

SOUNDTRACK RECOMMENDATION FOR IMAGES

Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Naturwissenschaftlich-Technischen Fakultät I
der Universität des Saarlandes

Aleksandar Stupar

Universität des Saarlandes

Saarbrücken
2013

Dekan der Naturwissenschaftlich-Technischen
Fakultät I

Prof. Dr. Mark Groves

Vorsitzender der Prüfungskommission

Prof. Dr. Dr. h.c. mult. Reinhard
Wilhelm

Berichterstatter

Dr.-Ing. Sebastian Michel

Berichterstatter

Prof. Dr.-Ing. Gerhard Weikum

Berichterstatter

Prof. Dr.ir. Arjen P. de Vries

Beisitzer

Dr.-Ing. Klaus Berberich

Tag des Promotionskolloquiums

04.10.2013

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form in einem Verfahren zur Erlangung eines akademischen Grades vorgelegt.

Saarbrücken, den 04.10.2013

(Unterschrift)

Kurzfassung

Der drastische Anstieg von verfügbaren Multimedia-Inhalten hat die Bedeutung der Forschung über deren Organisation sowie Suche innerhalb der Daten hervorgehoben. In dieser Doktorarbeit betrachten wir das Problem der Suche nach geeigneten Musikstücken als Hintergrundmusik für Diashows. Wir formulieren die Hypothese, dass die für das Problem erforderlichen Kenntnisse in öffentlich zugänglichen, zeitgenössischen Filmen enthalten sind. Unser Ansatz, Picasso, verwendet Techniken aus dem Bereich der Ähnlichkeitssuche innerhalb von Bild- und Musik-Domains, um basierend auf Filmszenen eine Verbindung zwischen beliebigen Bildern und Musikstücken zu lernen.

Um einen fairen und unvoreingenommenen Vergleich zwischen verschiedenen Ansätzen zur Musikempfehlung zu erreichen, schlagen wir einen Bewertungs-Benchmark vor. Die Ergebnisse der Auswertung werden, anhand des vorgeschlagenen Benchmarks, für Picasso und einen weiteren, auf Emotionen basierenden Ansatz, vorgestellt. Zusätzlich behandeln wir zwei Effizienzaspekte, die sich aus dem Picasso Ansatz ergeben. (i) Wir untersuchen das Problem der Ausführung von top-K Anfragen, bei denen die Ergebnismenge ad-hoc auf eine kleine Teilmenge des gesamten Indexes eingeschränkt wird. (ii) Wir behandeln das Problem der Ähnlichkeitssuche in hochdimensionalen Räumen und schlagen zwei Erweiterungen des Lokalitätssensitiven Hashing (LSH) Schemas vor. Zusätzlich untersuchen wir die Erfolgsaussichten eines verteilten Algorithmus für die Ähnlichkeitssuche, der auf LSH unter Verwendung des MapReduce Frameworks basiert. Neben den vorgenannten wissenschaftlichen Ergebnissen beschreiben wir ferner das Design und die Implementierung von PicassSound, einer auf Picasso basierenden Smartphone-Anwendung.

Abstract

The drastic increase in production of multimedia content has emphasized the research concerning its organization and retrieval. In this thesis, we address the problem of music retrieval when a set of images is given as input query, i.e., the problem of soundtrack recommendation for images. The task at hand is to recommend appropriate music to be played during the presentation of a given set of query images. To tackle this problem, we formulate a hypothesis that the knowledge appropriate for the task is contained in publicly available contemporary movies. Our approach, Picasso, employs similarity search techniques inside the image and music domains, harvesting movies to form a link between the domains. To achieve a fair and unbiased comparison between different soundtrack recommendation approaches, we proposed an evaluation benchmark. The evaluation results are reported for Picasso and the baseline approach, using the proposed benchmark. We further address two efficiency aspects that arise from the Picasso approach. First, we investigate the problem of processing top-K queries with set-defined selections and propose an index structure that aims at minimizing the query answering latency. Second, we address the problem of similarity search in high-dimensional spaces and propose two enhancements to the Locality Sensitive Hashing (LSH) scheme. We also investigate the prospects of a distributed similarity search algorithm based on LSH using the MapReduce framework. Finally, we give an overview of the PicasSound—a smartphone application based on the Picasso approach.

Zusammenfassung

Jeden Tag werden atemberaubende Mengen an Multimedialinhalten unter der Benutzung von Webportalen erstellt und weiterempfohlen. Das Paradebeispiel ist YouTube [YT], auf welches jede Minute mehr als 72 Stunden Videomaterial hochgeladen werden¹. Diese riesigen Mengen an Inhalt haben nicht nur Probleme in der Datenorganisation und Suche zur Folge, sondern führen auch zu einem harten Wettbewerb um die Aufmerksamkeit der Zuschauer.

Bilder werden normalerweise als Diashow mit einer Reihe an unterschiedlichen Effekten und Stilen untermalt, um die Aufmerksamkeit des Publikums zu gewinnen. Die Frage, wie Präsentationen automatisch generiert werden können, hat sehr viele wissenschaftliche Arbeiten hervorgebracht [CCK⁺06, LS07, CXG10], welche das Ziel verfolgen die Präsentationen ansprechender für die Zuhörerschaft zu gestalten. Auch Musik spielt eine wesentliche Rolle bei Diashows, weil die visuellen Eindrücke durch passende Musik, die während der Präsentation gespielt wird, verstärkt werden können. Allerdings ist die Aufgabe passende Musik für eine gegebene Menge an Bildern auszusuchen eine nicht-triviale Aufgabe, da große Musiksammlungen hierbei sehr viel menschliche Aufmerksamkeit und eine Menge Erfahrung verlangen. In der Filmwelt sind spezielle Preise etabliert, zum Beispiel der Oscar für die beste Filmmusik, um Meisterwerke auszuzeichnen, was die Bedeutung und Komplexität der Aufgabe bestätigt. In dieser Arbeit wird das Problem automatisch Musik als Soundtrack für gegebene Bilder zu empfehlen analysiert.

Gegeben eine Musiksammlung und eine Menge an Bilder als Anfrage ist es die Aufgabe eines Soundtrackempfehlungsdienstes die einzelnen Lieder nach Eignung als Soundtrack für die gegebenen Bilder zu ordnen. Die top-K Lieder dieser Rangordnung werden finale Empfehlung an den Benutzer gesehen. Die Arbeit führt Picasso ein - ein Framework das die Aufgabe Soundtracks für Bilder zu empfehlen angeht. Picassos Entscheidungen basieren auf dem Wissen von erfahrenen Regisseuren, welches direkt aus den Filmen selbst extrahiert wird. Auf diese Weise können die Trainingsdaten beliebig groß sein, sind kostengünstig, und stellen doch eine exzellente Informationsquelle dar. Das extrahierte Wissen wird in der Form von Bild-Musik Paaren verarbeitet, wobei das Bild ein Screenshot aus dem Film ist und die Musik diejenige Musik ist, die zu diesem Zeitpunkt im Film gespielt wird. Zur Anfragezeit benutzt Picasso eine Ähnlichkeitssuche

¹<http://www.youtube.com/yt/press/statistics.html>

für Bilder und Musik um die finale Empfehlung zu erstellen. Zuerst findet es die ähnlichsten Screenshots für ein gegebenes Anfragebild und wählt dann für jeden Screenshot die ähnlichsten Lieder für die entsprechende Filmmusik aus. Um die Umsetzbarkeit des Ansatzes zu evaluieren wurde eine Benutzerstudie durchgeführt.

Benutzerstudien durchzuführen um den Einfluss von Verbesserungen auf das System zu messen, oder um das System mit konkurrierenden Programmen zu vergleichen, ist ein mühsamer Prozess und bedarf einer Menge an menschlichem Arbeitsaufwand. Um diese Einschränkungen zu bewältigen, wird ein wiederverwendbarer Benchmark zur Messung der Leistung eines Soundtrackempfehlungsdienstes aufgestellt. Das Design wurde unabhängig von allen Soundtrackempfehlungsdiensten durchgeführt, was in einem unverfälschten und voll wiederverwendbaren Benchmark resultiert. Der Benchmark basiert auf Vorliebebewertungen, wobei die Bewertungen als Paare von Liedern im Bezug auf eine Anfrage, das heißt eine Menge von Bildern, aufgenommen wurden. Zwei verschiedene Typen von Relevanzbewertungen machen den Benchmark aus: (i) Bewertungen, die von einer Benutzerstudie auf dem Campus stammen, dienen als Goldstandard für (ii) Bewertungen, welche über Amazon's Mechanical Turk gesammelt wurden. Picasso und eine Basisimplementierung werden auf dem Benchmark im Bezug auf ihre Effektivität evaluiert.

Eine Smartphoneapplikation namens PicasSound, welche das Picasso System einem erweiterten Publikum demonstriert, wurde entwickelt. Da Smartphones meist sowohl Musik als auch Bilder enthalten, sind sie ein perfektes Vorzeigeprojekt für ein System wie Picasso. PicasSound ist als zweistufiges Programm implementiert. Der Empfehlungsprozess wird serverseitig ausgeführt, während die Smartphoneapplikation benutzt wird um durch die Ergebnisse zu durchstöbern und die vorgeschlagene Musik anzuhören.

Wenn die Empfehlungsanfrage auf der Serverseite empfangen wird, ist es wichtig den Empfehlungsprozess auf effiziente Weise zu erledigen, damit die Benutzer das Ergebnis mit so wenig Verzögerung wie möglich erhalten. Die Aufgabe des Empfehlungsprozesses ist es die passendsten Lieder für ein Anfragebild auszuwählen, sodass die Lieder auf dem Smartphone des Benutzers vorhanden sind. Das bedeutet, dass die Auswahl der top-K Lieder auf die Menge an Liedern, die der Benutzer besitzt beschränkt ist. Deshalb werden Effizienzaspekte des Verarbeitens von top-K Anfragen mit mengendefinierter Auswahl analysiert. Die Auswahlbeschränkung beeinflusst drastisch die Vor- und Nachteile eines nach Ids geordneten Indexes gegenüber einem nach Werten geordneten Index. Daher wird ein kombinierter Index eingeführt, welcher das Beste beider Indizes ausnutzt, indem zur Anfragezeit entschieden wird, welcher eingesetzt wird. Außerdem wird ein partitionierter Index vorgestellt, welcher die Latenz am Break-Even-Punkt des kombinierten Indexes senkt. Des Weiteren kann der partitionierte Index dazu eingesetzt werden approximierete top-K Ergebnisse mit Qualitätsabschätzung zu erhalten.

Eine andere Aufgabe des Empfehlungsprozesses ist es die ähnlichsten Screenshots gegeben ein Anfragebild zu finden. Die Merkmalrepräsentation der Bilder

ergibt für diese Aufgabe der Ähnlichkeitssuche eine K-nächste-Nachbarn-Suche in hochdimensionalen Räumen. Lokalitätssensitives Hashen (LSH) wurde als effiziente Technik vorgestellt um das Problem anzugehen. Im Rahmen dieser Arbeit werden zwei heuristische Verbesserungen zu LSH vorgestellt. Die erste Verbesserung basiert auf zusätzlichen Verknüpfungen für jeden Punkt im Merkmalsraum, die auf den exakten nächsten Nachbarn zeigen. Die zweite Verbesserung namens *Peek-Probing* spezifiziert LSH Einträge nur dann ganz zu lesen, wenn sie ein gewisses Maß an nützlicher Information enthalten. Die Techniken sind orthogonal und können deshalb in Kombination verwendet werden um eine verbesserte Leistung zu erzielen. Zusätzlich zu den zwei Verbesserungen untersuchen wir die Verteilungen der LSH Techniken unter Verwendung des MapReduce Frameworks für verteiltes Rechnen. Sowohl die vorgeschlagenen LSH Verbesserungen als auch die Indices für top-K Anfragen finden eine Vielzahl Anwendungen weit über Picasso hinaus.

Summary

Staggering amounts of multimedia content are created and shared online on a daily basis. The prime example is YouTube [YT], having more than 72 hours of video content uploaded to every minute². Not only do these large amounts give rise to the issues in data organization and retrieval but they also introduce a strong competition for the spectators' attention.

Images are usually presented in a slide show with a variety of transition styles and effects used to capture the audience attention. The question how to automatically generate slide shows has received a lot of research attention [CCK⁺06, LS07, CXG10] with the aim of making them more appealing to the general audience. Music plays an important role in slide shows, as the impressions conveyed by images can be emphasized when appropriate music is played during the presentation. However, selecting an appropriate piece of music for a given set of images is a non-trivial task as large music collections require a lot of human attention and a good deal of experience. In the world of movies, special awards are established to distinguish master pieces, e.g., the Academy Award for Best Original Score, emphasizing the importance and the complexity of the task. In this thesis, we study the problem of automatically assigning appropriate music pieces to a picture or, in general, series of pictures, that is, the problem of recommending songs as soundtracks for query images.

Given a music collection and a set of images as query the task of a soundtrack recommendation system is to rank the song in the collection by their appropriateness to serve as a soundtrack for the given images. The top-K songs from this ranking are considered as a final recommendation to the user. We propose Picasso—a framework to address the task of soundtrack recommendation for images. Picasso is based on the knowledge of experienced movie directors extracted directly from the movies themselves. This way, the training set can be arbitrarily large and is also inexpensive to obtain but still provides an excellent source of information. The extracted knowledge is in the form of image-music pairs, where the image is a screenshot from the movie and the music is the music played in the movie at the time of the taken screenshot. At query time, Picasso utilizes similarity search in image and music domains to create a final recommendation. First, it finds the most similar screenshots to a given query image, and then selects for each screenshot the most similar songs to the corresponding

²<http://www.youtube.com/yt/press/statistics.html>

movie music. We report on the results of a user study conducted to evaluate the feasibility of our approach.

Conducting user studies to evaluate the impact of improvements made to the system, or to compare competing systems, is a cumbersome task and requires a lot of human effort. To overcome this limitation, we propose a reusable benchmark to evaluate the retrieval performance of soundtrack recommendation systems. The building process is done independently of any soundtrack recommendation approach resulting in an unbiased and fully reusable benchmark. We base the proposed benchmark on preference judgments, where judgments are collected for pairs of songs with respect to a query (i.e., a set of images). The benchmark consists of two types of relevance assessments: (i) judgments obtained from a user study on campus, that serve as a “gold standard” for (ii) relevance judgments gathered through Amazon’s Mechanical Turk. We evaluate Picasso and a baseline system using the benchmark and report on their effectiveness results.

We have built a smartphone application, named PicasSound, which demonstrates the Picasso system to a wider audience. As smartphones contain both music and images this makes them an ideal showcase for a system such as Picasso. PicasSound is implemented as two-tiered application. The recommendation process is done on the server side while the smartphone application is used to browse through the results and listen to the recommended music.

When the request for recommendation is received on the server side, it is important to perform the recommendation process in an efficient manner such that users receive the recommendation with as little delay as possible. The task of the recommendation process is to select the most appropriate songs for the query image such that they are contained on the user’s phone. This means, selecting the top-K songs is constrained to the set of songs the user possesses. We consider efficiency aspects of processing such top-K queries with set-defined selections. The selections drastically influence the pros and cons of an id-ordered index vs. a score-ordered index. We propose a combined index which harnesses best of both indices by deciding which one to use at query time. We further propose a partitioned index which lowers the latency of the combined index around the break even point. When appropriate, approximate top-K results can be retrieved using the partitioned index with an indication of the expected results quality.

Another task in the recommendation process is to find the most similar screenshots to a given query image. The feature representation of images translates this task of similarity search into K-Nearest Neighbor (KNN) search in high-dimensional spaces. Locality Sensitive Hashing (LSH) has been proposed as an efficient technique to address this problem. In this thesis, we propose two heuristic enhancements to LSH. The first enhancement is based on additionally introduced links for each point in the feature space, referring to its exact nearest neighbor. The second enhancement, coined *Peek-Probing*, specifies reading LSH buckets fully only if they indicate a certain amount of useful information. The techniques are fully orthogonal and, hence, can be used in a combined way for

improved performance. In addition to two proposed enhancements, we investigate the distribution of the LSH techniques using the MapReduce framework for distributed computing. The proposed LSH enhancements, as well as the indices for top-K processing, reach beyond the Picasso system and find their application in multitude of settings.

Contents

1	Introduction and Problem Statement	1
1.1	Research Challenges	3
1.2	Contributions	4
1.3	Selected Publications	5
1.4	Outline of the Thesis	6
2	Background	9
2.1	Multimedia Information Retrieval	9
2.1.1	Image Features	11
2.1.2	Music Features	13
2.2	Similarity Search	15
2.2.1	Locality Sensitive Hashing	19
2.3	Evaluation Techniques	21
2.3.1	Effectiveness Measures	22
2.4	MapReduce Framework	23
3	Related Work	27
3.1	Soundtrack Recommendation	27
3.1.1	Soundtrack Recommendation for Images	28
3.1.2	Soundtrack Recommendation for Videos	29
3.1.3	Context-Aware Music Recommendation	31
3.2	Image Retrieval for Soundtracks	33
3.3	Automated Slide Show Generation	35
3.4	Collecting Relevance Assessments through Pairwise Comparisons	36
3.5	Processing Top-K Queries with Constraints	38
4	Picasso–A Soundtrack Recommendation Framework	41
4.1	Framework	42
4.1.1	Image Similarity Measure	45
4.1.2	Music Similarity Measure	46
4.2	Approach	48
4.2.1	The Case of Multiple Images	49
4.3	Feasibility Study	52
4.3.1	Study Setup	52
4.3.2	Study Results	54

4.3.3	Lessons Learned	56
5	Effectiveness Benchmark	57
5.1	Evaluation Dataset	58
5.1.1	Song Collection	59
5.1.2	Query Collection	60
5.2	Relevance Measure	61
5.2.1	Pairwise Preference Judgments	61
5.2.2	System Effectiveness Measures	63
5.3	The Benchmark	64
5.3.1	Benchmark Statistics	65
5.4	Approaches Evaluation	69
5.4.1	Results	70
6	Processing Top-K Queries with Set-Defined Selections	73
6.1	Indices and Cost Model	75
6.2	Partitioned Index Organization	77
6.2.1	The Partition Selection Phase	78
6.2.2	Index Partitioning	81
6.3	Approximate Query Answering	82
6.3.1	Tunable Expected Precision	82
6.4	Experimental evaluation	83
6.4.1	Model Verification	86
6.4.2	Exact Top-K Retrieval	89
6.4.3	Approximate Top-K Retrieval	91
7	Enhancing and Distributing Locality Sensitive Hashing	95
7.1	Linked-LSH	96
7.1.1	Query processing	97
7.2	Peek-Probing	98
7.2.1	Bucket Organization	99
7.3	RankReduce Framework	100
7.3.1	Query Processing	102
7.4	Experimental Evaluation	103
7.4.1	LSH Enhancements Evaluation	105
7.4.2	RankReduce Evaluation	108
8	PicasSound	111
8.1	Design Rationale	111
8.1.1	Implementation Details	112
8.2	Application Functionality	113
9	Conclusion and Outlook	115

A Appendix	117
A.1 Benchmark Music Collection Sample	117
A.2 Benchmark Queries	121
A.3 Feasibility Study Query Images	125
List of Figures	128
List of Algorithms	129
List of Tables	131
References	132
Index	147

Chapter 1

Introduction and Problem Statement

Nowadays, recording audio and video information has never been easier. Every smartphone is a full-fledged audio/video recording device. With over one billion smartphones in the world¹ and low prices of digital content storage, production rates of multimedia content have skyrocketed. It is estimated that there are more than one billion photos taken every day². This together with the production capabilities of personal computer results in a staggering amount of multimedia content produced on a daily basis. YouTube [YT] reports that more than 72 hours of video content is uploaded to their service every minute³. Flickr [FLI] reports that 518 million of images have been uploaded to their service in 2012 alone, that is 1.42 million of images uploaded per day.

With the success of these content sharing portals, such as YouTube [YT], Flickr [FLI], or SoundCloud [SND], together with the technology advances in computer networks, the content uploaded by the users is now available to anyone with an internet connection within a click of a mouse.

The huge quantities of multimedia content, being easily accessible through a couple of mouse clicks, bring out a number of challenges in content organization to achieve efficient and effective exploration and retrieval. On the other hand, there is a strong competition for the spectators' attention. Every produced and uploaded piece of multimedia competes for a limited amount of spectator's time. This results in a constant improvement in the way multimedia content is presented to the audience.

We can see such improvements in the presentation of images, where slide shows, being the default means of image presentation, are constantly enhanced and improved. There are a multitude of slide show styles originating from various techniques in slide animations and image placement. A substantial

¹<http://www.businessinsider.com/15-billion-smartphones-in-the-world-2013-2>

²<http://blog.1000memories.com/94-number-of-photos-ever-taken-digital-and-analog-in-shoebox>

³<http://www.youtube.com/yt/press/statistics.html>



Figure 1.1: Soundtrack recommendation for images

amount of research has been conducted [CCK⁺06, LS07, CXG10] to find a presentation technique that appeals the most to the audience. Even online services, such as Animoto [ANI], have been established to help users create appealing slide shows from their images.

The effects of a visual presentation are greatly enhanced if they are supported by the aural effects of the accompanying music. If selected properly, music can emphasize the point being conveyed by the images. On the other hand, if selected poorly it can distract the audience and ruin the presentation. Selecting an appropriate music piece for a slide show is everything but trivial. A good solution to this problem must have a big repository of music to choose from, to allow for a high level of diversity. While the requirement of having many songs is relatively easy to fulfill, the challenge of selecting the right one, out of hundreds or even thousands of songs, becomes difficult and requires a lot of human attention.

Soundtrack recommendation systems have been proposed to relieve the stress of inspecting hundreds or thousands of songs. These systems recommend to the users a subset of songs that are appropriate to be used as soundtracks, i.e., the background music. This way users need to inspect only a small subset of a potentially huge music collection. In case of slide shows, the recommended music subset has to be in agreement with the content of the images that are being presented. A formal definition of the task of soundtrack recommendation for images is given in the following.

Problem Formulation

We formally define the problem of soundtrack recommendation for images as follows: a soundtrack recommendation system over a set of indexed songs $S = \{s_1, s_2, \dots\}$ takes as an input query a set of images $q = \{img_1, img_2, \dots\}$ and the size of the result set K . It returns a subset of the indexed songs $S_r \subseteq S$, with $|S_r| = K$, ordered with respect to their relevance to act as background music

for a slide show that features the given query images.

Figure 1.1 illustrates this problem scenario. As illustrated, the main components of the problem are images as query, music collection, and the resulting slide show with the selected background music.

The task of this thesis is to propose a soundtrack recommendation system for images that will efficiently produce high quality recommendations.

1.1 Research Challenges

We identify the following three major research aspects: the connection between music and images, measuring effectiveness, and addressing efficiency aspects such that the proposed approach becomes usable for a large number of users in everyday situations.

Connecting Images and Music

When addressing the problem of soundtrack recommendation for images, the first question that arises is how the connection between images and music is created. One common approach is to map both images and songs in a common space, usually referred to as a pivot space, such as the space of emotions [LS07], and find a match in that space.

In this thesis, we make the hypothesis that the connection between images and music for soundtrack recommendation task can be made through publicly available contemporary movies. This opens a new set of questions concerning the concrete approach, first how can we extract this connection from the movies, and secondly how can this extracted information be used for the recommendation process. The proposed approach needs to support arbitrary images as queries and arbitrary music pieces as a music collection.

Effectiveness

Once we have approaches for soundtrack recommendation in place, we need to measure how aligned are the recommendations with the users' preferences. An obvious approach to this problem is to perform a user study for every new approach or an aspect that is to be evaluated. Not only does this waste resources as every new user study requires substantial organizational efforts, but it also brings into question the comparability of the results between the studies.

For this reason, we create a reusable benchmark used to evaluate the effectiveness of soundtrack recommendation approaches for images. The task starts with selecting images for query collection and songs for music collection to form the evaluation dataset. Further, relevance assessments have to be collected to provide a large enough coverage while keeping the task tractable. The collected relevance assessments have to be coupled with effectiveness measures that enable a fair and unbiased comparison between different systems. In addition, we are interested in what knowledge, about the soundtrack recommendation task,

we are able to gather from the large human effort put in the collected relevance assessments. The ultimate goal of the benchmark is to elaborate on the performance of the current state-of-the-art including our system, and provide means for a comparison in further research with minimum of additional effort needed.

Efficiency

We identify two aspects concerning the efficiency of the recommendation process in Picasso, namely, top-K queries with set-defined selection and similarity search in high-dimensional data spaces.

Selecting the top-K songs, with respect to a precomputed similarity score, constrained to a set of songs contained on a user's phone formulates a family of queries we refer to as top-K queries with set-defined selections. Given a database, the question is how can we organize the data into an index structure to allow for an efficient execution of these queries. Two basic organizations are possible, leaving an open question which one should be used and at which time. We look how we can improve the query latency given the queries from historic workloads. Further, we investigate the case when approximate results are allowed as it can introduce additional efficiency improvements.

To search in a large collection of high-dimensional data points we need an efficient index structure. Locality Sensitive Hashing (LSH) provides such a structure used to retrieve an approximation of the exact nearest neighbor results. We investigate if the LSH scheme can be enhanced using an additional preprocessing on the indexed data. Additionally, we investigate how the LSH techniques can be distributed on top of the MapReduce framework to answer similarity search queries in large databases.

1.2 Contributions

With this thesis we make the following contribution:

- We propose Picasso—a framework to recommend music for a given set of images. Picasso utilizes similarity search techniques together with information extracted from movies to make a match between the world of images and the world of music.
- We put forward a reusable benchmark to evaluate the effectiveness of soundtrack recommendation approaches for images. The benchmark contains a large set of preference assessments, concerning a pair of songs with respect to a given set of images as a query. We describe the details of the benchmark, report on the effectiveness of Picasso, and compare it to the performance of a baseline approach.
- We address the problem of processing top-K queries with set-defined selections and propose a partitioned index to efficiently answer these kind of queries. We further show how this index organization can be used to calculate approximate answers with a tunable expected precision, resulting

in additional performance gains. We report on the extensive evaluation using real-world data originating from the PicasSound application profiles.

- We propose two heuristic enhancements to the Locality Sensitive Hashing (LSH) techniques used to process similarity search queries in high-dimensional data. In addition, we investigate how similarity search queries can be processed in a distributed manner using MapReduce together with LSH.
- We describe the implementation details behind PicasSound—the smartphone application built to demonstrate the features of the Picasso approach.

1.3 Selected Publications

The content of this thesis is largely based on the research results published in [SMS10, SM11a, SM11b, SM12a, SM12b, SM13]. In the following we give a high level overview of the most important publications.

Soundtrack Recommendation

We have presented Picasso, a soundtrack recommendation framework for images, in [SM11a]. With Picasso, we have shown how information extracted from publicly available movies can be used to address the problem of soundtrack recommendation for images. The Picasso framework is a major part of this thesis, described in Chapter 4.

- Aleksandar Stupar and Sebastian Michel. Picasso - to sing, you must close your eyes and draw. 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), 2011.

We have build a reusable benchmark dataset to enable continuous reevaluation of the recommendations quality as perceived by the end user. The benchmark and the statistics about the collected relevance assessments are published in [SM13]. More detailed description of the proposed benchmark and its application to the state-of-the-art systems is presented in Chapter 5.

- Aleksandar Stupar and Sebastian Michel. SRbench—A Benchmark for Soundtrack Recommendation Systems. 22nd ACM International Conference on Information and Knowledge Management (CIKM), 2013.

Efficiency Aspects

In [SM12a] we have addressed the problem of top-K queries where the set of possible results is constrained to a set of items. In our setting, the problem appears when top-K songs are selected such that they are contained in a user’s smartphone. This problem is, however, more general and appears in a multitude of scenarios. Chapter 6 of the thesis is based on this publication.

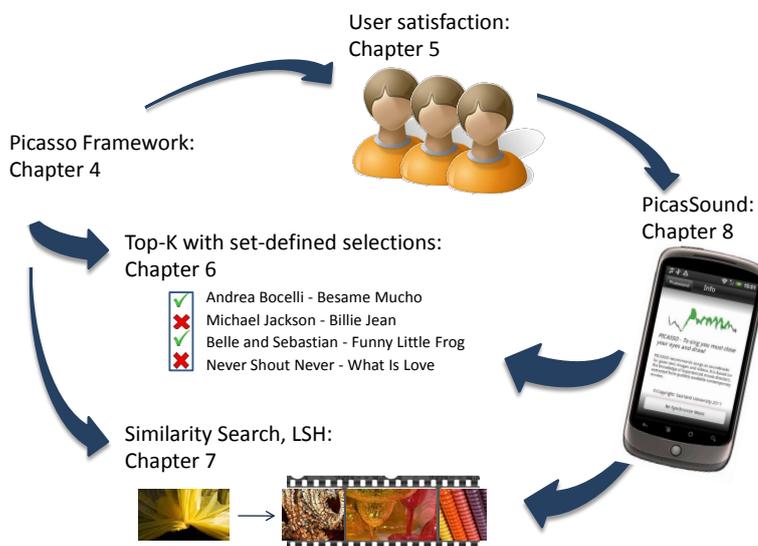


Figure 1.2: Outline with the interplay between different aspects of the thesis

- Aleksandar Stupar and Sebastian Michel. Being picky: processing top-k queries with set-defined selections. 21st ACM International Conference on Information and Knowledge Management (CIKM), 2012.

Locality Sensitive Hashing (LSH) is used for efficient similarity search in high-dimensional spaces, such as image similarity search based on the feature vector representation. We have proposed two enhancements to the LSH techniques in [SM12b], described in Chapter 7 of this thesis.

- Aleksandar Stupar and Sebastian Michel. Enhancing Locality Sensitive Hashing with Peek Probing and Nearest Neighbor Links. 15th International Workshop on the Web and Databases (WebDB), 2012.

1.4 Outline of the Thesis

The organization of the thesis is as follows. Chapter 2 starts with an overview of multimedia information retrieval with a special attention given to similarity search techniques. It further gives an insight into evaluation techniques and MapReduce—a framework for distributed computing. Chapter 3 discusses work related to the various aspects of the thesis. It gives an overview of the soundtrack recommendation approaches for various types of queries, recommendation of images to music, and automated slide show generation. Additionally, it covers approaches for top-K query processing and collecting relevance assessments. The Picasso soundtrack recommendation framework for images is described in Chapter 4. The benchmark dataset to evaluate the effectiveness of soundtrack recommendation systems for images is proposed in Chapter 5. It also contains the results of the evaluation of Picasso and a baseline system using the proposed

benchmark. Chapters 6 and 7 address efficiency aspects of Picasso. Processing top-K queries with set-defined selections is addressed in Chapter 6, while Chapter 7 proposes enhancements to the Locality Sensitive Hashing (LSH) scheme. Chapter 7 also shows how the LSH techniques can be distributed to enable large-scale similarity search. PicasSound—a smartphone application based on the Picasso approach is described in Chapter 8. Chapter 9 concludes the thesis and gives an outlook on future work.

Figure 1.2 describes the interplay between different aspects of the thesis.

Chapter 2

Background

This chapter contains background knowledge helpful for understanding the work described in this thesis. It starts with an overview of the multimedia information field in general, and gives a detailed description of image and music features. Similarity search using these features is described together with the data structures for efficient retrieval. Further, evaluation techniques for information retrieval systems are introduced with emphasis on system effectiveness measures. The chapter ends with the brief overview of the distributed data processing system MapReduce that we use to process similarity search queries in case of large multimedia collections.

2.1 Multimedia Information Retrieval

Massive quantities of multimedia content in form of songs, images, and videos, available on the Web, require search capabilities for content exploration, as well as for plagiarism and copyright infringement detection. These search capabilities are enabled by multimedia information retrieval (IR) systems in efficient and effective manner (cf., [BBFV07, Mue07] for an overview). Although, these systems can be considered as traditional textual IR systems extended to support multimedia documents, peculiarities of these documents result in additional required features, which are not considered for text retrieval.

The first challenge of multimedia IR systems is the limited power of users for expressing their information need. The most common approach, taken from traditional IR, is for users to formulate their information need using keyword queries. In this case, multimedia documents are stored in the system with their textual description and existing approaches for text retrieval are used to answer the queries. The textual description is either provided directly by users or extracted from text, containing references to the multimedia document. A special form of short textual descriptions, named tags, became common for this task.

Due to the lack of user engagement in creating textual descriptions there is a large portion of documents without description which limits the applicability

of text search. This gave rise to research on automatically describing multimedia documents [ELBMG07, HBC09, SWSK10]. Basic principles behind these approaches are the usage of features in both textual and multimedia domains to derive *rules* that tell when a term can be applied to a multimedia document. The rules are *learned* from the documents with descriptions using machine learning techniques.

On the other hand, querying by example tries to avoid the problem of textual descriptions by completely avoiding keyword queries. In this case, human effort is slightly increased when specifying a query as a rough example of the searched document must be provided. This creation procedure requires at least sketching an image [SB10] or whistling the melody [PT01] of a song.

In some cases, users might have an example of what they are searching for, an existing image or a song, and want to find the most similar documents for that example [LCL04, LO04]. This kind of search provides means for interactive exploration, where one of the search results is used as a query in the following steps. This research direction resulted in multitude of approaches [HLES06, ZG02, CS08], trying to minimize the so called semantic gap, i.e., the gap between the similarity calculated by the algorithms and the similarity perceived by humans.

The extreme case of similarity search is near duplicate detection [KSH04, WLLM06, BPP⁺05] where only duplicates of the document are of interest to be discovered. This has applications not only in copyright and plagiarism areas but also in tracking a document's spread online and, hence, assessing its popularity. Techniques for near duplicate detection are also used to remove duplicates from search results, such that users are not overwhelmed by multiple copies of the same document.

Another kind of systems, especially interesting for the music domain, assume an exploration process of multimedia content without an explicitly specified query. The idea is to allow for exploration of yet unseen (unheard) content which might appeal to the user, based on the user's profile containing information about previously seen (heard) documents. This kind of systems are commonly known as recommendation systems [Cel10b, NRSM10].

The main ingredient in all of the described tasks for multimedia IR systems are the features that are automatically extracted from the documents. These features are considered to contain the essence of the information from the document, in form of a summarized, numerical representation. Different types of features summarize documents from different perspectives. Generally, there are two types of features: low-level and high-level features. Low-level features are usually extracted directly from the documents, representing some patterns in that document and are in general not understandable by the end user. High-level features are concepts about documents that are understandable by the end user and are usually derived from low-level features. In the following we describe low-level features for images and music used throughout the thesis.

2.1.1 Image Features

The Moving Experts Picture Group (MPEG) recognized the importance of features for multimedia documents and published the MPEG-7 [CSP01] standard on the various aspects of multimedia content descriptors, including low-level features for images [MOVY01]. By this standard, low-level features for images are divided in three groups based on the aspect they are summarizing, namely color-based, texture-based, and shape-based features. The work described in this thesis uses color- and texture-based features, thus, most of the focus will be put in explaining these.

As information is spread over multiple regions in the image, the MPEG-7 standard introduced the concept of grid layout by dividing the image in equal sized rectangles. The low-level features are then computed not only for the whole image but also for the rectangle regions defined by the layout.

Color-Based Features

As the name suggests, color-based features summarize information about the distribution of colors inside the image. The standard supports colors encoded in a large variety of color spaces ranging from RGB, over YCrCb to Monochrome. Information on the **color space**, together with information on **color quantization**, is the base information that accompanies all other feature descriptors. Uniform color space quantization with configurable number of bins is supported by the standard.

In addition to color space and color quantization there are five color descriptors, namely: dominant color, scalable color, color layout, color structure, and group of frames/group of pictures.

The **dominant color** descriptor summarizes the image through its most representative colors, which are discovered through a color quantization procedure. The top- N representative colors are specified together with information on the percentage of the image containing that color. Optionally, for each dominant color there is a variance specified indicating the variability of colors from the cluster of the representative color. The overall spatial homogeneity of the colors is calculated and presented as a single measure for the whole image.

Scalable color describes all the colors found in the image by aggregating them in a single color histogram in the HSV color space using a fixed color space quantization. Each bin of the histogram represents one level of the color quantization. To lower the memory space needed for the histogram, the Haar transform [Haa10, SS99] is used for encoding. The standard defines three possible values for the number of Haar coefficients used for representation: 128, 64, and 32.

The **color structure** descriptor represents the colors found in the image in the HMMD color space together with the structure of their appearance. With this descriptor, two images with the same ratios of identical colors can be distinguished if the structure of the color distributions is different. The final representation in this case is also a color histogram, where each bin represents the

quantization of the color space. The value of the bin is the number of structuring elements, of 8×8 pixels in size, in which the respective color appears. Color appearances are counted while sliding the structural element over the whole image. The number of color quantization levels is a parameter for this descriptor and can be 184, 120, 64, and 32 as defined by the standard.

The **Color layout** descriptor summarizes the spatial distribution of colors in a given image using the YCbCr color space. First, a grid layout of the image is created by dividing the image into 64 (8×8) equal sized rectangles. In the second step, one representative color for each of the 64 parts is calculated, as recommended by the standard, by calculating an average color of all pixels in that part. The discrete cosine transform (DCT) is performed on these 64 parts and only low frequency coefficients, for each color component, are used as descriptor features. The number of coefficients is flexible by the standard, with the recommendation that 6 coefficients are used for luminance and 3 for each chrominance.

The **group of frames/group of pictures** descriptor extends scalable color descriptor to the case of multiple images, usually used as a video descriptor, summarizing information from multiple consecutive frames. The extraction of the descriptor is performed by aggregating histograms of multiple images into a single histogram and applying the Haar transform on this histogram as in case of the color structure descriptor. The aggregation is simply done as the average of all the histograms for each image. To avoid the influence of outliers to the mean operator, median histograms can be used instead of average histograms.

Texture-Based Features

For the texture-based features MPEG-7 defines three kind of descriptors: edge histogram, homogeneous texture, and texture browsing.

The **edge histogram** descriptor is used for the spatial representation of the edges in the image. The image is divided by creating a grid layout of 16 (4×4) parts. A local edge histogram, with five bins each, is calculated for each part of the image. Four bins are used for edges with four different orientation directions and one is used for non orientation specific edges. Concatenating local histograms and applying histogram levels quantization produces final values of the descriptor.

The **homogeneous texture** descriptor shows how homogeneous the texture patterns in the image are and is usually used for the search of similarly looking patterns in the image dataset. The feature extraction is done by applying 30 Gabor filters [Fei98], coming from the combination of 5 scale and 6 orientation values. The first and the second moment of the energy for each of the filters are considered as the features of homogeneous texture descriptor.

The **texture browsing** descriptor is a compact representation of the homogeneous texture in the image used for browsing purposes. Similarly to the homogeneous texture descriptor, texture browsing describes textures in terms of regularity, coarseness, and directionality. Features of the descriptor are calculated by first determining two dominant orientations and then projecting the

image on these orientations to determine regularity and coarseness.

Shape-Based Features

Defined by MPEG-7 standard, there are three kinds of shape-based features: region shape, contour shape, and shape 3D. The distribution of the shape's region is summarized through the **region shape** descriptor supporting complex objects that are consisting of multiple disconnected regions. The **contour shape** descriptor summarizes the shape by the features of its contour using the curvature scale space representation [MB11]. It is robust against partial occlusions and reflects properties of the human visual system. Through **shape 3D** descriptors, the MPEG-7 standard enables the description of 3D mesh models, which are used for the search, retrieval, and browsing tasks.

2.1.2 Music Features

There is a large variety of music features currently used among researchers in the multimedia retrieval field. Among the most known and widely accepted are Mel Frequency Cepstral Coefficients (MFCC) and Chroma features. In addition, we use a following set of features: spectral centroid, spectral rolloff, spectral flux, and time domain zero crossing [TC02]. The spectral representation of a music signal is used as a basis for extracting most of the music features described here. Short time Fourier transform [All77] (STFT) is used as a spectral representation for each time frame of the signal.

Mel frequency cepstral coefficients (MFCC) features originated from the field of speech recognition [ME05], extending later to the wide area of applications in music IR [MCM01, TC02, Mue07]. The main characteristic of MFCC features is that they are based on the mel scale—a model of the human auditory system—emphasizing the spectral distribution of a signal in the similar manner as perceived by humans. The mel frequency scale is defined to be linear under the reference frequency of 1000Hz, and logarithmic above that frequency. Mapping between the frequency in Hz and frequency in mel scale is given as:

$$M = \begin{cases} f, & \text{if } f < 1000 \\ f_c \cdot \log_2(1 + \frac{f}{f_c}), & \text{if } f \geq 1000 \end{cases} \quad (2.1)$$

where f is the frequency in Hz and f_c is the given corner (reference) frequency. There is no universal agreement about the corner frequency, however, the value of 1000 is frequently used.

To extract the MFCC features, first the spectral representation of a musical signal is transformed to the presented mel scale and then the discrete cosine transform (DCT) is performed to reduce the correlation between the final features. Finally, the first n coefficients of the DCT are used as the MFCC feature values. Although it is common for speech recognition tasks to use 13 DCT coefficients, for music IR tasks the number of coefficients can be reduced with minimal loss in quality.

Chroma [EP07] features represent the spectrum of the music by dividing it in twelve bins, each corresponding to one semitone. The final feature values are the energy (intensity) values of each of the semitones in the given time frame. The motivation for using twelve semitones is obvious as most of the western music is based on musical scales containing a subset of twelve traditional pitch classes. As semitones one octave away are interpreted as similar by humans the values of their intensity are aggregated into a single value, as illustrated in Figure 2.1.

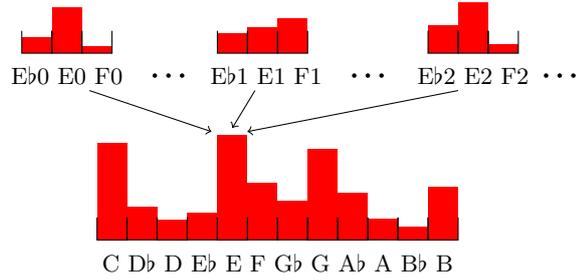


Figure 2.1: Aggregating semitone energy for chroma features

Chroma is a very valuable descriptor, used for musical similarity measures, as it strongly emphasizes melodic and harmonical characteristics of the analyzed musical piece. In addition, these features are highly robust to the change in instrumentation and timbre.

The **spectral centroid** descriptor represents a shape of the music signal spectrum by its center of gravity. For each time frame the spectral centroid is calculated as follows:

$$C_t = \frac{\sum_{f=1}^N f \cdot M_t(f)}{\sum_{f=1}^N M_t(f)} \quad (2.2)$$

where f represents a frequency bin produced by the STFT, and $M_t(f)$ is the magnitude of frequency bin f for the time frame t . If spectral centroid is high there are more high frequencies in the music, similarly when the centroid is low there are more low frequencies. This results in a high correlation between the perceived “brightness” of the sound and the value for the spectral centroid.

Spectral rolloff also represents a shape of the music signal spectrum by specifying how quickly the spectrum tails down towards the high frequencies, motivated by the fact that music signals tend to have less energy in high frequencies. It is measured as 85th percentile of the spectrum, i.e., the value is given as the frequency under which 85% of the signal’s magnitude is located, defined as:

$$\sum_{f=1}^{R_t} M_t(f) = 0.85 \cdot \sum_{f=1}^N M_t(f) \quad (2.3)$$

where R_t is the value of spectral rolloff and f and $M_t(f)$ are frequency and its magnitude, respectively. Spectral rolloff can be considered as a cutoff position between harmonic and noise frequencies.

Spectral flux describes a local change of spectrum throughout the time domain. It is calculated as a squared difference between two successive spectral distributions:

$$F_t = \sum_{f=1}^N (N_t(f) - N_{t-1}(f))^2 \quad (2.4)$$

where $N_t(f)$ is the normalized magnitude of frequency bin f at time t . Normalizing the magnitude of the spectrum avoids the influence of the change in the energy and only considers the change in the distribution of spectrum. Spectral flux is found to correlate with timbre of music making it valuable for the timbre classification. In conjunction with rising energy values it can also be effectively used for onset detection.

Time domain zero crossing is one of the rare features not directly based on the spectrum of the signal. It is defined in the time domain as a number of times a signal crosses between positive and negative values in the time frame, calculated as:

$$Z_t = \frac{1}{2} \sum_{t=1}^T |sgn(x[t]) - sgn(x[t-1])| \quad (2.5)$$

where $x[t]$ is the value of the signal at time point t and sgn is the sign function returning 1 for positive arguments and 0 otherwise. Time domain zero crossing provides a good measure of the noisiness of the signal, the noisier the signal is the higher the value is. It has applications in the field of rhythm extraction and percussive sound classification.

2.2 Similarity Search

Searching for the most similar documents plays an important part in multimedia IR. Not only does it enable exploration tasks and retrieval of similar documents, but it also enables inference of document attributes based on the similarity with other documents. A common way of performing similarity search is to calculate the distance between the feature representation of the documents; the closer the features are the higher the similarity between documents is.

As we saw in previous sections, all described feature representations represent a document as a point in a multidimensional space where the dimensionality of the space is defined by a specific feature descriptor. In these cases, searching for the most similar document translates into a more general problem of searching for the closest data points in the feature space. This problem is commonly known as the K-Nearest Neighbor problem.

The **K-Nearest Neighbor (KNN)** problem is defined for a metric space M and a collections of data points C as follows: given a query point q find the K closest points from C , i.e., find a set of points $N \subseteq C$, $|N| = K$, such that $d(q, n) \leq d(q, p)$ for each $n \in N$ and $p \in C \setminus N$, where $d(x, y)$ is the distance between two points x and y . Alternatively, similarity search is done using range queries which have a similar notion to KNN queries.

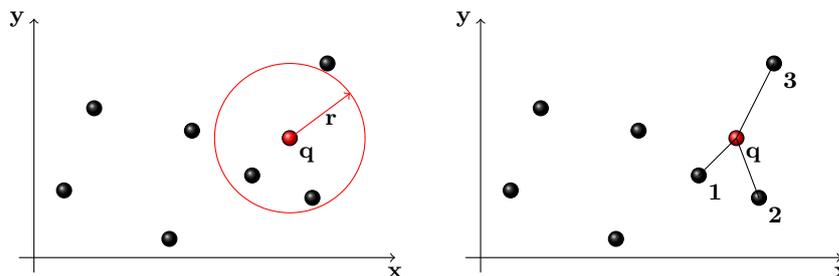


Figure 2.2: Similarity search: a) range query b) K-Nearest Neighbor query

A **range query** over a metric space M and a collection of data points C is defined by a query point q and a range r . The result of a range query is a subset of points whose distances to q are smaller than r , that is, a set of points $N \subseteq C$ such that $d(q, n) \leq r$ and $d(q, p) > r$ for each $n \in N$ and $p \in C \setminus N$. Figure 2.2 illustrates a range query (a) and a KNN query (b) in two-dimensional space.

A straightforward solution to both of the stated problems is achieved by calculating the distance between the query point q and each of the points $p \in C$. While the distances are calculated the closest K points are maintained for the case of KNN queries and all the points in the range r are kept for range queries. As we can see, this approach does not require any additional space utilization for indexing but requires distance calculation for each point in the collection. A large number of indexing approaches have been developed to tackle this problem and minimize the effort used to answer similarity queries. Among the most prominent are approaches based on space partitioning such as the k-d tree [Ben90], R-tree [Gut84], and X-tree [BKK96] (cf., see [Sam06] for an overview of multidimensional indexing structures).

The **k-d tree** [Ben90] is a binary tree used to index a set of data points with an arbitrary dimensionality k . This is achieved by splitting points into two sets at each level of the tree by inspecting the point's value at one of the dimensions. Choosing a partitioning dimension at each level of the tree is dependent on the variant of the k-d tree. However, the most common approach is to partition based on the values of the points inserted in the tree and in the constant predefined order of dimensions. For instance, in two dimensions (x, y) , we adopt a strategy that dimension x is compared at levels $\{0$ (root), $2, 4, \dots\}$ and dimensions y is compared at levels $\{1, 3, 5, \dots\}$, such that smaller values than the node value are placed in left subtree and larger and equal values are placed in the right subtree. An example of the space partitioning for two-dimensional data is shown in Figure 2.3(a), where numbers indicate the level of the tree defining a split.

When inserting points in the k-d tree we perform the traversal of the tree according to the point that is inserted until the first empty subtree is found. A new node is created with the value taken from the inserting point at the dimension that corresponds to the depth of the empty subtree. The created

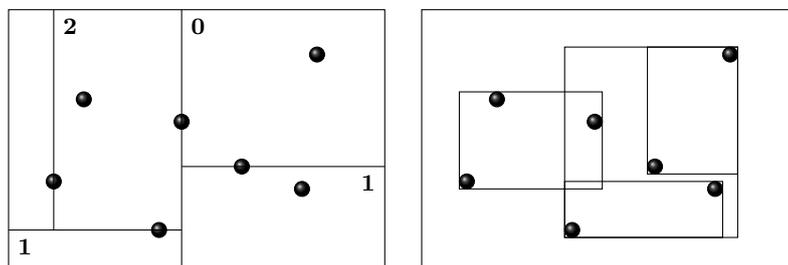


Figure 2.3: Space partitioning: a) k-d tree b) R-tree

node is then added to the tree at the place of the empty subtree. It is important to note that this inserting algorithm can lead to an unbalanced tree, which can be avoided in case all data points are known upfront. In that case, the median of the values at the corresponding dimension is taken for the node value, resulting in a balanced tree.

To remove a data point from the k-d tree we can employ a simple solution by first locating the node of the point and then removing the subtree of that node. Once the subtree is removed, we add each data point again to the tree except the point that is to be removed. A more elegant method would be to remove the node directly from the tree but find an adequate substitution from the subtree and apply this procedure recursively for the substitution node. An adequate substitution is a node from the right subtree having the smallest value for the dimension at level of the removed node, or a node from the left subtree having the largest value at the same dimension.

A range query with a query point q and a range r can easily be answered using a k-d tree. The tree is traversed from the root using a distance r from the value of q at the corresponding dimension to prune away subtrees that can not contain points in the range. On the other hand, answering KNN queries using k-d tree is more complicated as we do not know the radius r containing the closest K points. To answer a KNN query we first traverse the tree using the query point q as a guiding reference until the leaf node is encountered. The leaf node is then added to a priority queue containing results and we move one node up the tree. At this node we check if we need to visit the other branch in which case we visit it recursively. It is important to note that branch needs to be visited only if it might contain points closer than the current K^{th} farthest neighbor.

The **R-tree** [Gut84] is based on the principle of minimal bounding rectangles, i.e., a minimal rectangle that bounds the area containing a set of data points. These bounding rectangles are used as a rough approximation of the contained data points such that it can easily be determined when the query region or a query point does not intersect with an area of the represented point set.

An R-tree is organized such that each node contains between m and M bounding rectangles representing corresponding subtrees, with $m \leq \lceil \frac{M}{2} \rceil$, while

indexed data points are contained in the leaf nodes of the tree. When inserting or removing an element all affected bounding rectangles have to be updated. In addition to this, care must be taken that each node still has between m and M elements. This is achieved by removing nodes with less than m elements and reinserting their children elements and by splitting nodes with more than M elements. Figure 2.3 (b) illustrates an example of space partitioning with minimum bounding rectangles in two dimensions with $M = 3$ and $m = 2$.

Inserting elements in an R-tree is done by starting from the root node and following the path of the child whose bounding rectangle needs to be extended the least to include the inserting object. This strategy minimizes the coverage of the resulting bounding rectangles. Alternative strategies can be used taking into account the overlap between the rectangles, as done in the R*-tree [BKSS90]. In a similar manner, different strategies can be used to split a node with more than M elements into two nodes. The proposed approach for the original R-tree consists of finding two seed rectangles and extending them for each remaining rectangle always choosing the one with the smaller extension. Seed rectangles are chosen either by looking at pairs of rectangles, resulting in a quadratic algorithm, or by looking at individual rectangles at individual dimensions resulting in a linear algorithm.

Answering range and KNN queries using an R-tree is done in a similar manner as with the k-d tree. Answering range queries starts from the root node and continues by traversing all the child nodes whose bounding rectangle intersects with the region of the range queries. For KNN queries, the query point q is used to initially navigate through the tree until the leaf element is encountered. Afterwards, a recursive unwinding is done with the pruning of bounding rectangles using the fact that the distance of the query point to the bounding rectangles is smaller or equal than the distance to the points contained in them.

The **X-tree** [BKK96] is based on the R-tree structure trying to address problems caused by high-dimensional data. The increased dimensionality often results in a unavailability of good data partitioning resulting in a low pruning power of bounding rectangles. The idea behind the X-tree is to avoid splitting nodes in case the overlap between two split partitions is too large. In such cases, the node size is enlarged and the node becomes a so called *supernode*.

Splitting of node data into partitions is done along a single dimension in two phases. In the first phase, a topological split is done, minimizing the overlap between two partitions. If a good enough partitioning is produced in the first phase it is used, otherwise, a partitioning is done based on the split history in the second phase. The partitioning of the second phase always results with the minimal overlap between the partitions, however, the sizes of the partitions may be very unbalanced. In the case of unbalanced partitions a *supernode* is created.

As the X-tree only redefines splitting policy and enlargement of the nodes, the search through the tree to answer range and KNN queries is performed as in an R-tree.

2.2.1 Locality Sensitive Hashing

With increasing number of dimensions, answering range and KNN queries becomes intrinsically hard to solve. This stems from the fact that increasing dimensions increases the volume of the space exponentially. This results in increase of the data sparseness, commonly referred to as the “curse of dimensionality”. Beyer et al. [BGRS99] inspected the meaningfulness of the KNN search with respect to the increasing number of dimensions and conclude that KNN is not meaningful in cases when

$$\lim_{k \rightarrow \infty} \frac{\text{Variance}[d(p, q)]}{\text{Expected}[d(p, q)]} = 0 \quad (2.6)$$

where p and q are two randomly chosen data points, d is a distance measure, and k the dimensionality of the space. That is, KNN search makes sense if the distance between the farthest and the closest neighbor does not converge to the same value with increasing dimensionality.

Due to the *curse of dimensionality*, the pruning power of tree structures becomes very weak. This often results in scanning the whole tree structure which, due to complexity of the data structure, downgrades the performance below the full scan of the data, starting with dimensionality as low as 10 to 15 dimensions [WSB98].

To avoid these problems in high-dimensional spaces, range and KNN queries are reformulated into approximate queries. For a range query with a range r approximate methods return results that are in range $r(1 \pm \varepsilon)$, where $\varepsilon > 0$ is given as a parameter to trade off between the accuracy and the processing effort. Similarly, approximate KNN methods return K data points that lie in the $r(1 + \varepsilon)$ range, in which case r is the distance to the real K^{th} nearest neighbor.

Locality Sensitive Hashing (LSH) [AI06, DIIM04, GIM99] is proposed as a solution to approximate KNN problem, rendering KNN processing efficient in high-dimensional space. The basic principle behind LSH is the usage of *locality preserving* hash functions which map, with high probability, close points from the high-dimensional space to the same hash value (i.e., hash bucket). Formally defined: function f is said to be (r_1, r_2, P_1, P_2) -sensitive for metric space $\mathfrak{M} = (M, d)$ if for any two points p and q from M following holds:

$$\begin{aligned} d(p, q) \leq r_1 &\Rightarrow P(f(p) = f(q)) > P_1 \wedge \\ d(p, q) \geq r_2 &\Rightarrow P(f(p) \neq f(q)) > P_2 \end{aligned} \quad (2.7)$$

where $P(f(p) = f(q))$ denotes a probability of points colliding and $P(f(p) \neq f(q))$ denotes the probability of points not colliding at the same hash value. A family of hash functions F is said to be (r_1, r_2, P_1, P_2) -sensitive if each function $f \in F$ is (r_1, r_2, P_1, P_2) -sensitive.

In this thesis, we consider a family of LSH functions based on p -stable distributions [DIIM04] which are most suitable for l_p norms, including the ubiquitous Euclidean l_2 norm. In this case, for each data point \mathbf{v} , the hashing scheme con-

siders k independent hash functions of the form

$$h_{\mathbf{a},B}(\mathbf{v}) = \left\lfloor \frac{\mathbf{a} \cdot \mathbf{v} + B}{W} \right\rfloor \quad (2.8)$$

where \mathbf{a} is a d -dimensional vector whose elements are chosen independently from a p -stable distribution, $W \in \mathbb{R}$, and B is chosen uniformly from $[0, W]$. Each hash function maps a d -dimensional data point into an integer. With k such hash functions, the final result is a vector of length k of the form $g(\mathbf{v}) = (h_{\mathbf{a}_1, B_1}(\mathbf{v}), \dots, h_{\mathbf{a}_k, B_k}(\mathbf{v}))$, representing a hash label for a given point \mathbf{v} .

To increase the probability of collision for close points and decrease the probability of collision for points that are far apart, multiple hash tables are used, each with its own hash function drawn randomly from the same family of hash functions. At query time, the query point is hashed using the hash function of the corresponding hash table and the resulting label is used to retrieve a bucket containing potential nearest neighbors. The distance to all points contained in that bucket is calculated and the nearest K points are returned as the final result.

Looking at the family of hash functions based on p -stable distributions we see that we have several ways to create a hash function, varying not only the vector \mathbf{a} but also varying the parameters W and k . These two parameters have a direct impact on the r_1 and r_2 values determining how many neighboring points will end up in the same bucket. Clearly, the more points end together will result in a higher accuracy but will downgrade the performance. Modeling of LSH has been done in [DWJ⁺08] such that appropriate values of parameters can be chosen depending on the data being indexed. However, once parameters are chosen at index creation time they are not adapting in case a newly added data changes the distribution.

To solve this problem, LSH Forest was proposed [BCG05], with the basic idea of varying the value k for each data point and storing data points into a tree structure. At query time, the tree structure is traversed first top-down to locate the node with the highest overlap in labels with the query labels, and then bottom-up to expand the search to nodes with smaller overlap but may contain potentially relevant data points. This way, an adaptive quality can be achieved at query answering time rather than having it constrained upfront at index creation.

Multi-Probe Locality Sensitive Hashing

At query time, the LSH index is used to retrieve all points that have the same hash value as the query point. The observation that a significant fraction of the closest neighbors are located in the buckets with neighboring hash values led to the development of the Multi-Probe approach [LJW⁺07], an alternative approach to query an LSH index.

At query time, the distance between query point and a set of buckets is calculated and the closest buckets are used to retrieve the points contained in them. The distance measure between the query point and a bucket is given as

a sum of distances for each locality sensitive function of that hash table, which is given as

$$d(v, L) = (L - h_R)^2 = \left(L - \frac{\mathbf{a} \cdot \mathbf{v} + B}{W}\right)^2, L > h_R \quad (2.9)$$

$$d(v, L) = (L + 1 - h_R)^2 = \left(L + 1 - \frac{\mathbf{a} \cdot \mathbf{v} + B}{W}\right)^2, L \leq h_R \quad (2.10)$$

where L is the label (the integer value) of the bucket, obtained by hashing any point of the bucket with a given LSH function, and h_R is a real value obtained by projecting the query point onto the hashing vector of the corresponding LSH function. The other symbols are the same as for the basic LSH approach above.

A set of buckets, for which the distance to the query point is calculated, is restricted to the buckets whose hash value differs to the query hash value at only one hash function with the difference not larger than 1. This restriction is empirically motivated as the largest fraction of closest neighbors is contained in this set of buckets. An efficient algorithm based on perturbations is used to find the closest buckets to a query point from this restricted subset.

The Multi-Probe approach results in a higher precision with the same number of hash tables used and the same query answering time. When the set of previously seen queries is available, probing multiple buckets can be done based on learned probability distributions, as described in [JB08], resulting in further improvements.

2.3 Evaluation Techniques

With the introduction of information retrieval systems, one of the biggest concerns has been the quality of the systems as perceived by the users. Although low retrieval time is important too, the satisfaction of a user's information need expressed through a query is essential. As it may be simple to measure the system quality in terms of efficiency, e.g., measuring query response time, it is intrinsically hard to measure its performance in terms of perceived result quality.

The common approach to measuring effectiveness in IR is based on the notion of relevance. Retrieved documents are considered either relevant or not relevant to a specified query. Clearly, high quality is achieved by retrieving as many relevant documents as possible and omitting the irrelevant ones. This binary view on relevance is further extended into a graded perspective where relevance is measured on a scale from completely irrelevant to highly relevant. Figure 2.4 shows an example of graded relevance assessments for image retrieval. The impression of the document's relevance may highly depend on other documents in the collection, which is not captured with absolute relevance measures. To capture this, preference-based relevance is proposed [Ror90, CBCD08], where more relevant document is selected from usually a pair of documents. Relevance can further be considered based on the topic of the document, the task user is solving, the current context, and so on (cf., for historical development of relevance [Miz97]).

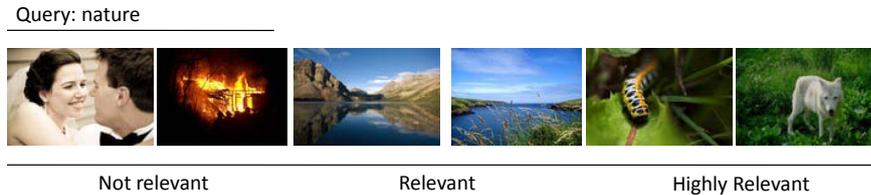


Figure 2.4: Graded relevance for image retrieval

Documents have to be assessed by human judges to determine whether they are relevant for a given query or not. This task requires a large human effort as there is most of the time a multitude of documents that have to be assessed for each query. In addition, there is an unlimited space of queries that can be explored. To alleviate this problem, a collective effort is made through venues such as TREC [Voo07, TRE], TRECVID [TVI], ImageCLEF [ICL], and MIREX [MIR]. These venues result in standardized document and query collections together with a set of relevance assessments gathered by the participants. Ideally, the collected relevance assessments are reusable, such that they can be used to test newly created systems.

Despite the collective efforts, assessing the relevance of the whole document collection for a given query is practically infeasible. To reduce the number of assessed documents, an initial filtering is performed based on the results of the existing systems. Existing systems execute a given query and the resulting documents are merged into a pool of documents, for which the assessments are done. This technique is commonly known as *pooling*.

2.3.1 Effectiveness Measures

Relevance assessments for documents enable us to calculate various measures of a system's effectiveness. Among the most used measures are precision and recall, together with their combination, the F measure [MRS08].

Precision is defined as the ratio between the number of retrieved relevant documents and all retrieved documents:

$$Precision = \frac{|\text{relevant documents retrieved}|}{|\text{documents retrieved}|} \quad (2.11)$$

Recall is defined as the number of relevant documents retrieved compared to the number of all relevant documents:

$$Recall = \frac{|\text{relevant documents retrieved}|}{|\text{relevant documents}|} \quad (2.12)$$

As we can see from the definitions, these two measures are opposing each other. We can easily achieve a recall of 1.0 by retrieving all the documents but this would ruin the precision. On the other hand, we can retrieve the documents we are completely sure about, increasing the precision but decreasing the recall. Thus, these two measures are often combined into a single measure by

calculating a harmonic mean between them, commonly referred to as F measure:

$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha) \frac{1}{R}} = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}; \quad \beta = \frac{1 - \alpha}{\alpha} \quad (2.13)$$

where P stands for precision, R for recall, and $\alpha \in [0, 1]$ is a weight parameter giving more weight to the precision or to the recall.

As defined, the precision and recall measures are rank agnostic, i.e., the importance of the retrieved documents is not dependent on the rank at which it was retrieved. In ranked retrieval, a simple extension is to look at the precision and recall depending on the number of retrieved documents k , resulting in a precision@ k and recall@ k .

The precision@ k for multiple values of k can be aggregated into a single measure by computing the average, commonly known as the average precision. Precision@ k is averaged for the values of k at which a relevant document has been retrieved. Calculating the mean value of an average precision over a set of queries results in the mean average precision (MAP) measure.

The normalized discounted cumulative gain (NDCG) [JK02] was proposed as a measure that is directly taking into account the rank information, discounting the gain of lower ranked relevant documents. NDCG supports graded relevance scores, i.e., multiple levels of relevance. NDCG for a ranking with k ranked documents is calculated as follows:

$$NDCG_k = Z_k \sum_{i=1}^k \frac{2^{rel_i} - 1}{\log_2(i + 1)} \quad (2.14)$$

where rel_i is the graded relevance for document at position i , and Z_k is the normalization factor, calculated such that the perfect ranking gets a score of 1. It is important to note that graded relevance has higher values for more relevant documents.

2.4 MapReduce Framework

MapReduce [DG04] is a framework for a large-scale, distributed computation. It is built on top of the Google Distributed File System [GGL03], which enables distribution of data over the cluster machines in a scalable and fault tolerant manner. Fault tolerance is achieved through data replication, such that common failures of commodity machines are easily handled. Further, MapReduce supports fail tolerance by replication results of the partially processed tasks so that they are not repeated after node failures. Tight integration of MapReduce with the distributed file system enables it to move initial calculations where data resides, minimizing network bandwidth bottlenecks caused by data shipping during processing.

The MapReduce framework gained a lot of attention in the open source community which resulted in a multitude of available implementations. The one most widely used is Hadoop [HAD], maintained by the Apache Foundation,

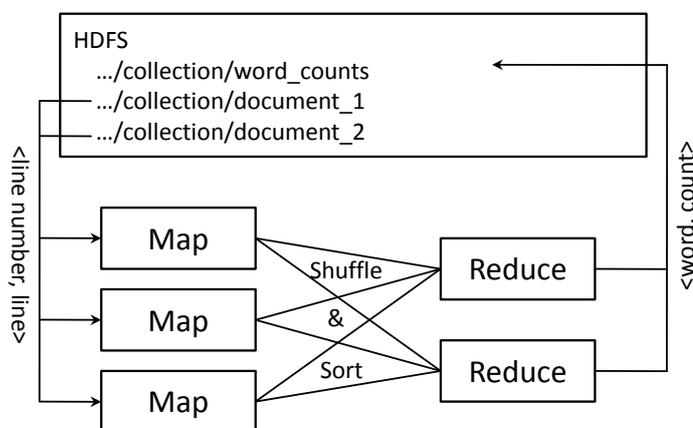


Figure 2.5: The MapReduce framework

which provides a Java-based implementation of both the MapReduce framework [DG04] and the Google like Distributed File System [GGL03] (coined HDFS for Hadoop Distributed File System). In the last years, Hadoop gained a lot of popularity in the open source community and is also part of many research efforts investigating large data processing [KBHR12, ZCW12, MBGZ13].

MapReduce provides a fairly simple programming model, based on two developer-supplied functions: *Map* and *Reduce*. Both functions are based on key-value pairs. The Map function receives a key-value pair as input and emits multiple (or none) key-value pairs as output. The output from all Map functions is grouped by key, and for each such key, all values are fed to the Reduce function, which then produces the final output from these values.

In the Hadoop implementation, the input data is grouped in so-called *input splits*, which often correspond to blocks in the distributed file system. A number of so-called *mapper* processes call the Map function for each key-value pair in such an input split. A number of mappers can run concurrently on each node in the cluster, and the mapper processes are in addition distributed over all nodes in the cluster. Ideally, a mapper is run on the same node where the input block resides, but this is not always possible due to workload imbalance.

Similarly, after all mappers have finished, dedicated *reducer* processes are run on nodes in the cluster. Each reducer handles a fraction of the output key space, copies those key-value pairs from all mappers' outputs (in the so-called *shuffle phase*), sorts them by key, and feeds the to the Reduce function. The output of the reducers is usually considered the final result but can also be used as input for following MapReduce jobs.

Optionally, users of MapReduce framework can specify a *Combine* function which operates on the same type of key-value pairs as does the Reduce function. Combine function is used as an optimization step, running a potential aggregation of Map results before they are shuffled and sent over a network.

Figure 2.5 illustrates a MapReduce framework with the well-known word count example. The task of the word count application is to count the number

of occurrences for each word appearing in a given document collection. As an input to the Map function a $\langle \text{line number}, \text{line} \rangle$ pair is given. The Map function emits a $\langle \text{word}, 1 \rangle$ pair for each word in a given line. After *shuffle and sort* phase, reducers sum up individual occurrences of the word to produce the total count as a result.

Chapter 3

Related Work

In this chapter, we give an overview of the state of the art in soundtrack recommendation, first describing competing approaches that address soundtrack recommendation for images, followed by approaches when a video or a context of a user is given as a query. Further, approaches that retrieve images for a given music piece, solving the inverted problem of soundtrack recommendation, are discussed. Having images and music selected, we discuss approaches that automatically arrange them in an appealing slide show video. We discuss approaches to collect relevance assessments that are related to the approach we utilize while building the benchmark to evaluate soundtrack recommendation systems. Last, we give an overview of the approaches for efficient top-K query processing focusing on the queries with constraints.

3.1 Soundtrack Recommendation

There is a growing interest in the problem of soundtrack recommendation, which resulted in a variety of approaches (cf., [KR12] for an overview). The problem is considered for a wide spectrum of available input data types, i.e., query types. Soundtrack recommendation with images given as a query has already been addressed in couple of approaches [LS07, DPC11, Woo09], and given the vast amount of potential users, in particular with the increasing number of digital cameras and the big success of portals like Youtube [YT] and Flickr [FLI] we can expect further growth in interest. The same potential is recognized also for the problem of soundtrack recommendation for videos, resulting in a range of approaches [MKYH03, CPD⁺10, NSK03]. Still, the majority of soundtrack recommendation approaches address the problem of context-aware music recommendation [LL07, BKL⁺11, RM06, CZW⁺07, KFTRC12], where a user's context, such as the currently performed activity or current weather state, is considered as a query.

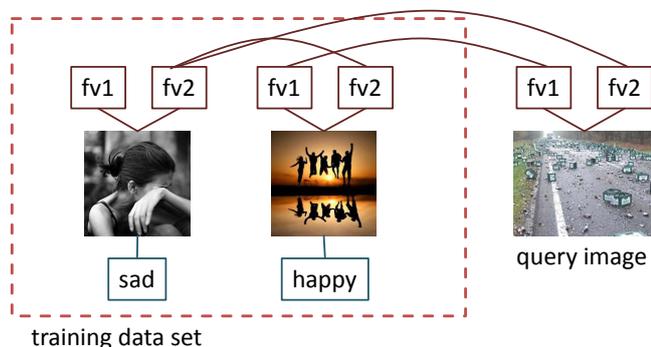


Figure 3.1: Detecting emotions in query images

3.1.1 Soundtrack Recommendation for Images

The approach developed by Li and Shan [LS07] addresses the problem of soundtrack recommendation for images. It was originally developed with the aim of recommending music to impressionism paintings, but it can be applied to arbitrary sets of query images. The key idea behind the approach is to detect emotions in both images and music and to employ this information for the match making. That is, emotions are the common ground that connects both worlds. The detection of emotions and the recommendation of music is done through methods based on the graph representation of multimedia objects, called *mixed media graph* [PYFD04].

In the mixed media graph, each multimedia object and the associated attributes are represented as vertices. Usually, there are two vertex types for associated attributes: one for the labels associated with the object and one for the low-level features extracted from the object. Edges connect object vertices to the label vertices that describe the object and to the low-level feature vertices that are extracted from the object. Additional edges are created between the vertices containing low-level features based on K-Nearest Neighbor search (cf., Section 2.2): For each feature vector, edges are created to its K closest neighbor vertices that contain the same type of features.

To detect emotions in the given query images, a training set of images with labeled emotions is represented in the mixed media graph. Additional vertices are added for each query image together with the vertices of the extracted low-level features. After creating the nearest neighbor edges, a random walk with restarts is applied to find the labels, i.e., emotions, with the largest weight. Figure 3.1 illustrates the emotion discovery with two emotions: *happy* and *sad*, using the mixed media graph.

The mixed media graph is again used in the second step of the soundtrack recommendation—this time with songs as multimedia objects. In this step a “dummy” object is created as a query, with emotions from the previous step as labels. Once the edges are created between the labels, again a random walk with restarts is applied, but this time with the aim of finding the songs with the

highest weight. Two sets of songs are contained in the graph, songs from the training set being labeled with emotions and songs from the song collection used for the recommendation. These sets are interconnected through the nearest-neighbor edges. The songs from the collection are recommended in decreasing order of their weight.

Another approach to recommend soundtrack for images, also based on the detected emotions, has been proposed in [DPC11]. The approach is based on the classification of emotions expressed in music into the emotional valance/arousal space [Rus80]. At query time, faces are detected in images together with the analysis of their expressions indicating the mood. In addition to face detection, color intensity, and brightness of the images are also mapped into the valance/arousal space. This way the closest song in the emotional space is used as the soundtrack. The dependence on the face detection limits the approach to images with faces, falling back to using only brightness and colorfulness information when faces are not detected. There has not been a user study performed evaluating the effects of the soundtrack recommendation with this approach.

Recommending soundtrack for images of an event have been addressed in [Woo09]. Submitted images are analyzed to detect event type they represent, scene and material they contain, place of their creation and the holidays they may relate to. A set of classifiers is used to detect the event, scene and material types. These classifiers provide membership results to classes such as party, outdoor sport, beach, urban, snow, or water. Latitude and longitude of captured images are used to detect features describing the location through a database of location-feature mappings. Detecting whether an image is related to holidays is done solely based on the date of the image capturing. This image analysis together with user provided keywords is used to generate a set of keyword queries, where each detected class of events, scenes or material types has a corresponding set of keywords. These queries are then used to query the database of songs indexed by their lyrics. This approach is clearly constrained to the use of music with lyrics only, not addressing large amounts of instrumental music including the whole classical music corpus.

3.1.2 Soundtrack Recommendation for Videos

The problem of soundtrack recommendation in a video setting has been addressed by Mulhem et al. [MKYH03]. It is based on the vector space model for both music and video where the number of dimensions depends on the number of features used. These vector space models are further mapped to a unified pivot vector space to make a match between different media, i.e., match between video and music.

The approach uses light, color, and motion features extracted from videos. Light features describe the contrast in brightness between lighted and shaded objects, while color features represent the distribution of colors in the frame. Motion vectors between frames are used as the motion features. Beside low-level features, such as spectral centroid, zero crossing, and volume (cf., Section 2.1.2

for an overview of music low-level features), the approach is based on the perceptual features of the music such as dynamics, tempo, and pitch. Dynamics relate to the change in music loudness and softness, while tempo indicates the periodical flow of music in the time. The pitch indicates how many high or low tones exist and how their volume is perceived by a listener.

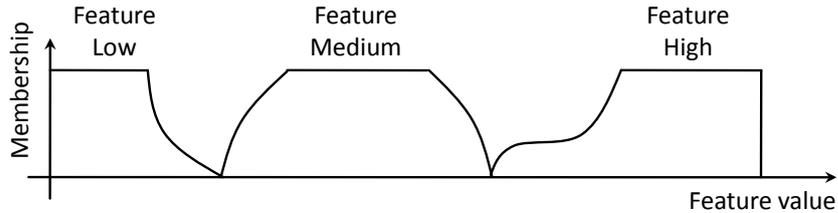


Figure 3.2: Fuzzy mapping of feature values to feature buckets

For each feature used, three levels of values are created, namely attribute low, attribute medium, and attribute high, using fuzzy linguistic variables, as illustrated in Figure 3.2. As we can see, the value of the membership to the attribute low, medium, or high is defined based on the mapping from the actual value of the extracted feature. This mapping is defined by the membership function which is learned from the distribution of values for the corresponding feature. For each video frame, or a music excerpt, one value is calculated for each feature at all three quantization levels, resulting in a point in a multidimensional vector space.

The mapping from the feature vector space to a pivot vector space is done using a projection matrix. This allows for mapping of one or more dimensions of one vector space onto one or more dimensions of the other vector space. The mapping used by this approach is shown in Table 3.1. Once the features are projected onto the pivot space the distance between them (e.g., Euclidean distance) is taken as a measure of similarity between media, used to recommend the closest songs to the video.

		Color			
Video feature:	Light	Energy	Hue	Brightness	Motion
Music feature:	Dynamics	Dynamics	Pitch	Dynamics	Tempo

Table 3.1: Mapping between video and music features

Similarly, Cristani et al. [CPD⁺10] consider as input a video of the scenery taken by the camera installed in the car. The authors propose a recommendation policy based on discovered correlations between audio and video features. The idea is that given a video, features are extracted and the required values for highly correlated audio features are identified. These values are then used to retrieve the music that contain such features. The details of the recommendation process are mentioned as left for future work as the focus of the current approach is only to investigate the correlations between the audio and video features.

To discover these correlations a set of professional documentary movies is used, each movie belonging to one of the following three categories: cities, nature, and mountains. Movies are processed by hand to remove scenes containing speech and human or animal acting.

The approach uses brightness, dynamics, and rhythm as audio features and saturation, brightness, and optical flow as video features. All of these features are based on the low-level features directly extracted from the medium. As in the previous approach, for each of the features their values are divided into three buckets with the labels: high, medium, and low. The correlation between all nine pairs of features are calculated and only the statistically significant ones ($p < 0.05$) would be used for the recommendation policy.

The identified correlations have been evaluated using music generated from midi files, showing that user decisions are in agreement with correlations extracted from the documentary movies.

The approach in [NSK03] also addresses the problem of having an appropriate music for a given video. However, the task that is considered is to synthesize customized music piece for a given video, rather than to select among existing ones. The idea is to extract low-level features of a video and apply them in music synthesis, resulting in a music that follows the *video rhythm*.

The approach uses two steps in generating the music. In the first step, mapping rules are used to map the values of video features to the values of music features, resulting in a generated music contour, i.e., a sequence of music notes. In the second step, a given example of the music piece is used to adjust the generated contour giving it form and structure. The matching of the given example music and the generated music contour is done minimizing the edit distance between the two sequences of notes.

3.1.3 Context-Aware Music Recommendation

Advances in mobile phone technology have drastically increased the number of sensors available in our everyday life. This resulted in a variety of systems that are context-aware, i.e., are dependent on the current context of the user. Likewise, a large interest has been developed in context-aware music recommendation which resulted in a variety of approaches [LL07, BKL⁺11, RM06, CZW⁺07, KFTRC12]. The approaches differ among each other based on their perception of the user's context. Considered context factors vary from weather information, over users' location and mood, to traffic conditions.

Beside demographic data of the users, such as age and gender, the approach in [LL07] considers the context of the current user by inspecting the current date, location of the user, season, month, weekday, and the weather. Contextual information is utilized for music recommendation using a training dataset and a case-based reasoning procedure. Case-based reasoning methodology assumes that similar problems will have similar solutions. Based on this, solution to new problems are found by analyzing old problems and their solutions. The recommendation process starts by retrieving users from the training dataset who

listened to the music in context similar to the query context. Further on, these users are filtered to include only the users similar to the user for whom the recommendation is done. In the last step, the listening history of the retrieved users is used to make a final recommendation. A cross-fold evaluation showed that the usage of environmental data increases the precision of the system compared to the case when only listening history is used.

Music recommendation for a specific context of a user while driving a car has been addressed in [BKL⁺11]. Characteristic contextual factors for this scenario are driving style, road type, and traffic conditions. Beside these additional factors like weather are also used. The list of all used contextual factors and their conditions is shown in Table 3.2.

Contextual factor	Contextual condition
driving style	relaxed driving, sport driving
road type	city, highway, serpentine
landscape	coast line, country side, mountains/hills, urban
sleepiness	awake, sleepy
traffic conditions	free road, many cars, traffic jam
mood	active, happy, lazy, sad
weather	cloudy, snowing, sunny, rainy
natural phenomena	day time, morning, night, afternoon

Table 3.2: Contextual factors used for in car music recommendation

The approach is based on the techniques of matrix factorization, extending them to context-aware scenarios. The score for each song, given a user and a context, is calculated as the product between the user and song latent vectors adjusted for the score calculated based on the genre information of the song and the dependence between the genre and the context. Latent vectors and the dependencies between contexts and genres are learned using stochastic gradient descent [Ber99] minimizing a squared error to a given training set together with a regularization term for achieving better generalization. Results of the evaluation show that context information improve the precision compared to the personalized recommendation without contextual information.

Similar to other context-aware approaches, Lifetrak [RM06] uses the information about the users' location, time of usage, users' movement speed, and urban environment while recommending music to the users. For the urban environment weather, traffic, and sound loudness factors are used. The approach is based on the one-time user effort to tag all songs for a certain set of contextual situations using a fixed vocabulary of tags. A sample of tags from the vocabulary is shown in Table 3.3. Once the songs are tagged, sensor information are continuously translated into the context information corresponding to the used tags. Ranking of songs is done through a vector space model where dimensions are the tags in the vocabulary. Scores for each dimension (tag) for songs and current context are one if the tag is applicable otherwise they are zero. The score for the song is given as the vector product between the context values and

the song values. Feedback about selected songs is used to adjust the scores of the tags for that song and a given context.

MusicSense [CZW⁺07] has been proposed to recommend music based on the currently read Web documents. It is based on the proposed *Emotional Allocation Model* that characterizes both Web pages and the music in a common emotion space. Each page and music piece are represented as a mixture of a fixed set of emotions and relatedness between them is calculated using Kullback-Leibler [KL51] divergence. The closest songs for a given Web page are used for recommendation. Mapping songs and Web pages to emotions is done based on the text of the Web page and the text on the Web describing the song, obtained through search engines. A probabilistic model is used to detect the emotions in text with parameters learned from the co-occurrences of the words with the word describing an emotion. Hand-labeled data is used to evaluate both emotion allocation and music recommendation. Results show that there is a positive correlation between labeled emotions and allocated emotions and that the precision for music recommendation is stable around 45% with recall increasing to 70% for top-10 recommendations.

Context describing tags

inside	afternoon	friday	run	calm	rain	sunny
outside	morning	monday	driving	chaotic	haze	clear

Table 3.3: Tags used for context description

Recommending musicians that are suited for a place of interest (POI) has been addressed in [KFTRC12]. In this work, a structured knowledge base DBpedia [ABK⁺07] is used as a basis to create a mapping between the musicians and POIs. The recommendation process is done in three phases. In the first phase, domain experts identify class nodes in the knowledge base graph that are relevant for the task together with the relevant paths between these classes. An example are “City” and “Opera composer” as class nodes, and a *location path* as a path that specifies that musician is linked to the place where he was born, lived, or died. Identified classes and paths are used as an input in the second phase to create a full network of class instances and the paths between them by aggregating all subgraphs that link POIs and musicians. In the final stage, a graph algorithm based on the weight spreading strategy is proposed to find the musicians that are most relevant to a certain POI node. A user study was performed indicating high precision values of the approach.

3.2 Image Retrieval for Soundtracks

The soundtrack recommendation problem for images can also be inverted, which is then a problem of retrieving relevant images for a given soundtrack [SPH05, CZJ⁺07, XJL08, CWJC08]. This task considers images only as a support to enhance the presentation of a song. Although an effective and appealing slide

show is a common goal for both tasks, the means and the final result differ substantially.

In their work, Shamma et al. [SPH05] propose a retrieval of images from the Web to automatically generate a video for a given music piece. The retrieval of images is based on the lyrics and the meta-data of a song. First, the meta-data of a song, such as the title and the author, are used to retrieve its lyrics, which are then used to create queries to image search engines. This is done by removing the stop words (i.e., frequently occurring words like the, at, etc.) from the lyrics and using the remaining ones as keywords to formulate search queries.

The approach in [CZJ⁺07] builds also on the principle of retrieving images from the Web based on the lyrics and the song meta-data. However, the approach is based on a more sophisticated keyword identification and style matching between the music and the images. After removing stop words from lyrics, keywords are identified using heuristic rules such as identifying a location or a mention of a person, or identifying nouns and noun phrases. Once the images are retrieved using these keywords, they are further re-ranked based on face detection and scenery classification. The images that contain faces or a nature scenery are ranked higher than the rest of the images. A final post-filtering step is performed to ensure all the retrieved images are in one style and that this style matches the emotions expressed in music. This is achieved by color mapping: blue and cyan colors are mapped to sad emotions while red and orange colors are mapped to happy emotions. The emotion classification is done for music piece based on its content and the images with corresponding colors are selected in the final step.

Retrieving personal photos instead of photos retrieved from Web to generate a video for a given music piece has been proposed in [XJL08]. The retrieval process starts with the detection of the song's keyframes by comparing distances between frame pairs. Subsequently an image is retrieved for each of the keyframes using corresponding lyrics. After removing the stop words, lyrics are used to retrieve images from the Web which are then used to calculate the similarity to the images in the personal collection. Low-level features extracted from images are used to calculate the similarity between them. Images, which are most similar to the images retrieved from the Web, are used for video generation.

An emotion-based approach [CWJC08] has been proposed to retrieve images for a given music piece by detecting and matching emotions. Emotion detection is done based on the low-level features using trained classifiers. The music piece is split in parts up to five seconds long using beat detection. The emotion detection is done through classification for each of the parts. As detected emotion retrieves a large number of images additional constraints are used when selecting an image. The first constraint is based on the matching between the brightness and contrast of an image with the timber of music. While the second constraint requires visual coherence between the successive images.

A user study was performed for the emotion-based approach showing that results are preferred to the results of randomly chosen image. For all other approaches, the authors present example results achieved by their systems. How-

ever, none of them have been evaluated by users, measuring the user perceived quality.

3.3 Automated Slide Show Generation

Once both music and a set of images are selected for the slide show, the problem of creating an effective and appealing slide show video remains. This is not an easy task as it requires determining image sequence, overlays, and transitions which in addition need to be synchronized with the music. Multiple approaches [CCK⁺06, LS07, CXG10] have been proposed addressing this task.

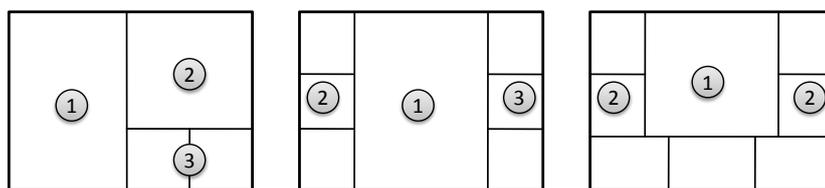


Figure 3.3: Slide layouts for multiple images

The tiling slide show approach [CCK⁺06] is a new form of slide show presentations where multiple images are displayed at the same time, coordinated with the pace of music. Photos with similar features are clustered together and displayed in form of tiles such that one larger image represents the topic of the slide and multiple smaller images support the topic, as illustrated in Figure 3.3. Determining the layout of the images is modeled as a constrained optimization problem taking into account an assessed importance of each photo. The importance of the photo is assessed based on user-attention models that take into account detected faces in the image and the contrast between objects and the background of the image. Detected beats in the incidental music are used as a time points at which new images are introduced or the transitions between slides are made. A performed user study shows that this form of slide show presentations provides a significantly higher level of satisfaction to the users than traditional *one-after-another* image slide show.

Besides the soundtrack recommendation task, the approach by Li and Shan [LS07] also addresses the problem of slide show generation. Similar to the tiling slide show approach, images are first clustered based on their content, more precisely based on the emotions detected from the extracted low-level features. A linear arrangement between images in one cluster and between clusters is determined using a modified traveling salesman algorithm to minimize the overall distance between consecutive displayed images. Each image is presented using a motion pattern such as panning or zooming to highlight detected regions of interest. The presented approach is evaluated through a user study, showing improvements in user satisfaction over system without accompanying music.

The approach in [CXG10] supports two modes for automatic slide show creation: story-telling and person-highlighting mode. In the story-telling mode,

first, images are clustered based on the distance in time, colors, and detected faces. Images from one cluster are combined in one slide using dynamic tiling and seamless blending on the edges. Transitions between frames are created simulating camera motions of zooming and panning. In a person-highlighting mode face detection is utilized to create clusters of images. The transition between images in this case is based on the regions of interest of an image, such as detected faces. Pixels in regions of interest of one image are distorted using one of the predefined distortion functions and undistorted to the region of interest of the other image, forming a transition between these images. A user study was conducted showing that users prefer results of this approach to the results achieved by the tiling slide show approach.

3.4 Collecting Relevance Assessments through Pairwise Comparisons

We have built a reusable benchmark to evaluate the effectiveness of soundtrack recommendation systems for images. To collect user assessments for the benchmark we use the notion of pairwise comparisons, as described in Chapter 5. Pairwise comparisons refer to the process of comparing pairs of objects to determine the preference towards one of them. They were first mentioned and analyzed in the field of psychology by Fechner [Fec60] and made popular later by Thurstone [Thu27]. Thurstone was mainly concerned how pairwise comparisons can be used to determining the scale of perceived stimuli. He formulated a general model to obtain scale measurements from a set of pairwise comparisons and referred to it as the *law of comparative judgment*.

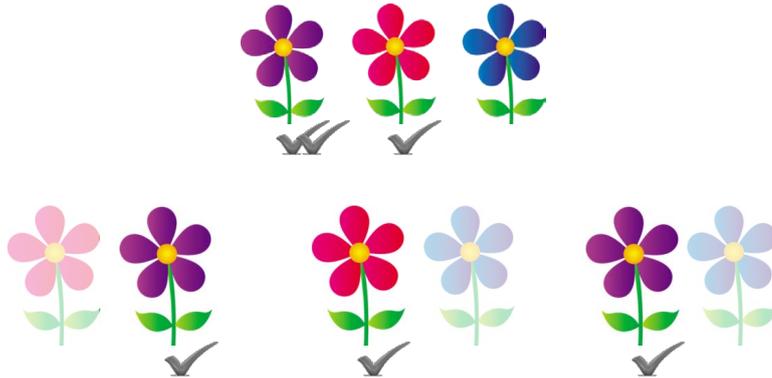


Figure 3.4: Pairwise comparisons to determine the final ranking

Moreover, a general problem of reconstructing the final ranking from a set of pairwise comparisons has received a substantial amount of research [Mik03, CP06, Jan08]. The approaches vary substantially from fuzzy numbers with linear programming optimization [Mik03] to learning approaches with logistic

regression models or SVM classification [CP06]. Figure 3.4 illustrates the process of creating a final ranking of flowers based on a set of pairwise comparisons.

For information retrieval tasks, Thomas and Hawking [TH06] use pairwise comparisons in order to compare systems in real settings, where interactive retrieval is used in specific context over ever-changing heterogeneous data. The comparison is achieved by presenting the results of two system side-by-side during the query session and asking the users to indicate the preferred result set. The evaluation of the approach showed that click-through data highly correlates with perceived preference judgments. The method of displaying results in side-by-side manner was also used by Sanderson et al. [SPCK10] to obtain the correlation between user preference for text retrieval results and the effectiveness measures computed from a test collection.

Using preference-based test collections is introduced by Rorvig [Ror90] and later developed for text retrieval by Carterette et al. [CBCD08]. In this work, the authors show that preference judgments are faster to collect and provide higher levels of agreement, compared to absolute relevance judgments. Preference-based effectiveness measures are proposed by Carterette and Bennett in [CB08], showing that they are stable and adhere to the measures based on absolute relevance judgments. Preference judgments between blocks of results are used by Arguello et al. [ADCC11] to evaluate aggregated search results. The small number of such blocks enabled the collection of preferences between all pairs of blocks. A suitable effectiveness measure in this case is the distance between the ranking produced by the system and the reference ranking created based on the all-pair preferences. In our setting, the huge number of possible song pairs prohibits an exhaustive evaluation, in which case the quality measure is more appropriate based directly on pairwise comparisons rather than using the reference ranking.

For music similarity, Typke et al. [TdHdN⁺05] conclude that coarse levels of relevance measure, usually used in text retrieval, are *not applicable*. Instead, they use a large number of relevance levels created from partially ordered lists. The ground truth in this case is given as ranked list of document groups, such that documents in one group have the same relevance. The work by Urbano et al. [UMML10] addresses some limitations of this approach by proposing different measures of similarity between groups of retrieved documents. Measuring retrieval effectiveness with these large number of levels is achieved using the Average Dynamic Recall [TVW06] measure.

Due to its low price and high scalability, crowd sourcing is a popular technique to obtain relevance assessments for information retrieval tasks [ABY11, AST10, SPCK10, KMFC09]. The work by Alonso and Baeza-Yates [ABY11] addresses the design and implementation of assessments tasks in a crowd-sourcing setting, indicating that workers perform as good as experts at TREC [TRE] tasks. Similar results have also been achieved by Alonso et al. [AST10] in the context of XML retrieval. Snow et al. [SOJN08] show that Mechanical Turk workers were successful in annotating data for various natural language processing tasks, even correcting the gold standard data in specific occasions.

3.5 Processing Top-K Queries with Constraints

In our approach to soundtrack recommendation, we are concerned with retrieving K songs that have the largest similarity score to a specific movie music, constrained to a set of songs that user possesses on a smartphone. Retrieving K top ranked objects constitutes a set of queries commonly known as top- K queries. There is a large body of existing work in the area of processing top- K queries, ranging from database systems [LCIS05, IBS08, BCG02], over distributed systems [CW04, MTW05], to information retrieval [FLN03, TWS04, DS11, AdKM01]. Among the most prominent approaches are the so called *threshold algorithms* [Fag99, GBK00, FLN03], used to answer top- K queries when the score of an item is aggregated from multiple different sources. These approaches scan the score-sorted lists of items and, at the same time, maintain a score threshold used for early termination. Accessing sorted lists is done in a sequential manner, usually complemented with random accesses based on item ids.

Although top- K queries provide focus on the top of the ranking, alone they do not provide enough flexibility to constrain rankings to a specific subset of items. For this reason, selection constraints are used in addition to ranking functions. The work in [XHCL06, XH08], supports selection constraints on multiple categorical attributes (e.g., producer = 'Ford' and production_year=2011) for top- K queries. The mentioned approaches extend the well known principle of *data cubes for data warehousing* [CD97] to incorporate ranking functionality. To achieve this, the P-cube approach [XH08] proposes the partitioning in the space of ranking dimensions and maintaining a signature for each of the partitions indicating its content. At query time, partitions are accessed in order of how promising they are for the ranking while using signatures to prune away the ones that do not satisfy selection constraints.

Retrieving the top- K documents that contain a certain phrase is another form of top- K queries with constraints, where constraints are phrases which must be contained in the retrieved documents. These queries can be transformed into a top- K queries with range selection using data structures such as suffix trees. Data structures and algorithms for processing top- K queries with range selection, also known as top- K *color queries*, have been proposed in [Mut02, KN11]. Efficient processing of these queries is in general achieved by supplementing the suffix tree structure to contain information about the number of suffix occurrences in a document.

In traditional information retrieval problems, the part of the data to look at is determined by query terms, not upfront by a set of documents concerning their characteristics. In this case, the formulated query is a top- K query with score aggregation. A notable exception is the work by Singh et al. [SSL07] on efficient enterprise search, where results are restricted based on the file access rights. Knowing all access rights upfront with a clear grouping in a small number of disjunctive groups enables the creation of so called access control barrels—sets of files accessible by the same group of users. At query time only the barrels

with appropriate access rights are queries. This way, the trade off between index blow up (index materialized for each user) and query processing time (querying over non accessible files) is achieved. Another exception is the work by Bast and Weber [BW06, BW07] on auto-completion search where query is executed while user is typing it in the search box. In this case, the selection constraint is given as a set of documents that correspond to the last entered query prefix. However, the work does not consider ranking documents, hence, early termination techniques can not be applied.

Ranking join results based on aggregated scores obtained from multiple tables (i.e., top-K join processing) and embedding such ranking concepts in a query optimizer have been addressed in [IAE04, WBEM⁺10]. These approaches again utilize the concept of accessing sorted lists of items while maintaining a threshold that corresponds to join criterion to enable early termination. In general, the problem of joining multiple tables with corresponding indices requires identification of the *break-even point* between index lookups (e.g., a B+ tree) vs. a full table scan—for which standard textbook solutions based on cost models exist [GMUW08]. However, this “index join” has not been considered in top-K setting yet. For general top-K join queries [IAE04, WBEM⁺10], our studies in Chapter 6 can be of use for the direct access to the base tables in a *rank-aware* query plan.

To achieve lower latency for top-K queries with set-defined selections, we create partitioned index based on the query logs (i.e., historic workloads). Using query logs with the aim of tuning a system’s performance is encountered quite frequently [NPS11, IKM07, WMB⁺07, CZJM10]. Applications where the workload is used to improve performance vary from index defragmentation [NPS11], over cache replacement [WMB⁺07] to range queries [IKM07]. Query logs have been used in [CZJM10] with the aim to reduce the number of distributed partitions by allocating tuples that are frequently used together to the same partition. In our index organization, the graph based approach proposed in [CZJM10] is used for data partitioning.

Chapter 4

Picasso—A Soundtrack Recommendation Framework

This chapter gives a detailed description of Picasso—our approach to soundtrack recommendation for images. The content of this chapter is based on our previous publication [SM11a] where Picasso was initially introduced. The naming of the system was inspired by the famous painter, Pablo Picasso, and his quote “*To draw you must close your eyes and sing.*” We tamper a bit with Pablo Picasso’s quote trying to figure out if the inverse holds too, namely, if we are able to find music (singing) by looking at pictures (drawings), that is “*To sing you must close your eyes and draw.*” At the same time the name stands as an acronym for *PI*cture *CA*tegorization for *S*uggesting *SO*undtracks.

The basic hypothesis behind Picasso approach is that the connection between the world of music and the world of images can be made through the knowledge of experienced movie directors, extracted directly from the movies themselves. Picasso extracts this knowledge in a fully automated way, utilizing it to create a match between query images and the songs in the collection and provide a final soundtrack recommendation.

Sketch of the approach: Given a set of movies, we take samples of them at equidistant time points. For a given sample point, we investigate if the corresponding sound surrounding the sample by a couple of seconds resembles a piece of music. If so, we take the screenshot and the music piece and consider it as training data. We do this for very many samples, for dozens of movies. The set of considered movies should be large enough and also diverse such that the training base is big enough and the resulting categorization captures all (or almost all) situations appearing in pictures people take.

At query time, two levels of similarity search are used to retrieve songs for the recommendation. In the first level, screenshots similar to the query image are retrieved. In the second level, songs similar to the music played at corresponding

places in movies are retrieved. This process is performed for each query image individually and the results are combined to recommend songs for a group of images.

With Picasso we make the following contributions:

- (i) We present an algorithm for soundtrack recommendation for an arbitrary wide range of user generated images as input. Besides complete image support, the presented approach also supports arbitrary variants of music, without limitations to songs with lyrics or specific music format such as midi files.
- (ii) We show how a training dataset can be extracted out of popular common movies in a fully automated way, capturing the knowledge of experienced movie directors.
- (iii) The soundtrack recommendation process for one submitted image is described, together with the approach of combining individual recommendations when multiple images are submitted. An additional feature of image grouping based on soundtrack recommendation is proposed.
- (iv) We report on the results of a user study evaluating the feasibility of our approach. The detailed and extensive evaluation concerning the effectiveness of Picasso and the baseline approach is given in the Chapter 5.

In the following, we first present our framework with the detailed description of training data extraction and a brief recap of basic techniques and similarity measures in Section 4.1. Section 4.2 presents the core concepts of Picasso describing the recommendation process and soundtrack-based image grouping in details. User study results are shown in Section 4.3.

4.1 Framework

Picasso is based on the expertise of movie directors to select appropriate songs for specific scenes in their movies. This knowledge is made public, naturally, when movies are presented in cinemas, TV, or DVD, generating a huge knowledge base that perfectly fits our goal. We extract samples of screenshot/soundtrack pairs from a certain amount of these movies and form a training dataset creating a basis for our framework.

As a final training dataset we aim at having, for each of the screenshots taken, a list of songs (for instance, the user's private mp3 collection), which are available for recommendation, in decreasing order of likelihood to be recommended for a specific screenshot. The structure of the training dataset is shown in Figure 4.1. Building this dataset is done in the following steps:

- (i) the soundtrack of the movie is extracted
- (ii) music/speech classification is done on the soundtrack
- (ii) speech parts are discarded

- (iv) screenshots, during the musical part, are taken
- (v) parts of the same scene are detected
- (vi) the soundtrack is split according to the scenes
- (vii) for each soundtrack part the distance to all songs is calculated
- (viii) the song lists are sorted in increasing order of distance, resulting in our training set

Soundtrack and image extraction, from the movies, is easily achievable using one of many available tools and will not be covered here in depth. However, music/speech classification is not that straightforward. For this task, we use a Naive Bayes [Ris01] classifier, trained using the labeled dataset available at [MAR]. This training dataset contains 64 speech samples with spoken text coming from different languages, and 64 music samples, covering variety of music genres. Low-level music features described in Section 2.1.2 are used for classifier training and for the later classification task. The Marsyas [Tza09] tool is used for both the features extraction and the classification. The output of the classifier for a given soundtrack is a label—“*music*” or “*speech*”—for each second of the soundtrack together with a confidence value in the performed classification. To be sure to use only the music parts of the soundtrack, all parts that are classified as a music with the confidence value less than 95% and all speech parts are discarded. Music parts of the soundtrack with length shorter than five seconds are also discarded, as they do not contain enough information to make the similarity measurement with songs meaningful.

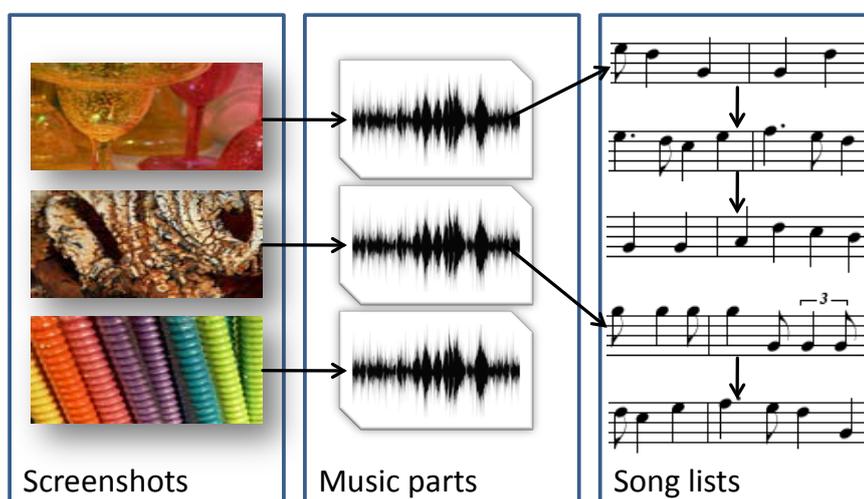


Figure 4.1: Training dataset structure

Screenshots are taken from the movies at each second of the remaining music parts. To obtain a more logical grouping, we divide these music parts further into scenes. The scene detection is implemented by splitting a sequence of screenshots on positions where the image-to-image distance, measured as described in

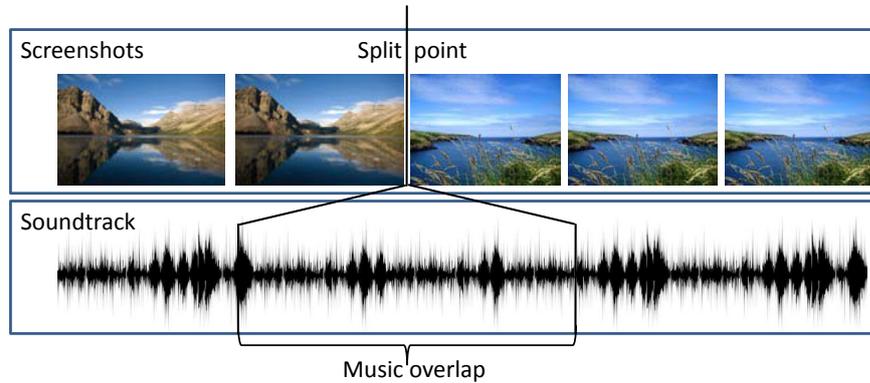


Figure 4.2: Splitting of a long scene

Section 4.1.1, is larger than a given threshold. The sequence of the screenshots from one split to the second one is considered a scene. This approach may not lead to exact movie scenes but, as we are only interested in image-to-image similarities for the further processing, it provides a reasonable grouping of similar images. To eliminate abrupt scene changes, for the same reason we have eliminated short music parts, i.e., we filter out and discard detected scenes whose length is shorter than 5 seconds.

Additionally, very long music themes can cause problems as a too large amount of information can render them too specific and, hence, would not match to any parts of the available songs. This is why scenes that are longer than 10 seconds are split in multiple parts. Splitting these long scenes in multiple parts also improves the locality connection between the soundtrack part and the taken screenshot. To split the long scenes into smaller parts and to make music parts of approximately the same length, we split the long scenes such that screenshots sequences are disjunctive, but with the possibility of having overlap between the music parts, as illustrated in Figure 4.2. The splitting algorithm starts greedily and chops away the first 8 seconds of the scene as a separate part, repeating this until the last piece is left. The last piece is either 8 seconds long or less than that. In case the last piece is less than 8 seconds the music part is extended to include previous missing seconds to make the total length of 8 seconds, creating an overlap in the soundtrack parts but no overlap between images. Screenshots together with links to their corresponding music parts are then saved for further processing.

After the music parts are cut, first based on the music/speech classification and then based on scene detection, the distances between each of the music parts of the soundtrack and all of the given songs are calculated. The distance measure between the music parts and the songs is described in Section 4.1.2. Songs are then ordered in increasing order of their distance to each of the music parts.

These ordered lists of songs together with the links from the taken screenshots represent our training dataset, which is later used for soundtrack recom-

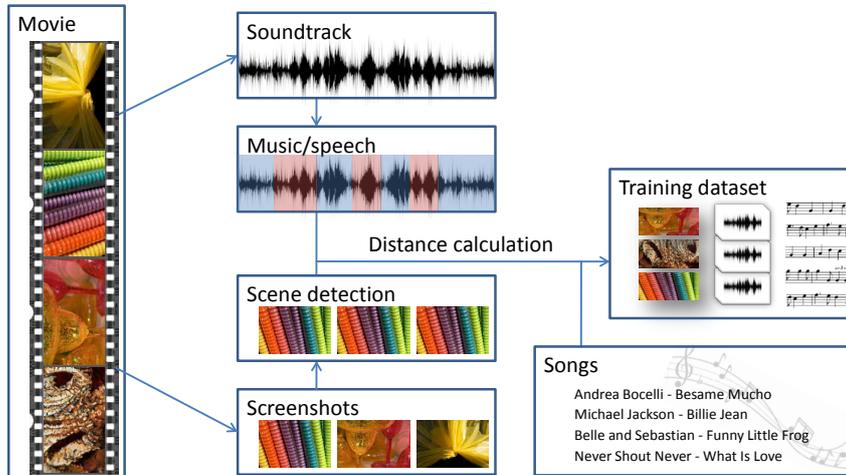


Figure 4.3: Training dataset extraction process

mentation, c.f., Figure 4.1 for an illustration. Each screenshot taken from the movie has a link to the part of the accompanying soundtrack which has a list of songs ordered by their distance to the soundtrack. The complete process of building the training dataset, from soundtrack extraction to song distance calculation, is illustrated in Figure 4.3.

4.1.1 Image Similarity Measure

For the image-to-image similarity measure we use low-level features proposed by the MPEG-7 standard [CSP01], namely: scalable color, color structure, color layout, and edge histogram. All the features are extracted directly from the image and represent color and edge distributions in the image. In the following, we give a brief description of the used features, for more details refer to Section 2.1.1.

Scalable color describes all the colors found in the image by aggregating them in a color histogram. Each bin of the histogram represent one level of the color quantization. To lower the memory space needed for the histogram, the Haar transform [Haa10, SS99] is used for encoding. The standard defines three possible values for the number of Haar coefficients used: 128, 64, and 32.

The **color structure** descriptor is represented also by a histogram, where the value of the bin is a counter for the structuring elements, of 8×8 size, in which the respective color appears. Counting these values is done while sliding the structural element over the whole image. The number of color quantization levels is a parameter for this descriptor and can be 184, 120, 64, and 32 as defined by the standard.

Color layout summarizes the distribution of colors in a given image, first by dividing the image into 64 (8×8) equal sized parts and, and then calculating the average color for each of the parts. The discrete cosine transform (DCT)

is performed on these 64 parts and only low frequency coefficients are used for each component. The number of coefficients used is defined as a parameter as well.

The **edge histogram** descriptor is used for the spatial representation of edges in the image. The image is divided into 16 (4×4) equal sized parts and the local edge histogram for each part is calculated. Each histogram has 5 bins representing 4 orientation directions and one used for non orientation-specific edges. Concatenation of the local histograms with the histogram levels quantization produces the final descriptor.

In this thesis, we use common configurations of the MPEG-7 image descriptor coefficients, with 64 Haar coefficients for scalable color descriptor, 64 color quantization levels for color structure descriptor and 6 coefficients for the Y component, 3 for the Cr component and 3 for the Cb component of the color layout descriptor.

After the distance calculation, for each of the descriptors, we need to combine them in one distance measure for further comparison. We do this by first calculating the *standard score* (*z-score*) for each of the descriptors and then summing up all of the standard scores into a single score. The standard score is calculated by Formula 4.1, where μ is the expected value (mean) of the distances for the given descriptor and σ is the standard deviation of the distances for the same descriptor.

$$z = \frac{x - \mu}{\sigma} \quad (4.1)$$

The mean μ and the standard deviation σ for the descriptors are approximated from the training dataset. The distances between each pair of the images, for each of the descriptors, are calculated, and then the mean and the standard deviation for the descriptor are approximated by the estimated mean and standard deviation of this set of calculated distances.

4.1.2 Music Similarity Measure

For the music similarity measures, the following low level music descriptors are used: *MFCC*, *Chroma*, *spectral centroid*, *spectral rolloff*, *spectral flux*, and *time domain zero crossing*, c.f., [TC02] for an overview.

The **Mel Frequency Cepstral Coefficients (MFCC)** descriptor is based on the mel scale, a model of the human auditory system. First, the spectral representation of a musical signal is transformed by the model and then the coefficients of the discrete cosine transform (DCT) are used as features.

As western music is based on a musical scales containing a subset of twelve basic musical semitones, the **Chroma descriptor** [EP07] represents the intensity of each of the twelve semitones in the analyzed music part. It is a very valuable descriptor, used for musical similarity measure, as it highly emphasizes melodic and harmonical characteristics of the analyzed musical piece.

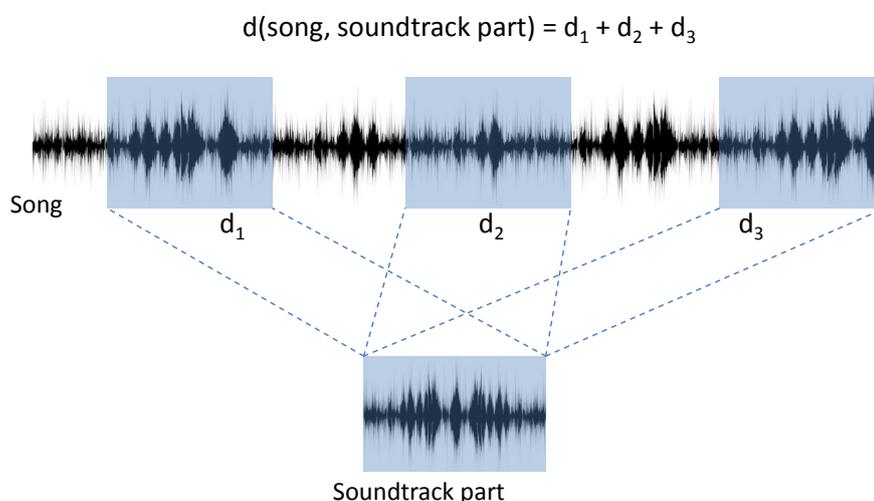


Figure 4.4: Song to soundtrack part distance

The **spectral centroid** is defined as a center of gravity of a musical signal’s spectral representation, while the **spectral rolloff** represents the frequency under which 85% of the signal’s magnitude is located. The local spectral change, in the time domain, is represented by the **spectral flux** which is calculated as a squared difference between two successive spectral distributions. The **time domain zero crossing**, describing the noisiness of the signal, is defined as a number of times a musical signal crosses between positive and negative values in the time domain. For more details on the used music low-level features see Section 2.1.2.

To calculate the similarity between two songs first the feature vectors of each of the descriptors are extracted for every 23.25 milliseconds of the song, resulting in 43 feature vectors per second (the default configuration for the Marsyas tool). Then, the pairwise similarity between the feature vectors is calculated and combined. The combination of distances of different descriptors into one distance measurement is done also using the sum of standard scores for each descriptor, as described in Section 4.1.1. The mean and the standard deviation for the standard scores are approximated using the mean and the standard deviation of the pairwise distances between the songs in the training dataset.

Calculating only pairwise distances between feature vectors is not enough as music also has a time dimension. One option to handle the time dimension is to calculate the sum of successive pairwise distances. As this option results in a rigid comparison in time, we use a more flexible option called Dynamic Time Warping [SC78]. **Dynamic Time Warping (DTW)** enables sequence matching with the variations in speed, making it possible to compare songs that are similar in content but with different speed measures. As speed change can be beneficial for flexibility it also has to be limited such that the similarity

between songs with different speed measures is less than between the songs with the same speed measure. This is also achievable using DTW by specifying the slope constrained condition and by specifying *Weighting Coefficient* for the case of speed change (time warp) and the case of no speed change (no time warp). In this work, we use a symmetric variant of DTW with slope constrained condition set to 1, and with the value of $\sqrt{2}$ for the *Weighting Coefficient* in case of speed change and value 1 for the case of no speed change.

Beside giving us a distance DTW also gives us a part of the song that matches the best to the other song. Because soundtrack parts are a lot shorter than the songs in general, we use DTW to find *three* positions in the song that are most similar to the soundtrack part. The sum of distances between the soundtrack part and these three positions in the song is used as the resulting distance between the song and the soundtrack part, as illustrated in Figure 4.4.

4.2 Approach

Once the training dataset is extracted we can use it for soundtrack recommendation. We consider two types of recommendations, single image recommendation and multiple images recommendation.

- In the single image recommendation, one image is submitted as a query and a list of songs is returned, in decreasing order of their suitability to the query image.
- In case of multiple images, we consider the images as the input for the slide show generation and try to recommend a grouping of the images together with the songs recommended for each of the groups. We base the grouping of the images on the similarity between the lists of recommended songs for each of the image. Obviously, the grouping step may be left out if the grouping is specified upfront.

The key idea of finding the most suitable song for a query image is to employ two phases of similarity search, first, in the image domain, and second, between music pieces. As we saw in the Section 2.2, the similarity search translates into a K-Nearest Neighbors (KNN) search, with documents represented as points in multidimensional space. Selecting the K-Nearest Neighbors (KNN) means that, for a given feature vector, the K nearest feature vectors are selected, that is, the K most similar images/songs are retrieved.

Phase 1: When the query image is submitted, its distance to each of the images in the training dataset is calculated. The images in the training dataset are then ordered in increasing order of distances to the query image and only the top- K images (i.e., the K-Nearest Neighbors) are used for the further recommendation process. This process is known as linear scan, as the distance to every image in the training dataset is calculated. To optimize for efficiency we could use approximate techniques for KNN search, described in Section 2.2,

and we will see in Chapter 7 how these techniques can further be enhanced to increase the efficiency and also distributed for large scale similarity search.

Phase 2: After the top- K images are found, the list of the songs together with the score for each image is retrieved. First, the name of the soundtrack sample for each image is retrieved and then the list of songs, ordered in increasing order of distance to the soundtrack sample, is retrieved for each soundtrack sample. As a result, each image, retrieved from the training set, has a list of songs in decreasing score order to that image. We aggregate these lists to create one final list that serves as a recommendation for a given image. Aggregating song lists could be avoided if only the closest screenshot, i.e., top-1 screenshot, is used with its song lists as a final recommendation. However, this would introduce a negative impact of outliers to the recommendation process. Retrieving multiple screenshots and aggregating their song lists can be considered as a smoothing technique enabling multiple movies to have an impact of recommendation to one query image.

To aggregate the song lists, we first assign a score to each of the top- K retrieved images based on the similarity. The score is normalized to the $[0.1, 1]$ interval by Formula 4.2, where d is the distance of the image to the query image, m is the distance to the most similar image (i.e., the smallest distance), and M is the distance to the least similar image (i.e., the largest distance). Each of the image scores is then further assigned to the top- K songs in the list of corresponding images. An average score for each song is then calculated and used to reorder the songs. The ordered list of songs, by their new calculated score, is returned as the final list of recommendations.

$$z = 1 - 0.9 * \frac{d - m}{M - m} \quad (4.2)$$

For each of the K images we get K songs, leading to a final number of songs between K (in case of heavy redundancy) and K^2 (in case of no redundancy), which, after duplicate elimination and a final ranking, assembles the final result list. In all of our setups we used $K = 10$.

While a fully automated process requires only one suggestion, in a semi-automated approach we return the top- K results to let users do the final assignment. However, rarely would users submit only one image to get a recommendation. For that reason we have to address the case when multiple images are submitted as a query.

4.2.1 The Case of Multiple Images

Given multiple images as input, we group them using a clustering algorithm and recommend a soundtrack for each of the groups. To achieve this, we first perform the recommendation for each of the images. This results in the list of recommended soundtracks for each input image. The similarity between these recommendation lists is then used as the grouping criteria. Computing

similarity between ranked recommendation lists is done using a slightly modified Spearman’s footrule distance measure [FKM⁺06], as follows.

First, we find the length of the shortest list as the length of the list depends on the number of unique songs in the top- K songs for each of the top- K similar images from the dataset. The lists are cut at the length of the shortest list to make the pairwise similarity measure comparable. The similarity between two lists l_1 and l_2 is calculated by Formula 4.3, where L is the length of the lists, and pos_1 and pos_2 are positions where the song s is located in each of the lists. If a song is found in only one of the lists, score is left unchanged, as specified by the formula.

$$sim(l_1, l_2) = \sum_{s \in l_1 \wedge s \in l_2} L - |pos_1 - pos_2| \quad (4.3)$$

As we can see, this kind of similarity measure takes into account not only the overlap between the lists but also an ordering of songs in the lists, imposing that the lists with the same ordering will be more similar than the ones with the same overlap but different ordering. As the Spearman’s footrule distance is defined over full (i.e., complete) rankings, we adjust it to be used also for our partial rankings (top- K lists) (cf., [FKM⁺06]) by ignoring the songs not present in one of the lists. Also, the list length L is added to the formula with subtracted $L1$ distance, turning the measure into a similarity measure.

In case of the large music collection it can happen that recommendation lists have no overlap in terms of songs they contain, which can easily be solved by increasing the value of K , resulting in more songs being recommended.

The similarity matrix, containing all pairwise similarities between the lists, is calculated and used for a *bottom-up hierarchical clustering* which produces a grouping of images as a result. We use a hierarchical clustering with the explicit number of clusters and the maximum cluster size specified. The process of clustering begins with creating a cluster for each image instance (i.e., list) and then iteratively merging the clusters until the required number of clusters is achieved. In each step, all pairs of clusters, except those whose cumulative size becomes larger than the specified cluster size limit, are considered for merging. Two clusters whose pairwise similarity is smallest, compared to other cluster pairs, are merged. The pairwise similarity between clusters is defined as the average pairwise similarity between their instances as shown in Formula 4.4, where c_1 and c_2 are clusters, i_1 and i_2 are their instances respectively. Cluster sizes, represented by $|c_1|$ and $|c_2|$, are defined as the number of instances in each of the clusters.

$$sim(c_1, c_2) = \frac{1}{|c_1| \cdot |c_2|} \sum_{i_1 \in c_1 \wedge i_2 \in c_2} sim(i_1, i_2) \quad (4.4)$$

In our setup, we use the floor of the number of input images divided by 5 for the number of clusters, aiming at the clusters with average size of 5 images,

					Average Position		Least Misery
1	Besame Mucho	Billie Jean	Billie Jean	1.3	Billie Jean	2	Billie Jean
2	Billie Jean	What Is Love	Besame Mucho	2.3	Besame Mucho	3	What Is Love
3	What Is Love	Feel	What Is Love	2.6	What Is Love	4	Besame Mucho
4	Feel	Besame Mucho	Feel	3.6	Feel	4	Feel

Figure 4.5: Group recommendation strategies used for soundtrack recommendation task

and we use 7 as the specified cluster size maximum.

After the grouping of the images is finished, a soundtrack needs to be recommended for each of the groups. The problem again is to aggregate the recommendation lists of the individual image recommendations into a final recommendation. To achieve this, we cast the problem into a group recommendation problem [AYRC⁺09]. Group recommendation is nicely illustrated with an example of a movie recommendation for a group of people where each person has her own preferences. In our case, the images are persons and songs are movies, where the preferences are specified in terms of individual recommendation lists.

We use two common strategies for group recommendation, namely *average position* and *least misery*. First, we find all the songs that are present in all of the n lists and calculate a score based on the group recommendation strategy that is chosen. We form a final ranking and add these songs, ranked by the calculated score. Then the songs that are found in $n - 1$ lists are ranked and appended to the final ranking. The step continues for the songs found in $n - 2$ lists, $n - 3$ lists, and so on. The aggregation step with both *average position* and *least misery* strategies is illustrated in Figure 4.5.

- For the **average position** strategy, the final score for a song is calculated as the average of all the positions in the recommendation lists, where the positions are given as the song rank. A small value of the score means that the song is high in rankings of recommendation lists, in average, making a good fit for query images. Songs with large score are ranked lower in the final ranking as they are in low rank positions in the initial recommendation lists, in average. A ranking based on the average position can sometimes be misleading for a group recommendation—some group members can be highly disappointed by the recommendation but the average satisfaction is still high because of the high satisfaction of the other group members.
- The **least misery** strategy is used in group recommendation with the intention to circumvent this problem. The basic idea behind the least misery strategy is that recommendation should minimize the misery of any of the group participants. With this strategy, the group recommendation is valued by the group member who likes the recommendation the least.

For our case, that means that we need to find the maximum position (lowest rank) for each of the songs found in the previous step and use this position as a score for recommendation. Like for the average ranking approach, songs are reordered on their new score in increasing order, and the song with the smallest score are recommended first.

4.3 Feasibility Study

In the course of developing Picasso we conducted a feasibility study to see the potential of the approach. In the following, we describe the study setup and the results. The detailed evaluation of the system’s effectiveness is done through the benchmark we have designed, cf., Chapter 5, while the efficiency issues addressed in Chapters 6 and 7, contain the evaluation of the these efficiency aspects.

To perform the feasibility study, we extracted the training dataset out of 28 movies of different genres through the procedure described in Section 4.1.

We use 275 songs as potential soundtrack recommendations. We obtained this dataset by downloading songs from the music2ten site [MUS], a site which contains only music given away by its creators for free use, as stated on the site: “All the music selections at music2ten.com are MP3s that are given away free with the artists’ blessings.”. This songs dataset contains fairly unknown material, i.e., artists are not promoted by the big labels.

4.3.1 Study Setup

We conducted a user study for both cases of recommendation, the single image recommendation and the recommendation for the case of multiple images. We have asked users from our university environment to grade the suitability (aka., relevance, appropriateness) of a suggested soundtrack w.r.t. a given image or a given slide show, in case of multiple images. Users were provided with the following options: “fits very good”, “fits good”, “fits ok”, “does not fit” and “total miss”. We assign grades from 5 to 1 for each of the option for the further numerical analysis of the results. Grade 5 is assigned to “fits very good”, 4 to “fits good”, and so on, having grade 1 for “total miss”.

We include a random recommendation to the user study to help us interpreting the results. Additionally, to be able to see how much the ranking of the recommended soundtracks make sense, we include also the recommended soundtrack at rank 10 in the evaluation of the single-image case. That means, during the evaluation, users were asked to grade the first ranked, the tenth ranked, and a randomly recommended soundtrack for each query image. To get consistent results for the random recommendation, a song was randomly chosen for every query image and then used consistently throughout the whole study (i.e., for all users).

The study is performed using a tool, which we have developed, shown in Figure 4.6. This tool enables users to play three recommended songs for each

Query



Results

Please enter your name:

and please rate how these songs fit to the query image.

<input type="radio"/> Fits very good <input checked="" type="radio"/> Fits good <input type="radio"/> Fits ok <input type="radio"/> Doesn't fit (...Id-Sick.wav) <input type="radio"/> Total miss	<input type="radio"/> Fits very good <input type="radio"/> Fits good <input type="radio"/> Fits ok <input type="radio"/> Doesn't fit (...Burnout.wav) <input checked="" type="radio"/> Total miss	<input type="radio"/> Fits very good <input type="radio"/> Fits good <input checked="" type="radio"/> Fits ok <input type="radio"/> Doesn't fit (...ny-Pack.wav) <input type="radio"/> Total miss
---	---	---

Figure 4.6: Screenshot of our evaluation tool

query image, and to choose how well each song fits to the query. To avoid the ordering bias, the evaluation tool places the songs on the page in random order for a specific session.

The study for the single image recommendation is done using 12 pictures, shown in Appendix A.3. These pictures were selected to be very different in content, as we want to evaluate how well the recommendation performs in the general case, rather than in one or a few specific cases. We had to limit the number of query images to 12, as each user was asked to assess the suitability of 3 recommended soundtracks, for each of the query images, resulting in a tedious and time consuming task of listening to 36 different songs.

For the case of multiple images, we also add a random recommendation to the evaluation, together with the evaluation of the average position and the least misery approaches. We use 36 images taken indoors and outdoors, on various locations all around the world, for the slide show creation. The grouping of the images is done using hierarchical clustering, as described in Section 4.2.1. Three slide show videos are made with the recommended soundtracks: by average position approach, by least misery approach, and by the random selection of the songs as a soundtrack. The slide show videos are made with a light “dissolve” transitions between the images, inside the groups, and with the heavier “page fold” transition between the images from different groups. Each image is shown for five seconds, resulting in the total length of three minutes for each of the slide shows. The transition between the songs is done with the fade out effect on the end of each song and with the fade in effect on the beginning of each of

the songs. The users are asked to rate how well the soundtrack fits to the slide show, for each of the three cases. This evaluation is also done with our online tool, where the slide shows were played and graded by the users.

4.3.2 Study Results

A total of 13 users participated in the evaluation of the generated slide shows, i.e., the multiple images case. The aggregated results of the evaluation are shown in Table 4.1 where each option for the soundtrack’s suitability was assigned a grade, as already explained in the previous section. As we can see, the average grade for the average position approach is a lot higher than the random soundtrack recommendation. The least misery approach has an average grade a lot higher than random, but slightly lower than the average position approach.

	Avg. position	Least misery	Random
Avg. grade	3.69	3.31	2.38
Std. dev.	0.95	0.95	1.26

Table 4.1: Multiple images grades

Besides having just a good average grade for the recommendation, it is also very important that users actually agree on how well the soundtrack fits to the image (or the set of images). The agreement between users is very important as high average grades could still have users with high disappointment, what we want to avoid. The agreement on the suitability of the recommended soundtrack is represented in Table 4.1 by the standard deviation of the results. The lower the standard deviation the larger the level of agreement between users. We see from the standard deviation that users agree more for the soundtracks recommended by our two approaches compared to the random recommendation, as expected.

	First rec.	Tenth rec.	Random
Avg. grade	3.21	3.08	2.97
Std. dev.	1.21	1.20	1.32

Table 4.2: Single image grades

Table 4.2 summarizes the results of the single image evaluation. Again, 13 users participated in the evaluation, but not all of them have evaluated the soundtracks for all of the 12 images. Each image is evaluated 10.83 times in average. We can see that the average grade for the first recommended song is higher than the random recommended song and that the tenth recommended is graded in between the first recommended and the random, showing that our ranking is indeed in line with the users’ perceived ranking. As we can also observe, the difference in average score between the first recommended and the random one is a lot smaller than the difference between the average position approach and the random one in case of multiple images evaluation. This kind of

result is well expected as the slide show contains multiple images, accumulating the users perceived soundtrack relevance over all of the images.

For the case of single images the average agreement over all 12 tests, in terms of standard deviation, is better for the first recommended and tenth recommended song than for the random one. This shows that not only the first ranked and ten ranked recommended soundtrack have higher score but that users also agree more on these scores compared to randomly recommended song.

Runtime & Scalability

We measured a query processing time with this setup of Picasso. The average runtime for the 12 single image queries and the training dataset extracted from 28 movies is 0.629 seconds per query image, with a standard deviation of 0.728. Due to the small training dataset, most of the running time was spent on extracting the image features for the query images, as these were not precomputed. The recommendation for the multiple image case was done in 14.933 seconds for our case of 36 input images. Here, again, most of the runtime was spent on the features extraction of each of the 36 images. The efficiency aspects of the approach, once the training dataset and the song collection become large, are addressed in details in Chapters 6 and 7.

Thread(s)	4	3	2	1
Runtime (sec.)	86.13	99.19	132.09	213.84
Queries/sec.	9.75	8.46	6.35	3.92

Table 4.3: Scalability measurements

At query time the training dataset is read-only, creating even on a single machine with multiple cores no conflicts between memory accesses, rendering the approach “embarrassingly scalable”, a common term for these kinds of parallelization. To show the scalability of the approach on a single machine, we measure the running times and the throughput (i.e., images per second), varying the number of threads used, for 840 query images, shown in Table 4.3. The measurements were done on the single machine with single four core processor. We did not measure the case of parallelizing the tasks on several separate machine, as the scale-up in throughput is obvious.

Once the memory consumption hits the limit of available memory, we can partition the training dataset on a per-movie basis. Then, for each machine, we obtain recommendations based only on a subset of all movies. These “local” results can get, however, easily merged by their similarity score in a final aggregation step (i.e., a very simple list merging task). We did not investigate this further. In a real world deployment one would install the approach directly in the cloud to benefit from the provided elasticity in terms of required computing power.

4.3.3 Lessons Learned

The lessons learned from the study presented above can be put in the following statements. Naturally, in most cases these are not hard facts being universally true, rather reflecting the first insights obtained throughout the study.

- Publicly available movies contain expert knowledge on matching the soundtrack to the given images
- It is possible to extract that knowledge and use it for soundtrack recommendation
- Fairly simple methods for recommendation, in conjunction with the extracted training dataset, work quite well
- Users find the recommendation of the soundtracks, for single image or set of images, satisfactory
- Users agree on the ranking of the recommended soundtracks
- The approach can be applied in user-centric responsive environments with easy scaling in cases of high throughput need

Chapter 5

Effectiveness Benchmark

Evaluating the quality of multimedia IR systems introduces various challenges as interpreting abstract associations—such as similarity between images—is complex and can be done in numerous ways. Due to this complexity a multitude of evaluation campaigns have been established in the field of multimedia IR. Similar to the text retrieval evaluations considered in TREC [Voo07, TRE], the MIREX [MIR] evaluation campaign addresses the assessment of music information retrieval systems, while the evaluation of image and video retrieval systems is considered in venues such as TRECVID [TVI] and ImageCLEF [ICL] .

One of the largest contributions made by these venues is the standardization of the retrieval corpus, i.e., a document collection and a set of queries. In addition to this, the defined benchmarks contain human-created relevance judgments that assess the quality of results with respect to the information need specified in the query. In order to estimate standard retrieval measures such as precision and recall, it is essential for these assessments to be complete and reusable. If constructed properly, they enable a fair and unbiased comparison among systems—which increases the competition and the pace of the improvements in the field.

In this chapter we describe the process of creating a benchmark dataset that can be used to assess the quality of soundtrack recommendation systems for images. To address this problem we have defined a set of queries, i.e., image sets, a set of songs for music collection, and collected judgments on the fitness between the songs and the images. This chapter describes the created benchmark, the process of collecting user assessments, and benchmark statistics. In addition, the results of Picasso and an emotion-based baseline approach, over the proposed benchmark are given and discussed. The content of this chapter is largely based on our publication in [SM13].

Problem formulation: create a benchmark to evaluate the retrieval performance of soundtrack recommendation systems. The proposed benchmark $B = (Q, S, R)$ contains a set of queries $Q = \{q_1, q_2, \dots\}$, a set of songs $S = \{s_1, s_2, \dots\}$, and a set of human relevance judgments $R = \{r_1, r_2, \dots\}$, with each query q_i defined as a set of images $q_i = \{img_1, img_2, \dots\}$. The proposed benchmark fulfills

the following important requirements:

- it enables an unbiased comparison between different recommendation systems
- it is reusable, that is, once created it can be used to evaluate systems with no additional human intervention
- it provides high coverage in terms of “document” collection (songs) and evaluated queries (images)
- it contains judgments with high agreement between assessors
- it is publicly available: <https://sites.google.com/site/srbench/>

Intuitively, the task of soundtrack recommendation appears to be highly subjective as the taste in music largely varies. However, as we will see, the agreement level between the assessors is quite high, indicating that it makes sense to address the problem for the general case, i.e., to recommend soundtracks for the “average” user. It is important to note that the proposed benchmark can also be used to evaluate personalized recommendation systems where the evaluation is performed with respect to assessments of the individual assessors.

The rest of the chapter is organized as follows: Section 5.1 describes the used document collection and the queries. Section 5.2 shows how relevance assessments are used to measure retrieval effectiveness. Section 5.3 explains the process of collecting relevance assessments and elaborates on various statistics of the collected assessments. Section 5.4 gives the results of the Picasso and the emotion-based baseline evaluation using the benchmark.

5.1 Evaluation Dataset

A suitable evaluation dataset has to provide a wide coverage of both documents and queries. A common approach in traditional text retrieval is to use a large number of documents (e.g., obtained by crawling parts of the Web) and to perform an initial filtering of documents based on existing approaches.

First, existing approaches are used independently to retrieve the top ranked documents and then these documents are combined (merged) to create, a so called, “pool” of documents. Relevance assessments are then collected only for the documents in the pool in order to minimize the effort of the human judges. This technique is commonly referred to as *pooling*. Due to the small number of existing soundtrack recommendation approaches, pooling would result in a highly biased dataset. Hence, we have to assemble the set of queries (images) and documents (songs) independently from existing approaches in a way that ensures wide coverage while keeping the collection size tractable.

As defined above, an evaluation benchmark $B = (Q, S, R)$ consists of a set of queries Q , a set of documents S (that are songs in our case), and a set of human relevance judgments R . The first step in creating the dataset is to select songs as documents and image groups as queries.

5.1.1 Song Collection

While building the song collection, we focus on popular music and try to achieve high coverage through understanding common music aspects. There are two major aspects that people refer to when talking about music: the *feelings* induced by the music and the *genre* it belongs to. We use Wikipedia¹ to obtain a hierarchy of modern popular music genres and focus on the genres that appear in the top level of the hierarchy. According to the creators of this Wikipedia page, music styles that are not commercially marketed in substantial numbers are not included the list. Additionally, in order to avoid the complexity of working with a large number of nation-specific music styles, we eliminate genres specific to the origin of music, such as “Brazilian music” and “Caribbean music”. The resulting genres, shown in Table 5.1, ranges from Country and Blues, over Metal to Hip Hop and Rap.

Music Genres			
Blues	Classical	Country	Easy listening
Electronic	Hip Hop and Rap	Jazz	Metal
Folk	Pop	Rock	Ska

Table 5.1: List of music genres

Next, we collect a set of feelings and organize them in two high-level groups: *positive and negative feelings*. We obtained an exhaustive list of fine-grained feelings from *Psychpage*². As the obtained list contains generic feelings, some are rarely conveyed by music, such as admiration or satisfaction. To identify feelings expressed through music we used the data from the *last.fm* [LST] music portal. For each general feeling, we check how frequently an artist or a song is annotated with the tag (term) that describes a feeling, for instance, “Sad”. This “*wisdom of crowds*” is gathered using Last.fm’s search capabilities that retrieves all artists and songs annotated with a specific tag. While building the list, we employ a policy that a given feeling is not related to music if there are less than 500 users who used this feeling as a tag. As the result, we get 7 positive and 7 negative feelings conveyed by music, shown in Table 5.2. We see that not only apparent feelings such “Happy” and “Sad” are there, but also less frequent ones are contained, such as “Tragic” and “Optimistic”. This way the number of feelings is limited while still supporting high coverage.

For each of the genres and feelings in the lists, we retrieve the top-10 played (listened to) artists. Again the *last.fm* portal is used for this task, as it contains the number of times an artist is listened to and enables the search for the top-K artists for a given query tag. For each artist, we acquire two representative songs, and automatically cut them to 30 seconds length—from minute 1:00 to 1:30. As some artists appear in multiple groups, (e.g., in the “easy listening” genre and in optimistic feeling), the document collection consists in total of 470

¹http://en.wikipedia.org/wiki/List_of_popular_music_genres

²<http://www.psychpage.com/learning/library/assess/feelings.html>

Positive Feelings			
Happy	Love	Calm	Peaceful
Energetic	Positive	Optimistic	

Negative Feelings			
Sad	Hate	Aggressive	Angry
Depressing	Pathetic	Tragic	

Table 5.2: Feelings induced by music

Image Themes			
Architecture	Aviation	Cloudscape	Conservation
Cosplay	Digiscoping	Fashion	Fine art
Fire	Food	Glamour	Landscape
Miksang	Nature	Old-time	Panorama
Portrait	Sports	Still-life	Street
Underwater	Vernacular	War	Wedding
Wildlife			

Table 5.3: List of query image themes

songs. Having a total of 470 songs make a moderate collection size, while all major music genres and feelings are covered. A sample of the song collection is listed in Appendix A.1.

5.1.2 Query Collection

In the addressed soundtrack recommendation scenario, a query is represented by a set of images. We create a list of 25 queries, each containing 5 images, such that all images of a query follow a specific image theme. The initial list of image themes is retrieved from a list of photography forms, specified on Wikipedia³. For each of these themes, we retrieve images that are annotated with the theme, using the search functionality of Google’s *Picasa* [PIC] photo sharing portal. We manually inspect the returned results and use only themes that provided at least 5 coherent and meaningful images. This filtering step results in the final list of 25 image themes shown in Table 5.3. As we can see, image themes vary from photos take underwater, over photos of people playing sports, to photos of special cloud forms. For each theme, a query is formed by manually selecting 5 publicly available images from Picasa, again keeping in mind the coherence and the meaningfulness of the image theme. Appendix A.2 displays all images from the query collection.

³<http://en.wikipedia.org/wiki/Photography>

5.2 Relevance Measure

Estimating the effectiveness of a retrieval engine is based on measuring the relevance of the returned results with respect to the given query. In traditional text retrieval, relevance is represented by absolute judgments that usually make use of a binary variable indicating that a document is either relevant or not relevant to a given query. Järvelin and Kekäläinen [JK00] proposed a larger grading scale that allows for a finer separation of relevant documents. We adopt such a fine-grained grading scale to assess the suitability of songs for series of images, extending it to the extreme such that for each document (song) there is one level of relevance. Note that such fine-grained scales emphasize the point of possible disagreement between human assessors, when determining how relevant a document is [Voo98].

In the task of soundtrack recommendation, there is no such a notion of fulfilling a particular information need expressed by the query. This renders the assessment less strict in the sense that in general *all* songs can be used as background music. That is, we do not explicitly have the notion of a document (song) being not relevant. Further, user perceived relevance of a song with respect to images highly depends on knowledge of other available songs—it is a very relative assessment task: we can not simply present users small subsets of songs and let them perform the assessment. A consistent full ranking of all available songs, for each query, is required. Thus, we define the relevance $R(s|S, q)$ of the song s , given a song collection S , and a query $q \in Q$, as the rank of that document in the *perfect ranking*. With a “perfect ranking” we denote the full ranking that would be created by the “expert” users. For a result list computed by a specific system for a given query, we can easily aggregate the relevance scores of the individual documents to obtain a final (non-zero) score.

A similar measurement is proposed for the task of similarity search in *sheet music* [TdHdN⁺05], with expert users providing a full ranking of the documents. In contrast to our setup, there, it can indeed be decided if two music sheets are completely not related (relevant to each other), which enables the use of pooling to obtain a filtered and shorter, list of documents for which the full ranking is done.

5.2.1 Pairwise Preference Judgments

What remains is the problem of obtaining the full relevance ranking, for each benchmark query. Doing this in an exhaustive way is prohibitively expensive, though. Instead, the idea is to let users evaluate a large number of song pairs for relevance, for each benchmark query.

We ask human judges to evaluate a large number of song pairs, answering which one of the two presented songs fits better for a given query. This method of assessing is known as *preference judgments*. It is a convenient way to obtain relevance assessments, compared to obtaining absolute relevance judgments [CBCD08]. Ideally, the number of pairs judged for one query is large enough to reconstruct the whole ranking—which is in practice not achievable.

Thus, we collect judgments for only a subset of song pairs.

In addition to selecting the best out of two proposed songs, each judge is asked to assess how much better the selected song fits to the query compared to the other song, on the scale from 1 to 5. A rating of 1 means “almost the same” while 5 means “large difference”. The result of one human assessment is given in the following form $r = (q, s_1, s_2, p, d)$, where q is the image theme query, s_1 and s_2 are songs, p is the preferred song and d is the difference between the songs. Optionally, assessors can provide a textual description (justification) of their decision.

The task at hand is, however, often influenced by the individual taste of the human judges—for some queries more than for others. To capture this factor, we ask multiple assessors to judge the same song pair and use only the ones that show a high level of agreement. This way, the benchmark can serve to evaluate generic soundtrack recommendation approaches.

To isolate the subjectivity of an individual assessor, based on the agreement level, we can check if the selection performed by the judges is *statistically significant*. In case the performed selection is statistically significant we know that the agreement level between judges is high. In case selection is not statistically significant but there is still one song selected more than the other, we can take this pair into consideration, keeping in mind that this was not an easy task—even for human judges.

To check the statistical significance of the agreement between the judges, we formulate the following null and alternative hypotheses:

H₀: assessors are selecting songs randomly, i.e., do not consider the given query images

H₁: assessors are selecting songs based on the given query images

If the null hypothesis is true, each song (of a song pair) is independently selected with probability $p = 0.5$. In that case, the songs are selected independently from the given query and due to the independent trials we can calculate the probability of the final outcome using a binomial distribution. Applying a binomial test [How09] gives us the probability of the outcome, given that the null hypothesis is true. In case the probability of the assessment outcome is smaller than the required significance level ($\alpha = 0.05$) we reject the null hypothesis and say that the agreement level for this question is statistically significant.

We create song pairs for human assessment by first creating song pairs in four different categories: genre, positive, negative, and positive-negative. The pairs in the genre category are all song pairs of the songs gathered based on the genre information. Similarly, the positive category contains all song pairs that have a positive feeling and negative category contains all song pairs with songs having a negative feeling. The positive-negative category consists of song pairs where one song is selected from the positive-feeling group and the second song is selected from the negative-feeling group. The first and the second song are shuffled before presenting them to the user to avoid an ordering bias.

Creating questions posed to human assessor is done by creating all possible triples where one element is an image-theme query and the other two are songs coming from song pairs of one of the four categories created in the previous step. All question triples are stored and the next question to be assessed by judges is selected randomly among all non-assessed questions.

5.2.2 System Effectiveness Measures

As each question, i.e., song pair for a certain query, is answered by 6 assessors, as explained in Section 5.3, we need to reconcile preference judgments made by different assessors. To achieve this, we compute the majority vote for each of the different agreement levels (four out of six (4/6), 5/6, and 6/6). Note that for the agreement level 3/6 there is no majority vote, so we leave this level out. For a certain agreement level, we then obtain a set of relevance judgments R with each $r \in R$ of the form $r = (q, s_1, s_2, p, d)$, where q is an image query, s_1 and s_2 are songs, p is the indication of the song preferred by the majority of users, and d is the difference between songs averaged over multiple users assessing the same pair.

Then, the quality (goodness) G of a ranking can be computed using preference precision [CB08] defined as:

$$G = \frac{\# \text{ correctly ordered pairs}}{\# \text{ evaluated pairs}} \quad (5.1)$$

where the pair of songs is correctly ordered if the song preferred by the most assessors is located higher in the ranking compared to the other song, and the evaluated pairs are all song pairs that are assessed by the judges and are contained in the top- K ranking. A pair of songs is contained in the final top- K ranking if at least one of the songs appear in the top- K results. If only one song is in the top- K results the rank of the second song is considered to be $K + 1$. Intuitively, this measure rewards a system if its ranking agrees with a user's perceived preference, resulting in a higher value with a higher agreement between the two.

Although G is normalized to the $[0, 1]$ interval, due to possible inconsistencies in the transitivity caused by pairwise comparisons, this interval might shrink. An example of the inconsistency can be seen in three pairwise comparisons between three elements, $\{a, b, c\}$, where a is preferred to b , b preferred to c , but c is preferred to a . We see that it is impossible to create a ranking satisfying all comparisons so the upper bound is lower than 1, and the lower bound is higher than 0. Of course, this kind of situations arise because pairwise comparisons are created independently from each other and potentially by other assessors. Still, the actual lower and upper bound can easily be computed once all preference judgments are collected.

Weighted Effectiveness Measure

The specified difference between the songs, denoted as d , can be considered as the strength of preference and, hence, can be taken into account when assessing

the quality of the system. As multiple judges are evaluating the same pair of songs for a given query, the final value of the difference between songs is taken as the average of the single evaluations.

The obvious way to extend the preference precision measure using the preference strength is as follows:

$$G_w = \frac{\sum_{\text{correctly ordered pairs } p_s} p_s}{\sum_{\text{evaluated pairs } p_s} p_s} \quad (5.2)$$

where p_s is the preference strength, having higher value if the preference is stronger. For instance, the preference is strong toward one song if the difference between the two songs is large. Thus, we can use this difference between songs directly as preference strength. Clearly, this measure gives more weight to the preference judgments which were obvious for humans, and dampens the effect of judgments for which even the assessors were not sure about their preference.

5.3 The Benchmark

Processing large amounts of human-involved tasks can be achieved using Amazon’s Mechanical Turk [MTU]. This service represents a mediator between the requester—a person or organization posting tasks to be done—and a number of workers—people willing to perform these tasks, while getting paid for it.

There are studies [ABY11, AST10, SPCK10] concerning the usage of the Mechanical Turk service to collect relevance assessments. All of these studies face the same problem: determining whether the worker really prefers a selected document (song), or if the selection is done randomly to simply gain money, without spending sufficient effort on the given assessment task.

To remove assessments of such “cheaters”, a certain set of question with known answers is inserted in the evaluation task. These questions are referred to as “trap questions”, “honey pots” or “gold standard” questions. Creating trap questions for text retrieval, in case of preference judgments, is an easy task: a pair of one relevant and one obviously irrelevant document are presented to the evaluator. *Cheating* evaluator are then identified by the percentage of times the obviously irrelevant document is selected as the preferred answer.

As our task is more prone to subjectivity, we collect judgments in two phases. In the first phase, we build a set of “gold standard” questions by collecting judgments from students on our campus, in the controlled environment of our offices. These gold standard questions are used as trap questions for the second phase of acquiring assessments, using Mechanical Turk workers. The main hypothesis behind this approach is that evaluators (workers) employed in our offices would have less incentives to cheat as they are payed by hour and not by number of performed assessments.

Each question (song pair and query image theme) is answered by six assessors. We chose six assessors, as the significance level of $\alpha = 0.05$ is achieved in case of all six assessors agree on the preferred song. Only the questions with an agreement level of “six out of six” are used to create trap questions for the next

phase—considering only the preferred song, not the level of difference d . The probability of achieving this level of agreement randomly is quite low, with a p-value of 0.03125, which makes it safe to use these questions as trap questions.

Note that we can chose to preform the quality assessment using only the second phase assessments obtained from Mechanical Turk to indisputably avoid potential student population bias.

Collecting through Mechanical Turk

Collecting a larger amount of assessments is achieved in the second phase with assessments being made by Mechanical Turk workers. To obtain a robust benchmark, again, each question is answered by six workers. This enables an evaluation based on different agreement levels.

Cheating workers are identified as the ones that have performed a large number of questions—expecting high money reward—while choosing answers at random. Due to the binary nature of the questions, cheaters answer approximately 50% of all trap questions correctly. We used a threshold of at least 100 answered questions and less than 65% correct trap questions to *reject* a work of a *cheating* worker. We used one trap question per five regular questions.

Because workers prefer small tasks [ABY11], we created for each question one HIT (Human Intelligent Task). We set the reward to \$0.02 for each performed task, as the reward per task has only a small impact on the quality but rather influences a quantity of the performed tasks [MW09].

5.3.1 Benchmark Statistics

To obtain trap questions for the Mechanical Turk workers, we collected assessments from 30 students. Students were able to choose whether they want to participate in the study for one hour or two hours, while being payed on an hourly basis. 29 out of 30 students participated in the study for two hours, resulting in the total of 665 questions, each question assessed by 6 students.

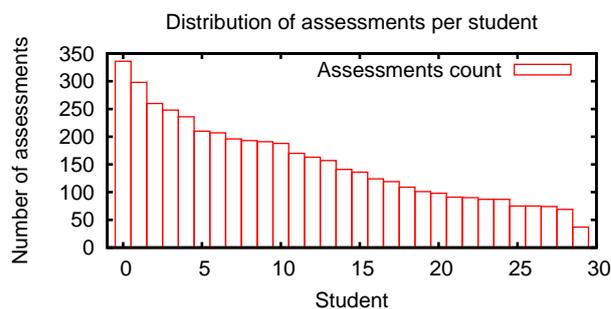


Figure 5.1: Distribution of assessments per student

Figure 5.1 shows the number of assessments per student, sorted in descending order. We observe a high variance in assessment performance, even if we exclude the student that assessed songs for only one hour. Using Pearson’s correlation

coefficient, we investigate if this variance comes from the open ended question, used to elaborate their decision. Pearson’s correlation coefficient between total text length (word count) and a number of assessments is only -0.0875 , indicating that typing the explanation answer is not the reason for the variance in individual assessment performance. If we characterize the agreement with other assessors as a quality estimate of the assessor, we can also check if assessors that produced a large number of assessments have a drop in quality, i.e., have low agreement with others. Pearson’s correlation coefficient between assessments made and agreement with other assessors is only 0.04141 indicating that the quality of the work is also independent of the assessors performance.

	Agreement level			
	3/6	4/6	5/6	6/6
percentage	12.33	32.18	26.17	29.32
average difference	2.75	2.90	3.11	3.65

Table 5.4: Agreement levels for student assessors

Table 5.4 shows the percentages of questions with different agreement levels for student evaluations. Agreement level “x/y” means that x out of y assessors agreed on one song. The observed values in Table 5.4 suggest that we can reject the null hypothesis that student answers were done by randomly choosing songs, supported by the Chi-square test $\chi^2 = 1586.86$, $df = 3$, $p < 0.0001$. This level of significance shows that such a high level of agreement between assessors is almost impossible to achieve by pure chance, but that the task at hand is reasonable and meaningful for the assessors. As we can see, 29.32% (i.e., 195) of all questions have agreement level of 6/6, making them applicable as trap questions for the second phase. The global agreement statistics shows that student assessors agree with each other in 66.94% of all cases. This is lower than the agreement level for the traditional text retrieval task (75.85%) [CBCD08], which also shows that the task of soundtrack recommendation is more subjective.

The averaged difference between songs is also reported for student evaluations in Table 5.4. We see that the average difference is smallest, 2.75, for the questions with low agreement level and gradually increases to 3.65 for the questions with the *six out of six* agreement level. Pearson’s correlation coefficient between agreement level and the average difference between songs is 0.3556, with a randomization test (100.000 permutations) showing that this result is not achievable randomly, $p < 0.0001$.

While assessing the songs, assessors were able to provide textual description of their decision. We analyzed the collected descriptions to see which concepts assessors use for music and images when they are being matched together. We manually extracted all concepts from the descriptions collected from students, shown in Figure 5.2 where the size of a word represent its frequency in the text.

In the second phase we used Amazon’s Mechanical Turk to collect a larger



Figure 5.2: Concepts used to describe matching between images and music

number of assessments. Our aim was to collect enough assessments such that each song for each image query had a chance of being judged once. This required us to have more than 5875 questions evaluated, each question assessed by six assessors. In the end, we collected assessments evaluating 5990 questions in total.

Overall, we had 269 assessors participating in the study. On average, each of them performed 138.69 assessments. As there was no time limit for each assessor, the skew in the number of performed assessments much larger than for the students in phase one, ranging from one evaluation up to 3845 evaluations per assessor. Gold standard questions enabled us to detect 15 *cheating* workers and to reject their work, being replaced by other workers' assessments.

	Agreement level			
	3/6	4/6	5/6	6/6
percentage	17.15	33.89	28.60	20.37
average difference	3.09	3.17	3.37	3.68

Table 5.5: Agreement levels for Mechanical Turk workers

The percentage of questions with respect to agreement levels for Mechanical Turk workers is shown in Table 5.5. As we can see, the percentages of questions with high agreement levels are lower than for student assessments. Still, we can safely reject the hypothesis of randomly provided answers, with $\chi^2 = 6605.18$, $df = 3$, $p < 0.0001$. The reduction in the agreement level might also be an effect of the more diverse population of the workers compared to the population of the students.

We see that the percentage of questions with agreement level of “5/6” and “6/6” is close to 50%, which renders almost half of the evaluated questions usable with high confidence. Overall, the agreement between Mechanical Turk workers was achieved in 62.10% of all cases, slightly less than the overall agreement of students, which corresponds to the drop in the number of high-agreement

questions.

Again we see that there is a correlation between average difference between the songs and the agreement level. Pearson’s correlation coefficient in this case is 0.2928, with randomization test (100.000 permutations) showing again that the probability of randomly achieving this value is $p < 0.0001$.

Query Type Statistics

After merging evaluations performed by students and by the Mechanical Turk workers we calculate the percentage of questions at different levels of agreement for each of the question types, shown in Table 5.6. The average difference between songs is also reported for each agreement level and each question type.

As we can see, the largest percentage of high agreement questions is achieved for questions where both songs have a negative feeling. Inspecting the assessments for these questions revealed that melancholic songs with slow rhythm were usually preferred to fast, loud, and aggressive songs. It is interesting to see that questions formed from different music genres had the least amount of high agreement. This might indicate that songs from different genres might not always be largely different, or that assessments were biased towards preferred music genre, which could be a cause of disagreements.

		Agreement level			
		3/6	4/6	5/6	6/6
Genres	percentage	18.24	37.51	28.30	15.95
	avg. difference	3.16	3.23	3.37	3.65
Positive	percentage	17.28	35.57	29.27	17.88
	avg. difference	2.98	3.06	3.24	3.56
Negative	percentage	13.89	30.01	28.84	27.26
	avg. difference	3.00	3.10	3.37	3.67
Pos.-Neg.	percentage	17.35	31.85	26.95	23.85
	avg. difference	3.10	3.17	3.43	3.79

Table 5.6: Statistics by question type

The percentage of questions with *five out of six* and *six out of six* agreement levels together with the average difference between songs is shown for each image theme in Table 5.7: The average difference between songs does not change a lot over different image themes, varying from 3.19 for architecture up to 3.41 for fashion and wedding themes. On the other hand, the number of high-agreement questions varies substantially, ranging from 35.7% for the war theme to 60.9% for the fine art theme. As expected, emotionally intense themes such as the war, fire, and aviation themes have a substantially lower level of agreement than the “calm” themes such as fine art, portrait, and nature.

Theme	Agr.	Diff.	Theme	Agr.	Diff.	Theme	Agr.	Diff.
architecture	41.2	3.19	fire	38.1	3.30	sports	42.1	3.29
aviation	41.4	3.26	food	53.9	3.33	still life	48.2	3.26
cloudscape	49.8	3.23	glamour	47.4	3.27	street	40.0	3.29
conservation	44.9	3.28	landscape	51.7	3.24	underwater	58.8	3.39
cosplay	42.1	3.26	miksang	51.8	3.32	vernacular	52.1	3.26
digiscoping	55.5	3.34	nature	58.8	3.34	war	35.7	3.36
fashion	45.0	3.41	old-time	54.1	3.27	wedding	58.6	3.41
fineart	60.9	3.29	panoram	51.6	3.23	wildlife	53.6	3.30
			portrait	60.1	3.40			

Table 5.7: Statistics by image theme

5.4 Approaches Evaluation

Using the collected assessments we evaluate our approach Picasso (cf., Chapter 4) with the emotion-based approach in [LS07] (cf., Section 3.1.1) used as a baseline. We execute both systems for each of the 25 queries from the benchmark requesting the top-20 songs as a recommendation result. Preference precision and weighted preference precision (cf., Section 5.2.2) are calculated and reported for both systems.

We extend the Picasso index from the feasibility study (cf., Section 4.3) by extracting information from additional 23 movies. This results in the final index extracted from the total of 50 publicly available movies. All the movies originate from Hollywood production but cover a wide variety in genres and styles. In total, the final index contains 10,454 snapshots and the same number of corresponding soundtrack parts.

For the emotion-based approach to operate we need two training datasets, that is, a set of images and a set of songs with labeled emotions. As part of the songs in the benchmark were acquired based on their emotion labels, we already have a training dataset for the songs.

As a training dataset for images we use the International Affective Picture System [LBC08] dataset. It contains 1196 images, each placed on the three dimensional space of emotions it evokes. The three dimensional space consists of two primary dimensions, namely valence—ranging from pleasant to unpleasant—and arousal—ranging from calm to excited. The third, less strongly related dimension, represents a dominance expressed in the image.

To create a match between music and images, we need a unified representation of emotions. This is achieved by mapping emotions, used as music labels, into the three dimensional space of emotions, used as image labels. Each image is labeled by one emotion, where the emotion label corresponds to the area of the space indicated by the two primary dimensions valence and arousal. The used mapping is shown in Figure 5.3.



Figure 5.3: Mapping emotion labels to two dimensional emotion space

5.4.1 Results

The preference precision results for both systems are shown in Table 5.8. The first column contains the preference precision measures when the systems are evaluated using only the questions with *six out of six* level of agreement. Further, adding questions with *five out of six* agreement level to the evaluation results in precision is shown in the second column, and finally, the evaluation with *four out of six* agreement level question added is shown in the third column.

Fisher’s exact test is used to examine the probability of achieving these differences in precisions in case the results come from the *same* system (hypothetically). The contingency tables for the Fisher’s exact test are created by counting the number of correctly and incorrectly ordered pairs for both approaches.

System	Agreement levels		
	6/6	+5/6	+4/6
Picasso	0.782	0.690	0.614
Emotion-based	0.658	0.595	0.559
Fisher’s exact test (two-tailed)	0.0530	0.0249	0.0938

Table 5.8: Preference precision results

We see that both systems perform best when the questions used for evaluation are the ones for which assessors agreed on the answers. The performance of both systems drops when questions, for which users did not easily agree on the answers, are added to the evaluation. The achieved precision numbers indicate that Picasso performs better with regard to questions at all levels of agreement. Fisher’s exact tests shows that it is not likely that this difference in precision is

achieved by chance. Although the systems achieve precision up to 0.782 (Picasso for *six out of six* agreement level) there is still a large space for improvements in both systems.

We calculate also the weighted preference precision that takes into account the difference between songs specified by the assessors. As the difference between songs is larger when one song fits a lot better to the query more emphasis on these song pairs is put, rewarding/penalizing a system for correct/incorrect ordering of these pairs.

System	Agreement levels		
	6/6	+5/6	+4/6
Picasso	0.818	0.728	0.645
Emotion-based	0.667	0.607	0.570
Student's t-test (two-tailed)	0.0148	0.0042	0.0197

Table 5.9: Weighted preference precision results

The weighted preference precision of both systems is shown in Table 5.9. As we can see, the weighted precision for both systems is higher than the preference precision. This shows that incorrectly ordered song pairs were the ones with small difference between the songs. Again, the best precision is achieved for high agreeing questions as the number of correctly ordered song pairs is higher. We also see that Picasso performs better than the emotion-based approach. By calculating Student's t-test, also shown in Table 5.9, with positive differences for correctly ordered pairs and negative for incorrectly ordered ones, we can reject the hypothesis that the means for the two systems are the same.

Picasso Robustness

We investigate the robustness of the proposed Picasso approach with respect to the size of the training dataset. To achieve this, we take the training dataset extracted from 50 movies and remove a random sample of n movies to form a new, reduced, dataset. The recommendation is then performed using the reduced dataset and the preference precision is calculated. This procedure is repeated 50 times and the average value of the preference precision is taken. We do this repeatedly for $n \in \{10, 20, 30, 40\}$, resulting in dataset sizes of 10, 20, 30, and 40 movies processed.

The resulting average preference precision for each dataset size is shown in Figure 5.4. The preference precision is again calculated depending on the agreement levels of the evaluated song pairs. As before, the precision in all cases is the highest when the evaluation is done with high agreeing questions and the lowest when all the questions are included in the evaluation.

A very important observation we can make from Figure 5.4 is that the system performs better with the growth of the dataset, i.e., the number of movies indexed. Moreover, this correlation between the number of movies indexed and

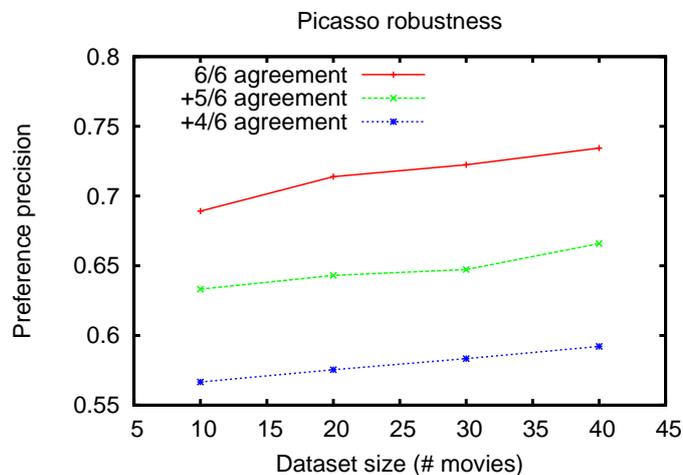


Figure 5.4: Picasso preference precision for different dataset sizes

the system's performance happens for evaluations at all levels of agreement. When evaluated with *six out of six* agreement level, the preference precision goes from 0.69 for ten indexed movies up to 0.73 for forty indexed movies. This indicates, as expected, that further adding of movies into the collection would produce even better soundtrack recommendation results. Of course, this growth in the system's performance by pure increase in the dataset size would saturate at a certain level, requiring further research in the direction of movie selection/filtering to create an appropriate dataset for the soundtrack recommendation task.

Chapter 6

Processing Top-K Queries with Set-Defined Selections

So far we have been mainly concerned with the quality of soundtrack recommendation task. In this and the following chapter we address efficiency issues that arise while solving the soundtrack recommendation task. More concretely, in this chapter we look at top-K queries with set-defined selections, created when the top-K songs are selected such that they are contained in a given music collection. The content of this chapter is based on our previous publication in [SM12a]. Although the addressed efficiency issues arise from the soundtrack recommendation task the core problems are generic and our solutions apply to a multitude of settings.

As we saw in Chapter 4, the recommendation process in Picasso requires the selection of the top-K songs ranked by their similarity to a specific movie soundtrack. Rankings are essential in many applications. Not only do they provide an ordered view on the data, according to some criteria, but they also allows users to focus on a few important results instead of inspecting hundreds or thousands of result items. Research on top-K query processing investigates how only the top portion of a hypothetically full ranking can be computed without full execution of the query—accessing only data items that are elementary for the final result (cf., [IBS08] for an overview).

We consider set-defined selections, where the items of interest are—by whatever means—identified upfront and represented in the query as a set of item ids. In addition, the query specifies the ranking attribute of interest and the result set size referred to as parameter K . Figure 6.1 gives an example: $Attr_4$ is the attribute used to rank; the set $\{2, 4\}$ restricts the result to $id=2$ or $id=4$. In case of a top-1 query with descending ordering, the result would be the tuple with $id=2$, which has a score of 21.6.

This problem setup is very generic and occurs in many different scenarios: Set defined selections occur frequently between independent Web services that offer different views on the same data items—like in case of Linked Open Data

(LOD) portals, where different datasets capture different aspects of (the same) entities. For instance, US-born computer scientists are identified based on a knowledge base [ABK⁺07] and subsequently ranked based on the number of publications obtained from DBLP [DBL].

Id	Attr1	Attr2	Attr3	Attr4	Attr5	Attr6
1	34.1	15.2	13.7	34.1	7.2	23.1
2	38.1	32.8	22.6	21.6	15.2	4.5
3	11.5	11.8	27.0	39.1	24.0	3.5
4	8.5	24.0	5.2	18.2	14.3	24.1
5	21.3	3.4	9.6	22.4	10.2	11.5

Figure 6.1: Top-K query with set-defined selection.

In our setting, a large index contains information about one million songs, where each song is described by a numeric attribute of how good it fits to a specific soundtrack part, i.e., to a specific snapshot taken from the movie. In this case each taken snapshot corresponds to one attribute in Figure 6.1. A query consists of a query image—identifying the attribute—and the selection set—selecting a subset of songs, which in this case is simply a list of songs on a user’s smartphone. Queries are uploaded to the system through an app installed on the user’s smartphone, like the one described in Chapter 8. The index is 120GB big and contains information for 50 sampled movies and one million known songs—to provide enough diversity in both audio and visual dimensions. At query time, though, only the songs the user possesses are relevant for the top-K results.

The size of the selection set drastically influences the design of an ideal index: When the selection set contains a small number of ids, the query is efficiently answered using an index on the id attribute. In case the selection set contains most of the ids, the best performance is reached by reading the ids from score-sorted lists. To be able to choose the appropriate index at query time, we develop cost models for both data organizations. Still, as we will see, the *latency around the break even point* (the point of same performance for both indices) is the main cause of a sub-optimal performance. We show how this latency can be decreased by partitioning the data based on query logs.

Formal Problem Definition

Given a relation R over the attributes $\{id, A_1, \dots, A_N\}$, in which id is the unique identifier of an item (e.g., real world entity, document, video, image). The attributes A_i describe properties of an item and are numeric (integer or floating point numbers).

Definition A top-K selection query is defined by the triple (K, A_i, S) , that is, the size of the result ranking K , the attribute used for ranking A_i , and a set

$S \subseteq \text{dom}(id)$. The task is to efficiently compute those ids that have the K largest values for attribute A_i among all ids that appear in R and the query specific set S . The result is ordered by attribute A_i (without loss of generality in descending order).

That means, a top- K query with set-defined selection can be expressed as a traditional top- K query over the subset of relation R that is given by the selection $\sigma_{id \in S} R$. A SQL-like notation for this kind of query would look like:

```
SELECT id, Ai FROM R
WHERE id IN S
ORDER BY Ai LIMIT K
```

where A_i is any of the numerical attributes of R . In this work, the case of a query specifying exactly one numerical attribute is considered.

In the following, we investigate the trade-offs between id- and score-ordered indices and present a cost model that allows to pick the right index at query time, Section 6.1; In Section 6.2 we propose means to benefit from a partitioned score-ordered index and show how to create such partitions using query logs. Where approximate answers are acceptable, our algorithm can enjoy an even larger gain in latency with provided estimates for the result quality, Section 6.3. We report on the results of a performance evaluation using real-world and synthetic data, Section 6.4.

6.1 Indices and Cost Model

Considering a relation R with attributes $\{id, A_1, \dots, A_N\}$, for each pair of attributes (id, A_i) two basic indices can be created:

- an index on the id attribute (called **id-ordered index**)
- an index on the numerical attribute A_i (called **score-ordered index**)

The id attribute are assumed to be *densely populated in sequential order*, such that the position of a score on the disk can be calculated directly from the id value in id-ordered index. Hence, only scores need to be stored—not the ids themselves. If the ids are not sequential, existing techniques based on B+ trees (cf., e.g., [GMUW08]) can be used for indexing. Because of the specific nature of the queries, we adapt a *column-store data layout* where the relation R is stored on disk in a per-attribute fashion (not row-by-row).

Both index organizations come with advantages and disadvantages: The id-ordered index is ideal if the size of the query set is rather small, resulting in a small number of index lookups. In contrast, the scored-ordered index is ideal if the size of the query set is large, such that K items out of the query set are found very early when scanning the sorted list on disk.

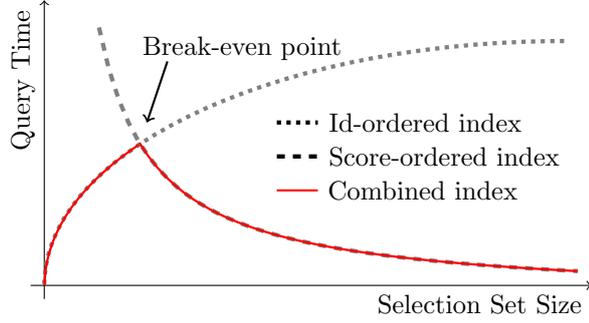


Figure 6.2: Characteristic performances of id-ordered index, score-ordered index, and the ideal combined index.

To benefit from both sweet spots at the same time, a cost model is required to decide at query time which index to use. We optimize for low query response time, which is modeled as $t = c_1 + c_2 \cdot D_b$, where D_b is the size of the data read from disk and c_1 and c_2 are constants which *minimize the squared error* on real-world measurements. The intuition is that c_2 represents the data transfer time, while c_1 approximates the time needed for a random access to disk. The read data size D_b is represented in number of disk blocks, as the access to disk is naturally done in a block-based manner.

To keep the analysis tractable, we ignore influences of distributions of ids and scores, and, hence, treat the size of the selection set and the value of K (specified in the query) as the main ingredients for the amount of data read from disk.

For the **id-ordered index**, Equation 6.1 is used to estimate the number of disk blocks read, D_b . N represents the total number of tuples in the index, b is the number of tuples stored in a single disk block, and P_b is the probability that a single block is going to be read.

$$\begin{aligned}
 D_b &= \frac{N}{b} P_b; \quad P_b = 1 - \overline{P}_b; \quad \overline{P}_b = \frac{\binom{N-b}{|q|}}{\binom{N}{|q|}} \\
 \Rightarrow D_b &= \frac{N}{b} \left(1 - \frac{\binom{N-b}{|q|}}{\binom{N}{|q|}} \right) \tag{6.1}
 \end{aligned}$$

The probability P_b is the probability of reading one or more ids from a single block and is inverse to the probability \overline{P}_b that a block is not going to be read. This probability is approximated by all possible combinations of ids from all blocks without the ids from one block, $N - b$, falling into a query of size $|q|$, divided by all possible combination of all ids, N , falling into query of size $|q|$. This probability approximation assumes independence between ids when they are chosen into a query. This, however, does not affect the precision of the estimate as tuple ids (determining the disk position) are assigned independently of their co-occurrences in the queries (no id reassignment).

For the **score-ordered index**, Equation 6.2 is used to estimate the number of disk blocks read (D_b). Here, b is the number of tuples in one disk block, K is the number of requested results, and P is the portion of the index to be read. The probability that an id just read from disk is contained in the selection set is estimated as $\frac{|q|}{N}$, where $|q|$ is the number of ids in the selection set and N is the total number of ids in the index. The expectation of a geometric distribution with this *hit probability*, i.e., $\frac{N}{|q|}$, gives us the portion of the index needed to be read to find one id that is contained in the selection set. As K ids are needed for the final result, the portion size for one id is then multiplied by K . The total number of blocks is given as the ceiling of the ratio between the index portion and the block size.

$$D_b = \left\lceil \frac{1}{b} \times P \times K \right\rceil; P = \frac{N}{|q|}; \Rightarrow D_b = \left\lceil \frac{N}{b \times |q|} \times K \right\rceil \quad (6.2)$$

Once the query is submitted to the system, the execution-time estimates are calculated and the index with the smaller execution time is used to answer the query. This procedure is totally hidden from the application layer: it appears as one index that combines the best of two index organizations. We call this index **combined index**. With an ideal cost model at hand, the combined index has a performance as illustrated in Figure 6.2.

6.2 Partitioned Index Organization

The above model provides a solid mechanism to identify the best index to use. In fact, experiments show that it is (almost) perfect. On the other hand, the characteristics of the underlying indices lead to a degenerated performance of the combined index around the break-even point (cf., Figure 6.2).

The above cost model reveals a way to improve runtime beyond the point of simply choosing the right index: The index size in terms of number of tuples stored, has a strong influence on the query response time, while the query features themselves can not be influenced.

Thus, the main idea of a partitioned index is to organize the original index into multiple chunks, such that a large fraction of queries is answered by reading only from one of them. This has a high potential: partitioning the score-ordered index into m parts lowers the query answering time by a factor of m (the number of blocks read from the disk would be m times smaller).

The large score-ordered index is chopped up in a set of non-overlapping partitions. Each partition is organized as a score-ordered index. The decision which tuples to put together in a partition is done using a graph-based clustering approach.

To fully harness such a partitioning, we check at query time whether the selection set is

- i) **entirely contained** in one partition,

- ii) **mostly contained** in one partition,
- iii) **distributed** between many partitions.

Answering a query in the first case is done using only the selected partition, while the second case requires a lookup of missing tuples using the id-ordered index. In the third case the query is answered using the combined index.

To determine if the partitioned index should be used for query answering, we employ the cost models introduced in Section 6.1. If one partition captures the entire selection set, only the model for the partitioned index is used, which essentially is the model for a score-ordered index, where the index size is adjusted accordingly.

In case not all of the ids from the selection set are found in one partition, the intersection size is used to estimate the query response time of the partitioned index. The number of the remaining ids (which are not covered by the partition) is used to estimate the lookup cost of the scores in the id-ordered index. The sum of the two estimates is used as a final response time estimate in case of this *mixed access* to the partitioned and the id-ordered index.

6.2.1 The Partition Selection Phase

To determine the partition which contains the largest subset of the selection set, data structures for computing set intersections are required. A bit-set structure is created for each partition, where each id is represented by one bit. The bit indicates whether or not the id is contained in the partition. This representation is exact (no false positives, no false negatives).

First, we consider the case when these bit set structures are kept in main memory, and then we focus on the case when they are stored on the disk with compact sketches used to decide which partition to access on disk.

Bits Sets in Main Memory

Bit sets in main memory allow a fast calculation of the intersection between the query selection set and each of the partitions in the index. *Only the partition with the largest intersection is used*, in case the time estimate using this partition is less than the time estimate for the combined index. Otherwise the query is answered using the combined index. This cost assessment is easy to achieve as the available bit sets give precise (exact) numbers of the contained and missing ids in a partition.

Algorithm 6.1 shows query processing procedure when the bit sets are kept in main memory. First, the approximate query answering time is calculated for the id- and score-ordered indexes (cf., lines 1 and 2). The best matching partition is the partition that has the largest intersection with a selection set specified in the query (cf., line 3). The ids that are in query and are not found in the best matching partition, named *rest*, are computed and later used to retrieve their missing scores in the id-ordered index (cf., line 5). The size of the intersection is used to calculate the approximate query answering time for

Algorithm 6.1 Query processing with bit sets in main memory

```

1   idTime = models.calcId(selectionSet)
2   scoreTime = models.calcScore(selectionSet)

3   Partition p = partitions.getBestMatch(selectionSet)
4   BitSet intersection = p.getBitSet() ∩ selectionSet
5   BitSet rest = selectionSet - intersection
6   partTime = models.calcPart(intersection)+
7       models.calcId(rest)

8   if (idTime.isMin(scoreTime, partTime):
9       return idIndex.query(q)
10  if (scoreTime.isMin(idTime, partTime):
11      return scoreIndex.query(q)
12  if (partTime.isMin(idTime, scoreTime):
13      topK results = p.query(intersection)
14      results.merge(idIndex.read(rest))
15  return results;

```

the partition index, which is combined with an estimate of the time needed to look up the scores not found in the partition (cf., lines 6 and 7). If one of the estimated response times for the original id- or score-ordered indices is smaller than the estimate for the partitioned index, the corresponding index is used for query answering (cf., lines 8 to 11). In case the estimated response time for the partitioned index is the most promising, the best matching partition is queried to retrieve the top-K result (cf., line 13). The scores of the ids that were not found in the best matching partition are retrieved from the id-ordered index and merged with previously retrieved results (cf., line 14).

Bits Sets on Disk

If the bit sets do not fit in main memory, reading them from disk would in most cases consume more time than answering a query using the combined index. Our solution to this problem keeps only compact sketches [FM85, BCFM98, BYJK⁺02] of partitions in main memory and the full sketch information on disk in the header of the corresponding partition. At query time, these sketches are used to determine the most promising partition to access by estimating the intersection size between partitions and the query selection sets. The accuracy of this estimation can be tuned using sketch specific parameters. In this work, KMV sketches (*k*-Minimum Values) [BYJK⁺02] are used.

The structure for the partitioned index with sketches is shown in Figure 6.3: The bit set structure for a given partition is located on the disk at the beginning of the partition itself. This is done such that once the bit set is read—and it tells that the partition is useful—the reading of the actual partition content can

continue without an additional disk seek.

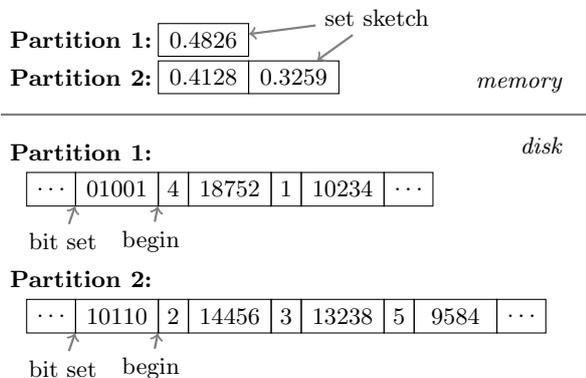


Figure 6.3: Partitioned index structure with sketches

Having only rough sketches of the partition contents in main memory requires changes to the querying algorithm. Exchanging lines 3 and 4 of Algorithm 6.1 with the lines 3 to 8 of Algorithm 6.2 gives us the algorithm for query answering with bit sets stored on disk. First, the most promising partition is identified using sketches (cf., lines 3 and 4). As sketches deliver only approximate results, there are no guarantees that the correct (best) partition will be identified. In case of a high risk to pick a wrong partition, discussed in the following paragraph, we fall back to the existing hybrid index organization (cf., lines 5 and 6). Otherwise, if the best matching partition is identified, we read the bit set from the beginning of the partition (cf., line 7), and after calculating the intersection with a selection set the steps from Algorithm 6.1 are executed.

Algorithm 6.2 Query processing with bit sets on disk

```

:
3   KMVSketch querySketch = selectionSet.sketch()
4   Partition match = partitions.findMatch(querySketch)
5   if (match == null):
6       return hybridIndex.query(q)

7   BitSet ids = partition.readBitSet()
8   BitSet intersection = ids ∩ selectionSet
:

```

It is important to avoid accessing a partition on disk that turns out to be of little use once the bit set is inspected. To limit these wrong decisions, made by the estimation inaccuracy of the sketches, a partition is identified as promising only if we are highly confident that it will be useful for the query optimization later on. The problem with sketches is that comparing a huge list of ids with a

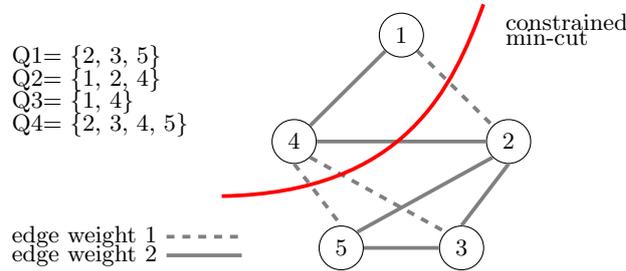


Figure 6.4: Graph based data partitioning

relatively small selection set often results in an empty intersection. We identify a partition to be useful, if and only if the intersection with the selection set is estimated to be larger than zero and there is no other partition for which the estimate is non zero.

In case partitioned indices are created for multiple attribute pairs of the same relation (e.g., (id, A_1) , $(id, A_2), \dots$), the content of the partitions will be the same for all of them. Hence, bit sets describing the partitions are materialized only once—they are shared to reduce the overhead.

6.2.2 Index Partitioning

The problem of data partitioning is formulated as follows: given selection sets from a query log, create m **disjoint data partitions** such that the probability of finding a randomly selected pair of ids from a randomly selected selection set in a single partition is maximized. The partitions should further be approximately equal in size. Determining the optimal number of partitions is not trivial: A large number of ideal partitions (i.e., each selection set is completely found in one partition) would decrease the runtime. However, increasing the number of partitions would increase the error introduced by the partitioning, that means, less and less queries could be answered by a single partition. In this work, we determine the optimal number of partitions experimentally (cf., Section 6.4).

For partitioning, we employ a technique used in recent work on data partitioning in distributed database systems, by Curino et al. [CZJM10]. The basic idea of their approach, coined Schism, is to create a graph based on a database workload (query logs). The vertices of the graph are tuples with edges connecting frequently co-occurring tuples in the transactions. The edge weight is given by the number of transactions in which two connected tuples occur together. Once the graph is constructed, the actual partitioning is done using *constrained k -way graph partitioning* [KL70].

An example of the graph data partitioning, with a query log of four queries, is shown in Figure 6.4. We see that the vertices 1 and 4 are connected with an edge with weight 2 as they occur together in two queries (Q2 and Q3), while the vertices 4 and 5 are connected with an edge of weight 1 as they have only

one query in common (Q4). Using the constrained k-way graph partitioning, we obtain the following two partitions: (1, 4) and (2, 3, 5).

This way, each query contributes $\frac{(|q|-1)(|q|-2)}{2}$ edges to the graph, where $|q|$ is the size of the query selection set. To avoid a quadratic explosion, in particular for queries with larger selection sets, we introduced *edge sampling*. For each of the ids in the given selection set, 10 randomly selected ids from the same set are used to create edges between them, rather than using all of the other ids. This way we get $10 \cdot |q|$ edges for each query, while preserving the density in the graph.

The basic assumption behind index partitioning is that selection sets are clustered in a meaningful way. Although this might not hold in general, selection sets are usually coherent in a semantic way (e.g., most people possess songs from mainly one or two genres).

Constrained k-way graph partitioning is NP-complete, but there exist efficient and accurate approximation techniques. In this work, we are using the software package METIS [KK99, MET], a freely available software library that offers approximate constrained k-way partitioning based on multilevel coarsening techniques.

6.3 Approximate Query Answering

Given the partitioned index organization, even higher performance gains can be achieved by returning approximate top-K results instead of the exact ones. By approximate top-K results, we refer to the case when the selected partition does not cover all of the ids from the selection set. The missing ones could be retrieved based on the id-ordered index, but this is not done now. Hence, there is a risk that some of the missing ids would contribute to the actual top-K result, in which case the returned result is not exact. Although such approximations bring performance gains, without a quantification of the expected error, such approximate results are in most cases not acceptable, as the result quality can arbitrarily vary.

6.3.1 Tunable Expected Precision

Approximate results are often very acceptable, but only up to a point where the precision is still above a certain level, for instance, above 80%. With precision we refer to the fraction of the returned top-K results which are also in the hypothetically exact result.

In the following, we derive an estimate for the results quality provided by the approximate query answering algorithm. Using this estimate at query time enables us to fall back to the exact query answering mode, in case this expected result quality is below the specified minimum threshold.

We describe the probability distribution of the precision as a binomial distribution with “success probability” p_h . This p_h is the probability that an id is found in the partition, given that it is in the selection set and also a true

top-K element. Note that, due to the nature of the partitions and the retrieval algorithm, p_h is the same as the probability that a retrieved top-K element is a true top-K element.

The estimated precision is then given by the expectation of the binomial distribution, divided by the size of the results,

$$P(X \leq x) = \sum_{i=0}^{\lfloor x \rfloor} \binom{K}{i} p_h^i (1 - p_h)^{K-i}$$

$$prec = \frac{E[X]}{K} = \frac{p_h \times K}{K} = p_h$$

where K is the size of the results (K in top-K) and hit probability is p_h .

The success (or hit) probability p_h can be written as

$$p_h = P(id \in F | id \in S, id \in topK)$$

with F representing the partition and S representing the selection set. Applying Bayes' theorem and assuming *conditional independence* between an id being found in the top-K results and the id being found in partition, given that it is in the selection set, we can derive

$$p_h = P(id \in F | id \in S)$$

Applying the definition of conditional probability and using $P(X, Y) = P(X)P(Y) + cov(X, Y)$ for Bernoulli random variables, we obtain:

$$p_h = \frac{P(id \in F)P(id \in S) + cov(id \in F, id \in S)}{P(id \in S)}$$

Estimating that $P(id \in F) = \frac{|F|}{N}$ and that $P(id \in S) = \frac{|S|}{N}$ (where N is the total number of tuples), together with the covariance estimation $cov(id \in F, id \in S) = \frac{|F \cap S|}{N} - \frac{|F||S|}{N^2}$, gives us:

$$p_h = \frac{\frac{|F|}{N} \frac{|S|}{N} + \frac{|F \cap S|}{N} - \frac{|F||S|}{N^2}}{\frac{|S|}{N}} = \frac{|F \cap S|}{|S|}$$

This shows that the precision is estimated as the size of intersection between the selection set and partition, divided by the selection set size. For instance, if there are 90% of the selection set contained in the queried partition, we estimate the precision to be 90%.

To get an accurate precision estimate, we use the exact intersection size using the partition bit sets that are read from the disk once the partition is selected based on the sketches.

6.4 Experimental evaluation

We implemented the described indexing techniques and algorithms in Java 1.6 using direct access to the disk by a JNI (Java Native Interface) connection

to routines written in C. The direct IO access uses the `O_DIRECT` flag in the libraries provided by the operating system. One disk block is kept cached in main memory while data is read from the disk. As a direct access to the disk is used, the size and the speed of the main memory was of negligible influence to our measurements.

All experiments are conducted on a Linux machine (Ubuntu 11.04 64bit with kernel version 2.6.32-28) with a Intel Xeon W3530 CPU (2.8GHz, 8MB cache, 4 cores (8 threads)). The local disk used is a 3Gb/s Barracuda 7200.12 SATA (7200 rpm, 32 MB cache, avg. latency 4.16 ms, random read seek time: 8.5 ms). The routines are executed using Java SE runtime version 1.6 64-bit, with C routines compiled using gcc version 4.4.3.

Prerequisites: To estimate the query response time, the cost models (Section 6.1) need to know the block size. This, however, depends on how much the disk is actually reading ahead with each access to the physical disk. In our case, we measured a size of 896 KB.

Real-world Dataset

For the real-world dataset we get back to the soundtrack recommendation problem that originally motivated addressing these efficiency issues. We analyzed the Million Song Dataset [BMEWL11] which contains low-level feature descriptors of one million songs. This way we need only the titles of a user's mp3 collection and can recommend songs out of this set. A user provided set of songs is interpreted as a selection set.

In addition, 50 movies are processed and analyzed resulting in approximately 10,000 movie snapshots. Fitness information between each song and each movie snapshot is calculate and stored. With 4 bytes for an id and 4 bytes for the score encoding, the total size of the indexed information is around 120GB for both id-ordered and score-ordered indices.

We extended the 1 million songs to 14 million songs to account for a more realistic total count of songs in the world. This approximation is based on the overlap between the user profiles (i.e., sets of songs in users' collections) we received and the list of songs in the freebase [FRE] database. The scores for those artificially added songs are generated from the distribution of scores for the songs in the million songs dataset.

For one index entry, the score in case of the id-ordered index and both the score and id in case of the score-ordered index are stored—which sums up to 12 bytes in total. As one tuple needs one bit in the partition bit set, the size of the bit set for one partition is $12 \times 8 (= 96)$ smaller than the combined index size of one column (e.g., movie snapshot). As bit sets are the same for all columns, only one copy per partition is kept. In our case of 10,000 columns, thus, the required space for the bit set information of one partition is a factor of 960,000 smaller than the whole combined index (relation). When the bit sets are stored on disk, we used 28,000 minimum values in the KVM-sketch (each value is of 4 bytes size) for each partition, rendering the signature for one partition 60,000 times smaller than the combined index size of one column. As an illustration:

with only 1 GB of main memory used and 10 partitions one can index 5.85 TB of data on disk using sketches.

Queries: We use 150 user profiles, i.e., sets of songs, obtained through our smartphone app PicasSound (cf., Chapter 8). Each of those profiles represents one selection set that we use as the top-10 query in this study. We execute each query ten times and the average runtime is reported for that query. Each execution is accessing disk (no caching), which is a direct consequence of direct IO usage. The distribution of the selection set sizes for the smartphone queries is shown in Figure 6.5.

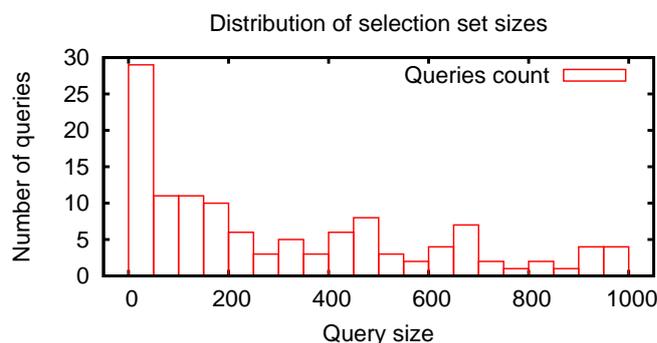


Figure 6.5: Distribution of selection set sizes

Partitioning: as 150 queries obtained from smartphone users are not enough for both testing and partitioning purposes and in particular to also show that the partitioning can be done on different but semantically related data, we use publicly available data [Cel10a] from last.fm [LST] for the partitioning. This dataset contains the most-listened-to artists for 360,000 last.fm users. Although this dataset contains artists rather than individual songs, we partition the artists and then assign songs to the partition which contains the corresponding artist. The partitioning of the artists is done as described in Section 6.2.2, where the top artists of one user are considered as a query used in graph creation.

As the partitioning dataset and the query dataset have different sources, we measured that around 82% of the artists present in the queries are also contained in the partitioning dataset. We removed song ids from the queries' selection sets if they were not contained in the partitioning set.

Synthetic DataSet

For the synthetic dataset, we use the same song collection with the same scores used in the real-world dataset, but create synthetic queries for execution and queries used in the partitioning process. This allows varying the query clustering density to study its effect on the performance of the partitioned index. To create the artificial clustering, ids are randomly split into 10 groups (artificial clusters). Then, we generate queries based on these groups.

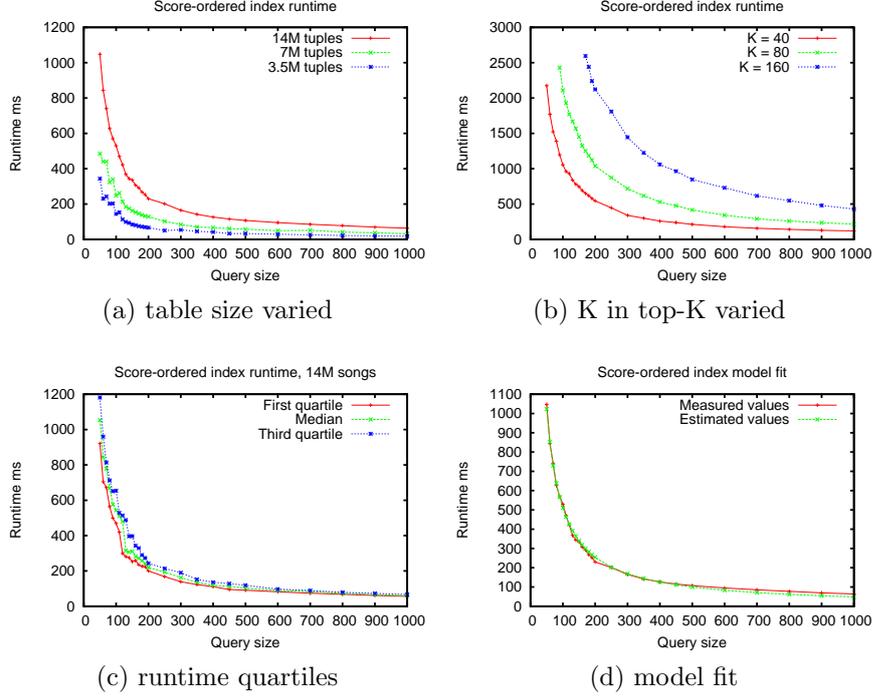


Figure 6.6: Score-ordered index runtime

We distinguish three types of queries: (i) all ids from the selection set are found in one cluster (ii) ids are split between two clusters and (iii) ids are uniformly spread between all clusters. Based on the ratio between these types of queries, we create three query logs for partitioning and testing purposes. The first query log, named “best” contains 80% of type (i) queries, 10% of type (ii) queries, and 10% of type (iii) queries. The second query log, named “middle”, contains 60% type (i), 20% type (ii), and 20% of type (iii) queries. Finally, the third, named “worst” query log contains 40% type (i), 30% type (ii), and 30% of type (iii) queries.

Queries: 200 queries for testing purposes were generated for each of the query log types with selection size uniformly distributed between 10 and 1000. Each query is generated by randomly selecting ids from one, two, or all artificial clusters. For instance, the “best” query log has 80% queries for which ids are selected from one randomly selected artificial cluster.

Partitioning: 20,000 queries for each of the query log types are generated, with the selection size uniformly distributed between 10 and 500.

6.4.1 Model Verification

First, we experimentally evaluate the accuracy of the runtime estimates obtained by the models proposed in Section 6.1. We use the songs dataset and create top-20 queries with uniformly selected ids in the selection set.

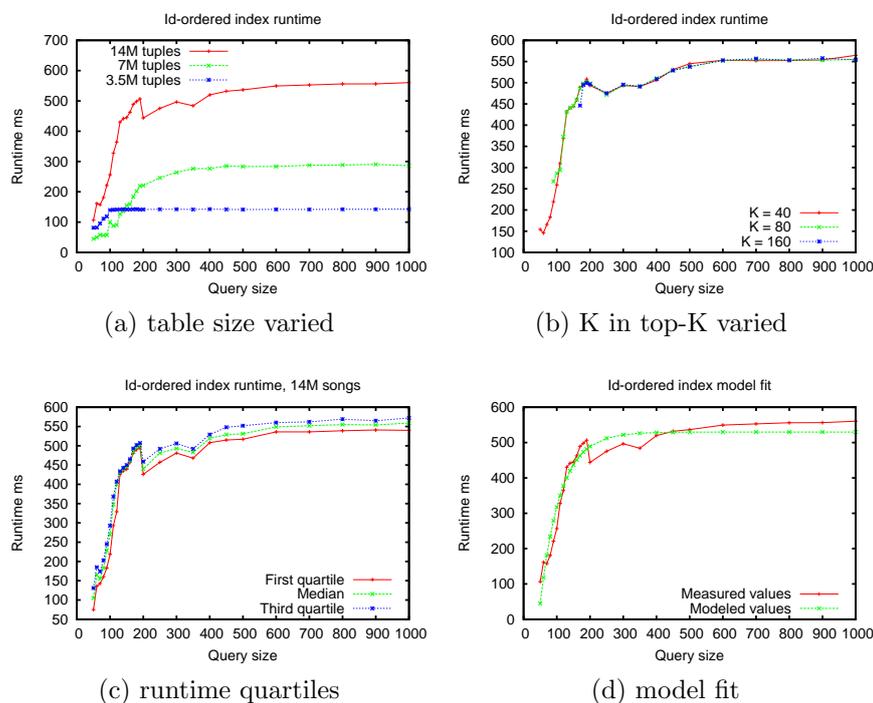


Figure 6.7: Id-ordered index runtime

First, we measure the effect of the table size (in terms of number of tuples) on the runtime of the score-ordered index. The results are shown in Figure 6.6(a): we distinguish three index sizes, 14 million tuples, 7 million tuples, and 3.5 million tuples. We can see that the runtime drops almost by a factor of two when decreasing the index by a factor of two. This shows that the runtime is proportional to the number of tuples stored in the index as proposed by the model (cf., Equation 6.2). The effects of changing K in the top- K queries on the runtime using the score-ordered index is shown in Figure 6.6(b). We observe that a larger value of K increases the runtime proportionally, which is again in accordance to the model. After fitting the c_1 and c_2 parameters, we see that the estimated values are almost a perfect fit to the measured values, see Figure 6.6(d).

The effects of the table size on the id-ordered index are shown in Figure 6.7(a), where we see that the runtime is proportional to the table size. This plot also reveals small anomalies for small query sizes, in which case the 7 million songs index is performing better than the 3.5 million songs index. This together with the sudden decrease in runtime for 14 million songs at query size 200 is most likely the effect of the disk's adaptive read-ahead strategy. Figure 6.7(d) shows that the values estimated the model are very close to the measured ones. Figure 6.7(b) shows that the parameter K has no effect on the runtime for id-ordered index. This is also in accordance to our model (cf., Equation 6.1).

The quartiles of multiple runtime measurements on the score-ordered index

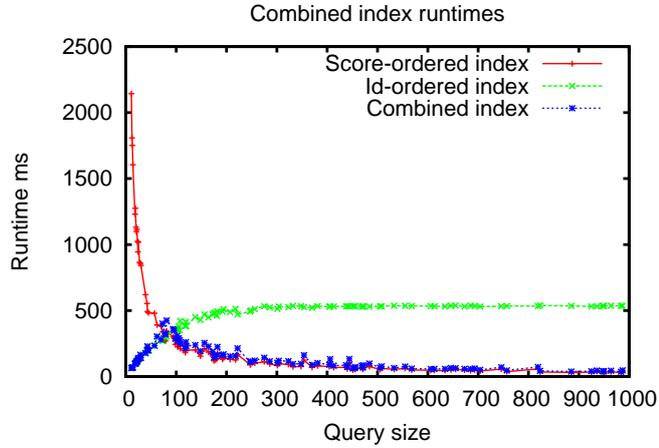
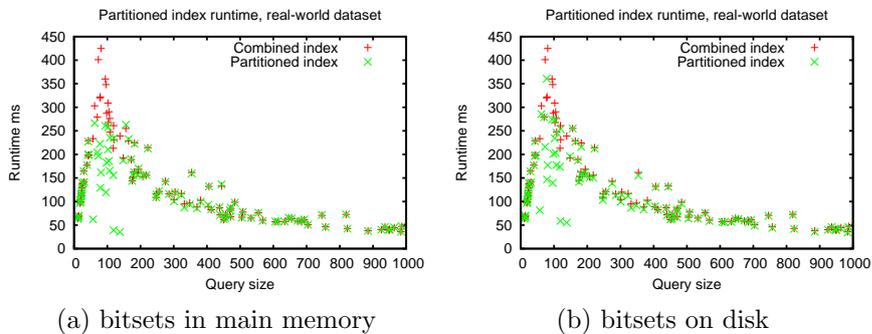


Figure 6.8: Runtime combined index

(Figure 6.6(c) and the id-ordered index (Figure 6.7(c)) are shown for 14 million songs and $K = 20$. We see that the runtimes have low variance, which means that time estimates are good in general, not only on average.

Overall, we see that the proposed models give a good estimate of the runtime for both indices. It is important to note that they do not have to be perfect in estimating the absolute runtimes, rather we need them to provide a reasonable decision of which index is better suitable to answer a query. To get a better understanding of this, we study the performance of the combined index which highly depends on the runtime estimates. The results for the real-world dataset are shown in Figure 6.8. We observe that the combined index performs almost optimally, choosing the right index for almost all of the queries, with only a couple of misses around the break-even point where the difference in runtime of the id- and score-ordered indices is negligible.



(a) bitsets in main memory

(b) bitsets on disk

Figure 6.9: Partitioned index runtime

6.4.2 Exact Top-K Retrieval

We report on the results for the partitioned index evaluation, when the index is used to retrieve the exact top-K results. We measure the runtime of the partitioned index on the real-world dataset with 10 partitions, when the bit sets are in main memory (Figure 6.9(a)) and when they are on disk (Figure 6.9(b)). The selection set size is shown on the x-axis, while the y-axis shows the runtime in milliseconds. We summarize these results in Figure 6.10, where query sizes are grouped into ranges of size 50. For a better readability, we do not use histogram plots (with bars). As expected, when the selection set size is small, the id-ordered index provides an efficient query answering. On the other hand, if the selection set size is large, the query can be processed more efficiently with the score-ordered index. Around the break-even point between these two indices, the partitioned index can greatly improve performance, as shown in Figure 6.10. We observe that the latency for queries of sizes between 50 and 100 drops down more than 140ms with bit sets in main memory and around 70ms for queries of size between 100 and 150. When the bit sets are kept on disk, the performance of the partitioned index is slightly worse than with bit sets in main memory, which is expected as reading bit sets from disk causes additional latency.

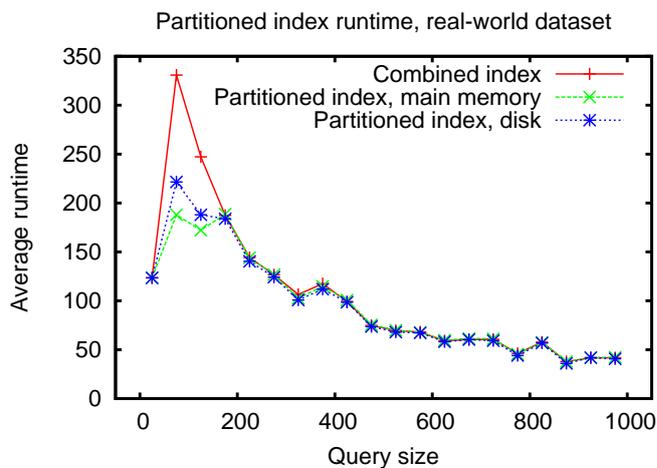


Figure 6.10: Partitioned index runtime, ten partitions

To study the implications of using different numbers of partitions on the performances of the partitioned index, we show results for 4, 6, 8, and 10 partitions. The runtime improvements (difference between combined and partitioned index) in terms of milliseconds for different numbers of partitions are shown in Table 6.1 and Table 6.2. Table 6.1 reports on the runtime improvements when the bit sets are kept in main memory, while Table 6.2 reports on the case when the bit sets are stored on disk. As the improvements were measured for queries with selection set size between 50 and 150, as seen in Figure 6.10, we report only on these. As we can see from these tables, some numbers of partitions

Query size	Number of partitions			
	4 part.	6 part.	8 part.	10 part.
0-50	0	0	0	0
50-100	112	130	145	142
100-150	82	97	59	75
>150	0	0	0	0

Table 6.1: Runtime improvement (ms), bit sets in memory

Query size	Number of partitions			
	4 part.	6 part.	8 part.	10 part.
0-50	0	0	0	0
50-100	50	87	119	109
100-150	40	47	47	60
>150	0	0	0	0

Table 6.2: Runtime improvement (ms), bitsets on disk

perform better than the others. We can also see that we have comparable improvements even if we are not using the optimal number of partitions, rendering our approach robust to changes in the number of partitions used.

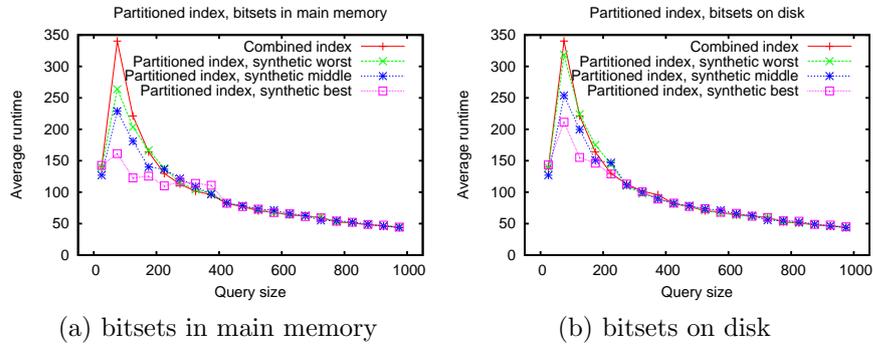


Figure 6.11: Synthetic dataset runtime

Figure 6.11 shows runtime experiments for the partitioned index on the synthetic dataset, when (a) bitsets are kept in main memory and (b) bitsets are kept on disk. As expected, the runtimes are high for the worst synthetic dataset and get smaller with the middle dataset and are smallest for the best synthetic dataset. We can see that the gain for the worst dataset, when the bit sets are kept on disk, is tiny, almost nonexistent. However, when the bit sets are kept in main memory, there is a gain for even the worst setup. Comparison between keeping bitsets in main memory and on disk is shown in Figure 6.12(a) for best synthetic dataset and in Figure 6.12(b) for worst synthetic dataset.

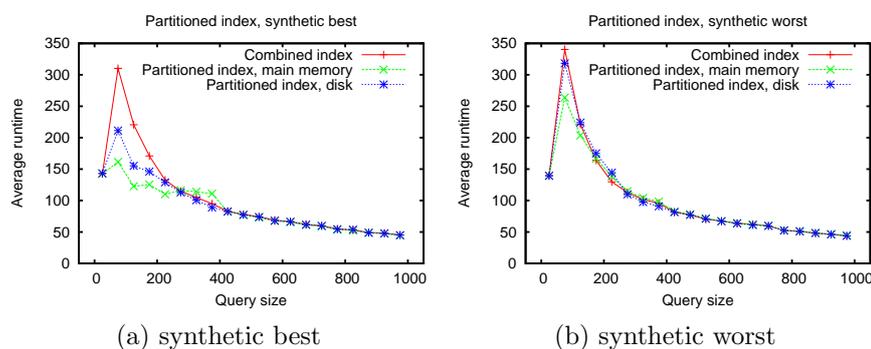


Figure 6.12: Partitioned index runtime

6.4.3 Approximate Top-K Retrieval

We have measured the runtime for the approximate querying, with and without guarantees, using the real-world dataset, shown in Figure 6.13. We see that the average runtime for the approximate querying without guarantees is much lower than for the variant with guarantees, e.g., for query sizes between 100 and 150 average runtime without guarantees is 83 milliseconds while runtime with guarantees is 140 milliseconds. This of course comes with the cost of decreased in precision shown in the same Figure 6.13. We used a requested precision of 80% for querying with guarantees in this case, which resulted in average precision around 95%.

Varying the requested (minimum) precision from 60% to 90% for approximate querying with guarantees is shown in Figure 6.14. We see that the precision drops down when the requested precision is lowered. Lowering the requested precision results also in lower runtimes, as reported in Figure 6.14.

The results of the approximate querying without guarantees over synthetic data are shown in Figure 6.15(a). As expected, the runtime for best synthetic dataset is lowest, the middle synthetic dataset has medium runtime, and the worst setup incurs the highest runtime. We see that even for the worst synthetic dataset, the improvement in runtime over the combined index is still high (up to 105 ms). As the modeling in these three cases has little control over the precision, it varies independently of the synthetic dataset use (Figure 6.15(a)).

Approximate querying with guarantees and a requested precision of 90% over synthetic data results in an average precision close to 100%, independently of the dataset used, cf., Figure 6.15(b). This emphasizes once more the influence of the requested precision over the actually observed result precision. The measured runtime is again as expected. It is lowest for the best synthetic dataset, highest for the worst dataset, and medium for the middle one, cf., Figure 6.15(b).

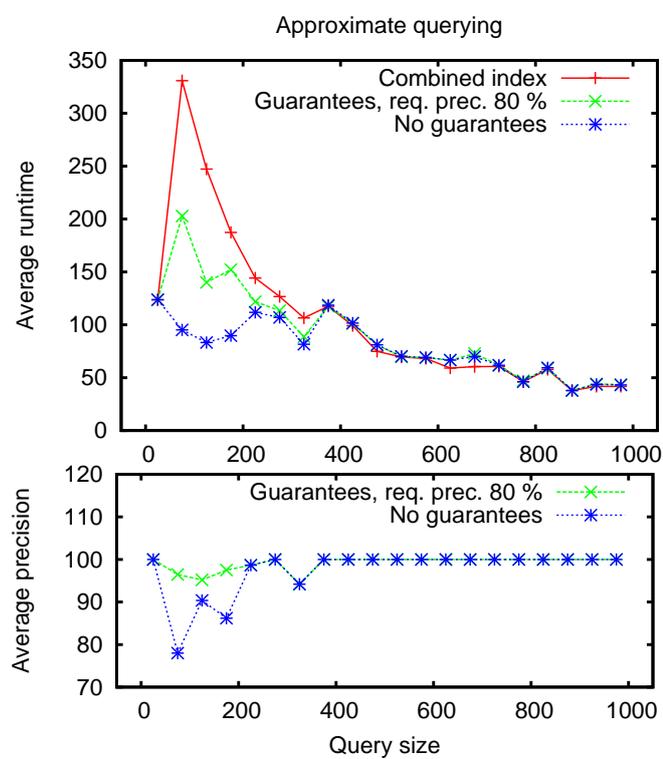


Figure 6.13: Approximate querying, real-world dataset

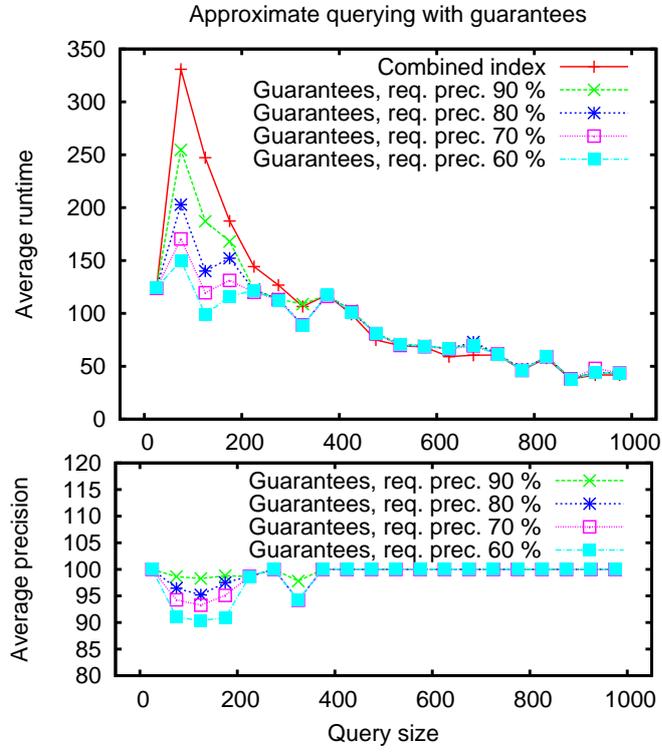
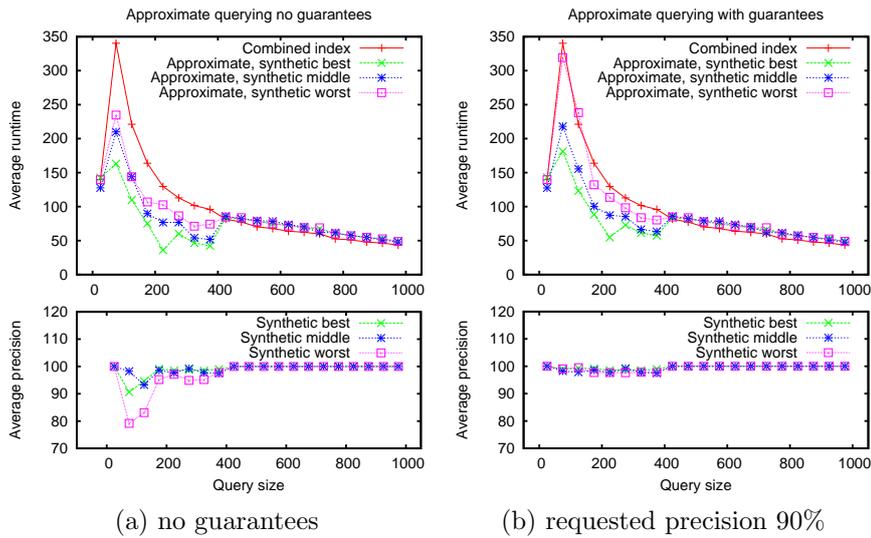


Figure 6.14: Approximate querying, real-world dataset



(a) no guarantees

(b) requested precision 90%

Figure 6.15: Approximate querying, synthetic dataset

Chapter 7

Enhancing and Distributing Locality Sensitive Hashing

As we saw in Section 2.2, searching for the most similar images in the collection for a given query image translates to the k-Nearest Neighbor (KNN) problem [Sam06], where each image is represented as a point in a multidimensional feature space. With increasing dimensionality this task becomes computation intensive. To achieve a desirable efficiency, approximate methods are used to solve the task: Locality Sensitive Hashing (LSH) [AI06, DIIM04, GIM99] allows the trade off between the memory used for the index, accuracy of the results, and the time needed to answer a query.

The proposed Picasso approach uses similarity search between images to retrieve the most similar movie screenshot to a given query image. To increase the quality of the soundtrack recommendations, a large number of movies is used for the training dataset, resulting in a large collection of movie screenshots. This motivated us to look at the efficiency aspects of LSH and its computation using the MapReduce framework to achieve scalable, large-scale similarity search.

In this chapter, we describe two enhancements to the standard LSH methods to accelerate the performance while keeping the answer accuracy at the same level. The first enhancement of LSH is based on additionally *introduced links* for each point in the feature space. These links refer to the exact nearest neighbor and are calculated and stored in a preprocessing phase. The second approach is coined *Peek-Probing*, where LSH buckets are only fully read if they indicate a certain amount of useful information. A salient property of our proposed techniques is their orthogonality, hence, they can be jointly applied, and their independence on the underlying LSH method.

In addition to the two LSH enhancements, this chapter describes RankReduce, an approach to distribute LSH techniques by implementing them on top of the highly reliable and scalable MapReduce [DG04] infrastructure. This task poses interesting challenges to the integration: most of the time, we face different characteristics of MapReduce and LSH which need to be harnessed at the same time to achieve both high accuracy and good performance.

The description of the two enhancements to LSH is based on our publication in [SM12b], while the LSH distributed scheme description is based on the publication in [SMS10].

The chapter is organized as follows. Section 7.1 presents the enhancement based on the nearest neighbor links, coined Linked-LSH approach. The Peek-Probing approach is described in Section 7.2. The distribution of LSH using MapReduce is described in Section 7.3. The description of the experimental setup together with the reported results on baselines and proposed approaches is contained in Section 7.4.

7.1 Linked-LSH

Locality Sensitive Hashing (LSH) is based on the principle of *locality preserving* hash functions. These functions map points from a high-dimensional space to hash values (i.e., hash buckets), such that close points have the same hash value, with high probability. To increase locality, multiple hash tables are used, each with its own hash function. For more details on the basics of LSH see Section 2.2.1.

As we can see, LSH indexes data points by computing hash bucket labels (i.e., hash values) of each object independently of the other data points contained in the same collection. The key idea behind *Linked-LSH* is to use additional information about the dataset, obtained at indexing time, with the goal of obtaining an improved query processing performance. More precisely, we use the (first) nearest neighbor in the indexed collection for all the data points as a global statistics descriptor of that collection.

The intuition behind using the exact first nearest neighbor as a descriptor is given by the triangle inequality $d(q, p_2) \leq d(q, p_1) + d(p_1, p_2)$, where q represents a query point, p_1 is the point indicated by LSH and p_2 is the point missed by LSH and is the exact closest neighbor in the collection for the point p_1 . As p_2 is the exact closest neighbor for the p_1 , there is a high probability that the distance between these two points $d(p_1, p_2)$ is small. If we make sure that the distance between the query point q and the point indicated by LSH p_1 is small enough, the triangle inequality tells us that distance between the query point q and point p_2 is also small and that there is high probability that the point p_2 is also in the exact top-K results for the given query.

Linked-LSH extends the LSH index by adding a pointer to each indexed data point (feature vector), which points to the closest neighbor in the indexed collection. Pointers for data points are precomputed in the indexing phase and the exact closest neighbor is used, which is found through the full scan of all data points. It is important to note that this extension of the index results in a negligible increase in index size as each data point contains values for multiple dimensions and only one pointer value in addition.

Figure 7.1 illustrates an LSH index with links introduced by Linked-LSH in the two dimensional space. The (red) rectangular data point represents a query point which after hashing to LSH index retrieves a first neighbor. The (green)

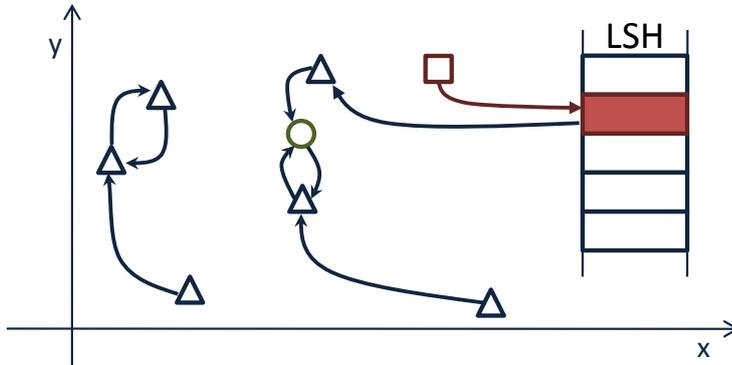


Figure 7.1: Linked-LSH index organization

circular point represents the second exact neighbor to the query point which is missed by LSH due to its approximate nature. We see that following the link from the first retrieved neighbor to its first neighbor results in retrieving the second exact neighbor of the query point, which would be lost if only LSH was used.

7.1.1 Query processing

Answering a query with Linked-LSH is done in two steps. In the first step the query is answered using an existing LSH approach, such as [LJW⁺07]. The obtained query results are then used in the second step, in which the links between data points are used to retrieve and evaluate additional points.

As stated above, we need to assure that the distance $d(q, p_1)$ is small enough. That is the reason why we use only the top- \tilde{K} results from the first step as an input to the second step, as it guarantees that we are using the points that have the smallest distance to the query point out of the points indicated by LSH. The parameter \tilde{K} should depend on the original K as specified in the query and can be experimentally tuned for the best performances. It is important to note, that \tilde{K} as well as the parameter n , introduced in the following paragraph, are parameters that are set at runtime and hence can be adjusted without index re-organization, which is usually needed for the parameters in the raw LSH approaches.

The approximate top- \tilde{K} results from the first step provide a starting point for further retrieval using links in the collection. For each point in the top- \tilde{K} points, we recursively retrieve the closest neighbors up to the depth n by consecutively following closest neighbor links. The depth n of the consecutive retrieval is again a parameter of the approach. The distance to each additionally retrieved data point is calculated, in case it was not already calculated in the previous step, and used to evaluate if the data point is in the top- K result list. The pseudo code of the query answering with Linked-LSH is shown in Algorithm 7.1.

By design, Linked-LSH can be applied to any of the existing LSH approaches

Algorithm 7.1 Query processing with Linked-LSH

```

1  intermediate = LSH.eval(query,  $\tilde{K}$ )
2  knn.add(intermediate)
3  for (point in intermediate):
4      last = point
5      while (steps  $\leq$  depth):
6          last = last.getNeighbor()
7          knn.eval(last)
8  return knn.results

```

[GIM99, LJW⁺07] by simply using them in the first step and following the links based on their results in the second step.

7.2 Peek-Probing

The LSH index consists of multiple hash tables such that each of them contains multiple buckets (cf., Section 2.2.1). Each bucket contains a subset of the data points, assigned by a hash function. In the query answering phase, multiple buckets are selected based on the hash value of the query or based on Multi-Probe techniques [LJW⁺07]. Our *Peek-Probing* approach assumes that not all of the selected buckets have the same importance to the query answering. We try to determine that importance before evaluating all the data points from all of the buckets.

The idea is to use existing LSH techniques to select buckets, and then to peek into each of these buckets and to predict how important it is for answering the given query. After the bucket importance values are approximated we use only the data points from the most important buckets and discard the rest. The key point here is that the importance of the bucket is determined for each specific query.

To approximate the importance of a bucket for the given query, we peek into all buckets indicated by LSH and perform a KNN evaluation over the seen data points. Peeking into a bucket means retrieving the first p elements from a bucket where p is proportional to the bucket size (number of data points in the bucket) and is given by

$$p = 1 + \left\lfloor \frac{b}{f} \right\rfloor$$

where b is the size of the bucket and f is the bucket fraction proportion (we used the value of $f = 8$ in all of the experiments). During the evaluation of the peeked data points, if the data point makes it in the *peeked top-K results*, we remember the bucket that point came from. We show below how we re-organize the bucket content to obtain a meaningful overview of the bucket content, as otherwise the p first points would represent a random sample.

After all the peeked data points are evaluated, we call a bucket important if there is at least a single data point in bucket that is also in the top-K peeked

results. This approach works well in practice as K is usually a small number. In case K is very large number we could instead judge the importance of the bucket as a total number of data points in the peeked top- K results that come from that bucket, rather than to just make a binary decision. In the following steps only the data points from the important buckets are used for evaluation. The querying algorithm is presented in Algorithm 7.2.

Algorithm 7.2 Query processing with Peek-Probing

```

1  buckets = LSH.probe(query)
2  for (bucket in buckets):
3      for (point in bucket.peek):
4          knn.eval(point, bucket)

5  important = knn.getBuckets()
6  for (bucket in important):
7      for (point in bucket.rest):
8          knn.eval(point)
9  return knn.results

```

It is important to note that as there are by design multiple hash tables in LSH, data points can be contained in multiple buckets originating from different hash tables. This means that we need to store multiple buckets for a data point at runtime, while evaluating peeked points. We have experimented with different number of buckets saved per data point and concluded that the best performances are achieved when only one bucket (the first one encountered) is saved for the data point. This small number is imposed by the large overhead in bookkeeping all of the information when multiple buckets are saved per data point.

7.2.1 Bucket Organization

Peeking into a bucket is performed to get an idea about the content of that bucket, and as already mentioned is done on the first p data points of the bucket. We can randomly select any p points of the bucket and place them in the beginning of the bucket. However, by doing this we may end up with a bad description of the content based only on the first p data points.

To avoid such situations we select first p points from the bucket by *clustering the bucket data* in p clusters and then selecting the medoid of each cluster to be put in the beginning of the bucket. We use the expectation-maximization algorithm for k-means clustering [HKP06] to cluster the data contained in the buckets. Figure 7.2 illustrates this process of data points selection and their placement in the beginning of the bucket.

The motivation behind this idea is given by the k-means optimization criterion $\arg \min_C \sum_{i=1}^p \sum_{x_j \in C_i} \|x_j - \mu_i\|^2$, where C_i is a cluster with centroid μ_i and x_j are data points from that cluster. This optimization criterion tells us

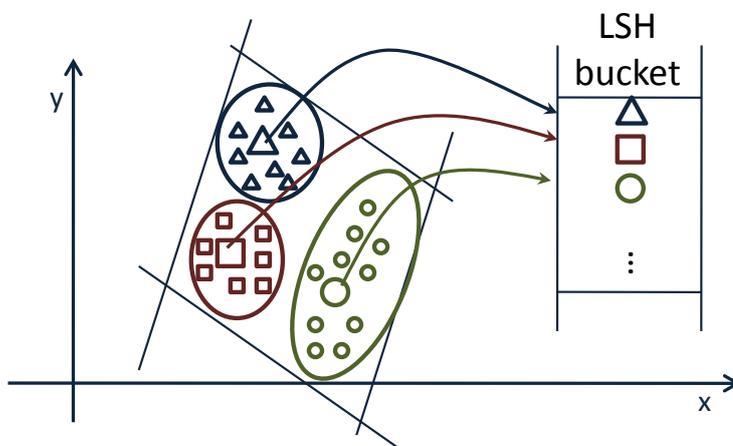


Figure 7.2: Peek-Probing bucket organization

that the distance between the centroid and the data point in that cluster is minimized, i.e., if the centroid is in the top-K results among the other centroids then there is a high probability that some other points from that cluster are also in the final top-K results. We use medoids, the closest point from the cluster to the centroid, instead of the centroids, as centroids are non-existent data points and would incur additional computation and storage cost.

As we mentioned earlier Linked-LSH and Peek-Probing can be applied to any existing LSH approach. We coin the name combined approach for the approach where the Linked-LSH is applied on top of the Peek-Probing approach, which is again applied on top of the Multi-Probe approach from [LJW⁺07].

7.3 RankReduce Framework

We now investigate the problem of processing K-Nearest Neighbor queries in large datasets by implementing a distributed LSH-based index within the MapReduce Framework [DG04]. The framework is designed to be used for large data processing in parallel. It is built on top of the Distributed File System [GGL03], which enables distributing the data over the cluster machines in a scalable and fault-tolerant way. Our implementation uses the open source software Hadoop [HAD], maintained by the Apache Foundation, which provides a Java-based implementation of both the MapReduce framework and the Distributed File System (coined HDFS for Hadoop Distributed File System).

MapReduce is a fairly simple programming model, based on two developer-supplied functions: *Map* and *Reduce*. Both functions are based on key-value pairs. The Map function receives a key-value pair as input and emits multiple (or none) key-value pairs as output. The output from all Map functions is grouped by key, and for each such key, all values are fed to the Reduce function, which then produces the final output from these values. For more details on the MapReduce framework and Hadoop implementation see Section 2.4.

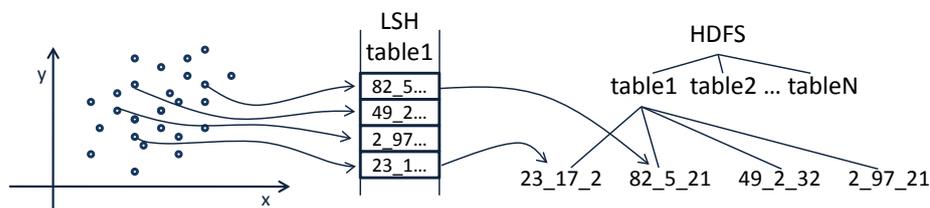


Figure 7.3: Mapping LSH buckets to HDFS

We distribute the LSH index by mapping each hash table from the index to one folder in HDFS. For each bucket in such a hash table, a corresponding file is created in this folder, where the file name is created by concatenating hash values into a string, with reserved character as a separator, as illustrated in Figure 7.3. This mapping of buckets to HDFS files enables fast lookup at query time and ensures that only data that is to be probed is read from the HDFS. Placing the bucket in one file also enables block based sequential access to all vectors in one bucket, which is very important as the MapReduce framework is optimized for such block based access rather than random access processing. Each of the buckets stores the complete feature vectors of all objects gped to this bucket in a binary encoding.

Indexing new feature vectors is easily done by appending them to the end of the appropriate bucket file. This can also be done in parallel with query processing as long as different buckets are affected; as HDFS does not include a transaction mechanism, appending entries to buckets that are being queried would be possible, but with unclear semantics for running queries. As HDFS scales well with increasing cluster size, the resulting growth of the LSH index can easily be supported by adding more machines to the cluster.

While an LSH index stored in-memory has no limitation on the number of buckets, too many files in HDFS can downgrade its performance, especially if these files are much smaller than the block size (which defaults to 64MB). The number of buckets, and therefore the number of files in HDFS for the LSH index, is highly dependent on the set up of LSH parameters.

Inspired by in-memory indexes, which can have references from buckets to materialized feature vectors, we considered storing only feature vector ids in the buckets instead of the actual feature vectors, and retrieving the full vectors only on demand at query time. However, this approach would result in poor performance due to many random accesses to the HDFS when retrieving the full vectors, so we decided to store complete feature vectors. This fact also needs to be addressed when setting up LSH parameters, while too many LSH hash tables can dramatically increase index size, as each feature vector is materialized for each hash table.

7.3.1 Query Processing

We implemented KNN query processing as a MapReduce job. Before starting this MapReduce job, the hash values for the query documents are calculated. These values are then used for selecting the buckets from the LSH index, which are to be probed. The selected buckets are provided as input to the query processing MapReduce job, generating multiple input splits. The generated input splits are read by a custom implementation of the *InputFormat* class, which reads feature vectors stored in a binary format and provides them as the key part of the Map function input. Queries are being distributed to mappers either by putting them in the *Distributed Cache* or by putting them in HDFS file with high number of replicas. They are read once by the *InputFormat* implementation and reused as value part of the Map function input between the function invocations.

The input to the Map function consists therefore of the feature vector to be probed as the key and the list of queries as the value. The Map function computes the similarity of the feature vector with all query vectors. While a standard MapReduce implementation would now emit a result pair for each combination of feature vector and query vector, we employ an optimization that delays emitting results until all feature vectors in the input split have been processed. We then eventually emit the final K-Nearest Neighbor for each query vector from this input split in the form of key-value pairs. Here, the query is the key and a nearest neighbor together with its distance to the query vector is the value. To implement this delayed emitting, we store the currently best K-Nearest Neighbor for each query in-memory, together with their distances from the query points. The results are emitted at the end of processing the input split in Hadoop's cleanup method¹. The Reduce method then reads, for each query, the K-Nearest Neighbor from each mapper, sorts them by increasing distance, and emits the best K of them as the final result for this query.

The final sort in the reducer can even be executed within Hadoop instead of inside the Reduce method, as a subtask of sorting keys in the reducer. It is possible to apply a so-called *Secondary Sort* that allows, in our application, to sort not just the keys, but also the values for the same key. Technically, this is implemented by replacing, for each (query, (neighbor, distance)) tuple that is emitted by a mapper, the key by a combined key consisting of the query and the distance. Keys are then sorted lexicographically first by query and then by distance. For assigning tuples to a Reduce method, however, only the query part of the key is taken into account. The reducer then only needs to read the first K values for each key, which then correspond to the K-Nearest Neighbor for that query.

It is worth mentioning that because one feature vector is placed in multiple hash tables, the same vector can be evaluated twice for the same query during processing. An alternative approach would be to have two MapReduce jobs for

¹This feature was introduced in the most recent version 0.20; before, it was only possible to emit directly from the Map function

query processing instead of one, which would eliminate this kind of redundancy. The first MapReduce job would create a union between buckets that need to be probed, and the second job would use the union as an input to similarity search. However, while this would possibly save redundant computations, it has the major drawback that the results from the first job need to be written to the HDFS before starting the second job. As initial experiments showed that overhead from multiple evaluations of the same feature vector has not been too large, we decided that it is better to probe slightly more data rather than to pay the additional IO cost incurred by using two Map Reduce jobs.

The approach can handle multiple queries at the same time in one MapReduce job. But it is not suitable for the cases when the number of queries becomes too large, as problem of KNN queries processing becomes the problem of set similarity joins [VCL10].

7.4 Experimental Evaluation

We have implemented all of the presented approaches in Java 1.6 and use the 64-bit variant of the Java VM to execute the code. The implementation is single threaded. The experiments are conducted on a dual CPU Intel Xeon E55302.4 GHz and 47.9 GB of main memory, running Microsoft Windows Server 2003 Enterprise x64 Edition (Service Pack 2).

Distributed setup: for distributed version of LSH we have used Hadoop version 0.20.2 installed on three virtual machines with Debian GNU/Linux 5.0 (Kernel version: 2.6.30.10.1) as operating system. Each of the virtual machines has been configured to have 200GB hard drive, 5 GB main memory and two processors. VMware Server version 2.0.2 was used for virtualization of all machines. The virtual machines were run on a single machine with Intel Xeon CPU E55302.4 GHz, 47.9 GB main memory, 4 TB of hard drive and Microsoft Windows Server 2008 R2 x64 as operating system. We used a single machine Hadoop installation on these virtual machines as described later on.

Image Dataset

To evaluate the above approaches on real-world data, we have obtained the **CoPhIR dataset** [BEF⁺09]. It consists of MPEG7 feature descriptors extracted from a large collection of images obtained from Flickr [FLI] image sharing portal. For each crawled image, the dataset contains MPEG7 feature vectors that are given in an XML based format together with a URL of the source photo. We have transformed the XML format in a convenient binary format before starting the experiments.

We use the following subsets of MPEG7 feature descriptors in the evaluation: color structure, scalable color, and edge histogram (cf., Section 2.1.1 for more information about the features used). Scalable color and color structure descriptors in CoPhIR are defined by 64 dimensional vectors, while an edge histogram descriptor is a 80 dimensional vector. We use these three different

feature representations for deeper insights on the performance of the algorithms under comparison.

LSH Setup

Locality Sensitive Hashing enables tradeoff between memory usage (index size) and the time needed to answer a query with a certain precision. Trading off index size against runtime is achieved by changing the number of hash tables used for indexing. The more hash tables are used, the less time is needed to answer a query with the same precision.

LSH is a parametric method with a common practice of tuning the parameters for each individual dataset. Having multiple parameters, yielding multi dimensional parameter space, usual practice is to fix all but one parameter and to vary that parameter until the optimum is found. This procedure is repeated until a global (or local) optimum approximation is found. To achieve the best results, we have performed parameter tuning of each approach independently.

We start LSH tuning by fixing the number of hash tables and then tuning the parameters to achieve required precision. Achieving a certain precision at fixed number of hash tables depends on the number of data points found in one LSH bucket. This number, in turn, depends on the number of hash functions per hash table as well as on the parameter W of the each of the function (see Equation 2.8). As W is a continuous variable it gives us more control over the bucket size, so we fix the number of hash functions per table and vary W to achieve a certain precision values.

Probing multiple buckets from the same hash table requires an additional parameter that describes the number of additional probes. We have experimented with different number of additional probes for each feature descriptor. It turns out that for color structure and scalable color descriptors Multi-Probe is not better than the original LSH and it achieves the best results with only two additional probes per table. However, for edge histogram descriptor Multi-Probe outperforms the original LSH, with the best performances at 30 additional probes per hash table. We have used the same number of probes also for Linked-LSH, Peek-Probing, and the combined approach.

Linked-LSH also introduced depth parameter which defines the depth of the recursive nearest neighbor traversal. Setting this parameter is easy as by intuition it has to have a low value. By experimenting we found out that best results are achieved when depth is two. For Linked-LSH we also need to determine the value of \tilde{K} , we do that by $\tilde{K} = c * K$, where c is determined experimentally for the best performance. We used the value of $c = 3$ when Linked-LSH was tested alone, and $c = 1.1$ when tested in the combined approach.

In the distributed setting each new hash table creates another copy of data and we may have only limited storage available, we need to tradeoff storage cost vs. execution time. Additionally, when only a fixed subset of the data should be accessed, a larger number of hash tables results in a large number of small sized buckets, which is not a good scenario for HDFS (it puts additional pressure on Hadoop's data node that manages all files). On one hand, we would like to

increase the number of hash tables and to decrease the probed data subset. On the other hand, we would like to use less storage space and a smaller number of files for storage and probing. Thus, as a general rule for distributed setting we suggest a smaller number of hash tables with larger bucket sizes, still set to satisfy the precision threshold.

7.4.1 LSH Enhancements Evaluation

To evaluate the proposed LSH enhancements we compare the following five approaches.

- **LSH:** This is the implementation of the original work on LSH presented in [GIM99].
- **Multi-Probe:** We have implemented the Multi-Probe algorithm presented in [LJW⁺07] and use it as the underlying LSH method for our approach. While this method outperforms LSH (in most cases), we still include the original LSH method for completeness.
- **Linked-LSH:** Represents the implementation of the enhancement of the LSH using the nearest neighbor links of the data points in the dataset, together with the recursive link traversal, as described in the Section 7.1.
- **Peek-Probing:** Implementation of the Peek-Probing strategy from Section 7.2, where LSH buckets are probed completely only if a certain amount of usefulness is indicated.
- **Combined:** This is the implementation of the Peek-Probing together with Linked-LSH, built on top of Multi-Probe LSH buckets selection.

Out of all available images in the dataset, we have randomly selected 100,000 images to index. Additionally, 10,000 images are randomly selected and used as query images. As we are interested in relative improvements we use 32 hash tables for all described LSH approaches. Preliminary experiments have shown that the best runtime, with 32 hash tables, is achieved when using 8 hash functions per table. Hence, in the following experiments, we use 8 hash function per table. We keep all of the data structures in main memory for all of the approaches in these experiments. As the non-deterministic nature of LSH can result in slight deviations of measurements obtained by the same parameter setup, we perform each experiment for each parameter setup 10 times and report average results.

Measured Values

As all LSH based approaches are by design approximate methods, i.e., a returned KNN result might or might not differ from the true K nearest neighbors. Although we tune the precision towards required value, it might still vary slightly. For this reason we measure the *precision* in addition to *runtime* and

inspected data portion. The precision is measured as the percentage of the returned approximate top-K results that are also found in the exactly computed (using a naive full scan approach) top-K results. The runtime was measured as the number of seconds (with millisecond resolution) needed to answer all of the 10,000 queries. The measured inspected data portion represents the percentage of the indexed 100,000 feature vectors for which the distance to the query data point was calculated while answering that query. Clearly, there is a correlation between the inspected data portion and the total runtime, as more distance calculations require more time. We report an average precision and average inspected data portion for 10,000 queries, while the reported runtime is the total time needed to answer all of 10000 queries.

Experimental Results

For each of the approaches we have performed measurements at three levels of precision: at 80%, 90%, and 95%. As described, the parameter W in Equation 2.8 is used to tune each of the methods towards a certain precision. As we are not able to strictly ensure the exact precision wanted, the precision is also measured and reported.

approach	prec. (%)	time (s)	insp. (%)
LSH	80.746	34.138	7.810
	90.384	61.504	13.659
	95.268	103.279	21.949
Multi-Probe	80.530	34.873	7.473
	90.493	69.012	14.496
	95.153	105.864	21.640
Linked-LSH	80.151	26.062	4.915
	90.237	50.773	10.118
	95.155	83.065	16.705
Peek-Probing	80.489	19.646	1.934
	90.245	31.558	3.208
	95.675	51.415	5.127
Combined	80.490	16.276	1.575
	90.354	25.490	2.617
	95.215	40.072	4.058

Table 7.1: Measurements for color structure descriptor

Table 7.1 contains the measurements for all approaches for color structure descriptor. As we can see, using Multi-Probe in the case of color structure descriptor does not result in an improvement over the original LSH. The runtime and inspected data portion for Multi-Probe and original LSH are almost the same, as we used small number (i.e., 2) of additional probes. Increasing the number probes made the results only worse for Multi-Probe in this case. We see that Linked-LSH provides a constant improvement in both runtime and inspected data portion over the baselines. The best improvement in runtime is

achieved for the precision at 80%, reducing the runtime by 23.65%, while the runtime improvements for 90% and 95% precision are 17.44% and 19.57% respectively. Due to the relation between runtime and inspected data portion, the improvements in the inspected portion are proportional to the runtime improvements. Even higher improvement in runtime and inspected data portion is achieved using Peek-Probing, which reduces runtime by 50.21% at 95% precision. Improvements using Peek-Probe are seen over all measured precisions, with runtime improvement of 42.45% at 80% precision and 48.68% at 90% precision. Combining Linked-LSH and Peek-Probe (i.e., Combined algorithm) yields the best results, with more than a factor of 2 improvement in runtime and more than a factor of 4 improvement in inspected data portion.

approach	prec. (%)	time (s)	insp. (%)
LSH	80.617	116.042	23.020
	90.287	183.178	35.243
	95.327	272.212	48.001
Multi-Probe	80.364	125.184	20.996
	90.046	174.275	29.282
	95.200	222.560	36.597
Linked-LSH	80.578	106.495	16.348
	90.135	160.724	25.703
	95.519	223.370	34.787
Peek-Probing	80.468	58.399	6.088
	90.180	70.624	7.760
	95.136	82.843	9.325
Combined	80.796	58.109	5.765
	90.301	70.695	7.459
	95.044	83.379	8.901

Table 7.2: Measurements for edge histogram descriptor

Measurements for the edge histogram descriptor are shown in Table 7.2. We see that Multi-Probe achieves a significant improvement in runtime over the original LSH method, at 95% precision, but is slightly worse at 80% precision. In the case of the edge histogram descriptor the benefit of using Linked-LSH is quite small at 80% and 90% precision, with an improvement of 8.22% and 7.77% respectively, and non-existent at 95% precision. However, using Peek-Probe results in a significant improvement in runtime, with 62.77% improvement at 95% precision. Improvement for Peek-Probing at 90% precision is 59.47% and at 80% precision is 49.67%. As there was no significant runtime improvement in using Linked-LSH there is no improvement in using the combined approach over only using Peek-Probing. The inspected data portion is still best for the combined approach, but values are close the Peek-Probing approach, showing once again that Linked-LSH has no impact for the edge histogram descriptor.

Table 7.3 shows measurements for the scalable color descriptor. We can see that relative improvements for Linked-LSH, Peek-Probing, and the combined

approach	prec. (%)	time (s)	insp. (%)
LSH	80.647	13.563	2.993
	90.563	25.562	5.741
	95.596	41.651	9.278
Multi-Probe	80.180	15.726	2.978
	90.318	27.254	5.631
	95.032	43.226	9.041
Linked-LSH	80.805	13.620	2.078
	90.849	22.418	4.063
	95.132	34.406	6.685
Peek-Probing	80.283	10.032	0.725
	90.977	14.948	1.264
	95.215	20.107	1.850
Combined	80.294	8.975	0.622
	90.650	12.451	1.029
	95.107	16.809	1.477

Table 7.3: Measurements for scalable color descriptor

approach are similar to the improvements for color structure descriptor. This is expected as both descriptors are based on the color distribution of the images. We can see that the combined approach performs best, for both runtime and inspected data portion. The best runtime improvement is achieved at 95% precision, reducing runtime for 59.64%.

Runtime measurement for all approaches and all descriptors at 95% precision are shown in Figure 7.4(a). We can see that searching for most similar image based on edge histogram descriptor takes a lot longer than based on the color descriptors. Although the edge histogram descriptor uses higher dimensional feature vectors (80 dimension) than for the color descriptors (64 dimensions), the main runtime difference comes from the fact that more data points are used in calculating the answer for edge histogram descriptor than for color descriptors, as shown in Figure 7.4(b), which shows inspected data portion for all approaches and all descriptors at 95% precision.

7.4.2 RankReduce Evaluation

We evaluate the RankReduce framework comparing it to the linear scan over all data, also implemented as a MapReduce job. As we did not have a real compute cluster at hand for running the experiments, we simulate the execution in a large cluster by running the mappers and reducers sequentially on our small machine. We measure run times of their executions and the number of mappers started for each query job. To avoid the possible bottleneck of a shared hard drive between virtual machines [LJL⁺09], we run each experiment on a single machine Hadoop installation with one map task allowed per task tracker. This results in sequential execution of map tasks so there is no concurrent access to a shared hard drive by multiple virtual machines.

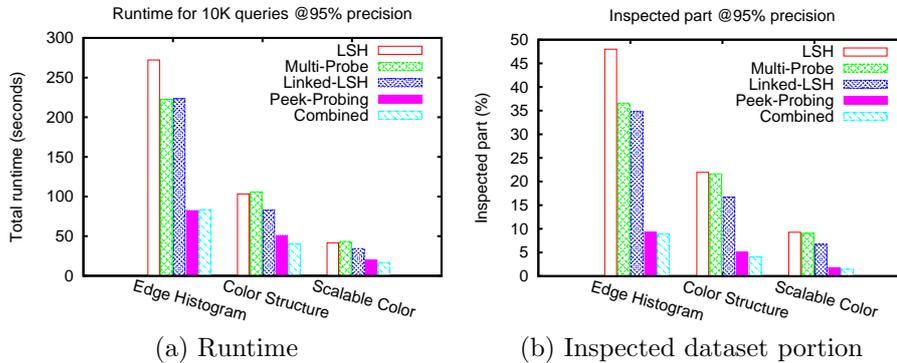


Figure 7.4: Measurements at 95% precision

Considering that the workload for the reducers is really small for both linear scan and LSH, we only evaluate Map execution times and the number of mappers run per query job. We measured the Map execution times for all jobs and found that they are approximately constant, with average value per mapper being 3.256 seconds and standard deviation of 1.702 seconds. Taking into account that each mapper has an approximately same data input size, defined by the HDFS’ block size, approximately constant mapper execution time is well expected.

We have evaluated performance of the approach using color structure feature vectors from the described CoPhIR dataset, and using synthetic dataset—randomly generated feature vectors.

Synthetic Dataset: For the synthetic dataset we used 32-dimensional randomly generated vectors. The synthetic dataset was built by first creating N independently generated vector instances drawn from the normal distribution $N(0, 1)$ (independently for each dimension). Subsequently, we created m near duplicates for each of the N vectors, leading to an overall dataset size of $m * N$ vectors. The rationale behind using the near duplicates is that we make sure that the KNN retrieval is meaningful at all. We set m to 10 in the experiments and adapt N to the desired dataset size depending on the experiment. We generated 50 queries by using the same procedure as the original vectors were generated.

Experimental Results

Because the execution time of the mappers is almost constant, the load of a query execution can be represented as a number of mappers per query. We measured the number of mappers per query and precision for 50 KNN queries, with $K=20$, for both datasets, with 2GB, 4GB, and 8GB of indexed data (~ 4000 , ~ 8000 , and ~ 16000 feature vectors for the real image dataset and ~ 8000 , ~ 16000 , and ~ 32000 feature vectors for the synthetic dataset, respectively). The number of mappers per query for image dataset is shown in Figure 7.5(a). And as we can see the number of mappers is more than four times smaller for LSH than for linear scan. The number of mappers per query for synthetic dataset is shown in Figure 7.5(b). Again, we see the large benefit of using LSH over linear scan.

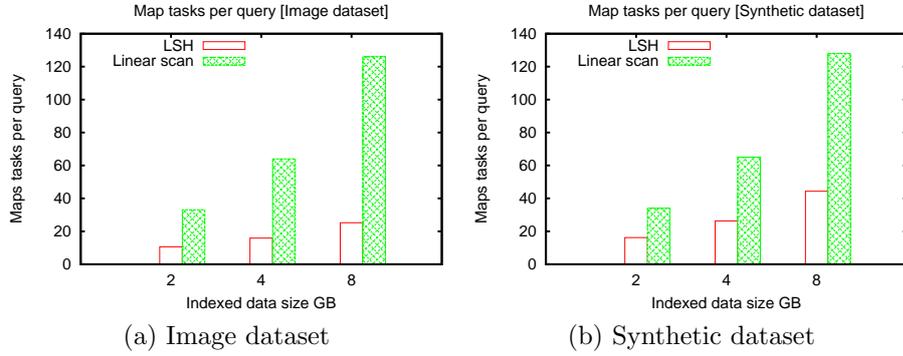


Figure 7.5: Map tasks per query

The precision, shown in Figure 7.6, for synthetic dataset is over 70% for 2GB and 4GB of indexed data, but drops down to 63.8% for 8GB. For real image data, the precision is almost constant, varying slightly around 86%.

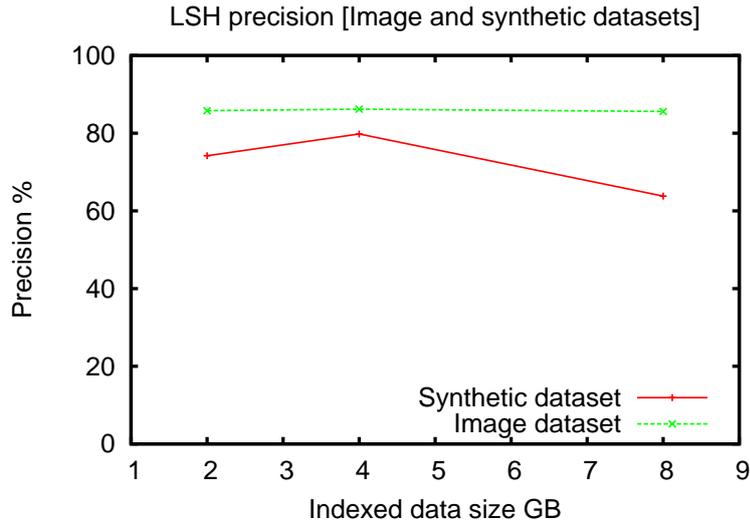


Figure 7.6: LSH precision on generated and picture data.

Chapter 8

PicasSound

PicasSound is a smartphone application based on the Picasso approach to recommend music for images. It is primarily developed to demonstrate the features of Picasso to a wider audience—anyone with a smartphone and a large enough music collection. PicasSound runs on iOS¹ and Android² operating systems, and hence, is available to a large portion of smartphone users. Having the application running on a smartphone makes an ideal showcase for Picasso as images and music are immediately available.

PicasSound enables users to upload images and receive the top ten songs as a music recommendation. The recommended songs are a subset of the songs found in the user’s phone. This way, the user is able to explore the recommendation by playing the songs directly within the application.

8.1 Design Rationale

We started off with the following goals in mind:

- The application should be able to recommend songs for all images contained in the phone.
- A user should be able to explore the recommendation by browsing the list of recommended songs.
- The application should be interactive, i.e, have response times below 1-2 seconds.

Ideally, the application should enable recommendation for multiple images, where images are displayed in a slide show manner after the recommendation process. For simplicity reasons, we decided to relax this goal and enable recommendation for one image at the time. This simplified the development while at the same time kept the essence of the approach.

¹<http://www.apple.com/ios/>

²<http://www.android.com/>

Beside the functionality-oriented goals we had the general goal for the application to be user friendly. This means that each operation is intuitively triggered by the basic interaction with the phone, together with a meaningful yet simple result representation. From the user friendliness perspective, the application also needs to be interactive, which puts an upper bound on the time needed to compute the recommendation.

Due to the limited available memory and the computing power of the phones, we made the first design decision to organize the application into a client/server architecture. With this, additional latency is introduced by the time needed to upload the images and to receive the response from the server. However, this also enables us to have high-end hardware at the server side, speeding up the recommendation process substantially.

The goal of having songs recommended from the set of songs contained in the phone introduced design challenges as the size of the songs and the limited bandwidth discarded the option to upload the songs to the server for their processing. Another option was to extract audio features, needed for the similarity computation, on the phone and upload only them. This however, would require a lot of processing on the users phone slowing down its performance and draining out the battery. Finally, we decided to use already extracted features from the one million songs dataset [BMEWL11] on the server, and use the names of the songs to find the match between the songs on the phone and the songs on the server. This approach comes at the price that we are able to recommend only songs out of the subset of songs contained on the phone, i.e., the songs found in both the phone and in the one million song dataset.

8.1.1 Implementation Details

On the client side—the application running on the phone—two values are stored: the signature of the music list and the application identifier. The initial values are both set to *null* when the application is installed. While starting the application the music list is read from the phone and the signature is calculated. If the signature is different than the previous one, the user is asked to synchronize the music list with the server.

The server exposes two functions to the clients using the HTTP protocol, one for the music synchronization and one for the recommendation. The music synchronization function is invoked with the list of songs contained on the phone and the application id as parameters. If the application id is null, the server recognizes this application installation as new, assigns a new id to it, and sends it back to the client as a return parameter. Thereafter, the matching between the songs on the server and the songs on the phone is made and the application profile is stored on the server.

The music recommendation function is invoked with the binary representation of the query image together with the application id sent to the server. Before issuing a request the image is scaled down to 640x480 pixels to decrease

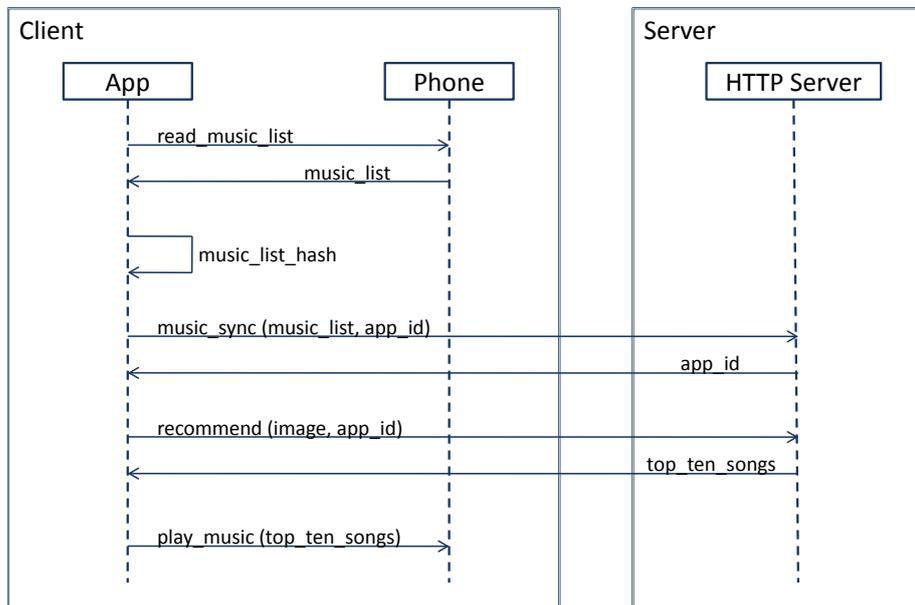


Figure 8.1: PicasSound interaction process

the time needed to send the request. The server retrieves the application profile based on the application id and makes a recommendation, sending the top ten songs as a response. The complete interaction process is illustrated in Figure 8.1.

8.2 Application Functionality

The basic functionality of the PicasSound application is organized in the following four steps:

- **Music synchronization:** is performed to identify the intersection between the songs on the user's phone and the songs contained in the index on the server. This way, only the songs from the user's phone are recommended in the following steps. The music synchronization step is performed at the user's request with the notification to the user that the names of the songs contained in the phone will be sent to the server. Once the step is completed, the user is notified about the intersection size between the songs.
- **Image selection:** a user selects an image that is to be uploaded to the server by either selecting the image from the existing image collection or by capturing a new image by the phone's camera. Selecting an image from the existing image collection is triggered by tapping twice on the free space of the application's background, while one tap triggers the *camera mode* used to capture a new image (cf., Figure 8.2(a)).

- **Soundtrack recommendation:** creates a request for the soundtrack recommendation on the server by uploading the selected/captured image. Once the recommendation process is done, the list of the recommended songs is received from the server and the application automatically changes to the recommendation exploration mode.
- **Recommendation exploration:** a user explores a list of recommended songs by playing them directly from the application. The application shows a ranked list of songs with the ability to jump between the played songs either incrementally or directly by scrolling to a specific song (cf., Figure 8.2(c)).

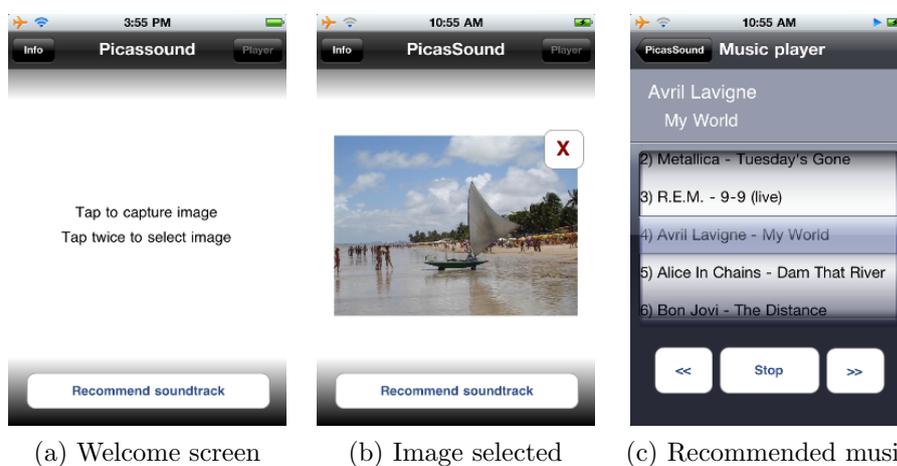


Figure 8.2: PicasSound screenshots

Figure 8.2 shows the screenshots of the PicasSound application in three different states. First screenshot (Figure 8.2(a)) shows the welcome screen, that is the screen that is first seen when the application is started. When an image is selected or captured it is displayed at the main screen as shown in the second screenshot (Figure 8.2(b)). Exploring the recommended songs is done by inspecting a ranked list of the recommendation and playing the songs from the list, as shown in the third screenshot (Figure 8.2(c)).

Chapter 9

Conclusion and Outlook

In this thesis we considered the problem of soundtrack recommendation for a given set of images to be presented in a slide show.

We proposed Picasso, an approach to solve the task of soundtrack recommendation based on information extracted from publicly available, contemporary movies. With Picasso we showed how similarity search in image and music domains can be used to create a connection between these domains using movies as a link. We created a reusable benchmark to evaluate the quality of soundtrack recommendation systems. We reported on detailed statistics of the collected relevance assessments giving a deeper insight into the problem. We evaluated and reported on the performance of Picasso and a competing approach using the proposed benchmark.

It is of importance that the produced recommendation are of high quality, but it is also very important to have an efficient recommendation process. For this reason, we have addressed two efficiency aspects that arise from the Picasso approach, however, both addressed problems are general and appear in multitude of situations: We have addressed the problem of processing top-K queries with set-defined selections and proposed a partitioned index which leverages the information from previous queries to organize the data in the index to minimize query processing latency. For the aspect of similarity search, we proposed two heuristic enhancements to the Locality Sensitive Hashing scheme. We further looked at the efficient distribution of the similarity search using the MapReduce framework.

So far, we addressed the problem of soundtrack recommendation for images without any knowledge about the user for which the recommendation is done. Including user preferences in the recommendation process appears to be promising to improve the quality of recommendations. This could be achieved directly using music preferences of the user or indirectly using preferences towards movies or movie genres. At its current state, Picasso uses all of the extracted information from movies with the same importance when recommending a soundtrack. It can be assumed that some movies contain more useful information than others

concerning the soundtrack selection. It might be beneficial to try and determine the quality of each movie concerning its soundtrack before including it in the training dataset. Picasso could also potentially benefit from the higher level features that could be extracted from images and music, such as face detection in images or tempo detection in music. Having a first benchmark in place, we would further address the problem of selecting a pair of songs for which a relevance assessment would bring the most benefit to the benchmark.

In the work on processing top-K queries with set-defined selections we only looked at the problem when one attribute is used for ranking. Obviously, the question arises what happens when multiple attributes are aggregated and the final score is used for ranking. Concerning Locality Sensitive Hashing enhancements, the major aspect remains unsolved whether we can mathematically show under which assumptions the proposed enhancements incur benefit in query processing times and in which cases it is better to use the basic LSH techniques.

Appendix A

Appendix

A.1 Benchmark Music Collection Sample

Classical

Andrea Bocelli	Besame Mucho
Andrea Bocelli	Con te Partiro
Frederic Chopin	Military Polonaise
Frederic Chopin	Revolutionary Etude
Johann Sebastian Bach	Oboe Concerto in D minor
Johann Sebastian Bach	Toccatina in D minor
Ludovico Einaudi	Nuvole Bianche
Ludovico Einaudi	Oltremare
Wolfgang Amadeus Mozart	symphony no. 40 1st
Wolfgang Amadeus Mozart	eine kleine nacht music
...	...

Table A.1: Sample of benchmark songs from genres category (part I)

Hip Hop and Rap

2Pac	Changes
2Pac	Keep Ya Head Up
Cypress Hill	Insane In The Brain
Cypress Hill	Superstar
Eminem	Not Afraid
Eminem	Without Me
Jay-Z	Empire State of Mind
Jay-Z	Young Forever
The Black Eyed Peas	I Gotta Feeling
The Black Eyed Peas	The Time Dirty Bit
...	...

Pop

ABBA	Dancing Queen
ABBA	Fernando
Lady Gaga	Bad Romance
Lady Gaga	Born This Way
Madonna	Die Another Day
Madonna	Like a prayer
Michael Jackson	Beat It
Michael Jackson	Billie Jean
Rihanna	Man Down
Rihanna	Only Girl
...	...

Rock

Bruce Springsteen	Dancing In The Dark
Bruce Springsteen	Fire
Foo Fighters	Learn To Fly
Foo Fighters	The Pretender
Queen	Don't Stop Me Now
Queen	Innuendo
Red Hot Chili Peppers	Otherside
Red Hot Chili Peppers	Scar Tissue
The Rolling Stones	Paint It Black
The Rolling Stones	Sympathy For The Devil
...	...

Table A.2: Sample of benchmark songs from genres category (part II)

Happy

Belle and Sebastian	Funny Little Frog
Belle and Sebastian	I Want The World To Stop
Jason Mraz	Im Yours
Jason Mraz	Lucky
Karen O	Hello Tomorrow
Karen O	Where the Wild Things Are
Never Shout Never	Trouble
Never Shout Never	What Is Love
Noah and the Whale	5 Years Time
Noah and the Whale	Life goes on
...	...

Peaceful

Above and Beyond	Alone Tonight
Above and Beyond	Oceanic
Mirah	Engine Heart
Mirah	The Garden
Never Shout Never	Trouble
Never Shout Never	What Is Love
Ryan Farish	Chasing The Sun
Ryan Farish	Pacific Wind
Zen Garden	Polymorfia
Zen Garden	moments behind glass
...	...

Positive

Brett Dennen	Ain't no reason
Brett Dennen	Comeback Kid
Chamillionaire	Good Morning
Chamillionaire	Ridin
Mando Diao	Dance With Somebody
Mando Diao	Gloria
Revolver	Helplessly Hoping
Revolver	This Boy
Train	Get to Me
Train	Hey, Soul Sister
...	...

Table A.3: Sample of benchmark songs from positive feelings category

Aggressive

Agonize	Circle Of Death
Agonize	I against me
Kreator	Enemy of God
Kreator	Violent Revolution
Metallica	Master Of Puppets
Metallica	Of Wolf And Man
Nonpoint	Alive and Kicking
Nonpoint	Bullet With A Name
Rammstein	Du Hast
Rammstein	Tier
...	...

Hate

Dark Fortress	Edge Of Night
Dark Fortress	Ylem
GG Allin	No Rules
GG Allin	You Hate Me and I Hate You
Ingested	Castigation and Rebirth
Ingested	Cremated Existence
Massemord	Obscura Symphonia
Massemord	Skogen Kaller Og Vi Svarer
X-Fusion	Dial D For Demons
X-Fusion	My Inner Storm Blows
...	...

Tragic

Leather Strip	Hate me
Leather Strip	Strap me down
Psyclon Nine	Parasitic
Psyclon Nine	Under the Judas Tree
Rufus Wainwright	Hallelujah
Rufus Wainwright	What would I ever do with a Rose
Solitary Experiments	Do you Feel
Solitary Experiments	Immortal
W.A.S.P.	Harder Faster
W.A.S.P.	Wild Child
...	...

Table A.4: Sample of songs from negative feelings category

A.2 Benchmark Queries

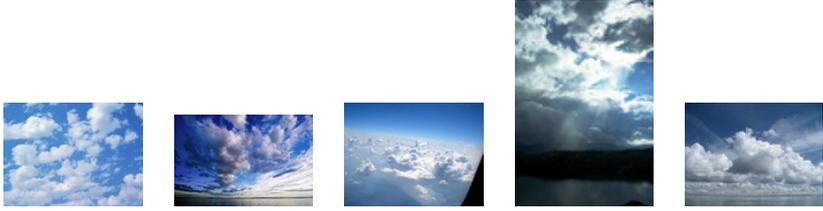
Architecture	
Aviation	
Cloudscape	
Conservation	
Cosplay	

Table A.5: Benchmark queries (part I)

Digiscoping					
Fashion					
Fine art					
Fire					
Food					
Glamour					
Landscape					
Miksang					

Table A.6: Benchmark queries (part II)

Nature					
Old-time					
Panorama					
Portrait					
Sports					
Still-life					
Street					
Underwater					

Table A.7: Benchmark queries (part III)

Vernacular					
War					
Wedding					
Wildlife					

Table A.8: Benchmark queries (part IV)

A.3 Feasibility Study Query Images

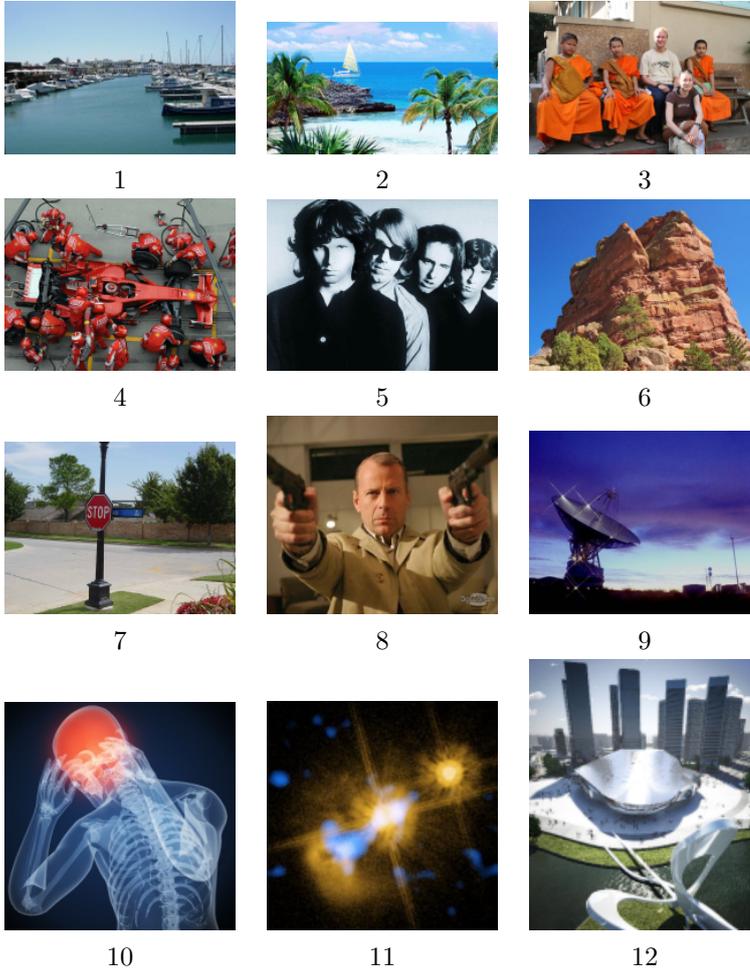


Table A.9: Query images used in the feasibility study

List of Figures

1.1	Soundtrack recommendation for images	2
1.2	Outline with the interplay between different aspects of the thesis	6
2.1	Aggregating semitone energy for chroma features	14
2.2	Similarity search: a) range query b) K-Nearest Neighbor query	16
2.3	Space partitioning: a) k-d tree b) R-tree	17
2.4	Graded relevance for image retrieval	22
2.5	The MapReduce framework	24
3.1	Detecting emotions in query images	28
3.2	Fuzzy mapping of feature values to feature buckets	30
3.3	Slide layouts for multiple images	35
3.4	Pairwise comparisons to determine the final ranking	36
4.1	Training dataset structure	43
4.2	Splitting of a long scene	44
4.3	Training dataset extraction process	45
4.4	Song to soundtrack part distance	47
4.5	Group recommendation strategies used for soundtrack recommendation task	51
4.6	Screenshot of our evaluation tool	53
5.1	Distribution of assessments per student	65
5.2	Concepts used to describe matching between images and music	67
5.3	Mapping emotion labels to two dimensional emotion space	70
5.4	Picasso preference precision for different dataset sizes	72
6.1	Top-K query with set-defined selection.	74
6.2	Characteristic performances of id-ordered index, score-ordered index, and the ideal combined index.	76

6.3	Partitioned index structure with sketches	80
6.4	Graph based data partitioning	81
6.5	Distribution of selection set sizes	85
6.6	Score-ordered index runtime	86
6.7	Id-ordered index runtime	87
6.8	Runtime combined index	88
6.9	Partitioned index runtime	88
6.10	Partitioned index runtime, ten partitions	89
6.11	Synthetic dataset runtime	90
6.12	Partitioned index runtime	91
6.13	Approximate querying, real-world dataset	92
6.14	Approximate querying, real-world dataset	93
6.15	Approximate querying, synthetic dataset	93
7.1	Linked-LSH index organization	97
7.2	Peek-Probing bucket organization	100
7.3	Mapping LSH buckets to HDFS	101
7.4	Measurements at 95% precision	109
7.5	Map tasks per query	110
7.6	LSH precision on generated and picture data.	110
8.1	PicasSound interaction process	113
8.2	PicasSound screenshots	114

List of Algorithms

6.1	Query processing with bit sets in main memory	79
6.2	Query processing with bit sets on disk	80
7.1	Query processing with Linked-LSH	98
7.2	Query processing with Peek-Probing	99

List of Tables

3.1	Mapping between video and music features	30
3.2	Contextual factors used for in car music recommendation	32
3.3	Tags used for context description	33
4.1	Multiple images grades	54
4.2	Single image grades	54
4.3	Scalability measurements	55
5.1	List of music genres	59
5.2	Feelings induced by music	60
5.3	List of query image themes	60
5.4	Agreement levels for student assessors	66
5.5	Agreement levels for Mechanical Turk workers	67
5.6	Statistics by question type	68
5.7	Statistics by image theme	69
5.8	Preference precision results	70
5.9	Weighted preference precision results	71
6.1	Runtime improvment (ms), bit sets in memory	90
6.2	Runtime improvment (ms), bitsets on disk	90
7.1	Measurements for color structure descriptor	106
7.2	Measurements for edge histogram descriptor	107
7.3	Measurements for scalable color descriptor	108
A.1	Sample of benchmark songs from genres category (part I)	117
A.2	Sample of benchmark songs from genres category (part II)	118
A.3	Sample of benchmark songs from positive feelings category	119
A.4	Sample of songs from negative feelings category	120
A.5	Benchmark queries (part I)	121

A.6	Benchmark queries (part II)	122
A.7	Benchmark queries (part III)	123
A.8	Benchmark queries (part IV)	124
A.9	Query images used in the feasibility study	125

Bibliography

- [ABK⁺07] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, pages 722–735, 2007.
- [ABY11] Omar Alonso and Ricardo A. Baeza-Yates. Design and implementation of relevance assessments using crowdsourcing. In *Advances in Information Retrieval - 33rd European Conference on IR Research, ECIR 2011, Dublin, Ireland, April 18-21, 2011. Proceedings*, pages 153–164, 2011.
- [ADCC11] Jaime Arguello, Fernando Diaz, Jamie Callan, and Ben Carterette. A methodology for evaluating aggregated search results. In *Advances in Information Retrieval - 33rd European Conference on IR Research, ECIR 2011, Dublin, Ireland, April 18-21, 2011. Proceedings*, pages 141–152, 2011.
- [AdKM01] Vo Ngoc Anh, Owen de Kretser, and Alistair Moffat. Vector-space ranking with effective early termination. In *SIGIR 2001: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, September 9-13, 2001, New Orleans, Louisiana, USA*, pages 35–42, 2001.
- [AI06] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*, pages 459–468, 2006.
- [All77] Jonathan Allen. Short term spectral analysis, synthesis, and modification by discrete fourier transform. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 25(3):235–238, 1977.
- [ANI] Animoto. <http://animoto.com/>.

- [AST10] Omar Alonso, Ralf Schenkel, and Martin Theobald. Crowdsourcing assessments for xml ranked retrieval. In *Advances in Information Retrieval, 32nd European Conference on IR Research, ECIR 2010, Milton Keynes, UK, March 28-31, 2010. Proceedings*, pages 602–606, 2010.
- [AYRC⁺09] Sihem Amer-Yahia, Senjuti Basu Roy, Ashish Chawlat, Gautam Das, and Cong Yu. Group recommendation: semantics and efficiency. *Proceedings of the VLDB Endowment*, 2(1):754–765, August 2009.
- [BBFV07] Henk M Blanken, Henk Ernst Blok, Ling Feng, and Arjen P Vries. *Multimedia retrieval*. Springer, 2007.
- [BCFM98] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 327–336, 1998.
- [BCG02] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Transactions on Database Systems (TODS)*, 27(2):153–187, 2002.
- [BCG05] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. Lsh forest: self-tuning indexes for similarity search. In *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, pages 651–660, 2005.
- [BEF⁺09] Paolo Bolettieri, Andrea Esuli, Fabrizio Falchi, Claudio Lucchese, Raffaele Perego, Tommaso Piccioli, and Fausto Rabitti. CoPhIR: a test collection for content-based image retrieval. *CoRR*, abs/0905.4627v2, 2009.
- [Ben90] Jon Louis Bentley. K-d trees for semidynamic point sets. In *Symposium on Computational Geometry*, pages 187–197, 1990.
- [Ber99] Dimitri P Bertsekas. *Nonlinear programming*. Athena Scientific, 1999.
- [BGRS99] Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings*, pages 217–235, 1999.
- [BKK96] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The x-tree : An index structure for high-dimensional data. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data*

- Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 28–39, 1996.
- [BKL⁺11] Linas Baltrunas, Marius Kaminskas, Bernd Ludwig, Omar Moling, Francesco Ricci, Aykan Aydin, Karl-Heinz Lücke, and Roland Schwaiger. Incarmusic: Context-aware music recommendations in a car. In *E-Commerce and Web Technologies - 12th International Conference, EC-Web 2011, Toulouse, France, August 30 - September 1, 2011. Proceedings*, pages 89–100, 2011.
- [BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990*, pages 322–331, 1990.
- [BMEWL11] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.
- [BPP⁺05] C.J.C. Burges, D. Plastina, J.C. Platt, E. Renshaw, and H.S. Malvar. Using audio fingerprinting for duplicate detection and thumbnail generation. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP'05). IEEE International Conference on*, volume 3, pages iii–9. IEEE, 2005.
- [BW06] Holger Bast and Ingmar Weber. Type less, find more: fast autocompletion search with a succinct index. In *SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, Washington, USA, August 6-11, 2006*, pages 364–371, 2006.
- [BW07] Holger Bast and Ingmar Weber. The completesearch engine: Interactive, efficient, and towards ir& db integration. In *CIDR 2007, Third Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*, pages 88–95, 2007.
- [BYJK⁺02] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Randomization and Approximation Techniques, 6th International Workshop, RANDOM 2002, Cambridge, MA, USA, September 13-15, 2002, Proceedings*, pages 1–10, 2002.
- [CB08] Ben Carterette and Paul N. Bennett. Evaluation measures for preference judgments. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in*

- Information Retrieval, SIGIR 2008, Singapore, July 20-24, 2008*, pages 685–686, 2008.
- [CBCD08] Ben Carterette, Paul N. Bennett, David Maxwell Chickering, and Susan T. Dumais. Here or there: preference judgments for relevance. In *Proceedings of the IR research, 30th European conference on Advances in information retrieval, ECIR'08*, pages 16–27, Berlin, Heidelberg, 2008. Springer-Verlag.
- [CCK⁺06] Jun-Cheng Chen, Wei-Ta Chu, Jin-Hau Kuo, Chung-Yi Weng, and Ja-Ling Wu. Tiling slideshow. In *Proceedings of the 14th ACM International Conference on Multimedia, Santa Barbara, CA, USA, October 23-27, 2006*, pages 25–34, 2006.
- [CD97] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and olap technology. *ACM SIGMOD Record*, 26(1):65–74, 1997.
- [Cel10a] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.
- [Cel10b] Òscar Celma. *Music Recommendation and Discovery - The Long Tail, Long Fail, and Long Play in the Digital Music Space*. Springer, 2010.
- [CP06] Ben Carterette and Desislava Petkova. Learning a ranking from pairwise preferences. In *SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, Washington, USA, August 6-11, 2006*, pages 629–630, 2006.
- [CPD⁺10] Marco Cristani, Anna Pesarin, Carlo Drioli, Vittorio Murino, Antonio Rodà, Michele Grapulin, and Nicu Sebe. Toward an automatically generated soundtrack from low-level cross-modal correlations for automotive scenarios. In *Proceedings of the international conference on Multimedia, MM '10*, pages 551–560. ACM, 2010.
- [CS08] Òscar Celma and Xavier Serra. Foafing the music: Bridging the semantic gap in music recommendation. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):250–256, 2008.
- [CSP01] Shih-Fu Chang, T. Sikora, and A. Purl. Overview of the mpeg-7 standard. *Circuits and Systems for Video Technology, IEEE Transactions on*, 11(6):688–695, June 2001.
- [CW04] Pei Cao and Zhe Wang. Efficient top-k query calculation in distributed networks. In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC*

- 2004, *St. John's, Newfoundland, Canada, July 25-28, 2004*, pages 206–215, 2004.
- [CWJC08] Chin-Han Chen, Ming-Fang Weng, Shyh-Kang Jeng, and Yung-Yu Chuang. Emotion-based music visualization using photos. In *Advances in Multimedia Modeling, 14th International Multimedia Modeling Conference, MMM 2008, Kyoto, Japan, January 9-11, 2008, Proceedings*, pages 358–368, 2008.
- [CXG10] Jiajian Chen, Jun Xiao, and Yuli Gao. islideshow: a content-aware slideshow system. In *Proceedings of the 2010 International Conference on Intelligent User Interfaces, February 7-10, 2010, Hong Kong, China*, pages 293–296, 2010.
- [CZJ⁺07] Rui Cai, Lei Zhang, Feng Jing, Wei Lai, and Wei-Ying Ma. Automated music video generation using web image resource. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 2, pages II-737 –II-740, 2007.
- [CZJM10] Carlo Curino, Yang Zhang, Evan P. C. Jones, and Samuel Madden. Schism: a workload-driven approach to database replication and partitioning. *Proceedings of the VLDB Endowment*, 3(1):48–57, 2010.
- [CZW⁺07] Rui Cai, Chao Zhang, Chong Wang, Lei Zhang, and Wei-Ying Ma. Musicsense: contextual music recommendation using emotional allocation modeling. In *Proceedings of the 15th International Conference on Multimedia 2007, Augsburg, Germany, September 24-29, 2007*, pages 553–556, 2007.
- [DBL] DBLP - The DBLP Computer Science Bibliography. <http://www.informatik.uni-trier.de/~ley/db/>.
- [DG04] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, pages 137–150, 2004.
- [DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*, pages 253–262, 2004.
- [DPC11] Peter Dunker, Phillip Popp, and Randall Cook. Content-aware auto-soundtracks for personal photo music slideshows. In *Proceedings of the 2011 IEEE International Conference on Multimedia and Expo, ICME 2011, 11-15 July, 2011, Barcelona, Catalonia, Spain*, pages 1–5, 2011.

- [DS11] Shuai Ding and Torsten Suel. Faster top-k document retrieval using block-max indexes. In *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011, Beijing, China, July 25-29, 2011*, pages 993–1002, 2011.
- [DWJ⁺08] Wei Dong, Zhe Wang, William Josephson, Moses Charikar, and Kai Li. Modeling lsh for performance tuning. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008, Napa Valley, California, USA, October 26-30, 2008*, pages 669–678, 2008.
- [ELBMG07] Douglas Eck, Paul Lamere, Thierry Bertin-Mahieux, and Stephen Green. Automatic generation of social tags for music recommendation. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, 2007.
- [EP07] D.P.W. Ellis and G.E. Poliner. Identifying ‘cover songs’ with chroma features and dynamic programming beat tracking. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 4, pages IV–1429 –IV–1432, 2007.
- [Fag99] Ronald Fagin. Combining fuzzy information from multiple systems. *Journal of computer and system sciences*, 58(1):83–99, 1999.
- [Fec60] Gustav Fechner. *Elemente der Psychophysik*. Breitkopf und Haertel, 1860.
- [Fei98] H.G. Feichtinger. *Gabor analysis and algorithms: Theory and applications*. Birkhauser, 1998.
- [FKM⁺06] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D. Sivakumar, and Erik Vee. Comparing partial rankings. *SIAM Journal on Discrete Mathematics*, 20(3):628–648, 2006.
- [FLI] Flickr. <http://www.flickr.com/>.
- [FLN03] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *Journal of computer and system sciences*, 66(4):614–656, 2003.
- [FM85] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.
- [FRE] Freebase. <http://www.freebase.com/>.

- [GBK00] Ulrich Gntzer, Wolf-Tilo Balke, and Werner Kiebling. Optimizing multi-feature queries for image databases. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 419–428, 2000.
- [GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *ACM SIGOPS Operating Systems Review*, 37(5):29–43, 2003.
- [GIM99] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 518–529, 1999.
- [GMUW08] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2 edition, 2008.
- [Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57, 1984.
- [Haa10] Alfred Haar. Zur theorie der orthogonalen funktionensysteme. *Mathematische Annalen*, 69(3):331–371, 1910.
- [HAD] Hadoop—open source implementation of MapReduce framework. <http://hadoop.apache.org>.
- [HBC09] Matthew D. Hoffman, David M. Blei, and Perry R. Cook. Easy as cba: A simple probabilistic model for tagging music. In *Proceedings of the 10th International Society for Music Information Retrieval Conference, ISMIR 2009, Kobe International Conference Center, Kobe, Japan, October 26-30, 2009*, pages 369–374, 2009.
- [HKP06] Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining: concepts and techniques*. Morgan kaufmann, 2006.
- [HLES06] J. S. Hare, P. H. Lewis, P. G. B. Enser, and C. J. Sandom. Mind the gap: Another look at the problem of the semantic gap in image retrieval. In *Proceedings of Multimedia Content Analysis, Management and Retrieval 2006 SPIE*, 2006.
- [How09] David C. Howell. *Statistical Methods fro Psychology*. Cengage Learning-Wadsworth, 2009.
- [IAE04] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Supporting top-k join queries in relational databases. *Proceedings of the VLDB Endowment*, 13(3):207–221, 2004.

- [IBS08] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top- k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*, 40(4), 2008.
- [ICL] ImageCLEF - Image Retrieval in CLEF. <http://www.imageclef.org/>.
- [IJL⁺09] Shadi Ibrahim, Hai Jin, Lu Lu, Li Qi, Song Wu, and Xuanhua Shi. Evaluating mapreduce on virtual machines: The hadoop case. In *Cloud Computing, First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings*, pages 519–528, 2009.
- [IKM07] Stratos Idreos, Martin L. Kersten, and Stefan Manegold. Database cracking. In *CIDR 2007, Third Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*, pages 68–78, 2007.
- [Jan08] Ryszard Janicki. Ranking with partial orders and pairwise comparisons. In *Rough Sets and Knowledge Technology, Third International Conference, RSKT 2008, Chengdu, China, May 17-19, 2008. Proceedings*, pages 442–451, 2008.
- [JB08] Alexis Joly and Olivier Buisson. A posteriori multi-probe locality sensitive hashing. In *Proceedings of the 16th International Conference on Multimedia 2008, Vancouver, British Columbia, Canada, October 26-31, 2008*, pages 209–218, 2008.
- [JK00] Kalervo Järvelin and Jaana Kekäläinen. Ir evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 41–48, 2000.
- [JK02] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.
- [KBHR12] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome A. Rolia. Skewtune: mitigating skew in mapreduce applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 25–36, 2012.
- [KFTRC12] Marius Kaminskas, Ignacio Fernández-Tobías, Francesco Ricci, and Iván Cantador. Knowledge-based music retrieval for places of interest. In *Proceedings of the second international ACM workshop on Music information retrieval with user-centered and multimodal strategies, MIRUM '12, Nara, Japan, October 29 - November 02, 2012*, pages 19–24, 2012.

- [KK99] George Karypis and Vipin Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999.
- [KL51] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.
- [KL70] BW Kernighan and S Lin. An efficient heuristic procedure for partitioning graphs. *Bell system technical journal*, 1970.
- [KMFC09] Gabriella Kazai, Natasa Milic-Frayling, and Jamie Costello. Towards methods for the collective gathering and quality control of relevance assessments. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009, Boston, MA, USA, July 19-23, 2009*, pages 452–459, 2009.
- [KN11] Marek Karpinski and Yakov Nekrich. Top-k color queries for document retrieval. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 401–411, 2011.
- [KR12] Marius Kaminskis and Francesco Ricci. Contextual music information retrieval and recommendation: State of the art and challenges. *Computer Science Review*, 2012.
- [KSH04] Yan Ke, Rahul Sukthankar, and Larry Huston. An efficient parts-based near-duplicate and sub-image retrieval system. In *Proceedings of the 12th ACM International Conference on Multimedia, New York, NY, USA, October 10-16, 2004*, pages 869–876, 2004.
- [LBC08] P. J. Lang, M. M. Bradley, and B. N. Cuthbert. International affective picture system (iaps): Affective ratings of pictures and instruction manual. Technical report, University of Florida, 2008.
- [LCIS05] Chengkai Li, Kevin Chen-Chuan Chang, Ihab F. Ilyas, and Sumin Song. Ranksql: Query algebra and optimization for relational top-k queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 131–142, 2005.
- [LCL04] Qin Lv, Moses Charikar, and Kai Li. Image similarity search with compact data structures. In *Proceedings of the 2004 ACM CIKM International Conference on Information and Knowledge Management, Washington, DC, USA, November 8-13, 2004*, pages 208–217, 2004.

- [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 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, pages 950–961, 2007.
- [LL07] Jae Sik Lee and Jin Chun Lee. Context awareness by case-based reasoning in a music recommendation system. In *Ubiquitous Computing Systems, 4th International Symposium, UCS 2007, Tokyo, Japan, November 25-28, 2007, Proceedings*, pages 45–58, 2007.
- [LO04] T. Li and M. Ogihara. Content-based music similarity search and emotion detection. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, volume 5, pages V–705. IEEE, 2004.
- [LS07] Cheng-Te Li and Man-Kwan Shan. Emotion-based impressionism slideshow with automatic music accompaniment. In *Proceedings of the 15th international conference on Multimedia, MULTIMEDIA '07*, pages 839–842, New York, NY, USA, 2007. ACM.
- [LST] Last FM. <http://www.last.fm>.
- [MAR] Marsyas - music/speech dataset. http://marsyas.info/download/data_sets.
- [MB11] F. Mokhtarian and M. Bober. *Curvature scale space representation: theory, applications, and MPEG-7 standardization*. Springer Publishing Company, Incorporated, 2011.
- [MBGZ13] Iris Miliaraki, Klaus Berberich, Rainer Gemulla, and Spyros Zoupanos. Mind the gap: Large-scale frequent sequence mining. In *ACM SIGMOD International Conference on Management of Data (SIGMOD 2013)*, New York, USA, June 2013. ACM.
- [MCM01] A. Martin, D. Charlet, and L. Mauuary. Robust speech/non-speech detection using lda applied to mfcc. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 1, pages 237–240 vol.1, 2001.
- [ME05] Michael I. Mandel and Dan Ellis. Song-level features and support vector machines for music classification. In *ISMIR 2005, 6th International Conference on Music Information Retrieval, London, UK, 11-15 September 2005, Proceedings*, pages 594–599, 2005.
- [MET] METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering. www.cs.umn.edu/~metis.
- [Mik03] L Mikhailov. Deriving priorities from fuzzy pairwise comparison judgements. *Fuzzy sets and systems*, 134(3):365–385, 2003.

- [MIR] MIREX - The Music Information Retrieval Evaluation eXchange. http://www.music-ir.org/mirex/wiki/MIREX_HOME.
- [Miz97] Stefano Mizzaro. Relevance: The whole history. *Journal of the American society for information science*, 48(9):810–832, 1997.
- [MKYH03] Philippe Mulhem, Mohan S. Kankanhalli, Ji Yi, and Hadi Hassan. Pivot vector space approach for audio-video mixing. *IEEE MultiMedia*, 10(2):28–40, 2003.
- [MOVY01] B. S. Manjunath, Jens-Rainer Ohm, Vinod V. Vasudevan, and Akio Yamada. Color and texture descriptors. *Circuits and Systems for Video Technology, IEEE Transactions on*, 11(6):703–715, 2001.
- [MRS08] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, 2008.
- [MTU] Amazon Mechanical Turk. <https://www.mturk.com/mturk/welcome>.
- [MTW05] Sebastian Michel, Peter Triantafillou, and Gerhard Weikum. Klee: A framework for distributed top-k query algorithms. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*, pages 637–648, 2005.
- [Mue07] Meinard Mueller. *Information Retrieval for Music and Motion*. Springer, 2007.
- [MUS] Music2ten. <http://music2ten.com>.
- [Mut02] S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 657–666, 2002.
- [MW09] Winter A. Mason and Duncan J. Watts. Financial incentives and the “performance of crowds”. In *Proceedings of the ACM SIGKDD Workshop on Human Computation, Paris, France, June 28, 2009*, pages 77–85, 2009.
- [NPS11] Vivek R. Narasayya, Hyunjung Park, and Manoj Syamala. Automatic workload driven index defragmentation. *Proceedings of the VLDB Endowment*, 4(12):1407–1409, 2011.
- [NRSM10] Alexandros Nanopoulos, Dimitrios Rafailidis, Panagiotis Symeonidis, and Yannis Manolopoulos. Musicbox: Personalized music recommendation based on cubic analysis of social tags. *IEEE Transactions on Audio, Speech & Language Processing*, 18(2):407–412, 2010.

- [NSK03] Meera Nayak, SH Srinivasan, and Mohan S Kankanhalli. Music synthesis for home videos: an analogy based approach. In *Information, Communications and Signal Processing, 2003 and Fourth Pacific Rim Conference on Multimedia. Proceedings of the 2003 Joint Conference of the Fourth International Conference on*, volume 3, pages 1556–1560. IEEE, 2003.
- [PIC] Picasa - Photo sharing portal. <https://picasaweb.google.com/>.
- [PT01] Lutz Prechelt and Rainer Typke. An interface for melody input. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 8(2):133–149, 2001.
- [PYFD04] Jia-Yu Pan, Hyung-Jeong Yang, Christos Faloutsos, and Pinar Duygulu. Automatic multimedia cross-modal correlation discovery. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, pages 653–658, 2004.
- [Ris01] Irina Rish. An empirical study of the naive bayes classifier. In *IJ-CAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.
- [RM06] Sasank Reddy and Jeff Mascia. Lifetrak: music in tune with your life. In *Proceedings of the 1st ACM international workshop on Human-centered multimedia*, pages 25–34. ACM, 2006.
- [Ror90] Mark E. Rorvig. The simple scalability of documents. *Journal of the American Society for Information Science*, 41(8):590–598, 1990.
- [Rus80] James A Russell. A circumplex model of affect. *Journal of personality and social psychology*, 39(6):1161, 1980.
- [Sam06] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures*. The Morgan Kaufmann Series in Computer Graphics, 2006.
- [SB10] Jose M. Saavedra and Benjamin Bustos. An improved histogram of edge local orientations for sketch-based image retrieval. In *Proceedings of the 32nd DAGM conference on Pattern recognition*, pages 432–441, Berlin, Heidelberg, 2010. Springer-Verlag.
- [SC78] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 26(1):43 – 49, February 1978.

- [SM11a] Aleksandar Stupar and Sebastian Michel. Picasso - to sing, you must close your eyes and draw. In *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011, Beijing, China, July 25-29, 2011*, pages 715–724, 2011.
- [SM11b] Aleksandar Stupar and Sebastian Michel. Picasso: automated soundtrack suggestion for multi-modal data. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management, CIKM 2011, Glasgow, United Kingdom, October 24-28, 2011*, pages 2589–2592, 2011.
- [SM12a] Aleksandar Stupar and Sebastian Michel. Being picky: processing top-k queries with set-defined selections. In *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*, pages 912–921, 2012.
- [SM12b] Aleksandar Stupar and Sebastian Michel. Enhancing locality sensitive hashing with peek probing and nearest neighbor links. In *Proceedings of the 15th International Workshop on the Web and Databases 2012, WebDB 2012, Scottsdale, AZ, USA, May 20, 2012*, pages 37–42, 2012.
- [SM13] Aleksandar Stupar and Sebastian Michel. Srbench—a benchmark for soundtrack recommendation systems. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 01, 2013*, 2013.
- [SMS10] Aleksandar Stupar, Sebastian Michel, and Ralf Schenkel. Rankreduce - processing K-Nearest Neighbor queries on top of MapReduce. In Roi Blanco, B. Barla Cambazoglu, and Claudio Luccese, editors, *LSDS-IR 2010 : Proceedings of the 8th Workshop on Large-Scale Distributed Systems for Information Retrieval*, volume 630 of *CEUR Workshop Proceedings*, pages 13–18, Geneva, Switzerland, 2010. CEUR-WS.org.
- [SND] Soundcloud. <https://soundcloud.com/>.
- [SOJN08] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y. Ng. Cheap and fast - but is it good? evaluating non-expert annotations for natural language tasks. In *2008 Conference on Empirical Methods in Natural Language Processing, EMNLP 2008, Proceedings of the Conference, 25-27 October 2008, Honolulu, Hawaii, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 254–263, 2008.

- [SPCK10] Mark Sanderson, Monica Lestari Paramita, Paul Clough, and Evangelos Kanoulas. Do user preferences and evaluation measures line up? In *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2010, Geneva, Switzerland, July 19-23, 2010*, pages 555–562, 2010.
- [SPH05] David A. Shamma, Bryan Pardo, and Kristian J. Hammond. Musicstory: a personalized music video creator. In *Proceedings of the 13th ACM International Conference on Multimedia, Singapore, November 6-11, 2005*, pages 563–566, 2005.
- [SS99] Zbigniew R. Struzik and Arno Siebes. The haar wavelet transform in the time series similarity paradigm. In *In proceedings of Principles of Data Mining and Knowledge Discovery*, pages 12–22, 1999.
- [SSL07] Aameek Singh, Mudhakar Srivatsa, and Ling Liu. Efficient and secure search of enterprise file systems. In *2007 IEEE International Conference on Web Services (ICWS 2007), July 9-13, 2007, Salt Lake City, Utah, USA*, pages 18–25, 2007.
- [SWSK10] K. Seyerlehner, G. Widmer, M. Schedl, and P. Knees. Automatic music tag classification based on block-level. *Proceedings of Sound and Music Computing 2010*, 2010.
- [TC02] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293 – 302, July 2002.
- [TdHdN⁺05] Rainer Typke, Marc den Hoed, Justin de Nooijer, Frans Wiering, and Remco C. Veltkamp. A ground truth for half a million musical incipits. *Journal of Digital Information Management*, 3(1):34–38, 2005.
- [TH06] Paul Thomas and David Hawking. Evaluation by comparing result sets in context. In *Proceedings of the 2006 ACM CIKM International Conference on Information and Knowledge Management, Arlington, Virginia, USA, November 6-11, 2006*, pages 94–101, 2006.
- [Thu27] Louis Thurstone. A law of comparative judgments. *Psychological Review*, 34(4):273–286, 1927.
- [TRE] TREC - Text REtrieval Conference. <http://trec.nist.gov/>.
- [TVI] TRECVID - TREC Video Retrieval Evaluation. <http://trecvid.nist.gov/>.

- [TVW06] Rainer Typke, Remco C. Veltkamp, and Frans Wiering. A measure for evaluating retrieval techniques based on partially ordered ground truth lists. In *Proceedings of the 2006 IEEE International Conference on Multimedia and Expo, ICME 2006, July 9-12 2006, Toronto, Ontario, Canada*, pages 1793–1796, 2006.
- [TWS04] Martin Theobald, Gerhard Weikum, and Ralf Schenkel. Top-k query evaluation with probabilistic guarantees. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pages 648–659, 2004.
- [Tza09] George Tzanetakis. Music analysis, retrieval and synthesis of audio signals marsyas. In *Proceedings of the 17th International Conference on Multimedia 2009, Vancouver, British Columbia, Canada, October 19-24, 2009*, pages 931–932, 2009.
- [UMML10] Julián Urbano, Mónica Marrero, Diego Martín, and Juan Lloréns. Improving the generation of ground truths based on partially ordered lists. In *Proceedings of the 11th International Society for Music Information Retrieval Conference, ISMIR 2010, Utrecht, Netherlands, August 9-13, 2010*, pages 285–290, 2010.
- [VCL10] Rares Vernica, Michael J Carey, and Chen Li. Efficient parallel set-similarity joins using mapreduce. In *Proceedings of the 2010 international conference on Management of data*, pages 495–506. ACM, 2010.
- [Voo98] Ellen M. Voorhees. Variations in relevance judgments and the measurement of retrieval effectiveness. In *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia*, pages 315–323, 1998.
- [Voo07] Ellen M. Voorhees. Trec: Continuing information retrieval’s tradition of experimentation. *Communications of the ACM*, 50(11):51–54, 2007.
- [WBEM⁺10] Minji Wu, Laure Berti-Equille, Amélie Marian, Cecilia M. Procopiuc, and Divesh Srivastava. Processing top-k join queries. *Proceedings of the VLDB Endowment*, 3(1):860–870, 2010.
- [WLLM06] Bin Wang, Zhiwei Li, Mingjing Li, and Wei-Ying Ma. Large-scale duplicate detection for web image search. In *Proceedings of the 2006 IEEE International Conference on Multimedia and Expo, ICME 2006, July 9-12 2006, Toronto, Ontario, Canada*, pages 353–356, 2006.

- [WMB⁺07] Xiaodan Wang, Tanu Malik, Randal C. Burns, Stratos Papadomanolakis, and Anastassia Ailamaki. A workload-driven unit of cache replacement for mid-tier database caching. In *Advances in Databases: Concepts, Systems and Applications, 12th International Conference on Database Systems for Advanced Applications, DASFAA 2007, Bangkok, Thailand, April 9-12, 2007, Proceedings*, pages 374–385, 2007.
- [Woo09] Mark D. Wood. Matching songs to events in image collections. In *Proceedings of the 3rd IEEE International Conference on Semantic Computing (ICSC 2009), 14-16 September 2009, Berkeley, CA, USA*, pages 95–102, 2009.
- [WSB98] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 194–205, 1998.
- [XH08] Dong Xin and Jiawei Han. P-cube: Answering preference queries in multi-dimensional space. In *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 1092–1100, 2008.
- [XHCL06] Dong Xin, Jiawei Han, Hong Cheng, and Xiaolei Li. Answering top-k queries with multi-dimensional selections: The ranking cube approach. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, pages 463–475, 2006.
- [XJL08] Songhua Xu, Tao Jin, and Francis Chi-Moon Lau. Automatic generation of music slide show using personal photos. In *Tenth IEEE International Symposium on Multimedia (ISM2008), December 15-17, 2008, Berkeley, California, USA*, pages 214–219, 2008.
- [YT] Youtube. <http://www.youtube.com>.
- [ZCW12] Xiaofei Zhang, Lei Chen, and Min Wang. Efficient multi-way theta-join processing using mapreduce. *PVLDB*, 5(11):1184–1195, 2012.
- [ZG02] Rong Zhao and William I. Grosky. Narrowing the semantic gap - improved text-based web document retrieval using visual features. *IEEE Transactions on Multimedia*, 4(2):189–200, 2002.

Index

- Amazon's Mechanical Turk, 62
- Android, 109
- Case-based reasoning, 29
- Chroma features, 12
- Color-based features, 9
- Combined index, 75
- Contextual factors, 30
- CoPhIR dataset, 101
- Curse of dimensionality, 17
- Direct IO, 82
- Dynamic time warping, 45
- Expectation-maximization, 97
- F measure, 21
- Gold standard, 62
- Graph partitioning, 79
- Group recommendation, 49
- Hadoop, 21, 98
- Hierarchical clustering, 48
- Id-ordered index, 73
- Image themes, 58
- ImageCLEF, 55
- InputFormat, 100
- International affective picture system, 67
- iOS, 109
- JNI (Java Native Interface), 81
- K-d tree, 14
- K-means, 97
- K-nearest neighbor, 13, 46, 93, 100
- KMV (k-Minimum Values) sketch, 77, 82
- LSH forest, 18
- MapReduce, 5, 93
- Marsyas, 41, 45
- Mel frequency cepstral coefficients, 11
- METIS software, 80
- Million song dataset, 82, 110
- MIREX, 55
- Mixed media graph, 26
- MPEG-7 standard, 9
- Multi-probe LSH, 18, 98, 102
- Music genres, 57
- Music/speech classification, 41
- MusicSense, 31
- Pablo Picasso, 39
- Pairwise comparisons, 34
- Places of interest, 31
- Pooling, 56
- Precision, 20
- Preference precision, 61, 67
- R-tree, 15
- Range query, 14
- Ranked lists aggregation, 47
- Recall, 20
- Relevance, 19
- Score-ordered index, 73
- Secondary sort, 100
- Set-defined selection, 71
- Shape-based features, 11
- Spearman's footrule distance, 48
- Standard score (z-score), 44
- Success probability, 80
- Texture-based features, 10
- Tiling slide show, 33

TREC, 55

TRECVID, 55

Triangle inequality, 94

User study, 50

Weighted preference precision, 62, 67

X-tree, 16

Z-score, 44