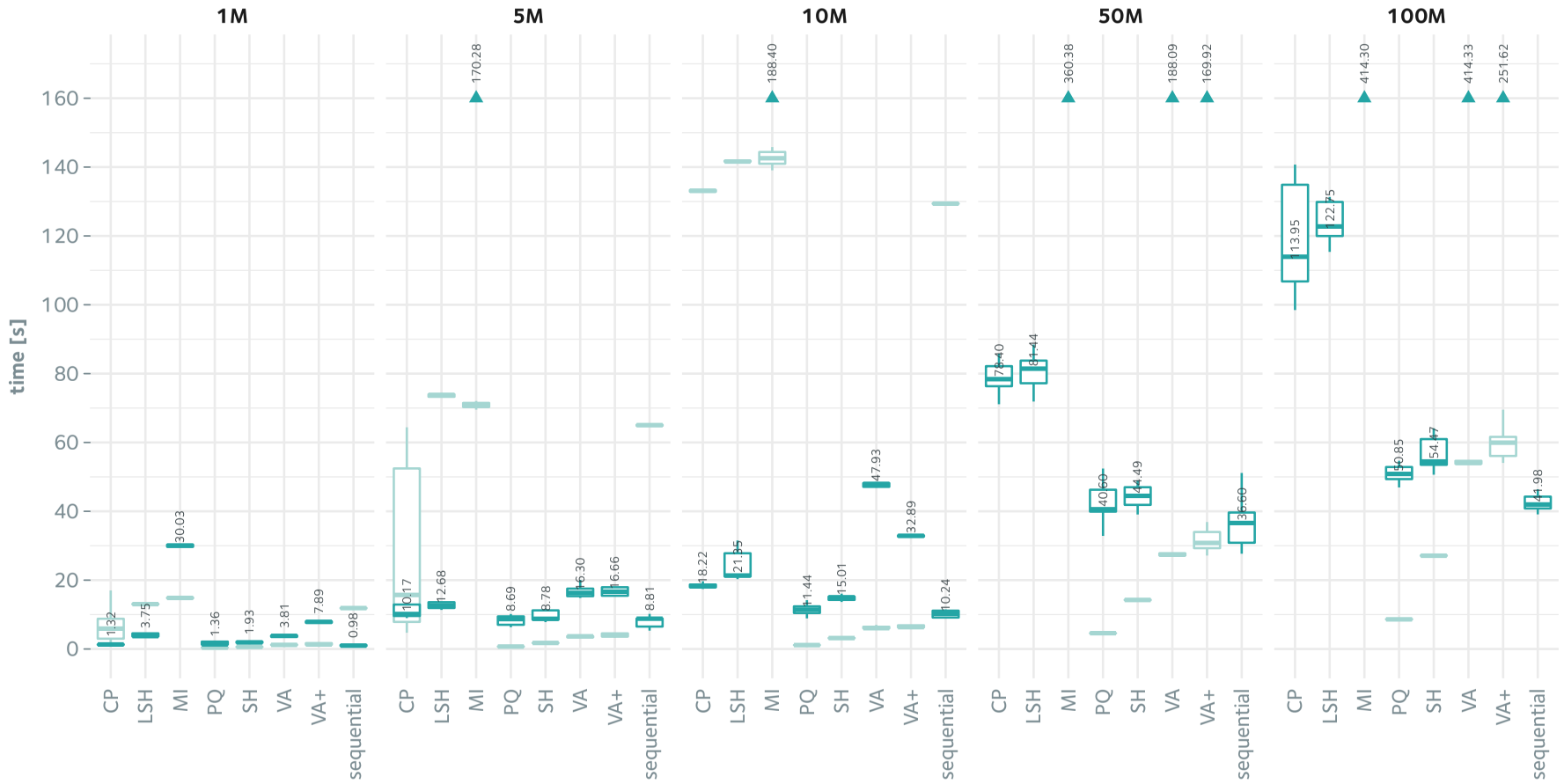


Figure 7.12 **Box plot of query time in a physically distributed setting:** The figure plots the query time in a distributed setting using four physically independent machines. Retrieval times outside the boundaries are denoted by ▲. The plot shows that a higher degree of distribution not necessarily improves the retrieval time, in particular for index structures which are not able to expose high data parallelism, but instead result in a larger set of candidate points to be scanned.

7.3 Summary and Discussion

This section briefly summarises and discusses the results presented in this chapter from a holistic point of view.

With these evaluations, we have given a first notion of the retrieval times and qualities for varying collection sizes and dimensionalities. Stating the obvious, with increasing collection sizes and dimensionalities, the retrieval time increases. Attempts to maintain the retrieval time constant, of course, have an effect on the retrieval quality. This evaluation has also presented comparable measurements of both the retrieval time and quality for a large variety of index structures for similarity-based searches in high-dimensional spaces. With this evaluation, we provide a means to compare index structures which have so far only been considered in isolation within incomparable systems.[3]

In this evaluation, we have considered results for collection sizes of up to 100M tuples and have reported the results both in terms of time and quality. To put the collection sizes into perspective, we note again that the Video Search Showcase/Video Browser Showdown uses a collection of a running time of 600 hours resulting in 300'000 tuples per feature. There has only been limited research on large collections such as the YFCC100M collection as the processing of such large collections is not trivial. For this reason, the YFCC100M collection is available together with a large set of features extracted from the collection already. Considering the evaluation measurements and given the machine setting described, our system is able to handle collection sizes of 100M (and possibly larger) with retrieval times of below 55 seconds to retrieve precise results. As stated in the introduction in Section 1.2, retrieval systems in the context of multimedia retrieval generally do not employ any particular data management system for storing their data; differently said, they are confronted with scanning their data in a non-distributed, sequential way. For collections of this size, our numbers are promising in that collections of 100M can be handled in reasonable time, while at the same time also handing over the responsibility for the organisation and the management of the data to a dedicated data management system.

With respect to the query parameters, we have shown the effect of the retrieval time when mixing Boolean and similarity predicates both in a sequential scan and using an index structure. On the executional parameters, stochastic scanning has shown a certain applicability in comparison to a sequential data scan. While stochastic scanning is not able to compete against the VA-File index, certain combinations might provide a retrieval time with a quality score possibly useful for very many retrieval applications. Similarly, for parallel, progressive scanning, the evaluation has shown promising results. As noted previously,

[3] We refer to [SGS+13; KSS14] as an attempt to evaluate indexes for high-dimensional data. These evaluation efforts, however, do not consider the retrieval problem in the same width as it is done in this thesis, but only consider the indexing aspect.

using hybrid indexes which do not only store an index value and a tuple identifier, but also further data, the retrieval times might be lowered further. More in-depth evaluations are, however, necessary to confirm this assumption. With respect to query optimisation, we were able to show that for simple similarity-based queries, in many cases the estimation by the optimiser matches well the true retrieval time. Further research and evaluations using more complex queries are necessary to determine the potential of such an optimisation approach.

The results for distributed setups, as shown in Figure 7.12, indicate that the distribution to multiple sites may increase – rather than decrease – the retrieval time for certain index structures. As noted previously and mentioned in [JDJ17], the bottleneck in similarity searches nowadays stems from using data structures which cannot saturate the high parallelism available. For a fixed collection size, hence, distribution may only achieve limited improvements. In the cases where an increase in retrieval time is visible, despite the distributed setting, an explanation might be that the higher degree of distribution results in more result candidates which are required to be retrieved and scanned from the underlying relation. Of course, the concerns on the distributability of our database have to be regarded in the context of our evaluation which considered a fixed collection size. Distribution mechanisms have been found to be important, in particular, for increasing collection sizes and, hence, we are convinced that these techniques become key for ensuring the scalability of the system. Nevertheless, the results point at the fact that more research is required, particularly in terms of scanning methods supporting high parallelism.

We have also discussed the difference between the storage engines available. We have noted the difference between our system particularly in comparison with a traditional database system which works on small data files, while Hadoop-oriented environments work on large files. To the best of our knowledge, there are no indexes or scanning methods within the research literature so far which are optimised for a largely distributed setting, in Hadoop-oriented environments for not batched, similarity-based queries. Analogue to the findings in the distributed setup, we have found such mechanisms to be important to be able to provide large-scale multimedia retrieval.

Obviously, our implementation leaves room for improvement to achieve lower retrieval times. Moreover, we note the large number of parameters of both ADAM$_{pro}$ and Apache Spark (in particular, the wide range of deployment parameters) which may alter the results presented in this evaluation. Obviously, the retrieval time and quality is strongly dependent on the context, the data and the queries, and the environment on which ADAM$_{pro}$ is deployed. It is, hence, evident that the results of our evaluation have to be put into a larger context of a full retrieval stack and, ultimately, also be evaluated in combination with a retrieval application posing a number of parallel queries on various features to scan for the queried

elements. Most importantly, we see a need in evaluating the full retrieval stack also on the basis of user studies. Dependent on the application at hand, the retrieval quality is subjective and, hence, the retrieval time, which is dependent on the accepted quality, is as well.

In summary, the results of our experiments confirm the benefits of our concepts for multimedia data management. The results assist in understanding the role of the introduced concepts and raise important implications for future research in this field of research.

Die gefährlichste Weltanschauung ist die Weltanschauung derer, welche die Welt nie angeschaut haben.

— Alexander von Humboldt

8

Related Work

This chapter reviews related research at the intersection of database and multimedia retrieval research. This area of research has no specifically focused research community, but receives contributions from various fields. In the following, we first sketch an overview of the research area. Subsequently, we review representative systems deemed relevant to this thesis.

The Lowell Database Research Self-Assessment [AGG+05] report envisioned in 2005 future research directions as seen by senior database researchers and identified important research questions for the research community. The authors of the report advocate – amongst other topics – focusing on the extension of database systems to support new (unstructured) data types, including text, sound, image, or video data. As a consequence of the increasing amount of multimedia data available, the authors propose the further investigation of multimedia queries and the associated challenges arising when analysing, summarising, searching and viewing such data. The topic has subsequently been greatly discussed in the database community (consider, for instance, [CRW05; ACR+05; Wei07; BC04]). Likewise, the following 2008 Claremont self-assessment report [AAB+08] recapitulates the challenges arising from the ubiquity of structured and unstructured data and the organisation of such data. The authors mention, for example, the need for new index structures for hybrid data, for algorithms providing best-effort services on loosely integrated data and possibly also the need for new notions of consistency and correctness.

Similarly to the database community, in 2016, the multimedia research community identified ten research questions addressing the scalability of multimedia analytics [JWZ+16]. The research questions are motivated by the rise in interest in Big Data found in various research areas. Most importantly, the authors of the report argue that methods from database management have to be applied in multimedia analytics to be able to gracefully scale to large collections in the long run, as questions regarding data management have so far not yet been explicitly considered in the field of multimedia research.

In recent years, vendors of traditional database systems have more and more adopted the ISO standard on SQL Multimedia and Application Packages (ISO/IEC 13249-1 to 6 [ISO13249:2003]). For instance, Oracle databases have already to some extent already received support for multimedia data already [Ora10ga; Kra05]. Nevertheless, compared to the extensions for spatial querying (as can be found, e.g., in Oracle Spatial[1] or in PostGIS[2]), the implementations for multimedia data lag largely behind. Obviously, given the breadth of the field of multimedia research, it would be unreasonable to assume that there exists a one-size-fits-all solution. Instead, a data management system for multimedia data would require on the one hand to provide support for low-level features, but at the same time also be able to handle concept-based queries, text retrieval and structured retrieval [JWZ+16]. On the other hand, the feature list would include also traditional functionalities of databases, including transaction support, query optimisation, caching, parallel and distributed processing, approximation and sampling [Özs99; JWZ+16].

Based on the taxonomy of [WLL+15] as presented in Section 4.1, in the following, we discuss related research of the area of databases and retrieval that provide integrated solutions to the problem. While in this chapter we focus on related work at the intersection between databases and retrieval system, we refer to Section 3.4 for more details on related work in the field of multimedia retrieval.

8.1 Retrieval on Top of a Database System

In this approach to integrate retrieval and database capabilities, the retrieval system makes use of the standard, built-in functions of a database system and provides retrieval functionality in a separate software layer on top of the data management system.

[Sch80] analysed in early information retrieval research the foundations for implementing a retrieval system using an underlying database. The author notes that there exists a need for a database management and information retrieval system (DBMIRS), however, points to the lack of support, for example, for retrieval concepts in SQL, for indexing techniques, etc. A more concrete implementation of a database-based text-retrieval system is given, for instance, in [Bla88]. The author applies a relational model to create a text-retrieval system for searching in documents and presents a possible data model which allows to search for keywords and co-occurring terms.

OVID [OT93] is an early multimedia retrieval system for videos implemented on the ideas of object-oriented database systems using Hypercard. The authors define a schemaless object model for video scenes which allows to describe various levels of the content of

[1] http://www.oracle.com/technetwork/database-options/spatialandgraph/
[2] http://postgis.net/

the scene. Moreover, to share descriptions between scenes, the authors introduce a model of inheritance. In the OVID prototype, the video objects have to be annotated manually.

In [RHM97; HMR96], the authors report on a system for multimedia analysis and retrieval, referred to as MARS. For storing and retrieving the low-level descriptions and the textual metadata (from manual descriptions), the authors make use of PostgreSQL. Their custom query processor allows users to pose complex queries describing visual properties, or text queries. The system performs the ranking of the images stored in the database and combines the results of the searches in the individual query modes.

The publication of the MPEG-7 standard to describe multimedia data in form of XML documents opened new areas of research to store and manage such data. The authors of [Kos02], for instance, discuss the possibilities of integrating an MPEG-7 based system into an object-relational database. In [TKB00], the authors perform a case study using soccer videos and integrate both high-level and low-level descriptors which are similar to the MPEG-7 standard in their prototype VideX. For a more comprehensive overview of means for managing large amounts of XML-based MPEG-7 data, we refer to [WK03].

The MUVIS system [GKC+02; Che04], a system for browsing and retrieving multimedia data, uses a traditional database for storing multimedia data and the corresponding metadata. The retrieval software layer takes care of the feature extraction, the search and similarity evaluation, etc. Moreover, the retrieval layer uses the Pyramid technique [BBK98] to generate a one-dimensional point that can be indexed using the built-in index of a database system [Che04, pp. 25].

The SIREN system [BRT+05; BRT+06] acts as an intermediary between a client application and a database. It implements an interpreter over a database realising an extension to the SQL language for similarity queries. Queries with no similarity constructs are directly forwarded to the database system, while similarity queries are rewritten to conventional operations. The authors advocate the adoption of SQL constructs for performing similarity queries in database systems and present possible language extensions.

Likewise, in [TSW07], the authors present a framework for indexing, querying and ranking unstructured and structured textual data over a relational database system. The TopX system, as it is referred to, comes with algorithms for both κNN ranked retrieval and approximate nearest neighbour retrieval.

More recent work in [MSL+14] calls for the use of column-oriented relational databases in retrieval research rather than writing own retrieval engines. The authors show how to build inverted indexes as relations and express textual ranking models (e.g., Okapi BM25) as SQL queries. Their experiments show that their advocated approach is on par with custom-built retrieval engines.

8.2 Middleware Layer subsuming a Database and a Retrieval System

This second class of systems is identified by making use of a middleware to integrate retrieval and database systems to answer queries for structured data and queries for unstructured information.

The Garlic approach [CHS+95] integrates diverse data storages which store either structured information or multimedia data into one federated, distributed database system. The storage systems include relational databases, files, text and image managers and video servers. An object-oriented data model, which can be queried using the object-oriented query language, is used for a unified view of the data. In [Fag99], the authors detail the mechanisms to retrieve results from the multiple subsystems and combine the results in a fuzzy way. The IBM QBIC [NBE+93; FSN+95] retrieval engine (see Section 8.4) is part of the IBM Garlic system.

DelosDLMS [ABB+07] is a large digital library management system that integrates diverse services to offer wide range of retrieval functionality, including visual image and video retrieval, audio retrieval, 3D retrieval, etc. Amongst others, the system builds on OSIRIS, a middleware supporting the definition and execution of distributed processes, and follows a service-oriented architecture. The ISIS system [BMR+07a], which builds upon the OSIRIS middleware as well, provides services for multimedia retrieval, including the feature extraction, the management of the indexes, relevance feedback, etc.

8.3 Retrieval Functionality based on the Database Extensibility Layer

One of the basic premises of this category is that the functionality for executing retrieval queries is implemented using the extensibility means of a database, for instance, by means of user-defined types (UDT) and functions (UDF). We separate this category from the first category (see Section 8.1) by requiring systems to truly implement some extended functionality into the database rather than only defining a schema.

Chabot [OS95] is a retrieval system that uses PostgreSQL for storing visual features extracted from images. The authors implement complex types, user-defined indexes and user-defined functions for supporting content-based queries. For example, the system allows to search for "images with some red in it", which is matched to the extracted image features by making use of user-defined functions.

In [LR95], the authors describe how to implement an audio data type into the object-oriented database system VODAK. The authors note the advantage of the object-oriented

model which supports both storing data and the appropriate manipulation/presentation methods. The system allows to search using the structured metadata and keywords, but not on the basis of the stored content.

[DDS+95] analyses what integration levels exist when combining a database and (text) retrieval system and note that the ISO SQL/MM standard is too strongly oriented towards a Boolean logic view of retrieval problems. The authors present a retrieval application based on an Oracle database system. Moreover, they describe the approach of cooperative indexing in which the retrieval components define the extracted parts of a document, while the database takes over the task of providing efficient access to the index.

DISIMA DBMS [OÖL+01; OÖI01; Özs99; OÖL+97] is an object-oriented database that allows to store for performing image retrieval both low-level features (including colour, shape, texture), and high-level features. The system supports content-based searches and searches on image semantics. The authors implement an extended version of the object-oriented query language for multimedia objects (MOQL) and VisualMOQL, a visual counterpart to MOQL for querying. To increase the performance of the system, the authors use three-dimensional extendible hashing (3DEH) that allows to pre-filter images based on low-level features.

The author of [Döl04] presents an extension to Oracle using the data cartridge technology which allows for a comparably tight integration into the database. The system is a multimedia database system which makes use of the MPEG-7 XML data structure for storing any sort of metadata corresponding to the multimedia object. Moreover, the authors present enhancements for similarity query processing, multi-dimensional index structures, etc.

[AV10] introduces a system that combines low-level and high-level features in a commercial object-relational database with the goal of performing content-based image retrieval. The database is extended by several user-defined types following the standardised MPEG-7 descriptors, and operations, for instance, to evaluate similarity measures, implemented in PL/SQL.

[FC13] implements content-based image retrieval on top of a PostgreSQL database. The authors implement the AH-tree using the Generalised Search Tree (GiST) mechanisms of PostgreSQL. GiST provides users a template to create indexing schemes for balanced, tree structures. The AH-tree is employed for improving the efficiency of retrieval when searching for high-level semantic information.

[KBP+11; KBT+09] make use of the Oracle interMedia [Ora10gb] functionality, which allows to store, manage and retrieve multimedia data. Moreover, the extension allows to define own feature extraction mechanisms, similarity measures, etc. The authors make use of user-defined functions for performing the similarity search. Moreover, in the full

picture, the system uses Oracle's extensible architecture framework to access the external indexing system Arboretum, which takes care of indexing the multi-dimensional data using a slim-tree.

POSTGRESQL-IE [GMR+09] is an extension to PostgresSQL. The extension not only implements similarity query mechanisms, but also includes the feature extraction procedures as part of the system specified in PL/SQL (e.g., for shape and colour features) and introduces a new data type for storing images.

VTApi [CPV+13] is a framework that is based on the one hand, on OpenCV for performing computer vision tasks, and on the other hand, on PostgreSQL for the data management. The authors extend PostgreSQL by means of the extensibility support given by the system to add vector-based similarity searches to the system. GEOS and PostGIS are integrated into the system for supporting multi-dimensional indexing. In their future work, the authors plan to integrate other database/data management systems, including NoSQL storage systems, within their framework.

8.4 Integration into the Database Engine

In contrast to the former class, the following systems aim at a much tighter integration of retrieval concepts into a database system by internalising the adaptions into the database engine and performing extensions to the database kernel which go beyond the mere extension using the provided programming interfaces.

The Atlas system [SKR+95] is based on a relational database that is extended to support a nested relational data model according to which the attributes of a tuple do not have to be atomic. The system has been designed for text-based applications, but can also be used to store multimedia data. It uses TQL, a SQL-like query language with additional operators supporting retrieval in textual documents.

The IBM QBIC system [NBE+93; FSN+95] has been mentioned previously already as part of the Garlic approach. It is a retrieval system that builds on the IBM relational database Starburst and allows to efficiently perform κNN searches using the R*-tree for indexing. For being able to efficiently make use of the R*-tree despite the high-dimensional features, a principal component analysis is used to reduce the dimensionality of the data points.

The INFORMIX Universal Server [Inf97] is an early commercial system that extended a relational database to support searching for images, audio, video, and other multimedia content. The system comes with R-trees for indexing multi-dimensional data.

Mirror [VB98] is a distributed multimedia information-retrieval-database based on the database system Monet [BK94]. In [VB98], the author not only describes the engineering factors for creating such a system, but also introduces a relational algebraic framework

based on the non-first normal form for modelling the data at hand.

Cobra [PJ00; PMJ02] integrates video processing and feature extraction techniques into the Monet DBMS [BK94] with a data model based on MPEG-7. Moreover, the database implements Hidden Markov Models, Dynamic Bayesian Networks and a rule-inference engine for extracting high-level concepts from the video data.

Similar to the model introduced in Mirror, the authors of [ACC+04] introduce a formal framework to address the integration of a retrieval model in an object-relational database system. The authors propose, amongst others, a similarity-based selection operator, a similarity-based join operation and an additive union. For demonstrating the use of their algebra, the authors implement EMIMS, a prototype allowing to search in medical image data.

[SAA+10] extends PostgreSQL to support similarity-aware operators, for instance, for grouping and joining. The authors extend the internals of the database for providing the functionality. In their future work, the authors note the need for indexing techniques to improve the efficiency of similarity-aware operations. Similarly, [AMO+16] focuses on extending the PostgreSQL database to support similarity-aware set operations.

In [WLL+15], the authors present their efforts to integrate text-retrieval capabilities into the Odysseus object-relational database system. The authors describe various indexes supported by their system (e.g., posting lists on top of a B^+-tree, Multi-Level Grid File, etc.), query operators, special join operations, etc.

With ADAM [GAS14a], the precursor system to ADAM_{pro}, we have presented a data management system for retrieval data based on PostgreSQL. The system supports both *κNN* similarity searches and Boolean retrieval. We extend the database with multiple distance functions and Vector-Approximation File for fast retrieval. For large collections, we introduce a middleware layer that is able to shard collections to multiple instances of the ADAM database system. The system has been extensively used for both image [GAS14b] and video retrieval [RGS+15]. The system ADAM_{pro} presented in this thesis builds on the early findings from [GAS14a]. However, the concepts have been refined and extended largely in this thesis; moreover, ADAM_{pro} is a completely new implementation of the system.

8.5 Library-based Approaches

In the following, we discuss approaches which do not consider integrating retrieval and database technologies, but rather provide some data management functionality for retrieval purposes as a library.

The Apache Lucene system[3] is a retrieval library that supports both Boolean retrieval

[3] http://lucene.apache.org/

and similarity retrieval. It indexes documents based on inverted indexes. The LIRe system for image retrieval [LCo8b] and the LIvRE system for video retrieval [BLG16], for example, make use of Lucene for multimedia retrieval. Both systems provide amongst others feature extraction algorithms by means of the Lucene API to index the multimedia data at hand. Lucene provides some database-like functionalities and is used as underlying storage engine for indexing and searching in the provided/extracted metadata. However, it should be noted that it is not a full-fledged database system, as its data model is oriented towards unstructured data and certain database functionalities, such as transactions, are missing.

FLANN [ML14] is a framework that allows to perform approximate nearest neighbour searches in high-dimensional spaces. The framework contains various algorithms for nearest neighbour searches and a system that optimises the choice of algorithm and parameters depending on the collection at hand.

8.6 Discussion

With ADAM$_{pro}$, we tightly couple database and retrieval functionality into one system.[4] While it is not exactly based on a true database, both the Boolean-based retrieval and the similarity-based retrieval are first class citizens within our model and the ADAM$_{pro}$ system. In the following, we highlight a few points of our approach which distinguish our model and our implementation from other systems.

Pragmatics From the pragmatics perspective, we note the difference of our approach compared to many previous approaches. Most of the systems introduced in this chapter (with very few exceptions only, e.g., Mirror [VB98]) integrate feature extraction logic and retrieval logic into one single system making the data management catered to one specific setting. Our implementation and model, on the other hand, are agnostic to the data used. Moreover, thanks to the large set of supported query paradigms, a wide range of use cases can be implemented.

Data model With respect to the data models, our approach is similar to many other systems providing data management means for multimedia data (cf. [SAA+10]). Data management systems in the context of multimedia retrieval have sometimes also made use of object-oriented data models (cf. [LR95; OT93]). Other approaches, including NoSQL, have seen some adoption as well in the given context (cf. [MSL+14]).

[4] Note that while ADAM$_{pro}$ makes use of database systems (and other persistence mechanisms) as storage engines for storing the data, from the perspective of the integration of retrieval and database functionality, we rather categorise our system as providing a tight integration.

Query model In this thesis, we have presented a query model which supports both logical and executional parameters. While, from a logical perspective, the combination of Boolean and similarity predicates is well known in research though not always formalised (cf. [Vri99; SAA+10; ACC+04]), the addition of executional parameters has not been studied so far. With ADAM$_{pro}$, we provide various scanning approaches (precise, approximate, stochastic, parallel), a different approach to query optimisation and a different execution paradigm. All these methods have only seen – if ever – limited support in previous systems.

Index structures From the perspective of index structures, ADAM$_{pro}$ presents a large and diverse set of index structures within one single system. In most similar systems, only a single index structure is implemented to demonstrate its working and to show its applicability. Many earlier systems have implementations of R-trees (cf. [NBE+93]); given their limited scalability in terms of the dimensions, we have refrained from adding any hierarchical index structure to our system. To the best of our knowledge, there exists no system with such a wide range of index structures for high-dimensional vector data as can be found in our implementation.

Distribution Distribution is a topic which is only seldom considered for data management systems in the context of retrieval. Most approaches integrate into existing, traditional databases which operate only on single nodes (cf. [FC13; GMR+09]). Distribution is considered, if at all, in very isolated use cases (cf. [MSG+13a] using map/reduce, [Vri99]). In our approach, we consider the distribution of data and work to multiple nodes and make use of distribution primitives for processing and answering queries. Moreover, ADAM$_{pro}$ also considers the partitioning of data which has only received very little attention in the context of multimedia retrieval.

Storage mechanisms The data management aspect in the context of multimedia retrieval data still considers monolithic system. Advances in the area of databases, such as the notion of polystores, has not received much attention so far (cf. [CPV+13]). With ADAM$_{pro}$, we present possibly one of the first systems combining multiple storage engines with the goal of storing multimedia data.

Heureux qui, comme Ulysse, a fait un beau voyage...

— Joachim Du Bellay

9 Conclusion and Outlook

9.1 Summary

In this thesis, we have taken an attempt at integrating efforts made in both the field of multimedia retrieval and database systems towards an integrated data management system for multimedia retrieval. The objective of this thesis has been to organise the management and storage of multimedia data and metadata providing an efficient and effective retrieval. We have, in particular, focused on the design of a logical data model which can be used to store structured information and (unstructured) multimedia documents. From the query perspective, we have studied and defined a query model allowing to search within the structured information and to perform retrieval based on the content metadata of multimedia documents. Given the high computational complexity of similarity-based queries, we have investigated ways to process and execute a query in an efficient way and have studied approaches to store and manage the data from a distribution and a physical perspective. One of the basic premises behind our approach is the supposition that the separation of data management questions from the pragmatics of the retrieval application is of the outmost importance.

To summarise, in this thesis, we have made the following contributions:

- We have identified the required building-blocks for supporting multimedia data and queries in a multimedia data management system.
- We have defined a blueprint for a multimedia data management system and its components.
- We have adopted the relational data model and the vector-space model to the end of creating an integrated data model for both structured and unstructured data.
- We have put strong emphasis on the query model which supports multimedia data and its corresponding metadata and which can be specified on a logical and an executional level.

- We have focused on the efficiency of the system by considering both distribution and indexing techniques. In particular, we have presented and compared a large set of state-of-the-art index structures for high-dimensional vector data which we employ in our system.
- This thesis has presented the working implementation ADAM$_{pro}$ which has successfully been used in the iMotion/vitrivr project. Moreover, we have presented a novel user interface to our implementation.
- We have presented an evaluation of the introduced concepts and the prototype ADAM$_{pro}$ with both synthetic and with real data. The results of our experiments confirm the benefits of our concepts for multimedia data management and raise important implications for future research in this field of research.

Ultimately, our work makes a much needed contribution to the field of multimedia data management for multimedia retrieval by providing a holistic perspective on the problem. Our contributions ensure that new developments on the retrieval level can easily be applied without having to take care of the data management perspective. With this work, we have bridged the gap between the fields of retrieval and databases and we hope that this works proves to provide a significant step towards this endeavour.

9.2 Future Work

In the following, we discuss potential future work which builds upon this thesis and may help to qualitatively and quantitatively improve the conceptual view and our implementation.

Architecture When it comes to databases for multimedia data, there is a great variety of retrieval paradigms necessary to tackle depending on the context. From an architectural perspective, hence, there might exist different flavours of the same system – making use of a different set of components – which may be applied dependent on the retrieval system accessing the data management system. For instance, in our requirements analysis, we have argued that transaction management and recovery topics are only secondary in the context of multimedia databases, as retrieval systems often clearly distinguish the off-line, loading phase from the on-line, querying phase. While this might be true for many settings, in certain contexts it might still be important to support transactional mechanisms. On the other hand, depending on the context, the database system may not have to support multiple storage engines or indexes when it is used with very small collections. We see, hence, a need for a more modular approach to the problem which allows to combine pluggable components to the end of creating a customised database system for multimedia. While our architecture and implementation to large extent support modularity, it might

be even considered on a more fine-granular level. The user employing the multimedia data management system in a context where the data is often updated may choose to use it together with the transactional semantics; on the other hand, a user in a small application setting may choose not to use transactional guarantees or a sophisticated storage, but instead only employ a simple file-based storage.

Indexing We have presented a large set of index structures for similarity-based retrieval on high-dimensional vector data. However, the index structures for high-dimensional vector data presented in the last decades all follow the assumption of accessing a tuple on a small data page of a few kilobytes. As we have seen in the evaluation, the usefulness of such index structures has proven to be only limited. We advocate for more research in indexes for data residing on in-memory systems or in distributed file systems (such as HDFS), which match the map/reduce working paradigm. Moreover, more research has to be done on highly parallelisable index structures which are able to saturate the parallel processing available nowadays in distributed systems or GPUs.

We have shown the necessity for supporting hybrid index structures which do not only store the indexed data, but possibly also further attributes to avoid that a data scan becomes necessary. Particularly in combination with stochastic scans and progressive querying, the use of hybrid index structures may yield good results at low retrieval times, as only the index structure possibly needs to be scanned.

Furthermore, we would like to see the construction of indexes in the long run in terms of building blocks. For example, new index structures may be built as a composition of a dimensionality reduction, projection, quantisation steps, etc. Following the ideas of database cracking [IKM07], such indexes may be generated dynamically by the underlying system based on the queries at hand to reduce the retrieval time. Very recent approaches found in [KBC+17] apply machine learning techniques for learning index structures based on the data and the access pattern. Possibly a sophisticated – "learned" – combination of index creation steps may yield index structures which are adapted to the context and provide a highly efficient retrieval.

Retrieval Our query model only considered Boolean and similarity retrieval based on εNN and κNN searches. [SHS+08], for example, lists a large set of alternative similarity searches, including reverse nearest neighbour, continuous nearest neighbour, etc. Our model could be extended to support other forms of retrieval. Moreover, it may include and support better the interactive style of querying known from retrieval systems and give support for what is generally known as relevance feedback.

Moreover, the empirically based query optimisation approach could further be researched in what features to use, what mechanisms to use for approximating the estimated

retrieval time. Also from the aspect of optimising queries, we note the efforts made in self-tuning databases (cf. [CN07]). While in our approach, we have employed empirical optimisation for simple queries only, we see a need to further research empirical query optimisation for more complex queries. Referring to potential future work mentioned previously, we see a strong case for combining the empirical optimisation strategy with adaptive scanning methods and learned index structures.

Query formulation and result visualisations While we have briefly touched upon the topic of query formulation, in this thesis, we have refrained from defining a query language for retrieval. Similarly, our implementation is lacking a declarative query language like SQL for this purpose. More research is necessary to study the requirements and elements of such a language. While this thesis has also suggested the use of visual query composition, as can also be found, for example, in [CAL+97; CC96], more in-depth analyses of how it could be used would be important.

On this topic, more research could go in studying data presentation techniques for result presentation to the user. A large set of visualisation methods for high-dimensional data have been researched in the past decades (cf. [LMW+17]), but are not used in the context of multimedia retrieval yet. Applying proper visualisations of the results may help in the retrieval of specific elements. Moreover, proper visualisations may help to detect problematic spots for retrieval in terms of the retrieval time (e.g., by having only very few, but large clusters of data).

Distribution We have touched upon distribution in multimedia databases and have performed experiments for partitioning the data at hand at selecting the shards to query at query time in [Hel16]. However, more research is necessary for an effective distribution which allows to efficiently process queries. In [Hel16], we have considered a number of partitioning mechanisms which have not proven to be successful. More research should go into this topic to be able to distribute the data at hand and only query a sub-set of shards for the data. Moreover, also the topic of replication, which has not been discussed in this thesis, needs more consideration.

Finally, while there exists a number of algorithms which try to optimise ranked results coming from different databases (e.g., Fagin's algorithm [Fag99], Threshold Algorithm [FLN01]), the methods and concepts available for similarity-based retrieval in large collections is only limited. As mentioned previously, more research is necessary in processing queries in a distributed fashion. To the best of our knowledge, both indexing and scanning methods optimised for largely distributed settings are missing, in particular for not batched, similarity-based queries. To ensure that systems are able to cope with increasing collection

sizes, it is of great importance to focus on the questions of indexing and retrieval, while keeping the topic of distribution in mind.

Storage management We have applied the concept of polystores in this thesis to store the data at hand in various sub-stores. It might be fruitful to research how to employ multiple physical, adaptable layouts for the data at hand. Following the ideas of [DJ11; AIA14], for instance, our model could be extended to support hybrid storage engines which allow to store data simultaneously in more than one storage engine and adapt the use of the engine to the accessing pattern. For example, a hybrid storage could allow to combine key-value stores supporting fast random access with a storage engine which supports the distribution of the data to multiple nodes; this could allow to make use of the most efficient storage engine depending on the given access pattern. Particularly in combination with empirical optimisation, such an approach might prove to be useful.

The workload we have envisioned for our model and our implementation is based on retrieval tasks. However, multimedia retrieval more and more also focuses on browsing. We see a lot of potential to support efficient (and effective) browsing based on the data, possibly also based on building nearest neighbour graphs. While this topic has not at all been considered in this thesis, it might be worthwhile providing solutions for browsing.

Content provisioning We have not discussed the topic of content provisioning and playout management, as it is a topic on its own to consider. In particular, we have also not addressed where and how to store the raw multimedia content. [Spr14, pp. 88] briefly discusses the advantages and disadvantages of storing a multimedia object within a digital library. Of course, for a proper multimedia data management system, this topic should be addressed and further researched.

Implementation Our prototypical implementation has successfully been used in various instances already. However, obviously, it is subject to endless improvements to make the prototype not only more stable, but also more efficient.

A

Index Parameters

Cluster Pruning

Maximum number of leaders 200

Number of representatives $\sqrt{n}$

Leader selection leader corresponds to representative

Locality-Sensitive Hashing

Number of hash tables 64

Number of hashes 64

Maximum number of buckets 256

Norm 2

Number of training tuples 500

Metric Inverted-File

Number of reference objects $\min(200, 2\sqrt{n})$

Number of hashes 64

Number of reference objects in indexing $\min(100, v)$

Number of reference objects in querying $\min(50, v)$

Product Quantisation

Number of training tuples 1000

Number of sub-vectors 8

Spectral Hashing

Number of training tuples 1000

Number of bits $2dim$

Vector Approximation-File/VA$^+$-File

Number of training tuples 5000

Number of bits per dimension $\max(5, 5 + 0.5 \log_2(dim/10))$

Type of marks equi-populated cells strategy

B

Evaluation Parameters

Environment Set-Up for Single-Machine Evaluation

The evaluation environment for evaluating ADAM$_{pro}$ has been started using the following commands.

```
export ENV_ADAMPRO_MASTER_HOSTNAME=local[20]
export ENV_ADAMPRO_MEMORY=50G

docker network create --driver bridge vitrvrnw

docker run
   --net=vitrvrnw
   -p 5890:5890
   -e "ADAMPRO_MEMORY=$ENV_ADAMPRO_MEMORY"
   -e "ADAMPRO_MASTER=$ENV_ADAMPRO_MASTER_HOSTNAME"
   -d vitrivr/adampro:2.1-eval

docker run
   --net=vitrvrnw
   -p 9042:9042
   --net-alias cassandra -d
   cassandra:3.7
```

The `adampro.conf` configuration file within the Docker container has been adjusted as follows.

```
adampro {
  ...

  defaultPartitions = 40
  defaultPartitionsIndex = 40

  evaluation = true

  filteringMethod = "isinfilter"
  VAGlobalRefinement = true

  engines = ["index", "parquet", "cassandra"]
}

storage {
  ...

  cassandra {
      engine = "CassandraEngine"
      url = "cassandra"
      port = "9042"
      user = "cassandra"
      password = "cassandra"
  }
}
```

Environment Set-Up for Distributed Evaluation

The evaluation environment for evaluating ADAM$_{pro}$ in a truly distributed setting has been started using the following commands.
The master node has been started as follows.

```
export ENV_ADAMPRO_MASTER_HOSTNAME=$HOSTNAME
export ENV_ADAMPRO_MEMORY=50G
docker run --name adamproHDFSMaster
   --network=host
   -p 2122:2122 -p 4040:4040 -p 5005:5005 -p 5432:5432
   -p 5890:5890 -p 6066:6066 -p 7001:7001 -p 7002:7002
   -p 7003:7003 -p 7004:7004 -p 7005:7005 -p 7006:7006
   -p 7077:7077 -p 8020:8020 -p 8030:8030 -p 8031:8031
```

```
   -p 8032:8032 -p 8080:8080 -p 8088:8088 -p 8983:8983
   -p 9000:9000 -p 9099:9099 -p 19888:19888 -p 38000:38000
   -p 39000:39000 -p 50010:50010 -p 50020:50020 -p 50070:50070
   -p 50075:50075 -p 50090:50090
   -e "ADAMPRO_DRIVER_MEMORY=$ENV_ADAMPRO_MEMORY"
   -e "ADAMPRO_MASTER=spark://$ENV_ADAMPRO_MASTER_HOSTNAME:7077"
   -e "ADAMPRO_MASTER_HOSTNAME=$ENV_ADAMPRO_MASTER_HOSTNAME"
   -e "ADAMPRO_EXECUTOR_MEMORY=$ENV_ADAMPRO_MEMORY"
   -e "SPARK_PUBLIC_DNS=localhost"
   --entrypoint="/adampro/bootstrap.sh" -d
   vitrivr/adampro:2.1-hdfs
   -d --masternode
```

The worker nodes are started as follows.

```
export ENV_ADAMPRO_MASTER_HOSTNAME= #set host name of master host
export ENV_ADAMPRO_MASTER_IP= #set IP address of master host
export ENV_ADAMPRO_MEMORY= 50G
docker run --name adamproHDFSWorker
   --network=host
   -p 2122:2122 -p 7012:7012 -p 7013:7013 -p 7014:7014
   -p 7015:7015 -p 7016:7016 -p 8020:8020 -p 8030:8030
   -p 8031:8031 -p 8032:8032 -p 8081:8081 -p 8881:8881
   -p 9000:9000 -p 38000:38000 -p 39000:39000 -p 50010:50010
   -p 50020:50020 -p 50070:50070 -p 50075:50075
   -e "ADAMPRO_MASTER=spark://$ENV_ADAMPRO_MASTER_HOSTNAME:7077"
   -e "ADAMPRO_MASTER_HOSTNAME=$ENV_ADAMPRO_MASTER_HOSTNAME"
   -e "ADAMPRO_EXECUTOR_MEMORY=$ENV_ADAMPRO_MEMORY"
   -e "SPARK_WORKER_INSTANCES=1"
   --add-host $ENV_ADAMPRO_MASTER_HOSTNAME:$ENV_ADAMPRO_MASTER_IP
   --entrypoint="/adampro/bootstrap.sh" -d
   vitrivr/adampro:2.1-hdfs
   -d --workernode
```

Evaluation Parameter Specification in UNIBAS Chronos

Evaluation of the Effect of Collection Size in Similarity Queries

Mode SEN (System Evaluation Random Data)

Number of tuples 100K, 500K, 1M, 5M, 10M, 50M, 100M

Dimensionalities 100

Sparsity, Minimum/Maximum, Distribution 0%, 0.0/1.0, uniform

Logical Metadata Long (0), Integer (0), Float (0), Double (0), String (0), Text (0), Boolean (0)

Storage handler Apache Cassandra

Number of retrieved results 100

Number of queries 10

Distance function Squared Euclidean, not weighted

Execution paths all (sequential, index scans with data access)

Evaluation of the Effect of Dimensionality in Similarity Queries

Mode SEN (System Evaluation Random Data)

Number of tuples 10M

Dimensionalities 10, 50, 100, 500

Sparsity, Minimum/Maximum, Distribution 0%, 0.0/1.0, uniform

Logical Metadata Long (0), Integer (0), Float (0), Double (0), String (0), Text (0), Boolean (0)

Storage handler Apache Cassandra

Number of retrieved results 100

Number of queries 10

Distance function Squared Euclidean, not weighted

Execution paths all (sequential, index scans with data access)

Evaluation using the YFCC100M Data in Similarity Queries

Mode SEE (System Evaluation Existing Data)

Storage handler Apache Cassandra

Number of retrieved results 100

Number of queries 10

Distance function Squared Euclidean, not weighted

Execution paths all (sequential, index scans with data access)

Evaluation of the Use of Logical Parameters

Mode SEN (System Evaluation Random Data)

Number of tuples 10M

Dimensionalities 100

Sparsity, Minimum/Maximum, Distribution 0%, 0.0/1.0, uniform

Logical Metadata Long (1), Integer (1), Float (1), Double (1), String (0), Text (0), Boolean (0)

Storage handler Apache Cassandra

Number of retrieved results 100

Number of queries 10

Distance function Squared Euclidean, not weighted

Execution paths sequential, VA-File with data access

Evaluation of the Use of Executional Parameters

Stochastic scanning

Mode SQE (Stochastic Query Evaluation)

Number of tuples 10M

Dimensionalities 100

Sparsity, Minimum/Maximum, Distribution 0%, 0.0/1.0, uniform

Logical Metadata Long (0), Integer (0), Float (0), Double (0), String (0), Text (0), Boolean (0)

Storage handler Apache Cassandra

Number of retrieved results 100

Number of queries 10

Distance function Squared Euclidean, not weighted

Execution paths all (index scans, with data access)

Parallel scanning and progressive execution

Mode PAE (Parallel Scan Evaluation)

Number of tuples 10M

Dimensionalities 100

Sparsity, Minimum/Maximum, Distribution 0%, 0.0/1.0, uniform

Logical Metadata Long (0), Integer (0), Float (0), Double (0), String (0), Text (0), Boolean (0)

Storage handler Apache Cassandra

Number of retrieved results 100

Number of queries 10

Distance function Squared Euclidean, not weighted

Execution paths SH, PQ, VAF with sequential

Empirical optimisation

Mode EQE (Empirical Query Evaluation)

Number of tuples 100K, 500K, 1M, 5M, 10M, 50M, 100M

Dimensionalities 10, 50, 100, 500

Sparsity, Minimum/Maximum, Distribution 0%, 0.0/1.0, uniform

Logical Metadata Long (0), Integer (0), Float (0), Double (0), String (0), Text (0), Boolean (0)

Storage handler Apache Cassandra

Number of retrieved results 100

Number of queries 10

Distance function Squared Euclidean, not weighted

Training tuples 100K, 500K, 1M, 5M, 10M*, 50M, 100M (* default)

Training dimensionalities 10, 50, 100*, 500 (* default)

Optimizer LR

Number of queries/runs for training 10/1

Execution paths all (sequential, index scans with data access)

Evaluation of the Use of various Storage Engines in Similarity Queries

Mode SEN (System Evaluation Random Data)

Number of tuples 10M

Dimensionalities 100

Sparsity, Minimum/Maximum, Distribution 0%, 0.0/1.0, uniform

Logical Metadata Long (0), Integer (0), Float (0), Double (0), String (0), Text (0), Boolean (0)

Storage handler Apache Parquet

Number of retrieved results 100

Number of queries 10

Distance function Squared Euclidean, not weighted

Execution paths sequential, PQ, VAF

Evaluation of the Distribution Mechanisms

Mode SEN (System Evaluation Random Data)

Number of tuples 10M

Dimensionalities 100

Sparsity, Minimum/Maximum, Distribution 0%, 0.0/1.0, uniform

Logical Metadata Long (0), Integer (0), Float (0), Double (0), String (0), Text (0), Boolean (0)

Storage handler Apache Cassandra

Number of retrieved results 100

Number of queries 10

Distance function Squared Euclidean, not weighted

Execution paths all (sequential, index scans with data access)

Bibliography

[AAB+08] Rakesh Agrawal, Anastasia Ailamaki, Philip A. Bernstein, Eric A. Brewer, Michael J. Carey, Surajit Chaudhuri, AnHai Doan, Daniela Florescu, Michael J. Franklin, Hector Garcia-Molina, Johannes Gehrke, Le Gruenwald, Laura M. Haas, Alon Y. Halevy, Joseph M. Hellerstein, Yannis E. Ioannidis, Hank F. Korth, Donald Kossmann, Samuel Madden, Roger Magoulas, Beng Chin Ooi, Tim O'Reilly, Raghu Ramakrishnan, Sunita Sarawagi, Michael Stonebraker, Alexander S. Szalay and Gerhard Weikum. The Claremont Report on Database Research. *Special Interest Group on Management of Data (SIGMOD) Records*, 37(3):9–19, September 2008.

[ABB+07] Maristella Agosti, Stefano Berretti, Gert Brettlecker, Alberto del Bimbo, Nicola Ferro, Norbert Fuhr, Daniel Keim, Claus-Peter Klas, Thomas Lidy, Diego Milano, Moira Norrie, Paola Ranaldi, Andreas Rauber, Hans-Jörg Schek, Tobias Schreck, Heiko Schuldt, Beat Signer and Michael Springmann. DelosDLMS — The Integrated DELOS Digital Library Management System. In *Proceedings of the International DELOS Conference on Digital Libraries: Research and Development*, pages 36–45, Pisa, Italy. Springer, 2007.

[ABS04] Sibel Adali, Corey Bufi and Maria Luisa Sapino. Ranked Relations: Query Languages and Query Processing Methods for Multimedia. *Multimedia Tools and Applications (MTAP)*, 24(3):197–214, December 2004.

[ACC+04] Solomon Atnafu, Richard Chbeir, David Coquil and Lionel Brunie. Integrating Similarity-Based Queries in Image DBMSs. In *Proceedings of the International Symposium on Applied Computing (SAC)*, pages 735–739, Nicosia, Cyprus. ACM, 2004.

[ACR+05] Sihem Amer-Yahia, Pat Case, Thomas Rölleke, Jayavel Shanmugasundaram and Gerhard Weikum. Report on the DB/IR Panel at SIGMOD 2005. *Special Interest Group on Management of Data (SIGMOD) Records*, 34(4):71, December 2005.

[AGG+05] Serge Abiteboul, Dieter Gawlick, Jim Gray, Laura Haas, Alon Halevy, Joseph M. Hellerstein, Yannis Ioannidis, Martin Kersten, Michael Pazzani, Mike Lesk, David Maier, Rakesh Agrawal, Jeff Naughton,

Hans-Jörg Schek, Timos Sellis, Avi Silberschatz, Mike Stonebraker, Rick Snodgrass, Jeff Ullman, Gerhard Weikum, Jennifer Widom, Stan Zdonik, Phil Bernstein, Mike Carey, Stefano Ceri, Bruce Croft, David DeWitt, Mike Franklin and Hector Garcia-Molina. The Lowell Database Research Self-assessment. *Communications of the ACM (CACM)*, 48(5):111–118, May 2005.

[AGL+07] Edoardo Ardizzone, Luca Gatani, Marco La Cascia, Giuseppe Lo Re and Marco Ortolani. A P2P Architecture for Multimedia Content Retrieval. In *Proceedings of the International Conference on Multimedia Modeling (MMM)*, pages 462–474, Singapore, Singapore. Springer, 2007.

[AGS+13] Ihab Al Kabary, Ivan Giangreco, Heiko Schuldt, Fabrice Matulic and Moira Norrie. QUEST: Towards a Multi-modal CBIR Framework Combining Query-by-Example, Query-by-Sketch, and Text Search. In *Proceedings of the International Symposium on Multimedia (ISM)*, pages 433–438, Anaheim, USA. IEEE, 2013.

[AGS14] Giuseppe Amato, Claudio Gennaro and Pasquale Savino. MI-File: Using Inverted Files for Scalable Approximate Similarity Search. *Multimedia Tools and Applications (MTAP)*, 71(3):1333–1362, August 2014.

[AGZ+15] Michael Armbrust, Ali Ghodsi, Matei Zaharia, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan and Michael J. Franklin. Spark SQL: Relational Data Processing in Spark. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1383–1394, Melbourne, Australia. ACM, 2015.

[AH14] Ahmed Abdelsadek and Mohamed Hefeeda. DIMO: Distributed Index for Matching Multimedia Objects using MapReduce. In *Proceedings of the International Conference on Multimedia Systems (MMSys)*, pages 115–126, Singapore, Singapore. ACM, 2014.

[AHK01] Charu C. Aggarwal, Alexander Hinneburg and Daniel A. Keim. On the Surprising Behavior of Distance Metrics in High Dimensional Space. In *Proceedings of the International Conference on Database Theory (ICDT)*, pages 420–434, London, UK. Springer, 2001.

[AI08] Alexandr Andoni and Piotr Indyk. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *Communications of the ACM (CACM)*, 51(1):117–122, January 2008.

[AI15] Alexandr Andoni and Piotr Indyk. *E²LSH 0.1 User Manual*. Cambridge, USA, 2015. URL: www.mit.edu/~andoni/LSH/manual.pdf (visited on 31/10/2017).

[AI16] Manos Athanassoulis and Stratos Idreos. Design Tradeoffs of Data Access Methods. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 2195–2200, San Francisco, USA. ACM, 2016.

[AIA14] Ioannis Alagiannis, Stratos Idreos and Anastasia Ailamaki. H_2o: A Hands-free Adaptive Store. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1103–1114, Snowbird, USA. ACM, 2014.

[AK17] Khushbu Agrawal and Omnia Kahla. *Evaluation of ADAM$_{pro}$*. Master Project, University of Basel, 2017.

[AKK+99] Michael Ankerst, Gabi Kastenmüller, Hans-Peter Kriegel and Thomas Seidl. Nearest Neighbor Classification in 3D Protein Databases. In *Proceedings of the International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pages 34–43, Heidelberg, Germany. AAAI, 1999.

[AKM+16] Manos Athanassoulis, Michael S. Kester, Lukas M. Maas, Radu Stoica, Stratos Idreos, Anastasia Ailamaki and Mark Callaghan. Designing Access Methods: The RUM Conjecture. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 461–466, Bordeaux, France. OpenProceedings.org, 2016.

[ALM96] Edoardo Ardizzone, Marco La Cascia and Davide Molinelli. Motion and Color-based Video Indexing and Retrieval. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, pages 135–139, Vienna, Austria. IEEE, 1996.

[AMO+16] Wadha J. Al Marri, Qutaibah Malluhi, Mourad Ouzzani, Mingjie Tang and Walid G. Aref. The Similarity-Aware Relational Database Set Operators. *Information Systems*, 59(C):79–93, July 2016.

[AMP+13] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden and Ion Stoica. BlinkDB: Queries with Bounded Errors and Bounded Respose Times on Very Large Data. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, pages 29–42, Prague, Czech Republic. ACM, 2013.

[Ams14] Laurent Amsaleg. *A Database Perspective on Large Scale High-Dimensional Indexing*. Habilitation à Diriger des Recherches, University of Rennes 1, France, 2014.

[AN97] Donald A. Adjeroh and Kingsley C. Nwosu. Multimedia Database Management-Requirements and Issues. *MultiMedia*, 4(3):24–33, July 1997.

[Apa17] Apache Software Foundation. *API Documentation of Spark 2.2, Dataset*. 2017. URL: https://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.Dataset (visited on 15/08/2017).

[Art16] Artive Inc. New Non-Profit Launches with Mission to Protect and Preserve the World's Cultural Heritage through Technology, November 2016. URL: https://artive.org/2016/11/16/new-non-profit-launches-with-mission-to-protect-and-preserve-the-worlds-cultural-heritage-through-technology/ (visited on 02/10/2017).

[AS08] Giuseppe Amato and Pasquale Savino. Approximate Similarity Search in Metric Spaces Using Inverted Files. In *Proceedings of the International Conference on Scalable Information Systems (InfoScale)*, number 28, Vico Equense, Italy. ACM, 2008.

[AS13] Ihab Al Kabary and Heiko Schuldt. SportSense: Using Motion Queries to Find Scenes in Sports Videos. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 2489–2492, San Francisco, USA. ACM, 2013.

[AV10] Carlos E. Alvez and Aldo R. Vecchietti. Combining Semantic and Content Based Image Retrieval in ORDBMS. In *Proceedings of the International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES)*, pages 44–53, Cardiff, UK. Springer, 2010.

[AVT+04] Adriano S. Arantes, Marcos R. Vieira, Caetano Traina Jr. and Agma J. M. Traina. Efficient Algorithms to Execute Complex Similarity Queries in RDBMS. *Journal of the Brazilian Computer Society*, 9(3):5–24, April 2004.

[Bac14] Lucius Bachmann. *"Bernstein": Online-Schaltung und Federation einer Wasserzeichen-Datenbank*. Bachelor Project, University of Basel, 2014.

[BBK+97] Stefan Berchtold, Christian Böhm, Daniel A. Keim and Hans-Peter Kriegel. A Cost Model For Nearest Neighbor Search in High-Dimensional Data Space. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 78–86, Tucson, USA. ACM, 1997.

[BBK01] Christian Böhm, Stefan Berchtold and Daniel A. Keim. Searching in High-Dimensional Spaces — Index Structures for Improving the Performance of Multimedia Databases. *Computing Surveys (CSUR)*, 33(3):322–373, September 2001.

[BBK98] Stefan Berchtold, Christian Böhm and Hans-Peter Kriegel. The Pyramid-Technique: Towards Breaking the Curse of Dimensionality. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 142–153, Seattle, USA. ACM, 1998.

[BBZ12a] Petra Budikova, Michal Batko and Pavel Zezula. Query Language for Complex Similarity Queries. In *Proceedings of the East European Conference on Advances in Databases and Information Systems (ADBIS)*, pages 85–98, Poznań, Poland. Springer, 2012.

[BBZ12b] Petra Budikova, Michal Batko and Pavel Zezula. Query Language for Complex Similarity Queries. arXiv, April 2012. URL: `https://arxiv.org/abs/1204.1185` (visited on 31/10/2017).

[BC04] Ricardo Baeza-Yates and Mariano Consens. The Continued Saga of DB-IR Integration. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 1245–1246, Toronto, Canada. Morgan Kaufmann, 2004.

[BCD+09] Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Kilian Thiel and Bernd Wiswedel. KNIME — The Konstanz Information Miner. *ACM SIGKDD Explorations Newsletter*, 11(1):26–31, November 2009.

[BCR09] Nicolas Bruno, Surajit Chaudhuri and Ravi Ramamurthy. Power Hints for Query Optimization. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 469–480, Shanghai, China. IEEE, 2009.

[Bel61] Richard E. Bellman. *Adaptive Control Processes*. Princeton University Press, Princeton, 1961.

[Ben75] Jon Louis Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM (CACM)*, 18(9):509–517, September 1975.

[BFL+10] Michal Batko, Fabrizio Falchi, Claudio Lucchese, David Novak, Raffaele Perego, Fausto Rabitti, Jan Sedmidubsky and Pavel Zezula. Building a Web-Scale Image Similarity Search System. *Multimedia Tools and Applications (MTAP)*, 47(3):599–629, May 2010.

[BGR+99] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan and Uri Shaft. When Is "Nearest Neighbor" Meaningful? In *Proceedings of the International Conference on Database Theory (ICDT)*, pages 217–235, Jerusalem, Israel. Springer, 1999.

[BGS08] Nouha Bouteldja, Valerie Gouet-Brunet and Michel Scholl. The Many Facets of Progressive Retrieval for CBIR. In *Proceedings of the Pacific-Rim Conference on Multimedia Information Processing (PCM)*, pages 611–624, Tainan, Taiwan. Springer, 2008.

[BHM15] Kai Uwe Barthel, Nico Hezel and Radek Mackowiak. Graph-Based Browsing for Large Video Collections. In *Proceedings of the International Conference on Multimedia Modeling (MMM)*, pages 237–242, Sydney, Australia. Springer, 2015.

[BK03] Christian Böhm and Florian Krebs. Supporting KDD Applications by the k-Nearest Neighbor Join. In *Proceedings of the International Conference on Database and Expert Systems Applications (DEXA)*, pages 504–516, Prague, Czech Republic. Springer, 2003.

[BK94] Peter A. Boncz and Martin L. Kersten. Monet. An Impressionist Sketch of an Advanced Database System. In *Proceedings of the Basque International Workshop on Information Technology (BIWIT)*, pages 240–251, Biarritz, France. IEEE, 1994.

[BKK01] Stefan Berchtold, Daniel A. Keim and Hans-Peter Kriegel. The X-Tree: An Index Structure for High-Dimensional Data. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 28–39, Rome, Italy. Morgan Kaufmann, 2001.

[BKS+90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider and Bernhard Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 322–331, Atlantic City, USA. ACM, 1990.

[Bla88] David C. Blair. An Extended Relational Document Retrieval Model. *Information Processing & Management*, 24(3):349–371, January 1988.

[BLG16] Gabriel de Oliveira Barra, Mathias Lux and Xavier Giró-i-Nieto. Large Scale Content-based Video Retrieval with LIvRE. In *Proceedings of the International Workshop on Content-based Multimedia Indexing (CBMI)*, Bucharest, Romania. IEEE, 2016.

[Blo70] Burton H. Bloom. Space/Time Trade-Offs in Hash Coding with Allowable Errors. *Communications of the ACM (CACM)*, 13(7):422–426, July 1970.

[BMR+07a] Gert Brettlecker, Diego Milano, Paola Ranaldi, Hans-Jörg Schek, Heiko Schuldt and Michael Springmann. ISIS and OSIRIS: A Process-Based Digital Library Application on Top of a Distributed Process Support Middleware. In *Proceedings of the International DELOS Conference on Digital Libraries: Research and Development*, pages 46–55, Pisa, Italy. Springer, 2007.

[BMR+07b] Gert Brettlecker, Diego Milano, Paola Ranaldi and Heiko Schuldt. DelosDLMS — A Next-Generation Digital Library Management System. In *Proceedings of the International Conference of Image Analysis and Processing - Workshops (ICIAPW)*, pages 83–88, Modena, Italy. IEEE, 2007.

[BMS+01] Klemens Böhm, Michael Mlivoncic, Hans-Jörg Schek and Roger Weber. Fast Evaluation Techniques for Complex Similarity Queries. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 211–220, Rome, Italy. Morgan Kaufmann, 2001.

[Böh00] Christian Böhm. A Cost Model for Query Processing in High Dimensional Data Spaces. *Transactions on Database Systems (TODS)*, 25(2):129–178, June 2000.

[BP98] Sergej Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proceedings of the International Conference on the World-Wide Web (WWW)*, pages 107–117, Brisbane, Australia. Elsevier, 1998.

[BR11] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology behind Search*. Addison Wesley, Harlow, UK, 2nd edition, 2011.

[BRT+05] Maria Camila Nardini Barioni, Humberto Luiz Razente, Caetano Traina Jr. and Agma J M Traina. Querying complex objects by similarity in SQL. In *Proceedings of the Brazilian Symposium on Databases (SBBD)*, pages 130–144, Uberlândia, Brazil. Brazilian Computer Society (SBC), 2005.

[BRT+06] Maria Camila Nardini Barioni, Humberto Luiz Razente, Agma J. M. Traina and Caetano Traina Jr. SIREN: A Similarity Retrieval Engine for Complex Data. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 1155–1158, Seoul, Korea. ACM, 2006.

[BRT+09] Maria Camila Nardini Barioni, Humberto Luiz Razente, Agma J. M. Traina and Caetano Traina Jr. Seamlessly Integrating Similarity Queries in SQL. *Software: Practice and Experience*, 39(4):355–384, March 2009.

[BTV06] Herbert Bay, Tinne Tuytelaars and Luc Van Gool. SURF: Speeded Up Robust Features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 404–417, Graz, Austria. Springer, 2006.

[Bus45] Vannevar Bush. As We May Think. *The Atlantic Monthly*:112–124, July 1945.

[BZF+16] Emad Barsoum, Cha Zhang, Cristian Canton Ferrer and Zhengyou Zhang. Training Deep Networks for Facial Expression Recognition with Crowd-Sourced Label Distribution. arXiv, August 2016. URL: https://arxiv.org/abs/1608.01041 (visited on 31/10/2017).

[CAL+97] Isabel F. Cruz, Michael Averbuch, Wendy T. Lucas, Melissa Radzyminski and Kirby Zhang. Delaunay: A Database Visualization System. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 510–513, Tucson, USA. ACM, 1997.

[CC96] Tiziana Catarci and Maria Francesca Costabile. Visual Query Systems. *Journal of Visual Languages & Computing*, 7(3):243–245, September 1996.

[CCM+97] Shih-Fu Chang, William Chen, Horace J. Meng, Hari Sundaram and Di Zhong. VideoQ: An Automated Content Based Video Search System Using Visual Cues. In *Proceedings of the International Conference on Multimedia (ACM MM)*, pages 9–13, Seattle, USA. ACM, 1997.

[CCM+98] Shih-Fu Chang, William Chen, Horace J. Meng, Hari Sundaram and Di Zhong. A Fully Automated Content-based Video Search Engine Supporting Spatiotemporal Queries. *Transactions on Circuits and Systems for Video Technology (TCSVT)*, 8(5):602–615, September 1998.

[CCM17] Gabriele Colombo, Paolo Ciuccarelli and Michele Mauri. Visual Geolocations. Repurposing Online Data to Design Alternative Views. *Big Data & Society*, 4(1), June 2017.

[CDM+15] Francesco Colace, Massimo De Santo, Vincenzo Moscato, Antonio Picariello, Fabio A. Schreiber and Letizia Tanca, editors. *Data Management in Pervasive Systems*. Data-Centric Systems and Applications. Springer, Cham, Heidelberg, New York, Dordrecht, London, 2015.

[CFN08] Edgar Chavez Gonzalez, Karina Figueroa and Gonzalo Navarro. Effective Proximity Retrieval by Ordering Permutations. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 30(9):1647–1658, July 2008.

[CG94] Richard L. Cole and Goetz Graefe. Optimization of Dynamic Query Evaluation Plans. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 150–160, Minneapolis, USA. ACM, 1994.

[CGM04] Surajit Chaudhuri, Luis Gravano and Amélie Marian. Optimizing Top-K Selection Queries over Multimedia Repositories. *Transactions on Knowledge and Data Engineering (TKDE)*, 16(8):992–1009, August 2004.

[Cha96] Shi-Kuo Chang. Extending Visual Languages for Multimedia. *MultiMedia*, 3(3):18–26, Fall 1996.

[Che04] Faouzi Alaya Cheikh. *MUVIS: A System for Content-Based Image Retrieval*. PhD Thesis, Tampere University of Technology, Finland, 2004.

[CHS+95] Michael J. Carey, Laura M. Haas, Peter M. Schwarz, Manish Arya, William F. Cody, Ronald Fagin, Myron Flickner, Allen Luniewski, Wayne Niblack, Dragutin Petkovic, Joachim Thomas II, John H. Williams and Edward L. Wimmers. Towards Heterogeneous Multimedia Information Systems: The Garlic Approach. In *Proceedings of the International Workshop on Research Issues in Data Engineering-Distributed Object Management (RIDE-DOM)*, pages 124–131, Taipei, Taiwan. IEEE, 1995.

[CJN+11] K. Selçuk Candan, Jong Wook Kim, Parth Nagarkar, Mithila Nagendra and Renwei Yu. RanKloud: Scalable Multimedia Data Processing in Server Clusters. *MultiMedia*, 18(1):64–77, January 2011.

[CK97] Michael J. Carey and Donald Kossmann. On Saying "Enough already!" in SQL. *Special Interest Group on Management of Data (SIGMOD) Records*, 26(2):219–230, June 1997.

[CMM+00] Ingemar J. Cox, Matt L. Miller, Thomas P. Minka, Thomas V. Papathomas and Peter N. Yianilos. The Bayesian Image Retrieval System, PicHunter: Theory, Implementation, and Psychophysical Experiments. *Transactions on Image Processing (TIP)*, 9(1):20–37, January 2000.

[CMO+96] Ingemar J. Cox, Matt L. Miller, Stephen M. Omohundro and Peter N. Yianilos. PicHunter: Bayesian Relevance Feedback for Image Retrieval. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, pages 361–369, Vienna, Austria. IEEE, 1996.

[CMP+00] Paolo Ciaccia, Danilo Montesi, Wilma Penzo and Alberto Trombetta. Imprecision and User Preferences in Multimedia Queries: A Generic Algebraic Approach. In *Proceedings of the International Symposium on Foundations of Information and Knowledge Systems (FoIKS)*, pages 50–71, Burg, Germany. Springer, 2000.

[CMQ09] John Collomosse, Graham McNeill and Yu Qian. Storyboard Sketches for Content Based Video Retrieval. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 245–252, Kyoto, Japan. IEEE, 2009.

[CN06] Edgar Chávez and Gonzalo Navarro. Metric Databases. In Laura C. Rivero, Jorge H. Doorn and Viviana E. Ferraggine, editors, *Encyclopedia of Database Technologies and Applications*, pages 367–372. Idea Group Reference, Hershey, 2006.

[CN07] Surajit Chaudhuri and Vivek Narasayya. Self-Tuning Database Systems: A Decade of Progress. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 3–14, Vienna, Austria. ACM, 2007.

[CNB+01] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates and José Luis Marroquín. Searching in Metric Spaces. *Computing Surveys (CSUR)*, 33(3):273–321, September 2001.

[Cod70] Edgar F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM (CACM)*, 13(6):377–387, June 1970.

[Cod90] Edgar F. Codd. *The Relational Model for Database Management. Version 2*. Addison-Wesley, Reading, 1990.

[CP02] Paolo Ciaccia and Marco Patella. Searching in Metric Spaces with User-Defined and Approximate Distances. *Transactions on Database Systems (TODS)*, 27(4):398–437, December 2002.

[CPR+07] Flavio Chierichetti, Alessandro Panconesi, Prabhakar Raghavan, Mauro Sozio, Alessandro Tiberi and Eli Upfal. Finding Near Neighbors Through Cluster Pruning. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 103–112, Beijing, China. ACM, 2007.

[CPV+13] Petr Chmelar, Martin Pesek, Tomas Volf, Jaroslav Zendulka and Vojtech Froml. VTApi: An Efficient Framework for Computer Vision Data Management and Analytics. In *Proceedings of the International Conference on Advanced Concepts for Intelligent Vision Systems (ACIVS)*, pages 378–388, Poznań, Poland. Springer, 2013.

[CR94] Chungmin Melvin Chen and Nick Roussopoulos. Adaptive Selectivity Estimation Using Query Feedback. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 161–172, Minneapolis, USA. ACM, 1994.

[CRW05] Surajit Chaudhuri, Raghu Ramakrishnan and Gerhard Weikum. Integrating DB and IR Technologies: What is the Sound of One Hand Clapping? In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, Asilomar, USA. CIDR, 2005.

[CS10] K. Selçuk Candan and Maria Luisa Sapino. *Data Management for Multimedia Retrieval*. Cambridge University Press, Cambridge, 2010.

[CSB+17] Claudiu Cobârzan, Klaus Schöffmann, Werner Bailer, Wolfgang Hürst, Adam Blažek, Jakub Lokoč, Stefanos Vrochidis, Kai Uwe Barthel and Luca Rossetto. Interactive Video Search Tools: A Detailed Analysis of the Video Browser Showdown 2015. *Multimedia Tools and Applications (MTAP)*, 76(4):5539–5571, February 2017.

[CTB+99] Chad Carson, Megan Thomas, Serge Belongie, Joseph M. Hellerstein and Jitendra Malik. Blobworld: A System for Region-based Image Indexing and Retrieval. In *Proceedings of the International Conference on Advances in Visual Information Systems (VISUAL)*, pages 509–516, Amsterdam, The Netherlands. Springer, 1999.

[CWW+10] Yang Cao, Hai Wang, Changhu Wang, Zhiwei Li, Liqing Zhang and Lei Zhang. MindFinder: Interactive Sketch-based Image Search on Millions of Images. In *Proceedings of the International Conference on Multimedia (ACM MM)*, pages 1605–1608, Firenze, Italy. ACM, 2010.

[Dat15] Chris J. Date. *SQL and Relational Theory. How to Write Accurate SQL Code*. O'Reilly, Sebastopol, 3rd edition, 2015.

[DDS+95] Samuel DeFazio, Amjad Daoud, Lisa Ann Smith and Jagannathan Srinivasan. Integrating IR and RDBMS using Cooperative Indexing. In *Proceedings of the International Conference on Information Retrieval (SIGIR)*, pages 84–92, Seattle, USA. ACM, 1995.

[DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM (CACM)*, 51(1):107–113, January 2008.

[DII+04] Mayur Datar, Nicole Immorlica, Piotr Indyk and Vahab S. Mirrokni. Locality Sensitive Hashing Scheme Based on p-Stable Distributions. In *Proceedings of the Annual Symposium on Computational Geometry (SCG)*, pages 253–262, Brooklyn, USA. ACM, 2004.

[DJ11] Jens Dittrich and Alekh Jinda. Towards a One Size Fits All Database Architecture. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 195–198, Asilomar, USA. CIDR, 2011.

[DJL+08] Ritendra Datta, Dhiraj Joshi, Jia Li and James Z. Wang. Image Retrieval: Ideas, Influences, and Trends of the New Age. *Computing Surveys (CSUR)*, 40(2), April 2008.

[DKN08] Thomas Deselaers, Daniel Keysers and Hermann Ney. Features for Image Retrieval: An Experimental Comparison. *Information Retrieval*, 11(2):77–107, December 2008.

[DM11] Neil Day and José M. Martínez. Introduction to MPEG-7 (v.3.0). 2011. URL: https://www.w3.org/2001/05/mpeg7/w4032.doc (visited on 31/10/2017).

[dMP+98] Alberto del Bimbo, Mauro Mugnaini, Pietro Pala and Francesco Turco. Visual Querying by Color Perceptive Regions. *Pattern Recognition*, 31(9):1241–1253, September 1998.

[Döl04] Mario Döller. *The MPEG-7 Multimedia Database System (MPEG-7 MMDB)*. PhD Thesis, Universität Klagenfurt, Germany, 2004.

[dP96] Alberto del Bimbo and Pietro Pala. Image indexing using shape-based visual features. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, pages 351–355, Vienna, Austria. IEEE, 1996.

[dP97] Alberto del Bimbo and Pietro Pala. Visual Image Retrieval by Elastic Matching of User Sketches. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 19(2):121–132, February 1997.

[Düb15] Pascal Düblin. *SanDBoX — A Modular Component-based and Scalable Multimedia Information Retrieval Framework*. Master Thesis, University of Basel, 2015.

[DZE+15] Jennie Duggan, Stan Zdonik, Aaron J. Elmore, Michael Stonebraker, Magda Balazinska, Bill Howe, Jeremy Kepner, Sam Madden, David Maier and Tim Mattson. The BigDAWG Polystore System. *Special Interest Group on Management of Data (SIGMOD) Record*, 44(2):11–16, August 2015.

[DZS+02] Nevenka Dimitrova, Hong Jiang Zhang, Behzad Shahraray, Ibrahim Sezan, Thomas S. Huang and Avideh Zakhor. Applications of Video-Content Analysis and Retrieval. *MultiMedia*, 9(3):42–55, July 2002.

[Eak96] John P Eakins. Automatic Image Content Retrieval — Are we Getting Anywhere? In *Proceedings of the International Conference on Electronic Library and Visual Information Research (ELVIRA)*, pages 123–135, Milton Keynes, UK, 1996.

[Elt17] Mohamed Y. Eltabakh. Data Organization and Curation in Big Data. In Albert Y. Zomaya and Sherif Sakr, editors, *Handbook of Big Data Technologies*, pages 143–178. Springer, Cham, 2017.

[EMH+06] Martin Eisenhardt, Wolfgang Muller, Andreas Henrich, Daniel Blank and Soufyane Allali. Clustering-Based Source Selection for Efficient Image Retrieval in Peer-to-Peer Networks. In *Proceedings of the International Symposium on Multimedia (ISM)*, pages 823–830, San Diego, USA. IEEE, 2006.

[EN11] Ramez Elmasri and Shamkant B. Navathe. *Fundamenthals of Database Systems*. Addison-Wesley, Pearson, Boston, 6th edition, 2011.

[Fag99] Ronald Fagin. Combining Fuzzy Information from Multiple Systems. *Journal of Computer and System Sciences*, 58(1):83–99, February 1999.

[FB74] Raphael Ari Finkel and Jon Louis Bentley. Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Informatica*, 4(1):1–9, March 1974.

[FC13] Fausto C. Fleites and Shu-Ching Chen. Efficient Content-based Multimedia Retrieval Using Novel Indexing Structure in PostgreSQL. In *Proceedings of the International Symposium on Multimedia (ISM)*, pages 500–501, Anaheim, USA. IEEE, 2013.

[FKS03] Ronald Fagin, Ravi Kumar and D. Sivakumar. Efficient Similarity Search and Classification via Rank Aggregation. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 301–312, San Diego, USA. ACM, 2003.

[FLN01] Ronald Fagin, Amnon Lotem and Moni Naor. Optimal Aggregation Algorithms for Middleware. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 102–113, Santa Barbara, USA. ACM, 2001.

[FSN+95] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele and Peter Yanker. Query by Image and Video Content: The QBIC System. *Computer*, 28(9):23–32, September 1995.

[FST+11] Mônica Ribeiro Porto Ferreira, Lucio Fernandes Dutra Santos, Agma Juci Machado Traina, Ires Dias, Richard Chbeir and Caetano Traina Jr. Algebraic Properties to Optimize kNN Queries. *Journal of Information and Data Management*, 2(3):385–400, October 2011.

[FTA+06] Hakan Ferhatosmanoglu, Ertem Tuncel, Divyakant Agrawal and Amr El Abbadi. High Dimensional Nearest Neighbor Searching. *Information Systems*, 31(6):512–540, September 2006.

[Fuh12] Norbert Fuhr. Salton Award Lecture: Information Retrieval as Engineering Science. *Special Interest Group on Information Retrieval (SIGIR) Forum*, 46(2):19–28, December 2012.

[Fuh14] Norbert Fuhr. Bridging Information Retrieval and Databases. In Nicola Ferro, editor, *Bridging Between Information Retrieval and Databases. PROMISE Winter School 2013*, pages 97–115. Springer, Bressanone, Italy, 2014.

[GAJ+17] Gylfi Þ. Guðmundsson, Laurent Amsaleg, Björn Þ. Jónsson and Michael J. Franklin. Towards Engineering a Web-Scale Multimedia Service. In *Proceedings of the International Conference on Multimedia Systems (MMSys)*, pages 1–12, Taipei, Taiwan. ACM, 2017.

[Gar09] Hector Garcia-Molina. *Database Systems: The Complete Book*. Pearson, Prentice-Hall, Upper Saddle River, 2nd edition, 2009.

[GAS14a] Ivan Giangreco, Ihab Al Kabary and Heiko Schuldt. ADAM — A Database and Information Retrieval System for Big Multimedia Collections. In *Proceedings of the International Congress on Big Data (BigData)*, pages 406–413, Anchorage, USA. IEEE, 2014.

[GAS14b] Ivan Giangreco, Ihab Al Kabary and Heiko Schuldt. ADAM — A System for Jointly Providing IR and Database Queries in Large-Scale Multimedia Retrieval. In *Proceedings of the International Conference on Information Retrieval (SIGIR)*, pages 1257–1258, Gold Coast, Australia. ACM, 2014.

[Gas17] Ralph Gasser. *Towards an All-Purpose, Content-Based Multimedia Information Retrieval System*. Master Thesis, University of Basel, Switzerland, 2017.

[GBS04] Torsten Grabs, Klemens Böhm and Hans-Jörg Schek. PowerDB-IR — Scalable Information Retrieval and Storage with a Cluster of Databases. *Knowledge and Information Systems*, 6(4):465–505, July 2004.

[GfdS17] Gesellschaft für deutsche Sprache e.V. (GfdS). Wort des Jahres, 2017. URL: http://gfds.de/aktionen/wort-des-jahres/ (visited on 18/08/2017).

[GGR+17] Prateek Goel, Ivan Giangreco, Luca Rossetto, Claudiu Tănase and Heiko Schuldt. "Hey, vitrivr!" — A Multimodal UI for Video Retrieval. In *Proceedings of the European Conference on Information Retrieval (ECIR)*, pages 749–752, Aberdeen, UK. Springer, 2017.

[Gia13] Ivan Giangreco. *ADAM — A Database and Information Retrieval System for Storing, Organizing and Retrieving Multimedia Data*. Master Thesis, University of Basel, Switzerland, 2013.

[GIM99] Aristides Gionis, Piotr Indyk and Rajeev Motwani. Similarity Search in High Dimensions via Hashing. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 518–529, Edinburgh, Scotland. Morgan Kaufmann, 1999.

[GJA10] Gylfi Þ. Gudmundsson, Björn Þ. Jónsson and Laurent Amsaleg. A Large-Scale Performance Study of Cluster-based High-Dimensional Indexing. In *Proceedings of the International Workshop on Very-Large-Scale Multimedia Corpus, Mining and Retrieval (VLS-MCMR)*, pages 31–36, Firenze, Italy. ACM, 2010.

[GKC+02] Moncef Gabbouj, Serkan Kiranyaz, Kerem Caglar, Bogdan Cramariuc, Faouzi Alaya Cheikh, Olcay Guldogan and Esin Karaoglu. MUVIS: A Multimedia Browsing, Indexing and Retrieval Framework. In *Proceedings of the Workshop on Digital Communications (IWDC), International Conference on Advanced Methods for Multimedia Signal Processing*, pages 379–383, Capri, Italy. IEEE, 2002.

[GMR+09] Denise Guliato, Ernani V. de Melo, Rangaraj M. Rangayyan and Robson C. Soares. POSTGRESQL-IE: An Image-handling Extension for PostgreSQL. *Journal of Digital Imaging (JDI)*, 22(2):149–165, April 2009.

[Goe16] Prateek Goel. *Natural Language Query Interface for vitrivr*. Google Summer of Code Project, University of Basel, 2016.

[Gro97] William I. Grosky. Managing Multimedia Information in Database Systems. *Communications of the ACM (CACM)*, 40(12):72–80, December 1997.

[GRT+17] Ivan Giangreco, Luca Rossetto, Claudiu Tănase and Heiko Schuldt. vitrivr: A Multimedia Search System supporting Multimodal Interactions. In *Proceedings of the International Conference on Multimodal Communication: Developing New Theories and Methods*, Osnabrück, Germany, 2017.

[GS16] Ivan Giangreco and Heiko Schuldt. ADAM$_{pro}$: Database Support for Big Multimedia Retrieval. *Datenbank-Spektrum*, 16(1):17–26, March 2016.

[GSA+12] Ivan Giangreco, Michael Springmann, Ihab Al Kabary and Heiko Schuldt. A User Interface for Query-by-Sketch Based Image Retrieval with Color Sketches. In *Proceedings of the European Conference on In-*

formation Retrieval (ECIR), pages 571–572, Barcelona, Spain. Springer, 2012.

[Gut84] Antonin Guttman. R-Trees — A Dynamic Index Structure for Spatial Searching. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 47–57, Boston, USA. ACM, 1984.

[HAK00] Alexander Hinneburg, Charu C. Aggarwal and Daniel A. Keim. What is the Nearest Neighbor in High Dimensional Spaces? In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 506–515, Cairo, Egypt. Morgan Kaufmann, 2000.

[HBH+04] Samira Hammiche, Salima Benbernou, Mohand-Saïd Hacid and Athena Vakali. Semantic Retrieval of Multimedia Data. In *Proceedings of the International Workshop on Multimedia Databases (MMDB)*, pages 36–44, Washington, USA. ACM, 2004.

[Hel16] Silvan Heller. *Index-Partitioning in the Distributed Database System $ADAM_{pro}$*. Bachelor Thesis, University of Basel, Switzerland, 2016.

[HG14] Rakebul Hasan and Fabien Gandon. A Machine Learning Approach to SPARQL Query Performance Prediction. In *Proceedings of the International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, pages 266–273, Warsaw, Poland. IEEE, 2014.

[HMR96] Tom Huang, Sharad Mehrotra and Kannan Ramchandran. Multimedia Analysis and Retrieval System (MARS) Project. In *Proceedings of the Conference on Clinic on Library Applications of Data Processing*, pages 100–117, Illinois, USA. Graduate School of Library and Information Science, University of Illinois at Urbana-Champaign, 1996.

[HS02] Arvind Hulgeri and S. Sudarshan. Parametric Query Optimization for Linear and Piecewise Linear Cost Functions. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 167–178, Hong Kong. Morgan Kaufmann, 2002.

[HS03] Gisli R. Hjaltason and Hanan Samet. Index-Driven Similarity Search in Metric Spaces. *Transactions on Database Systems (TODS)*, 28(4):517–580, December 2003.

[HSH07] Joseph M. Hellerstein, Michael Stonebraker and James Hamilton. Architecture of a Database System. *Foundations and Trends in Databases*, 1(2):141–259, November 2007.

[HWH15] Wolfgang Hürst, Rob van de Werken and Miklas Hoet. A Storyboard-Based Interface for Mobile Video Browsing. In *Proceedings of the International Conference on Multimedia Modeling (MMM)*, pages 261–265, Sydney, Australia. Springer, 2015.

[IKM07] Stratos Idreos, Martin L. Kersten and Stefan Manegold. Database Cracking. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, pages 68–78, Asilomar, USA. CIDR, 2007.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, pages 604–613, Dallas, USA. ACM, 1998.

[Inf97] Informix. Getting Started with INFORMIX-Universal Server. Technical report, Novell Inc., Menlo Park, USA, 1997. URL: `http://publibfp.dhe.ibm.com/epubs/pdf/3890.pdf` (visited on 31/10/2017).

[ISO13249-5:2003] International Organization for Standardization (ISO) / International Electrotechnical Commission (IEC). ISO/IEC 13249-5:2003 — Information Technology — Database Languages — SQL Multimedia and Application Packages — Part 5: Still Image, 2003.

[ISO13249:2003] International Organization for Standardization (ISO) / International Electrotechnical Commission (IEC). ISO/IEC 13249:2003 — Information Technology — Database Languages — SQL Multimedia and Application Packages, 2003.

[ISO15938-3:2002] International Organization for Standardization (ISO) / International Electrotechnical Commission (IEC). ISO/IEC 15938-3:2002 — Information technology — Multimedia Content Description Interface, 2002.

[JDJ17] Jeff Johnson, Matthijs Douze and Hervé Jégou. Billion-scale Similarity Search with GPUs. arXiv, February 2017. URL: `https://arxiv.org/abs/1702.08734` (visited on 31/10/2017).

[JDS11] Hervé Jégou, Matthijs Douze and Cordelia Schmid. Product Quantization for Nearest Neighbor Search. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 33(1):117–128, January 2011.

[Jég12] Hervé Jégou. Slides: "Large-Scale Visual Recognition — Part 1: Efficient Matching". 2012. URL: `https://sites.google.com/site/lsvrtutorialcvpr14/home/efficient-matching` (visited on 30/10/2017).

[JFS95] Charles E. Jacobs, Adam Finkelstein and David H. Salesin. Fast Multiresolution Image Querying. In *Proceedings of the International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 277–286, Los Angeles, USA. ACM, 1995.

[JHS07] Hervé Jégou, Hedi Harzallah and Cordelia Schmid. A Contextual Dissimilarity Measure for Accurate and Efficient Image Search. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, Minneapolis, USA. IEEE, 2007.

[JJ05] Corinne Jörgensen and Peter Jörgensen. Image Querying by Image Professionals. *Journal of the American Society for Information Science and Technology*, 56(12):1346–1359, October 2005.

[JRU14] Leskovec Jure, Anand Rajaraman and Jeff Ullman. *Mining of Massive Datasets*. Cambridge University Press, Cambridge, 2014.

[JSD+14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the International Conference on Multimedia (ACM MM)*, pages 675–678, Orlando, USA. ACM, 2014.

[JSX+10] Jiong Xie, Shu Yin, Xiaojun Ruan, Zhiyang Ding, Yun Tian, James Majors, Adam Manzanares and Xiao Qin. Improving MapReduce performance through data placement in heterogeneous Hadoop clusters. In *Proceedings of the International Symposium on Parallel & Distributed Processing, Workshops and PhD Forum (IPDPSW)*, Atlanta, USA. IEEE, 2010.

[JWZ+16] Björn Þ. Jónsson, Marcel Worring, Jan Zahálka, Stevan Rudinac and Laurent Amsaleg. Ten Research Questions for Scalable Multimedia Analytics. In *Proceedings of the International Conference on Multimedia Modeling (MMM)*, pages 290–302, Miami, USA. Springer, 2016.

[JZS+16] Wang Jingdong, Ting Zhang, Jingkuan Song, Nicu Sebe and Heng Tao Shen. A Survey on Learning to Hash. arXiv, June 2016. URL: `https://arxiv.org/abs/1606.00185` (visited on 31/10/2017).

[KAG10] Rajkumar Kannan, Frederic Andres and Christian Guetl. DanVideo: An MPEG-7 Authoring and Retrieval System for Dance Videos. *Multimedia Tools and Applications (MTAP)*, 46(2-3):545–572, January 2010.

[Kalo0] Oya Kalipsiz. Multimedia Databases. In *Proceedings of the International Conference on Information Visualization (iV)*, pages 111–115, London, UK. IEEE, 2000.

[KB96] Setrag Khoshafian and Brad A. Baker. *MultiMedia and Imaging Databases*. Morgan Kaufmann, San Francisco, USA, 1996.

[KBC+17] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean and Neoklis Polyzotis. The Case for Learned Index Structures. arXiv, December 2017. URL: https://arxiv.org/abs/1712.01208 (visited on 30/12/2017).

[KBP+11] Daniel S. Kaster, Pedro H. Bugatti, Marcelo Ponciano-Silva, Agma J. M. Traina, Paulo M. A. Marques, Antonio C. Santos and Caetano Traina Jr. MedFMI-SiR: A Powerful DBMS Solution for Large-Scale Medical Image Retrieval. In *Proceedings of the International Conference on Information Technology in Bio- and Medical Informatics (ITBAM)*, pages 16–30, Toulouse, France. Springer, 2011.

[KBT+09] Daniel S. Kaster, Pedro H. Bugatti, Agma J. M. Traina and Caetano Traina Jr. Incorporating Metric Access Methods for Similarity Searching on Oracle Database. In *Proceedings of the Brazilian Symposium on Databases (SBBD)*, Fortalez, Brazil. Brazilian Computer Society (SBC), 2009.

[KG05] Serkan Kiranyaz and Moncef Gabbouj. Novel Multimedia Retrieval Technique: Progressive Query (Why Wait?) *IEE Proceedings - Vision, Image, and Signal Processing*, 152(3):356–366, July 2005.

[KHZ09] Martin Kampel, Reinhold Huber-Mörk and Maia Zaharieva. Image-Based Retrieval and Identification of Ancient Coins. *Intelligent Systems*, 24(2):26–34, March 2009.

[Kir16] Anton Kirillov. Apache Spark: Core Concepts, Architecture and Internals, 2016. URL: http://datastrophic.io/core-concepts-architecture-and-internals-of-apache-spark/ (visited on 16/08/2017).

[KKO+92] Toshikazu Kato, Takio Kurita, Nobuyuki Otsu and Kyoji Hirata. A Sketch Retrieval Method for Full Color Image Database. Query by Visual Example. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, pages 530–533, The Hague, The Netherlands. IEEE, 1992.

[KKW+15] Holden Karau, Andy Konwinski, Patrick Wendell and Matei Zaharia. *Learning Spark. Lightning-Fast Data Analysis*. O'Reilly, Beijing, 2015.

[KL16] Andrej Karpathy and Fei Fei Li. Deep Visual-Semantic Alignments for Generating Image Descriptions. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 39(4):664–676, August 2016.

[KNK+99] Naoko Kosugi, Yuichi Nishihara, Seiichi Kon'ya, Masashi Yamamuro and Kazuhiko Kushima. Music Retrieval by Humming. Using Similarity Retrieval over High-Dimensional Feature Vector Space. In *Proceedings of the Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, pages 404–407, Victoria, Canada. IEEE, 1999.

[Kos00] Donald Kossmann. The State of the Art in Distributed Query Processing. *Computing Surveys (CSUR)*, 32(4):422–469, December 2000.

[Kos02] Harald Kosch. MPEG-7 and Multimedia Database Systems. *Special Interest Group on Management of Data (SIGMOD) Records*, 31(2):34–39, June 2002.

[KPB+16] Pooya Khorrami, Tom Le Paine, Kevin Brady, Charlie Dagli and Thomas S. Huang. How Deep Neural Networks Can Improve Emotion Recognition on Video Data. arXiv, February 2016. URL: https://arxiv.org/abs/1602.07377 (visited on 31/10/2017).

[KPZ+04] Eamonn Keogh, Thermistoklis Palpanas, Victor B. Zordan, Dimitrios Gunopulos and Marc Cardle. Indexing Large Human-Motion Databases. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 780–791, Toronto, Canada. Morgan Kaufmann, 2004.

[Kra05] Marcel Kratochvil. The Move to Store Images In the Database, 2005. URL: http://www.oracle.com/technetwork/testcontent/why-images-in-database-100783.html (visited on 31/10/2017).

[KS87] Hans-Peter Kriegel and Bernhard Seeger. Multidimensional Dynamic Quantile Hashing Is Very Efficient for Non-Uniform Record Distributions. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 10–17, Los Angeles, USA. IEEE, 1987.

[KS97] Norio Katayama and Shin'ichi Satoh. The SR-Tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 369–380, Tucson, USA. ACM, 1997.

[KSH12] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, Lake Tahoe, USA. NIPS, 2012.

[KSR+06] David Kirk, Abigail Sellen, Carsten Rother and Ken Wood. Understanding Photowork. In *Proceedings of the International Conference on Human Factors in computing systems (CHI)*, pages 761–770, Montreal, Canada. ACM, 2006.

[KSS14] Veit Koppen, Martin Schäler and Reimar Schroter. Toward Variability Management to Tailor High-Dimensional Index Implementations. In *Proceedings of the International Conference on Research Challenges in Information Science (RCIS)*, Marrakech, Morocco. IEEE, 2014.

[KT92] Hirata Kyoji and Kato Toshikazu. Query by Visual Example. Content based Image Retrieval. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 56–71, Vienna, Austria. Springer, 1992.

[KTH+12] Anagha Kulkarni, Almer S. Tigelaar, Djoerd Hiemstra and Jamie Callan. Shard Ranking and Cutoff Estimation for Topically Partitioned Collections. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 555–564, Maui, USA. ACM, 2012.

[KY95] George J. Klir and Bo Yuan. *Fuzzy Sets And Fuzzy Logic: Theory And Applications*. Prentice-Hall, Upper Saddle River, 1995.

[LA96] Marco La Cascia and Edoardo Ardizzone. JACOB: Just a Content-based Query System for Video Databases. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 1216–1219, Atlanta, USA. IEEE, 1996.

[Lan01] Doug Laney. 3D Data Management: Controlling Data Volume, Velocity, and Variety. 2001. URL: https://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf (visited on 31/10/2017).

[Lan10] Tom Lane. *A Tour of PostgreSQL Internals*. 2010. URL: https://www.postgresql.org/files/developer/tour.pdf (visited on 28/06/2017).

[LBS14] Jakub Lokoč, Adam Blažek and Tomáš Skopal. Signature-Based Video Browser. In *Proceedings of the International Conference on Multimedia Modeling (MMM)*, pages 415–418, Dublin, Ireland. Springer, 2014.

[LC08a] Jeongkyu Lee and Mehmet Emre Celebi. STRG-QL: Spatio-Temporal Region Graph Query Language for Video Databases. In *Proceedings of SPIE - Multimedia Content Access: Algorithm and Systems II*, San Jose, USA. SPIE, 2008.

[LC08b] Mathias Lux and Savvas A. Chatzichristofis. LIRe: Lucene Image Retrieval. In *Proceedings of the International Conference on Multimedia (ACM MM)*, pages 1085–1088, Vancouver, Canada. ACM, 2008.

[LCI+05] Chengkai Li, Chen-Chuan Kevin Chang, Ihab F. Ilyas and Sumin Song. RankSQL: Query Algebra and Optimization for Relational Top-k Queries. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 131–142, Baltimore, USA. ACM, 2005.

[LCW+14] Wei Kuang Lai, Yi-Uan Chen, Tin-Yu Wu and Mohammad S. Obaidat. Towards a Framework for Large-Scale Multimedia Data Storage and Processing on Hadoop Platform. *The Journal of Supercomputing*, 68(1):488–507, April 2014.

[LCW02] Baitao Li, Edward Chang and Ching-Tung Wu. DPF — A Perceptual Distance Function for Image Retrieval. In *Proceedings of the International Conference on Image Processing (ICIP)*, pages 597–600, Rochester, USA. IEEE, 2002.

[Lin08] Jimmy Lin. Slides: "Session 10: Information Retrieval". 2008. URL: https://github.com/lintool/UMD-courses/blob/master/INFM603-2014f/slides/session10.pdf (visited on 31/10/2017).

[LJF94] King-Ip Lin, Hosagrahar Visvesvaraya Jagadish and Christos Faloutsos. The TV-Tree: An Index Structure for High-Dimensional Data. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 517–542, Santiago de Chile, Chile. Morgan Kaufmann, 1994.

[LJW+06] Qin Lv, William Josephson, Zhe Wang, Moses Charikar and Kai Li. Ferret: A Toolkit for Content-based Similarity Search of Feature-Rich Data. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, pages 317–330, Leuven, Belgium. ACM, 2006.

[LJW+07] Qin Lv, William Josephson, Zhe Wang, Moses Charikar and Kai Li. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 950–961, Vienna, Austria. ACM, 2007.

[LK00] Jonathan K. Lawder and Peter J. H. King. Using Space-Filling Curves for Multi-dimensional Indexing. In *Proceedings of the British National Conference on Databases (BNCOD)*, pages 20–35, Exeter, UK. Springer, 2000.

[LKO02] Jorma Laaksonen, Markus Koskela and Erkki Oja. Picsom — self-organizing image retrieval with mpeg-7 content descriptors. *Transactions on Neural Networks*, 13(4):841–853, July 2002.

[LLN+13] Duy-Dinh Le, Vu Lam, Thanh Duc Ngo, Vinh Quang Tran, Vu Hoang Nguyen, Duc Anh Duong and Shin'ichi Satoh. NII-UIT-VBS: A Video Browsing Tool for Known Item Search. In *Proceedings of the International Conference on Multimedia Modeling (MMM)*, pages 547–549, Huangshan, China. Springer, 2013.

[LM13] Mathias Lux and Oge Marques. *Visual Information Retrieval Using Java and LIRE*. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool, 2013.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick and Piotr Dollár. Microsoft COCO: Common Objects in Context. arXiv, May 2014. URL: `https://arxiv.org/abs/1405.0312` (visited on 31/10/2017).

[LMW+17] Shusen Liu, Dan Maljovec, Bei Wang, Peer-Timo Bremer and Valerio Pascucci. Visualizing High-Dimensional Data: Advances in the Past Decade. *Transactions on Visualization and Computer Graphics (TVCG)*, 23(3):1249–1268, March 2017.

[Low99] David G. Lowe. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 1150–1157, Kerkyra, Greece. IEEE, 1999.

[LR95] Michael Löhr and Thomas C. Rakow. Audio Support for an Object-Oriented Database-Management System. *Multimedia Systems*, 3(5):286–297, November 1995.

[LS16] Paul Lee and Duncan Stewart. Photo Sharing: Trillions and Rising. Technical report, Technology, media, and telecommunications (TMT), Deloitte, 2016. URL: https://www2.deloitte.com/content/dam/Deloitte/global/Documents/Technology-Media-Telecommunications/gx-tmt-prediction-online-photo-sharing.pdf.

[LSC+12] Wei Lu, Yanyan Shen, Su Chen and Beng Chin Ooi. Efficient Processing of k Nearest Neighbor Joins using MapReduce. *Proceedings of the VLDB Endowment (PVLDB)*, 5(10):1016–1027, June 2012.

[LSE02] Michael S. Lew, Nicu Sebe and John P. Eakins. Challenges of Image and Video Retrieval. In *Proceedings of the International Conference on Image and Video Retrieval (CIVR)*, London, UK. Springer, 2002.

[MAA+15] Anastasia Moumtzidou, Konstantinos Avgerinakis, Evlampios Apostolidis, Fotini Markatopoulou, Konstantinos Apostolidis, Theodoros Mironidis, Stefanos Vrochidis, Vasileios Mezaris, Ioannis Kompatsiaris and Ioannis Patras. VERGE: A Multimodal Interactive Video Search Engine. In *Proceedings of the International Conference on Multimedia Modeling (MMM)*, pages 249–254, Sydney, Australia. Springer, 2015.

[MBN+06] Sebastian Michel, Matthias Bender, Nikos Ntarmos, Peter Triantafillou, Gerhard Weikum and Christian Zimmer. Discovering and Exploiting Keyword and Attribute-Value Co-Occurrences to Improve P2P Routing Indices. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 172–181, Arlington, USA. ACM, 2006.

[ME01] Jim Melton and Andrew Eisenberg. SQL Multimedia and Application Packages (SQL/MM). *Special Interest Group on Management of Data (SIGMOD) Records*, 30(4):97–102, December 2001.

[Mea80] Donald Meagher. Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer. Technical report, Rensselaer Polytechnic Institute. Image Processing Laboratory, 1980.

[MH03] Wolfgang Müller and Andreas Henrich. Fast Retrieval of High-Dimensional Feature Vectors in P2P Networks Using Compact Peer Data Summaries. In *Proceedings of the International Workshop on Multimedia Information Retrieval (MIR)*, pages 79–86, Berkeley, USA. ACM, 2003.

[ML14] Marius Muja and David G. Lowe. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 36(11):2227–2240, November 2014.

[MRT91] Carlo Meghini, Fausto Rabitti and Costantino Thanos. Conceptual Modeling of Multimedia Documents. *Computer*, 24(10):23–30, October 1991.

[MS98] Marjo Markkula and Eero Sormunen. Searching for Photos — Journalists' Practices in Pictorial IR. In *Proceedings of the Workshop and Symposium on the Challenge of Image Retrieval*, Newcastle, UK. British Computer Society, 1998.

[MSG+13a] Diana Moise, Denis Shestakov, Gylfi Þ. Gudmundsson and Laurent Amsaleg. Indexing and Searching 100M Images with Map-Reduce. In *Proceedings of the International Conference on Multimedia Retrieval (ICMR)*, pages 17–24, Dallas, USA. ACM, 2013.

[MSG+13b] Diana Moise, Denis Shestakov, Gylfi Þ. Gudmundsson and Laurent Amsaleg. Terabyte-scale Image Similarity Search: Experience and Best Practice. In *Proceedings of the International Conference on Big Data (Big Data)*, pages 674–682, Santa Clara, USA. IEEE, 2013.

[MSL+14] Hannes Mühleisen, Thaer Samar, Jimmy Lin and Arjen de Vries. Old Dogs are Great at New Tricks: Column Stores for IR Prototyping. In *Proceedings of the International Conference on Information Retrieval (SIGIR)*, pages 863–866, Gold Coast, Australia. ACM, 2014.

[MSS02] B.S. Manjunath, Philippe Salembier and Thomas Sikora, editors. *Introduction to MPEG-7: Multimedia Content Description Interface*. John Wiley & Sons, Chichester, 2002.

[Mul16] Multimedia Commons Initiative. YFCC100M Core Dataset, 2016. URL: `https://multimediacommons.wordpress.com/yfcc100m-core-dataset/` (visited on 19/12/2017).

[Nar96] Arcot D. Narasimhalu. Multimedia Databases. *Multimedia Systems*, 4(5):226–249, October 1996.

[NBE+93] Carlton W. Niblack, Ron Barber, Will Equitz, Myron D. Flickner, Eduardo H. Glasman, Dragutin Petkovic, Peter Yanker, Christos Faloutsos and Gabriel Taubin. QBIC Project: Querying Images by Content, using Color, Texture, and Shape. In *Proceedings of the International Symposium on Electronic Imaging*, number 1, pages 173–187, San Jose, USA. SPIE, 1993.

[NLF+15] Frank Austin Nothaft, Michael Linderman, Michael J. Franklin, Anthony D. Joseph, David A. Patterson, Matt Massie, Timothy Danford, Zhao Zhang, Uri Laserson, Carl Yeksigian, Jey Kottalam, Arun Ahuja and Jeff Hammerbacher. Rethinking Data-Intensive Science Using Scalable Analytics Systems. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 631–646, Melbourne, Australia. ACM, 2015.

[NYF08] Linh Thai Nguyen, Wai Gen Yee and Ophir Frieder. Adaptive Distributed Indexing for Structured Peer-to-Peer Networks. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 1241–1250, Napa Valley, USA. ACM, 2008.

[OÖI01] Vincent Oria, M. Tamer Özsu and Paul J. Inglinski. Querying Images in the DISIMA DBMS. In *Proceedings of the International Workshop on Multimedia Information Systems (MIS)*, pages 19–28, Como, Italy, 2001.

[OÖL+01] Vincent Oria, M. Tamer Özsu, Shu Lin and Paul J. Iglinski. Similarity Queries in the DISIMA Image DBMS. In *Proceedings of the International Conference on Multimedia (MULTIMEDIA)*, pages 475–478, Ottawa, Canada. ACM, 2001.

[OÖL+97] Vincent Oria, M. Tamer Özsu, Ling Liu, Xaiobo Li, John Z. Li, Youping Niu and Paul J. Iglinski. Modeling Images for Content-based Queries: The DISIMA Approach. In *Proceedings of the International Conference on Visual Information Systems (VISUAL)*, pages 339–346, San Diego, USA. Springer, 1997.

[Ora10ga] Oracle Corp. *New interMedia Features in Oracle10g*. URL: `http://www.oracle.com/technetwork/developer-tools/jdev/imedia-new-features-in-10g-092186.html` (visited on 31/10/2017).

[Ora10gb] Oracle Corp. *Oracle interMedia User Guide, Online Documentation for Oracle 10g, Release 2 (10.2)*. 2015. URL: `http://docs.oracle.com/cd/B19306_01/appdev.102/b14302/ch_intr.htm` (visited on 01/06/2017).

[ORC+98] Michael Ortega, Yong Rui, Kaushik Chakrabarti, Kriengkrai Porkaew, Sharad Mehrotra and Thomas S. Huang. Supporting Ranked Boolean Similarity Queries in MARS. *Transactions on Knowledge and Data Engineering (TKDE)*, 10(6):905–925, November 1998.

[OS95] Virginia E. Ogle and Michael Stonebraker. Chabot: Retrieval from a Relational Database of Images. *Computer*, 28(9):40–48, September 1995.

[OT93] E. Oomoto and K. Tanaka. OVID: Design and Implementation of a Video-Object Database System. *Transactions on Knowledge and Data Engineering (TKDE)*, 5(4):629–643, August 1993.

[ÖV11] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Springer, New York, Dordrecht, Heidelberg, London, 3rd edition, 2011.

[Özs99] M. Tamer Özsu. Issues in Multimedia Database Management. In *Proceedings of the International Database Engineering and Applications Symposium (IDEAS)*, pages 452–459, Montreal, Canada. IEEE, 1999.

[Pan05] Rina Panigrahy. Entropy based Nearest Neighbor Search in High Dimensions. In *Proceedings of the International Symposium on Discrete Algorithms (SODA)*, pages 1186–1195, Miami, USA. ACM, 2005.

[PC09] Marco Patella and Paolo Ciaccia. Approximate Similarity Search: A Multi-Faceted Problem. *Journal of Discrete Algorithms*, 7(1):36–48, March 2009.

[PCI+08] James Philbin, Ondřej Chum, Michael Isard, Josef Sivic and Andrew Zisserman. Lost in Quantization: Improving Particular Object Retrieval in Large Scale Image Databases. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, Anchorage, USA. IEEE, 2008.

[PGS16] Lukas Probst, Ivan Giangreco and Heiko Schuldt. PAN — Distributed Real-Time Complex Event Detection in Multiple Data Streams. In *Proceedings of the International Conference on Distributed Applications and Interoperable Systems (DAIS)*, pages 189–195, Heraklion, Greece. Springer, 2016.

[PJ00] Milan Petković and Willem Jonker. A Framework for Video Modeling. In *Proceedings of the International Conference on Applied Informatics (AI)*, Innsbruck, Austria. IASTED, 2000.

[PMJ02] Milan Petković, Vojkan Mihajlović and Willem Jonker. Extending a DBMS to Support Content-Based Video Retrieval: A Formula 1 Case Study. In *Proceedings of the International Workshop on Multimedia Data Document Engineering*, pages 318–341, Prague, Czech Republic. Springer, 2002.

[PPS93] Alexander P. Pentland, Rosalind W. Picard and Stanley Sclaroff. Photobook: Content-based Manipulation of Image Databases. Technical report 3, Massachusetts Institute of Technology, Cambridge, USA, 1993, pages 233–254.

[PPS96] Alexander P. Pentland, Rosalind W. Picard and Stanley Sclaroff. Photobook: Content-based Manipulation of Image Databases. *International Journal of Computer Vision*, 18(3):233–254, June 1996.

[Pro14] Lukas Probst. *PAN — A P2P Approach for Scalable Complex Event Detection in Distributed Data Streams*. Master Thesis, University of Basel, 2014.

[PSA+98] Dulce Ponceleon, Savitha Srinivasan, Arnon Amir, Dragutin Petkovic and Dan Diklic. Key to Effective Video Retrieval: Effective Cataloging and Browsing. In *Proceedings of the International Conference on Multimedia (MULTIMEDIA)*, pages 99–107, Bristol, UK. ACM, 1998.

[PY01] Christos H. Papadimitriou and Mihalis Yannakakis. Multiobjective Query Optimization. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 52–59, Santa Barbara, USA. ACM, 2001.

[RGG+17] Luca Rossetto, Ivan Giangreco, Ralph Gasser and Heiko Schuldt. Open-Source Column: Content-based Multimedia Retrieval using vitrivr. *Special Interest Group on Multimedia (SIGMM) Records*, 9(3), December 2017.

[RGG+18] Luca Rossetto, Ivan Giangreco, Ralph Gasser and Heiko Schuldt. Competitive Video Retrieval with vitrivr. In *Proceedings of the International Conference on Multimedia Modeling (MMM)*, volume 97861V, pages 403–406, Bangkok, Thailand. Springer, 2018.

[RGH+16a] Luca Rossetto, Ivan Giangreco, Silvan Heller, Claudiu Tănase and Heiko Schuldt. Searching in Video Collections Using Sketches and Sample Images — The Cineast System. In *Proceedings of the International Conference on Multimedia Modeling (MMM)*, pages 336–341, Miami, USA. Springer, 2016.

[RGH+16b] Luca Rossetto, Ivan Giangreco, Silvan Heller, Claudiu Tănase, Heiko Schuldt, Stéphane Dupont, Omar Seddati, T. Metin Sezgin, Ozan Can Altıok and Yusuf Sahillioğlu. IMOTION — Searching for Video Sequences using Multi-Shot Sketch Queries. In *Proceedings of the International Conference on Multimedia Modeling (MMM)*, pages 377–382, Miami, USA. Springer, 2016.

[RGS+15] Luca Rossetto, Ivan Giangreco, Heiko Schuldt, Stéphane Dupont, Omar Seddati, T. Metin Sezgin and Yusuf Sahillioğlu. IMOTION — A Content-based Video Retrieval Engine. In *Proceedings of the International Conference on Multimedia Modeling (MMM)*, pages 261–265, Sydney, Australia. Springer, 2015.

[RGS14] Luca Rossetto, Ivan Giangreco and Heiko Schuldt. Cineast: A Multi-Feature Sketch-Based Video Retrieval Engine. In *Proceedings of the International Symposium on Multimedia (ISM)*, pages 18–23, Taichung, Taiwan. IEEE, 2014.

[RGT+16a] Luca Rossetto, Ivan Giangreco, Claudiu Tănase and Heiko Schuldt. vitrivr — A Flexible Retrieval Stack Supporting Multiple Query Modes for Searching in Multimedia Collections. In *Proceedings of the International Conference on Multimedia (ACM MM)*, pages 1183–1186, Amsterdam, The Netherlands. ACM, 2016.

[RGT+16b] Luca Rossetto, Ivan Giangreco, Claudiu Tănase, Heiko Schuldt, Stéphane Dupont, Omar Seddati, T. Metin Sezgin and Yusuf Sahillioğlu. iAutoMotion — An Autonomous Content-Based Video Retrieval Engine. In *Proceedings of the International Conference on Multimedia Modeling (MMM)*, pages 383–387, Miami, USA. Springer, 2016.

[RGT+17a] Luca Rossetto, Ivan Giangreco, Claudiu Tănase and Heiko Schuldt. Multimodal Video Retrieval with the 2017 IMOTION System. In *Proceedings of the International Conference on Multimedia Retrieval (ICMR)*, pages 457–460, Bucharest, Romania. ACM, 2017.

[RGT+17b] Luca Rossetto, Ivan Giangreco, Claudiu Tănase, Heiko Schuldt, Stéphane Dupont and Omar Seddati. Enhanced Retrieval and Browsing in the IMOTION System. In *Proceedings of the International Conference on Multimedia Modeling (MMM)*, pages 469–474, Reykjavík, Iceland. Springer, 2017.

[RHM97] Yong Rui, Thomas S. Huang and Sharad Mehrotra. Content-based Image Retrieval with Relevance Feedback in MARS. In *Proceedings of the International Conference on Image Processing (ICIP)*, pages 815–818, Santa Barbara, USA. IEEE, 1997.

[Rij79] Cornelis Joost van Rijsbergen. *Information Retrieval*. Butterworths, London, 2nd edition, 1979.

[Ros14] Luca Rossetto. *Cineast: a Content-based VIdeo Retrieval Engine*. Master Thesis, University of Basel, 2014.

[RS17] Luca Rossetto and Heiko Schuldt. Web Video in Numbers — An Analysis of Web-Video Metadata. arXiv, July 2017. URL: `https://arxiv.org/abs/1707.01340` (visited on 31/10/2017).

[Rüg10] Stefan Rüger. *Multimedia Information Retrieval*. Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool, 2010.

[SAA+10] Yasin N. Silva, Ahmed M. Aly, Walid G. Aref and Per-Åke Larson. SimDB: A Similarity-aware Database System. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1243–1246, Indianapolis, USA. ACM, 2010.

[SAB+14] Klaus Schöffmann, David Ahlström, Werner Bailer, Claudiu Cobârzan, Frank Hopfgartner, Kevin McGuinness, Cathal Gurrin, Christian Frisson, Duy-Dinh Le, Manfred Del Fabro, Hongliang Bai and Wolfgang Weiss. The Video Browser Showdown: A Live Evaluation of Interactive Video Search Tools. *International Journal of Multimedia Information Retrieval (IJMIR)*, 3(2):113–127, June 2014.

[Sal68] Gerard Salton. *Automatic Information Organization and Retrieval*. McGraw-Hill, New York, Maidenhead, 1968.

[SBL+16] Klaus Schöffmann, Christian Beecks, Mathias Lux, Merih Seran Uysal and Thomas Seidl. Content-based Retrieval in Videos from Laparoscopic Surgery. In *Proceedings of the International Conference on Medical Imaging*, San Diego, USA. SPIE, 2016.

[SC03] Luo Si and Jamie Callan. Relevant Document Distribution Estimation Method for Resource Selection. In *Proceedings of the International Conference on Information Retrieval (SIGIR)*, pages 298–305, Toronto, Canada. ACM, 2003.

[SC05] Michael Stonebraker and Ugur Cetintemel. "One Size Fits All": An Idea Whose Time Has Come and Gone. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 2–11, Tokyo, Japan. IEEE, 2005.

[SC96] John R. Smith and Shih-Fu Chang. VisualSEEk: A Fully Automated Content-based Image Query System. In *Proceedings of the International Conference on Multimedia (MULTIMEDIA)*, pages 87–98, Boston, USA. ACM, 1996.

[Sch80] Hans-Jörg Schek. Methods for the Administration of Textual Data in Database Systems. In *Proceedings of the International Conference on Information Retrieval (SIGIR)*, pages 218–235, Cambridge, UK. ACM, 1980.

[SDM15] Omar Seddati, Stéphane Dupont and Said Mahmoudi. DeepSketch: Deep Convolutional Neural Networks for Sketch Recognition and Similarity Search. In *Proceedings of the International Workshop on Content-Based Multimedia Indexing (CBMI)*, Prague, Czech Republic. IEEE, 2015.

[SDP+07] Shi-Kuo Chang, Vincenco Deufemia, Giuseppe Polese and Mario Vacca. A Normalization Framework for Multimedia Databases. *Transactions on Knowledge and Data Engineering (TKDE)*, 19(12):1666–1679, December 2007.

[SFJ14] Miaojing Shi, Teddy Furon and Hervé Jégou. A Group Testing Framework for Similarity Search in High-dimensional Spaces. In *Proceedings of the International Conference on Multimedia (ACM MM)*, pages 407–416, Orlando, USA. ACM, 2014.

[SFK+14] Liwen Sun, Michael J. Franklin, Sanjay Krishnan and Reynold S. Xin. Fine-grained Partitioning for Aggressive Data Skipping. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1115–1126, Snowbird, USA. ACM, 2014.

[SGG+13] David Scott, Jinlin Guo, Cathal Gurrin, Frank Hopfgartner, Kevin McGuinness, Noel E. O'Connor, Alan F. Smeaton, Yang Yang and Zhenxing Zhang. DCU at MMM 2013 Video Browser Showdown. In *Proceedings of the International Conference on Multimedia Modeling (MMM)*, pages 541–543, Huangshan, China. Springer, 2013.

[SGS+13] Martin Schäler, Alexander Grebhahn, Reimar Schröter, Sandro Schulze, Veit Köppen and Gunter Saake. QuEval: Beyond High-Dimensional Indexing à la carte. *Proceedings of the VLDB Endowment (PVLDB)*, 6(14):1654–1665, September 2013.

[SGS14] Fabio Sulser, Ivan Giangreco and Heiko Schuldt. Crowd-based Semantic Event Detection and Video Annotation for Sports Videos. In *Proceedings of the International Workshop on Crowdsourcing for Multimedia (CrowdMM)*, pages 63–68, Orlando, USA. ACM, 2014.

[SHJ+05] Rut Sigurðardóttir, Hauksson Hauksson, Björn Þ. Jónsson and Laurent Amsaleg. A Case-Study of the Quality vs. Time Trade-off for Approximate Image Descriptor Search. In *Proceedings of the International Conference on Data Engineering Workshops (ICDEW)*, Tokyo, Japan. IEEE, 2005.

[SHS+08] Jie Shao, Zi Huang, Heng Tao Shen, Xiaofang Zhou, Ee-Peng Lim and Yijun Li. Batch Nearest Neighbor Search for Video Retrieval. *Transactions on Multimedia (TOM)*, 10(3):409–420, April 2008.

[SKR+10] Konstantin Shvachko, Hairong Kuang, Sanjay Radia and Robert Chansler. The Hadoop Distributed File System. In *Proceedings of the Symposium on Mass Storage Systems and Technologies (MSST)*, Lake Tahoe, USA. IEEE, 2010.

[SKR+95] Ron Sacks-Davis, Alan Kent, Kotagiri Ramamohanarao, James Thom and Justin Zobel. Atlas: A Nested Relational Database System for Text Applications. *Transactions on Knowledge and Data Engineering (TKDE)*, 7(3):454–470, June 1995.

[SLM+01] Michael Stillger, Guy M. Lohman, Volker Markl and Mokhtar Kandil. LEO — DB2's LEarning Optimizer. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 19–28, Rome, Italy. Morgan Kaufmann, 2001.

[SMG+13] Denis Shestakov, Diana Moise, Gylfi Gudmundsson and Laurent Amsaleg. Scalable High-Dimensional Indexing with Hadoop. In *Proceedings of the International Workshop on Content-Based Multimedia Indexing (CBMI)*, pages 207–212, Veszprem, Hungary. IEEE, 2013.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of the International Conference*

on Applications, Technologies, Architectures and Protocols for Computer Communications (SIGCOMM), pages 149–160, San Diego, USA. ACM, 2001.

[SOK06] Alan F. Smeaton, Paul Over and Wessel Kraaij. Evaluation campaigns and TRECVid. In *Proceedings of the International Workshop on Multimedia Information Retrieval (MIR)*, page 321, Santa Barbara, USA. ACM, 2006.

[Son96] Gabriele Sonnenberger. Exploiting the Functionality of Object-Oriented Database Management Systems for Information Retrieval. *Bulletin of the Technical Committee on Data Engineering*:14–23, March 1996.

[Spi95] Der Spiegel. "Das Ding der Zukunft". *Der Spiegel*, (34):22–26, August 1995.

[Spr14] Michael Springmann. *Building Blocks for Adaptable Image Search in Digital Libraries*. PhD Thesis, University of Basel, Switzerland, 2014.

[SRF87] Timos K. Sellis, Nick Roussopoulos and Christos Faloutsos. The R+-tree: A Dynamic Index for Multi-dimensional Objects. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 507–518, Brighton, UK. Morgan Kaufmann, 1987.

[Ste11] Mathias Stearn. Slides: "Indexing, Query Optimization, the Query Optimizer". February 2011. URL: `https://de.slideshare.net/mongodb/indexing-and-query-optimizer-mongo-austin` (visited on 30/10/2017).

[Sto15] Michael Stonebraker. The Case for Polystores, 2015. URL: `http://wp.sigmod.org/?p=1629` (visited on 31/10/2017).

[Sul14] Fabio Sulser. *Crowdsourcing Annotations for Video Retrieval in Sports Videos*. Bachelor Thesis, University of Basel, 2014.

[SWS+00] Arnold W. M. Smeulders, Marcel Worring, Santini Santini, Amarnath Gupta and Ramesh Jain. Content-Based Image Retrieval at the End of the Early Years. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 22(12):1349–1380, December 2000.

[SWY75] Gerard Salton, Anita Wong and Chung-Shu Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM (CACM)*, 18(11):613–620, November 1975.

[TES+16] Bart Thomee, Benjamin Elizalde, David A. Shamma, Karl Ni, Gerald Friedland, Douglas Poland, Damian Borth and Li-Jia Li. YFCC100M: The New Data in Multimedia Research. *Communications of the ACM (CACM)*, 59(2):64–73, January 2016.

[TGR+16] Claudiu Tănase, Ivan Giangreco, Luca Rossetto, Heiko Schuldt, Omar Seddati, Stephane Dupont, Ozan Can Altiok and T. Metin Sezgin. Semantic Sketch-Based Video Retrieval with Autocompletion. In *Proceedings of the International Conference on Intelligent User Interfaces (IUI)*, pages 97–101, Sonoma, USA. ACM, 2016.

[TK78] Dennis Tsichritzis and Anthony Klug. The ANSI/X3/SPARC DBMS Framework Report of the Study Group on Database Management Systems. *Information Systems*, 3(3):173–191, January 1978.

[TKB00] Roland Tusch, Harald Kosch and Lázló Böszörményi. VIDEX: An Integrated Generic Video Indexing Approach. In *Proceedings of the International Conference on Multimedia (MULTIMEDIA)*, pages 448–451, Marina del Rey, USA. ACM, 2000.

[TLF10] Engin Tola, Vincent Lepetit and Pascal Fua. DAISY: An Efficient Dense Descriptor Applied to Wide-Baseline Stereo. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 32(5):815–830, May 2010.

[TLS+16] Giovanni Taverriti, Stefano Lombini, Lorenzo Seidenari, Marco Bertini and Alberto del Bimbo. Real-time Wearable Computer Vision System for Improved Museum Experience. In *Proceedings of the International Conference on Multimedia (ACM MM)*, pages 703–704, Amsterdam, The Netherlands. ACM, 2016.

[TRG+16] Claudiu Tănase, Luca Rossetto, Ivan Giangreco and Heiko Schuldt. The vitrivr System at TRECVID 2016: The Ad-Hoc Video Search Task. In *Proceedings of the TRECVID Ad-Hoc Video Search Task*, Maryland, USA. TRECvid, 2016.

[TSW07] Martin Theobald, Ralf Schenkel and Gerhard Weikum. The TopX DB&IR engine. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 1141–1143, Beijing, China. ACM, 2007.

[TT07] Kengo Terasawa and Yuzuru Tanaka. Spherical LSH for Approximate Nearest Neighbor Search on Unit Hypersphere. In *Proceedings of the Workshop on Algorithms and Data Structures (WADS)*, pages 27–38, Halifax, Canada. Springer, 2007.

[TVM+11] George Teodoro, Eduardo Valle, Nathan Mariano, Ricardo Torres and Wagner Meira. Adaptive Parallel Approximate Similarity Search for Responsive Multimedia Retrieval. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 495–504, Glasgow, Scotland. ACM, 2011.

[Vau11] Tay Vaughan. *Multimedia — Making it work*. McGraw-Hill, Emeryville, 8th edition, 2011.

[VB98] Arjen P. de Vries and Henk M. Blanken. Database Technology and the Management of Multimedia Data in the Mirror Project. In *Proceedings of SPIE - The International Society for Optical Engineering*, pages 443–453. SPIE, 1998.

[Vog15] Marco Vogt. *Time-Check: Analyse der Parameter zur Evaluation verteilter Mehrversionen-Datenbanksysteme*. Bachelor Thesis, University of Basel, Switzerland, 2015.

[Vri99] Arjen P. de Vries. *Content and Multimedia Database Management Systems*. PhD Thesis, University of Twente, The Netherlands, 1999.

[Wan03] Avery Li-Chun Wang. An Industrial-Strength Audio Search Algorithm. In *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*, Baltimore, USA. The Johns Hopkins University, 2003.

[WBS00] Roger Weber, Klemens Böhm and Hans-Jörg Schek. Interactive-Time Similarity Search for Large Image Collections Using Parallel VA-Files. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 197–197, San Diego, USA. IEEE, 2000.

[WBS01] Roger Weber, Klemens Böhm and Hans-Jörg Schek. Parallel NN-Search for Large Multimedia Repositories. In Remco C. Veltkamp, Hans Burkhardt and Hans-Peter Kriegel, editors, *State-of-the-Art in Content-Based Image and Video Retrieval*, pages 319–343. Springer, Dordrecht, 2001.

[Web00] Roger Weber. *Similarity Search in High-Dimensional Vector Spaces*. PhD Thesis, Diss. ETH No 13974, Swiss Federal Institute of Technology Zurich (ETHZ), Switzerland, 2000.

[Web97] Roger Weber. Parallel VA-File. Technical report, Swiss Federal Institute of Technology Zurich (ETHZ), Switzerland, 1997.

[Wei07] Gerhard Weikum. DB & IR: Both Sides Now. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 25–30, Beijing, China. ACM, 2007.

[WFT12] Yair Weiss, Rob Fergus and Antonio Torralba. Multidimensional Spectral Hashing. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 340–353, Zurich, Switzerland. Springer, 2012.

[Wik17] Wikipedia. Simple example of an R-tree for 2D rectangles, 2017. URL: `https://en.wikipedia.org/wiki/R-tree#/media/File:R-tree.svg` (visited on 31/10/2017).

[WJ96] David A. White and Ramesh Jain. Similarity Indexing with the SS-tree. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 516–523, New Orleans, USA. IEEE, 1996.

[WK03] Utz Westermann and Wolfgang Klas. An Analysis of XML database Solutions for the Management of MPEG-7 Media Descriptions. *Computing Surveys (CSUR)*, 35(4):331–373, December 2003.

[WLL+15] Kyu-Young Whang, Jae-Gil Lee, Min-Jae Lee, Wook-Shin Han, Min-Soo Kim and Jun-Sung Kim. DB-IR Integration using Tight-Coupling in the Odysseus DBMS. *World Wide Web*, 18(3):491–520, May 2015.

[WMZ10] William Webber, Alistair Moffat and Justin Zobel. A Similarity Measure for Indefinite Rankings. *Transactions on Database Systems (TODS)*, 28(4), November 2010.

[WNM+95] Jian-Kang Wu, A. Desai Narasimhalu, Babu M. Mehtre, Chian-Prong Lam and Yong Jian Gao. CORE: A Content-Based Retrieval Engine for Multimedia Information Systems. *Multimedia Systems*, 3(1):25–41, February 1995.

[WSB98] Roger Weber, Hans-Jörg Schek and Stephen Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 194–205, New York, USA. Morgan Kaufmann, 1998.

[WSZ17a] Dongyao Wu, Sherif Sakr and Liming Zhu. Big Data Programming Models. In Albert Y. Zomaya and Sherif Sakr, editors, *Handbook of Big Data Technologies*, pages 31–63. Springer, Cham, 2017.

[WSZ17b] Dongyao Wu, Sherif Sakr and Liming Zhu. Big Data Storage and Data Models. In Albert Y. Zomaya and Sherif Sakr, editors, *Handbook of Big Data Technologies*, pages 3–29. Springer, Cham, 2017.

[WTF08] Yair Weiss, Antonio Torralba and Rob Fergus. Spectral Hashing. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1753–1760, Vancouver, Canada. Curran Associates, 2008.

[WYL+10] Brandyn White, Tom Yeh, Jimmy Lin and Larry Davis. Web-scale Computer Vision Using MapReduce for Multimedia Data Mining. In *Proceedings of the International Workshop on Multimedia Data Mining (MDMKDD)*, New York, USA. ACM, 2010.

[YIS12] Takuya Yokoyama, Yoshiharu Ishikawa and Yu Suzuki. Processing All k-Nearest Neighbor Queries in Hadoop. In *Proceedings of the International Conference on Web-Age Information Management (WAIM)*, pages 346–351, Harbin, China. Springer, 2012.

[YL13] DongSheng Yin and DeBo Liu. Content-Based Image Retrial [sic] Based on Hadoop. *Mathematical Problems in Engineering*, 2013:1–7, August 2013.

[YWS15] Jia Yu, Jinxuan Wu and Mohamed Sarwat. GeoSpark: A Cluster Computing Framework for Processing Large-Scale Spatial Data. In *Proceedings of the International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, number 70, Seattle, USA. ACM, 2015.

[ZAD+06] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal and Michal Batko. *Similarity Search: The Metric Space Approach*. Springer, New York, 2006.

[ZCD+12] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker and Ion Stoica. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NSDI)*, pages 15–18, San Jose, USA. USENIX, 2012.

[ZCF+10] Matei Zaharia, Mosharaf Chowdhury, Michael Franklin, Scott Shenker and Ion Stoica. Spark: Cluster Computing with Working Sets. In *Proceedings of the Conference on Hot Topics in Cloud Computing (HotCloud)*, pages 10–17, Boston, USA. USENIX, 2010.

[ZCO+15] Hao Zhang, Gang Chen, Beng Chin Ooi, Kian-Lee Tan and Meihui Zhang. In-Memory Big Data Management and Processing: A Survey. *Transactions on Knowledge and Data Engineering (TKDE)*, 27(7):1920–1948, July 2015.

[ZIP16] Kostas Zoumpatianos, Stratos Idreos and Themis Palpanas. ADS: The Adaptive Data Series Index. *International Journal on Very Large Data Bases (VLDB Journal)*, 25(6):843–866, December 2016.

[ZLJ12] Chi Zhang, Feifei Li and Jeffrey Jestes. Efficient Parallel kNN Joins for Large Data in MapReduce. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 38–49, Berlin, Germany. ACM, 2012.

[ZLX+14] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba and Aude Oliva. Learning Deep Features for Scene Recognition using Places Database. In *Advances in Neural Information Processing Systems (NIPS)*, pages 487–495, Montreal, Canada. NIPS, 2014.

Photo Credits

A. Stills from Woody Allen. Midnight in Paris, 2011; Woody Allen. Manhattan, 1979; Marc Forster. Finding Neverland, 2004; Ben Stiller. The Secret Life of Walter Mitty, 2013; Sam Mendes. Skyfall, 2012; Marc Webb. 500 Days of Summer, 2009.

B. Paintings by Vincent van Gogh. Wheat Fields with Sheaves (F561), Honolulu Museum of Art, Honolulu, Hawaii, 1888; Vincent van Gogh. Wheat Field with Alpilles Foothills in the Background (F411), Van Gogh Museum, Amsterdam, The Netherlands, 1888; Harvest in Provence (F558), Israel Museum, Jerusalem, Israel, 1888; Vincent van Gogh. Wheat Fields (F564), Van Gogh Museum, Amsterdam, The Netherlands, 1888; Vincent van Gogh. Green Wheat Field with Cypress (F719), Narodni Gallery, Prague, Czech Republic, 1889; Vincent van Gogh. Wheat Stacks with Reaper (F559), Toledo Museum of Art, Toledo, USA, 1890; Vincent van Gogh. Wheatfield with Crows (F779), Van Gogh Museum, Amsterdam, The Netherlands, 1890.

C. Images from Woody Allen. Midnight in Paris, 2011; Vincent van Gogh. The Starry Night (F612), The Museum Of Modern Art, New York, USA, 1889.

D. Paintings by Vincent van Gogh. Wheatfield with Crows (F779), Van Gogh Museum, Amsterdam, The Netherlands, 1890; Vincent van Gogh. Green Wheat Field with Cypress (F719), Narodni Gallery, Prague, Czech Republic, 1889.

E. Paintings by Harvest in Provence (F558), Israel Museum, Jerusalem, Israel, 1888; Vincent van Gogh. The Starry Night (F612), The Museum Of Modern Art, New York, USA, 1889; Vincent van Gogh. Wheatfield with Crows (F779), Van Gogh Museum, Amsterdam, The Netherlands, 1890; Vincent van Gogh. Green Wheat Field with Cypress (F719), Narodni Gallery, Prague, Czech Republic, 1889.

F. Stills from Woody Allen. Midnight in Paris, 2011.

G. Stills from Woody Allen. Midnight in Paris, 2011.

Index