Doctoral Dissertation
Academic Year 2016

A Multicontext-adaptive Query Creation and Search System for Large-scale Image and Video Data

Graduate School of Media and Governance
Keio University

Nguyen Thi Ngoc Diep

Abstract of Doctoral Disseration

Academic Year 2016

A Multicontext-adaptive Query Creation and Search System for Large-scale Image and Video Data

by

Nguyen Thi Ngoc Diep

Abstract

This thesis addresses the dynamic adaptability of a multimedia search system to the contexts of its users and proposes a new query creation and search system for large-scale image and video data. The contexts of a user in this system are defined as three types of preferences: content, intention, and response time. A content preference refers to the low-level or semantic representations of the data that a user is interested in. An intention preference refers to how the content should be regarded as relevant. And a response time preference refers to the ability to control a reasonable wait time.

The important feature of the proposed system is the integration of context-based query creation functions with high-performance search algorithms into a unified search system. This system adapts the inverted list data structure to construct a disk-resident database model for large-scale data in high-dimensional feature space. The database model has an intrinsic property, the orthogonality of indexes, which facilitates the functionality of the query creation and search system. The query creation process consists of two main operations: (1) context-based subspace selection and (2) subspace manipulation that includes several unary and binary functions to alter and combine subspaces in order to reflect content preferences of users. The pruning search mechanism consists of three search algorithms that are designed to adapt to different types of intention preferences of users. Three search algorithms have a same processing logic: (1) prioritizing feature indices for searching based on the input content

preferences, (2) initializing starting points of searching based on the input intention preferences, (3) iteratively finding relevant candidates, and (4) ranking candidates by actual relevance scores and returning top results to users.

This thesis contributes to the multimedia retrieval research field a solution to the tradeoff between functionality and usability of a search system. With the query creation method, the proposed system provides users with a flexible means to express their varying contextual preferences. Moreover, with the pruning search mechanism, the system maintains a comparative high search performance to other conventional search techniques without disrupting the feature space by dimension reduction or weakening retrieval capabilities by static indexing.

Two application systems has been implemented to demonstrate the effectiveness and applicability of the proposed system: a context-dependent image search system with dynamic query creation and a frame-wise video navigation system. Many quantitative experiments were intensively studied using these systems. The experimental results have shown a high usability of query creation functions to reflect imaginations of users into queries and the advantages of the search algorithms to obtain high performances comparative to other conventional search techniques.

Keywords: Multimedia Retrieval, Subspace Selection, Query Creation, Large-scale Data, Video Navigation.

To my sister.

Acknowledgments

It is the end of my apprenticeship in research. During the training period, I owe great debts of gratitude to my advisors, colleagues, families, and friends. There would be not enough words and space to describe everything I have received from those great minds and warm-hearted people, who have been encouraging me, cheering me, looking after me, watching my back, and keeping their faiths in me. Yet I feel obliged to express my deep thanks and appreciation to them.

Firstly, I would like to express my sincere gratitude to my supervisor and advisors at Keio University. I would like to thank Professor Yasushi Kiyoki for his continuous support of my Ph.D. study and related researches, for his patience, motivation, and immense knowledge. Besides his wise guidance that helped me in all the time of my research, his teaching in the art of crafting human knowledge with a lot of care and respect has always been inspiring and unique.

I would like to show my special thanks to Professor Yoshiyasu Takefuji for his encouragements and invaluable discussions that have motivated me to trust my intuition and work passionately on what I love. His attitude of absolute freedom in research is always admirable and influential.

I would like to thank Professor Rodney D. Van Meter for his precious comments and suggestions in completing my dissertation. His demands of making careful writing documents and his requests for theories meeting with experiments have been priceless lessons.

I would like to acknowledge my deep appreciation to Professor Hideyuki Tokuda for his insightful comments and guidance. He has taught me to look beyond the theoretical side of research, to aim for the original contributions to the society in

order to demonstrate the spirit of study and research practices of Keio University.

I owe my deepest gratitude to Professor Kuniaki Mukai. It has been an honor and a gift to be able to discuss my ideas and share my thoughts with him. I appreciate all his contributions of time, ideas, and comments to make my Ph.D. experience productive and stimulating. The joy and enthusiasm he has was invaluably motivational for me, especially during tough times in the Ph.D. study.

I gratefully acknowledge the funding sources that support my Ph.D. work. I thank the Japanese Government scholarship, the MEXT scholarship that gave me a great chance to continue my professional study in Japan for five years. I appreciate the Faculty of Information Science at UET, Vietnam National University, Hanoi for promoting my study at Keio University. I also thank the Taikichiro Mori Memorial Research Fund and the Keio University Doctorate Student Grant-in-Aid Program for the financial supports in every academic year.

And I am heartily thankful to Dr. Sasaki, Dr. Uraki, Ms. Itabashi, Ms. Yamamoto, Ms. Saita, and all members in Multimedia Database Laboratory (MDBL) who have continuously encouraged me and given me all necessary supports to complete my study. Besides their valuable comments and advices, I have enjoyed very much the daily discussions with them and appreciated greatly their patience and care.

I am particularly grateful to Mr. Kazuyuki Hirobe and his wife, Mrs. Mieko Hirobe for their kindness and hospitality. Mr. Hirobe has been like a dear father looking after me. Without his persistent support, generous care, and beneficial advices, it would be too difficult for me to focus on what is important.

Last but not least, I would like to thank my beloved families who always stand by me and encourage me with their best wishes. There are no better gifts for me than their unconditional love and beliefs in me. A love, even thousand miles away, can always matter and be no less powerful.

And now, shall the actual journey begin.

February 07, 2017

Nguyen Thi Ngoc Diep

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

> The best way of finding out the difficulties of doing something is to try to do it.
>
> *Vision, David Marr*

1.1 Research Motivation

1.1.1 Emergence of Large-scale Image and Video Data

Between 40,000 and 10,000 years ago, Early Human started to express images of themselves and animals on cave walls [1]. Human is not the only species that can "see" the outside world but we are the only one that can create a visual space out of our perception, memory, and imagination. Ever since that very first time of drawing, there have been many evolutionary moments that we invented techniques and devices to promote this rich and powerful space for ourselves: from drawing, painting to photography, from still pictures to moving pictures, from analog to digital imaging. Nowadays we no longer depend on only our hands to capture what we see but instead we have camera devices to record the world either at anytime we want or even all the time such in case of surveillance cameras.

At 19th century and early 20th century, the most productive painters could create

about one to two thousands artworks in their lifetime (e.g., Pierre Auguste Renoir (1702 artworks), Oscar-Claude Monet (1284 artworks), Vincent Willem van Gogh (913 artworks))[1]. Those are outnumbered in many large order of magnitude by the aid of cameras and recording devices. According to a recent technical report of Mary Meeker, '2016 Internet Trends' [2, p. 90], about 3.2 billion photos per day are shared on social media services including Facebook, Instagram, Messenger, and WhatsApp. Among them, about 2.0 billion photos per day at only Facebook. As of May 7, 2015, Flickr, one of image hosting and video hosting websites, announced there are over 10 billion images on its server[2] and more than 3.5 million new images uploaded daily[3].

This emergence of recording and communicating technologies has brought about plentiful of data at many large scales and at the same time, challenged us to make use of them. Undoubtedly, we use these resources intensively. A brief statistic at SimilarWeb[4] describes how often we visit websites looking for information as shown in Table 1.1.

Table 1.1: Monthly traffic overview of some selected websites during May - October 2016 (data from *www.similarweb.com*).

Website	**Total visits (millions)**	**Avg. Visit Duration**
google.com	31,600	00:08:29
images.google.com	49.9	00:02:01
flickr.com	143.3	00:05:02
youtube.com	22,700	00:19:02
tineye.com	11	00:04:03
pinterest.com	970.8	00:05:28
instagram.com	1.8	00:05:58
yandex.com	19.7	00:06:48
tumblr.com	971.6	00:07:39
dailymotion.com	306.7	00:04:55
netflix.com	1,500	00:09:15

While to end-users, search is the most basic and quickest way to find information, at the core of a search engine, it is the indexing and search algorithms that work clev-

[1]Source: The Athenaeum, http://www.the-athenaeum.org/art/counts.php?s=cd&m=a
[2]Flickr blog: http://blog.flickr.net/2015/05/07/flickr-unified-search/
[3]Source: The Verge, March 2013
[4]https://www.similarweb.com

erly to meet that information need. Large-scale multimedia data challenge a search engine in several ways: organizing the contents of unstructured data; indexing large amount of records into databases; interpreting and adapting to changing contexts of users at query time; and quickly returning relevant information from the databases to users.

1.1.2 Contextual Aspects in Searching for Information

The *information need* of users is the topic has been widely studied in information retrieval field [3, 4, 5]. While in principle text retrieval including document retrieval or Web page retrieval and multimedia retrieval share some principal concepts and methodology, in practice multimedia search seems to be more complex and exploratory [5]. Spink et al. [6] stated that "multimedia searching appears to require greater interactivity between users and the search engine." Comparing to the general Web page search, multimedia retrieval shows "a significant increase in the number of query terms, search session length, query reformulations, and number of search results clicks" [5].

Those differences can be explained by addressing the represented contents in a full-text document comparing to a multimedia datum. The contents of a document is often represented by the set of its words, whereas the contents of an image, for example, are often more unstructured and can vary depending on the low-level features are chosen to represent it such as color, shape or texture features. This gap between a low-level features and high-level semantics is often known as a *semantic gap*. Besides the semantic gap, recent researches in multimedia retrieval have also realized another gap, the *intention gap* between search intent of a user and the query at query time [7, 8]. However, instead of defining explicitly what can be an *intention*, the existing methods often suggests to use the user's feedbacks (i.e., the evaluation of a result is relevant or irrelevant) as an implicit interpretation of the search intent.

In this thesis, three distinctive contextual aspects comprised to form a set of preferences of a user at query time are defined and treated: content, intention, and response time.

Content of interest: This aspect refers to the low-level features of an image or video, which include color, shape, texture.

Intention of search: This aspect refers to how the content will be regarded as relevant. Intention preferences are classified into three categories: "dominant", "similar", and "exact". A "dominant" intention preference signifies that the strong characterized features should be considered as more important therefore a result will be counted as a match if it contains these features with high values. A "similar" intention preference refers indicates a desire to find similar data to the input datum given a preferred content. An "exact" intention preference suggests a search for only exactly matched data given the input datum and a preferred content.

Response time: This aspect refers to the ability of users to control the response time of a search engine. Currently, most search systems run to completion then return results to users. However, there are situations when the users want to choose a reasonable running time and want to get results by this time limit. This demand is supposedly raised when the dataset is large and the response time may be longer than an affordable wait time of a user.

Different persons, or the same person under different circumstances, may have different interest under one query. For a same input image in content-based image search, for example, one person may be more interested in the colors while another may be more interested in the shapes of objects in the input. Likewise, another person may wish to search for images that contain the vivid red colors as those are in the input image ignoring other light red colors or other colors. Only those returned images that match the user's interest and intention are evaluated as "relevant". This state of affairs challenges a search system to be able to treat those preferences and return reasonable results.

1.1.3 The Curse of High Dimensionality

Multimedia data especially image and video data are unstructured data and their contents are often represented by many high-dimensional features. An image can be represented by a set of hundreds or thousands of colors with their distributions in the image, or a set of hundred shape descriptors, or a set of words that describes objects are in the image. A video can be represented as a sequence of continuous frames, in which each frame is treated as an image. As a consequence, the dimensionality of the metadata describing the contents of the video can be in much higher. From a computation viewpoint, a higher dimensionality means a "curse" -the well-known "curse of dimensionality" in literature.

In high dimensional space, "the concept of proximity, distance or nearest neighbor may not even be qualitatively meaningful" [9]. Beyer et. al. [10] stated that under some assumption on the data distribution, the distances between the nearest and farthest neighbors to a given target in high dimensional space are almost the same for a wide variety of distance functions. Moreover, Weber et. al. [11] observed the rapid performance degradation of similarity indexing structures in high-dimensional spaces, as they exhibit linear complexity.

The straightforward and common solution to tackle this problem is to reduce the number of dimensions (or features) that used to represent the multimedia data such as feature selection methods [12] or dimension reduction methods [13]. However, reducing the number of dimensions causes reduced "subtleness" expressed in the original data, which is, most of the time, what users are looking for.

It is observed that despite the high dimensionality of data, the number of dimensions with respect to a content preference can be relatively small. For example, even if we use thousands of colors to represent an image, it is more likely that we want to use a part of them (some particular colors) at query time. For that reason, this thesis suggests to manipulate the dimensionality of data in a context-adaptive way.

1.2 Research Proposal and Objectives

The study in this thesis aims to answer the question "how to dynamically reflect contextual preferences into queries, yet still have high-performance search capabilities on large scale data with high-dimensional feature space?"

The thesis features the dynamic adaptability of a multimedia search engine and proposes a new multicontext-adaptive query creation and search system for large-scale image and video data. By realizing the importance of high-dimensional representations in describing delicate contents of multimedia but also acknowledging the challenges it may cause for a search algorithm on large-scale data, the proposed system adapts the inverted list data structure to construct a disk-resident database model and uses it to enable a dynamic pruning search mechanism for quickly finding relevant candidates.

The indexing method is designed following three principles:

- features are indexed independently to each other as inverted lists,
- each inverted list is sorted by a descending order of values,
- all indices and metadata are written to binary files on disk by data blocks so that each can be accessed by random access.

The proposed search mechanism works adaptively to the preferences input by users at query time. It reflects users' content preferences by selecting an appropriate subset of inverted lists as a subspace for search process, and then adapts to the intention preferences by prioritizing indexes to search and initializing starting points to search for candidates. The searching for high-possibility relevant candidates is based on a heuristic strategy to maximize the impact of important content and the relevance of content measured by some similarity distance functions.

In this thesis, two application systems, context-dependent image search system and frame-wise video navigation system, are implemented using the proposed query creation and search system in order to demonstrate its feasibility and effectiveness via several intensive experimental studies.

1.3 Scope and Limitations

1.3.1 Scope of Context

The scope of context is limited in this thesis. Unlike *context* that can be defined in ubiquitous computing as information including *location, identity, activity, time* [14], the definition of *context* in this thesis is limited to contain only information about content, intention, and response time preferences of users at query time.

Different from the "context-aware" computing, in which an application system infers automatically the user contexts based on data from environmental sensors or wearable devices, the "context-adaptive" computing that defined in this thesis assumes that the contexts are ***explicitly stated*** by users as input at query time.

1.3.2 Scope of Content

1.3.2.1 Feature Representations of Image and Video Data

Features can be divided into two classes: low-level and high-level features. The low-level features can often be extracted automatically whereas the extraction of high-level features usually requires human assistance to create metadata such as captions or keywords.

Contents used in this thesis refer to low-level features of image or video data which are feature vectors of real numbers. In other words, the contents of image and video data are represented by numerical vectors with each dimension represents a feature of their contents. The proposed system currently does not deal with symbolic representations such as names of objects in an image/video (as known as "bag-of-words"), or other kinds of features.

Most of the current researches in content-based access and manipulation of visual data have focused on using low-level features such as texture, shape, and color. Examples of features and their descriptors are shown in Table 1.2.

- **Texture** features are analyzed based on structural, statistical, spectral, stochastic model-based, morphology-based, or multiresolution techniques.

- **Shape** features are analyzed by various boundary-based (e.g., chain codes, geometric, and Fourier descriptors), or region-based (e.g., area, roundness) shape models.

- **Color** features are frequently represented by computing the average color, the dominant color, and the global/local histograms. Notably, color features are extensively used because of their invariance with respect to image scaling and rotation.

Table 1.2: General low-level features and their descriptors used to represent image and video data (from table 3, section 6.5 [15]).

Feature	Descriptor
Texture	Contrast, coarseness, edge density and direction, Markov model, co-occurrence matrix, DCT coefficients, wavelet coefficients, Wold coefficients
Shape	Geometrical descriptors (area, perimeter, etc.), Fourier descriptors, chain code
Color	Color histogram, color moments

Certainly using one low-level feature in many cases may not be sufficient to discriminate between several objects. Therefore, combinations of two or several low-level features are frequently used to improve significantly data access and manipulation by content-based visual information. The concrete features used in the implemented systems will be described in section 7.3.

1.3.2.2 Processing of Content

Moreover, the content is not processed in a device-independent way. The images or videos can be acquired from many sources and their low-level data (in pixels) can depend on the environmental conditions that they are captured and sensors of camera devices (e.g., color constancy problem). However, they will be processed to produce metadata in a uniform way.

1.3.3 Scope of Intention

Although, it is very often that the user uses a search engine to explore the data space rather than has already had a clear intention in mind [5]. The prominent assumption of the proposed system is that every preference that a user might have must be explicitly expressed as input at each query time.

The scope of intention is limited by the definition of how a result is regarded as a relevant match, including only three types "dominant", "similar", and "exact". Other intentions of users that do not belong to these types are limited.

The above limitations of intention preferences can question the applicability of the systems implementing the proposed architecture since from the viewpoint of users, "don't make me think" seems to be desirable [16]. However, it is suggested for such applications to configure a reasonable default setting of the proposed system, which requires the least input from users but when needed they can provide user interfaces for setting context preferences.

1.4 Significance of the Research

This thesis contributes to the multimedia retrieval research field a solution to the tradeoff between functionality and usability of a search system. With the query creation method, the proposed system provides users with a flexible means to express their varying contextual preferences. Moreover, with the pruning search mechanism, the system maintains a comparative high search performance to other conventional search techniques without disrupting the feature space by dimension reduction or weakening retrieval capabilities by static indexing. The concrete contributions can be reported from three main points as follows.

1.4.1 Context-adaptive Query Creation

The best "machine" that can sense and react effortlessly according to the surrounding environment is human. It seems to be the case that "'intelligent behavior' often has

more to do with simple situational understanding than complex reasoning" [17]. As a results, understanding "contexts" and making use of them as human do (or better) are the goals of many fields in computer science such as artificial intelligence (AI), ubiquitous computing, wearable computing [18, 19, 14].

This thesis promotes the use of contexts for more beneficial results in multimedia retrieval based on the idea of context-dependent semantic computing [20, 21]. The thesis clearly defines three contextual aspects further than the conventional treatment regarding contents of multimedia data and proposes an architecture to index data in such a way that a multimedia retrieval system can smoothly adapt to any new context at query time while effectively maintaining its performance.

1.4.2 Alleviating the Curse of High Dimensionality with Subspace Manipulation

As discussed in section 1.1.3, high dimensionality of data degrades the performance of nearest neighbor search algorithms that use similarity distance functions. This thesis intends to alleviate this phenomenon by a context-based subspace selection method. While indexing multimedia data, high-dimensional features are used so that the delicate contents are preserved, especially for color information of image data (as in chapter 6). But at search time, a subspace of the index space will be selected according to the input contexts of users.

Moreover, a method to combine subspaces can be further applied in order to project the whole index space into a space with a less number of dimensions for more efficient distance calculation. This method will be introduced in section 6.3.3

1.4.3 Adaptive Fast Search Mechanism

This thesis proposes a fast search mechanism to prune the index space in order to quickly find relevant candidates given an input query and context preferences. There have been many successful methods using space partitioning at index time [22, 23, 24, 25, 26], but using space partitioning at search time is intensitvely proposed and

studied in this thesis for large-scale image and video data. The original idea has been introduced in the works of Kiyoki and Miyagawa et al. [27, 28]. This thesis aims for an unified system for the context-based computing and heuristic search algorithms which include two search algorithms based on the related works [27, 28] and a new search algorithm. The detail of the algorithms are introduced in chapter 3.

1.5 Definition of Terms

- **candidate**

  a candidate or a **search candidate**, is a possibly relevant record found by a search algorithm given a query.

- **content preference**

  a content preference, refers to low-level or semantic representations of data that a user is interested in.

- **context**

  a context, refers to a content, intention, or response time preference of users.

- **dimension**

  a dimension or a **feature dimension**, refers to a type of low-level visual feature which is used to represent the content of data. Saying data are represented in d-dimensional space indicates each datum is represented by d types of low-level features and each type is assummedly independent to other types.

- **feature**

  a feature or a **descriptor**, refers to a low-level visual feature that presents an elementary content of an image such as color, shape, or texture content associated with the image.

- **feature space**

  a feature space, is a vector space associated with a set of feature vectors.

- **feature vector**

  a feature vector (sometimes called "a histogram") in d-dimensional space, is a tuple of d real numbers that describes the characteristics of a record.

- **image**

  a digital color image, is a two-dimensional array of *pixels* where a pixel is a sample of three color channels (e.g., RGB) as a tuple of three small integers.

- **intention preference**

  an intention preference, refers to how the content of a result should be regarded as relevant given a content reference.

- **key frame**

  a key frame or a **scene frame**, is a video frame that belongs to a scene in a video and representatively presents the scene.

- **metadata**

  data that contains information about a record including its feature vectors.

- **query**

  a query, is a set of input parameters which are used to state a question to a search system for relevant records.

- **record**

  a record (in this thesis), refers to an image or a video frame.

- **response time**

  time to get results from a search.

- **result**

  a result, a **search result** or an **answer**, is a relevant record returned by a search algorithm.

- **scene**

  a scene, is a series of semantically correlated video frames (e.g., same events).

- **score**

  a relevance score or a **similarity** score, is a real number returned by a function which calculates how much relevant a result is to a given query.

- **subspace**

  a subspace of a feature space, is a feature space created by a subset of dimensions of the feature space.

- **video**

  a digital video, is a sequence of time varying pictures.

- **video frame**

  a video frame or a **frame**, is a single picture from the video. This picture is equivalent to an image.

1.6 Thesis Organization

The contents of this thesis are divided into three parts. Part one including chapters 1–2 gives an overview of the study. Part two consisting of chapters 3–5 focus on the theoretical observation, design, and analysis of the indexing method and search algorithms. Part three including chapter 6–8 describes the implementation of the two application systems, discusses the experiments on those systems, and concludes the thesis with research findings and discussions on open questions in future work.

Part one Chapter 1 introduces the background, motivation, and objectives of this study. This chapter also describes the significance as well as the scope and limitations of the study and gives some definitions of main technical terms used in the thesis. Chapter 2 gives a survey of related works in literature while delineating the characteristics and originalities of this research by contrasting to those works in several perspectives.

Part two Chapter 3 introduces the geometric intuitions and the generalized framework of the multicontext-adaptive search mechanism. This chapter is the most

important chapter to understand the design and implementing principles of the proposed system, which follow throughout the rest of the thesis. Chapter 4 describes the details of the indexing method including indexing principles and the file structure of the disk-resident database model, which is named Bamboo Forest database. Chapter 5 describes three concrete algorithms that work compatibly with the Bamboo Forest database based on the search intuitions and the generalized framework in chapter 3: Maxfirst search, Combinatorial search, and Exact-match search algorithms. This chapter also gives a complexity analysis of the algorithms using the big-O notation.

Part three Chapter 6 focuses on the implementation and experimental studies on the context-dependent image search system with dynamic query creation. Likewise, chapter 7 is dedicated for the frame-wise video navigation system. This chapter discusses intensively the performances of the proposed search algorithms on the Bamboo Forest databases, and is the main chapter on experimental studies in this thesis. Chapter 8 gives some discussions regarding the applicability of the proposed system, sums up the advantages of the proposed query creation and search system with inclusive research findings, and encloses this thesis by introducing further research directions.

Chapter 2

Literature Review

> The greatest challenge to any thinkers is stating the problem in a way that will allow a solution.
>
> *Bertrand Russell*

Content-based multimedia retrieval as a research field has attracted many researchers from many different fields including both theoretical and experimental studies and application systems. The prominent goal is to satisfy the information need of end-users. This goal becomes non-trivial as soon as we start to analyze the challenges waiting ahead. The "information need" of end-users are often ambiguous and context-dependent, whereas the data that we are dealing with are unstructured by nature, contigent on recording devices and large in scales. Especially image and video data, as the fragmented records of our real world, are still so far to represent the complicated imaginations or memories of users for visual information, which are often the starting points for a querying and searching task.

In this chapter, a review of current literature of research trends which relate to the topic of this thesis is described. The review contains several parts that are the main components in the general framework of a content-based multimedia retrieval system: query processing, feature construction, indexing, and search.

2.1 Querying and Searching for Image and Video Data

Before investigating the technical aspects of how to index and search for image or video data from a viewpoint of computational systems, a description from a viewpoint of users will be described. In other words, this section discusses what users expect when searching for images or videos and what tools are existing to assist them.

2.1.1 Search Intention

In literature, the word "intent" and "intention" are used interchangeably, and both refer to what a user desires when searching for information. Datta et al. states "clarity of intent plays a key role in a user's expectation from a search system and the nature of her interaction" [29]. There are several different classification models have been proposed to define how to define "intention" in a search system, but among them, there are three main models.

"Intention" as "reason, purpose, or goal" This definition type relates to the question "why the user is searching" [30]. For example, Broder et al. [3] defines a taxonomy of three categories: "navigational", "transactional" and "informational". Lux et al. [31] defines five categories: "knowledge orientation", "mental image", "navigation", "transaction".

"Intention" as "behavior type" associated with a degree of goal clarity This definition type relates to the question "how the user is search" [29, 32]. For example, Datta et al. [29] classifies users based on their searching behaviors into three classes: "browser", "surfer", and "searcher". In this classification model, a browser is "a user browing for pictures with no clear end-goal, " "a surfer is a user surfing with moderate clarity of an end-goal," and a searcher is "a user who is very clear about what she is searching for in the system."

"Intention" as "implicit content" In this model, an intention is treated as an implicit information that can be learned from a repeated relevance feedbacks of

users [33, 34, 35]. A user of these systems repeatedly marks images in the results as "relevant", "not relevant", or "neutral" to progressively refine the search results.

In this thesis, a new classification model of "intention" is introduced. This model defines an intention of a user as her expectation of how a result should be regarded as relevant with three categories: "dominant", "similar", and "exact". This model focuses on an expectation-driven search behavior rather than a goal-oriented search. In addition, comparing to the relevance feedback approaches which deal with intentions implicitly, the proposed model deals with intentions explicitly suggesting an important role in search and evaluation of results.

2.1.2 Querying Paradigms

A querying paradigm refers to a user-system interaction model of a image (video) search system. In conventional retrieval systems, there are several interaction levels that relates to the complexity of queries supported by the systems. The following list describes the most common models.

Input keyword (or "query-by-keyword") The user inputs a keyword and the search system uses the keyword to match with annotated texts of image or video data. This is currently the most popular way to search images and videos.
Sample systems: Google image search engine[1], Yahoo image search[2], Youtube[3].

Input free-text The user inputs a complex phrase, sentence, question, or story that describes an image she is looking for. This query model is still not widely developed in literature. Google image search can handle an input of a sentence, for example, "a picture of a cute white cat" but it is believed that the algorithm is based on word-by-word matching using annotated texts of image or video data.

[1]https://images.google.com
[2]https://images.search.yahoo.com/
[3]https://www.youtube.com/

Input sketch (or "query-by-sketch") The user draws a rough picture describing what she is looking for.

Sample systems: QbS [36], Sketch4match [37], DrawSearch [38].

One input mage (or "query-by-example") The user inputs an image as the query and wishes to search for an image similar to the query. This is the most popular query model in content-based image search, which does not require any annotated texts but relies only on the metadata extracted from image data.

Sample systems: QBIC [39], survey article [40].

Multiple input images The user inputs multiple images and using combination operations to express their interested content.

Sample systems: imagination-based query creation [41, 42, 43, 44, 45, 46, 47, 48] (proposing set and algebraic operations such as "PLUS", "MINUS", "INTERSECTION", "UNION", "ACCUMULATION", "DIFFERENCE"), [49] (proposing "intersection" operation), [50] (proposing logical operations "AND", "OR", "NOT"),

Text and image The user inputs an example image and keywords as the query for search.

Sample systems: 5D World Map [44, 51], Google image search.

In this thesis, the combination of "multiple input images" and "text and image" querying models is introduced. This query creation method provides users with a highly dynamic query expression capability. The detail of the query creation method and experimental studies are described in chapter 6.

2.2 Presentation of Image and Video Data

2.2.1 Feature Space

Image and video data are basically unstructured. The annotation-based querying and search methods are very limited since they require a heavy labor of manually

annotating the data. Therefore there are more and more researches putting effort on the content-based search methods. In order to develop a content-based search method, a method to extract "content" (or "feature") of the data is needed and this is known as "feature extraction". The collection of features of data creates a representational space of data, called "feature space".

Datta et al. [29] describes two major methods of feature extraction: "global extraction" and "local extraction". "In global extraction, features are computed to capture the overall characteristics of an image" and "in local extraction, a set of features are computed for every pixel using its neighborhood" [29].

The extracted information will be represented by a computational structure, which is commonly a single vector (as known as a "histogram") in a space with the number of dimensions equal to the number of feature components, or a set of such vectors. In some methods, the extracted information can be summarized into "visual words" so that the image is represented as a set of natural language words [52].

As discussed in section 1.3.2, this thesis limits the content of image and video frame data to single-numerical-vector representation. Each element of a feature vector has a non-negative real value (numerical value) and represents a feature of content of an image or a video frame.

2.2.2 High Dimensionality

In section 1.1.3, the "curse of dimensionality" was introduced. This phenomenon was very early recognized by Bellman [53] when he was observing the behavior of choices in high dimensional space, which he called the "combinatorial explosion". Ever since, it is widely studied and found in many fields of computer science such as machine learning including clustering and classification which depends on distance functions, numerical analysis, combinatorics, or data mining [54, 10, 9, 55, 56, 57, 58, 59].

In information retrieval including multimedia retrieval, which majorly depends on a wide variety of distance functions to find nearest neighbors for a query (often called the k-nearest neighbor (KNN) problem), the curse of dimensionality is often hard to deal with.

There are some main trends of methods that deal with high dimensionality:

- Feature extraction including feature selection, dimension reduction methods,
- Adjusting in measure in higher dimensional space itself, like the work in [60] (using angle), share-neighbor distance: [61].

The objective of feature extraction is to reduce the number of dimensions that are used to represent the data including two main kinds of methods: (1) feature selection methods such as sequential feature selection such as FSS, SFBS, PIMR, BDS, FS [12], and (2) dimension reduction method such as PCA, SVD, or LDA [13].

However, reducing the number of dimensions causes reduced "subtleness" that is expressed in the original data and most of the time, this is what users are interested in finding out. While considering that, we also realize that the number of dimensions might be large, but for a query, a smaller number of dimensions are preferred either by the setting of users or by the input itself. As an intuitive example, if we use some thousands of colors to represent an image, it is unlikely that an input will contain all these colors but it is more likely that the input has some colors more dominantly distributed in it. Therefore, at the search time, a search algorithm should focus on these colors rather than others.

2.3 Efficient Indexing Structures and Fast Search In High-dimensional Spaces

Searching in a large amount of multimedia data whose semantic features are represented in high dimensional metadata has been main objective of many researches. There is a phenomenon called "curse of dimensionality" that arises when analyzing data in high-dimensional spaces. This phenomenon suggests that close objects might get separated by a partition boundary when partitioning the space. Many researches have utilized approximate nearest neighbors problems [62, 63] and the MapReduce paradigm [64] in order to solve large-scale multimedia retrieval tasks.

Content-based image retrieval systems have been able to manage collections having sizes that could be very difficult years back. Most systems can handle several million to hundred million images [65, 66, 67, 68, 69, 70], billions of descriptors [71, 68, 72], or address web-scale problems [73, 74, 70, 75], and so on.

The approaches to overcome large-scale datasets and high-dimensional presentation of data vary from many ranges, but in general can be categorized as follows:

- Cluster-based and its derivatives, e.g., [76, 68, 75, 77]
- Locality-sensitive hashing, e.g., [26, 75]
- Tree-based, e.g., [67, 71, 78, 22, 79]
- MapReduce paradigm, e.g., [68]

Most of the approximate high-dimensional near neighbor search methods based on the above categories are based on some kinds of segmentation of the data collection into groups named *clusters*, or partition the data space into areas so that *close* data are indexed in the same *cluster*, or with the same *hash code*, or at the same *node leaf*.

It is easy to see that the space partitioning is done at index time in these methods, but using space partitioning at search time is our original proposal. The importance of treating the data space dynamically depending on the contexts at query time has been recognized early [27, 28]. However, the current situation of emergent of data and demanding attention to users' preferences when searching for information have made this idea more alerted. Additionally, the computational resources as well as the storage resourses that we have nowadays enable us to extend this research direction further.

2.3.1 Feature Space Partitioning for Fast Nearest Neighbors Retrieval

The most efficient and well-known search approach is a group of spatial indexing methods including a family of hashing algorithms (e.g. LSH and its variants) and a family of tree-based algorithms (e.g. R-tree and its variants).

2.3.1.1 Locality-Sensitive Hashing

Some researches approach to high-dimensional image search for large datasets using "locality-sensitive hashing" (LSH), which is a solution for nearest neighbor problem such as those studied in [26, 75]. The first locality-sensitive hashing algorithm was introduced very early in 1998 [57] to overcome "curse of dimensionality". A LSH algorithm uses a family of locality-sensitive hash functions to hash nearby objects in the high-dimensional space into the same bucket. A similarity search is performed by hashing a query object into a bucket, using the data objects in the bucket as the candidate set of the results, and then ranks the candidate objects using the distance measure of the similarity search. The goal of LSH is to maximize probability of "collision" of similar items rather than avoid them such like perfect hashing techniques. Some methods such as [26] improved space efficiency of LSH using *multi-probe* by deriving probing sequence to look up multiple buckets that have a high probability of containing the nearest neighbors of a query object.

Locality-sensitive hashing is similar to cluster-based algorithms since datasets are pre-indexed into groups and at search time, a group will be called providing candidates for similarity calculation. Comparatively, LSH with fixed indexes have limited support for the variations in feature importance due to different user preferences.

2.3.1.2 Tree-based Indexing

The most widely used tree-based spatial indexing structure for high-dimensional data is R-Tree, which is firstly proposed by Guttman in [80]. Many variants of R-Tree including KD-Tree, Ball-Tree have been studied and show some improvements in performance by some aspect like "query retrieval, indexing cost, application specefic, and so on" [81]. Some good survey articles for R-Tree variants can be found in [82], [81], and [83].

Lejsek et al. in [67, 71] introduced a *Nearest-Vector-tree* data structure named NV-tree for approximate search in very large high-dimensional collections. Another author, Liu in [78] used a distributed hybrid Spill-tree, a variant of the Metric-tree, for

a collection of 1.5 billion global descriptors. When comparing to LSH, the performance was the same but the method used fewer disk reads [78]. A NV-tree is constructed after a repeated steps of projection and partitioning through the high-dimensional space. The search algorithms using NV-tree mainly depend on the selected projection lines. Each projection line can be seen as a concrete context to be search. Due to a limited number of projection lines and their fixed contexts, the tree-based techniques are restricted to apply for dynamic queries.

2.3.2 Data Partitioning for Fast Nearest Neighbors Search

Cluster-based techniques Many approximate high-dimensional near neighbor search methods are based on some kinds of segmentation of the data collection into groups named *clusters*, which are stored together on disk. At query time, an index is typically used to select the single nearest cluster for searching. Some survey articles on clustering for content-based image search are such as [84], [85], and [40].

The work [68] used cluster-based indexing method named the *extended Cluster Pruning* (eCP) proposed in [76] that is an extended method of *Cluster Pruning* (CP) [86]. The *Cluster Pruning* method [86], which is a simple algorithm comparing to the traditional K-Means algorithm, randomly chooses a subset of data points to be *leaders* and the remaining data points are partitioned by which leader is the closest. The eCP by Gudmundsson et al. gives some changes to improve CP by three additional parameters to control cluster size on disk, balance cluster size distribution in order to improve search in both IO and CPU costs. Other derivatives of cluster-based methods can also use *hierarchical clusters* and so-called *multiplevel clustering* [76] or *multi-index* in [75] to partition large clusters into smaller clusters. The very recent work in very large scale image search [68] used clustering concept adapting with distributed Map-Reduce programming paradigm, and tested with Hadoop framework by indexing 30 billion SIFT descriptors for roughly 100 million images (about 4 terabytes of data).

2.3.3 Inverted List Data Structure

Inverted index is a optimized data structure that facilitates efficient retrieval. This data structure is broadly used in information retrieval (IR) tasks where target data is documents. An inverted index includes a list of *inverted lists* for each *word* (or *term*). Each inverted list is a list of identifiers of the documents containing that *term*. The answers to a query "find the documents where word X occurs" can be retrieved quickly by looking up at the inverted list X. When querying one or more terms, documents are retrieved by looking up indexes of corresponding terms, and they are processed by computing their vectors of word frequencies and ranked to return as closer distance to the query. In either case, the time and processing resources to perform the query is dramatically improved.

Using inverted index structure for other multimedia information retrieval tasks such as for image and video are currently investigated, e.g., [35, 87, 65, 88, 28]. Researches in this direction treat either low-level features of images such as color and texture in [35] or "visual words" of quantized descriptors such as [87] as set of terms to be indexed. By searching time, the algorithms get out all the images which contain features appear in query and adding them to the pool of candidate images to be ranked.

There are two common ways to organize the inverted list: *document-sorted* [87, 65] and *frequency-sorted* as stated in [89]. Document-sorted inverted list means the list of documents are sorted by document identifiers, commonly by an increasing order. Frequency-sorted list means the list of documents are sorted by the frequency of the term occurring in the documents, commonly by a decreasing order. The document-sorted technique is useful for phrase and Boolean queries whereas the frequency-sorted technique is useful for ranked document retrieval. A combined technique named *dual-sorted* [90] can be used to maintain a single data structure that offers both two orderings of frequency-sorted and document-sorted inverted lists.

From storage viewpoint, the inverted index is stored either in-memory [87] or on-disk [87, 65] and either be compressed [91, 92] or not. For large-scale data, it's more

Table 2.1: Categories of methods use inverted index. The bold methods are used or newly proposed in this research.

Viewpoint	Categories
Organization	in-memory, **on-disk**
List structure	document-sorted, frequency-sorted, dual-sorted, **modified frequency-sorted**
Content representation	**low-level features**, "visual words"
search mechanism	pruning from head, **context-adaptive pruning**
Application	text retrieval, image retrieval, **video retrieval**

appropriate to use on-disk storage so that the memory consumption can be reduced significantly by transferring only needed part of each inverted list from disk [93].

Table 2.1 summaries categories of methods that use inverted index: organization of inverted index (either in-memory or on-disk), structure of inverted index (either document-sorted or frequency-sorted), representation of semantic content (either using low-level features or visual words), and its applications (either text, image or video retrieval). The next chapters of this thesis introduces a disk-resident database using a modified frequency-sorted inverted indexing method (chapter 4), which is as an improved method to support context-dependent image and video retrieval with a dynamic pruning search mechanism (chapter 3) and three search algorithms (chapter 5).

Chapter 3

Methodology

> If the facts don't fit the theory,
> change the facts.
>
> *Albert Einstein*

3.1 Geometric Intuition

Human are excellent at adapting to and using contexts. Despite our complex desires and preferences, we know when a thing is *relevant* or *irrelevant* effortlessly given a situation. Observing the processes of how we achieve that gives us a hint to formalize them in a computational model. It is likely that we organize all of known facts and beliefs (data) into a virtual *attribute* space with nearly infinite number of dimensions but when given a situation, we only use some of them to make some *orders* of those facts and make decisions based on those orders. *Sorting* and *searching* become two basic and interdependent processes of our daily life problem solving.

This context-dependent computing mechanism has been early recognized and proposed in the works of Kiyoki et al. [20, 21]. In a computation model, data are supposed to be in a high dimensional space and when a context is given, a subspace of a less number of dimensions is selected as the space for calculation, which is either searching or ranking or more complex integrated tasks such as semantic interpretation. By using the idea of Kiyoki et al., the importance of dynamic recognition of

contexts is emphasized, which will potentially leads to better computing performances by its intrinsic ability to alleviate the curse of high dimensionality discussed in section 1.1.3. Moreover, it also yields some degrees of interpretability of semantic computing processes. Figure 3-1 describes this intuition.

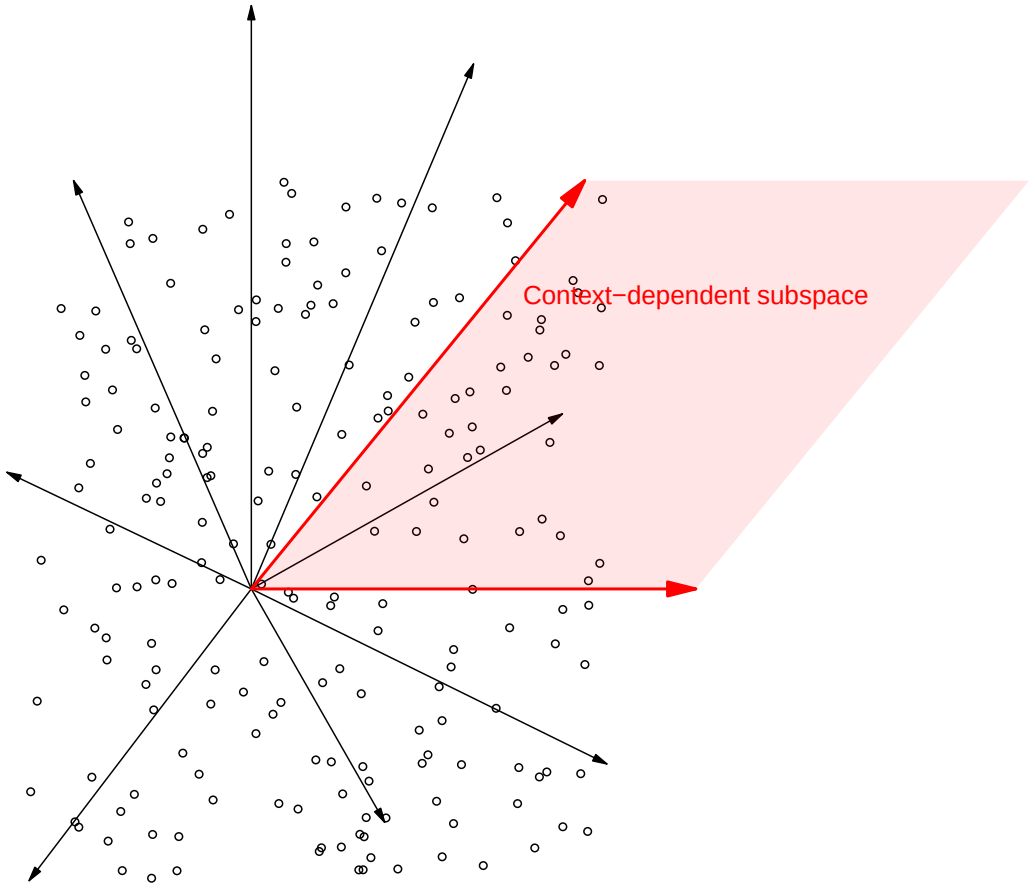


Figure 3-1: Data on high dimensional space and subspace selection.

Assume that we are interested in two *attributes* of data and want to proceed some search for information with respect to an existing *preference*. This situation is described in Figure 3-2 in which data and query from high dimensional space are already projected into a two-dimensional space. By assuming that the attributes are independent to each other, we can assume that the computing space is a subspace of high-dimensional space of real numbers in which orders can be induced from the

orders of values of real numbers. As a consequence, a subspace created by a selection of a set of appropriate dimensions is a subspace with orthogonal dimensions.

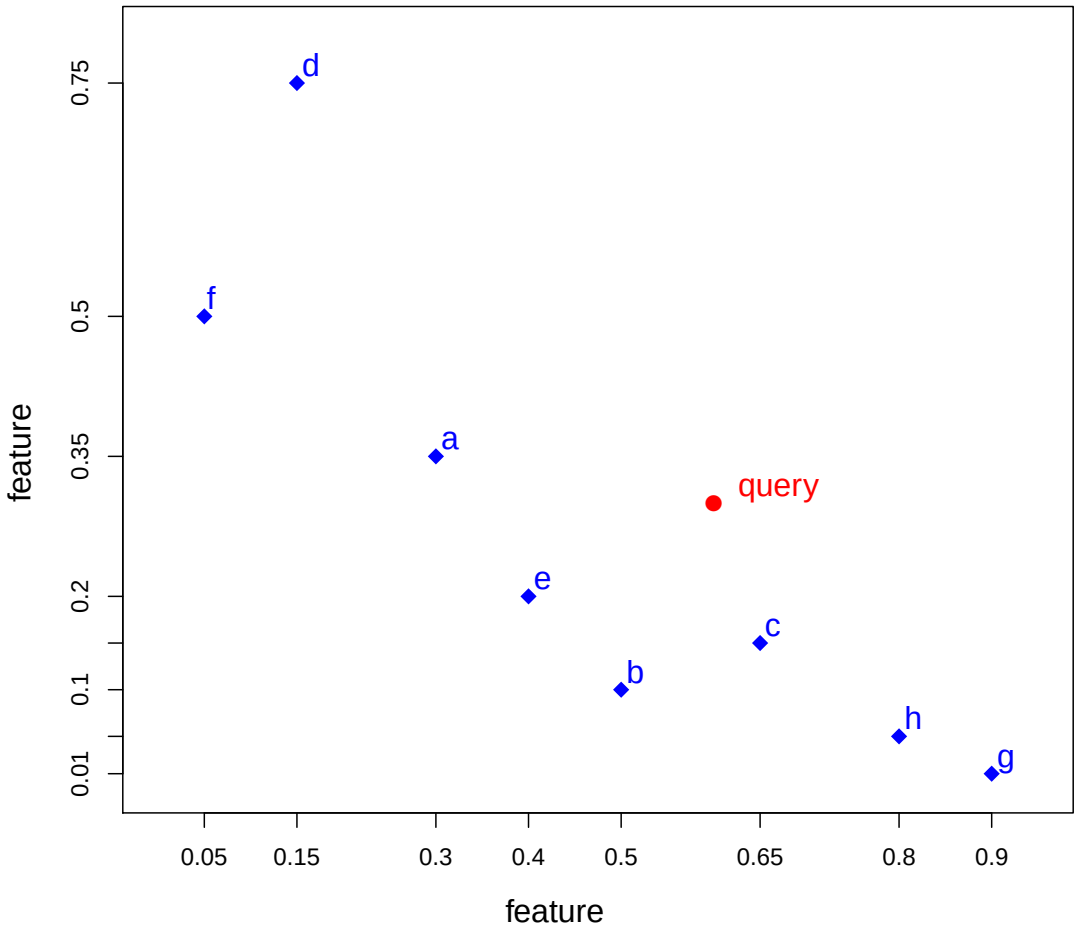


Figure 3-2: Sample data and query on a selected subspace $\mathbb{R}^2$.

In Figure 3-2, the *content preference* is supposedly expressed by the position of the *query* point. At this moment, we can have at least three intentions to *order* other data points and have them listed as the results of a search problem. The first intention is to find the answers to the question: "which is the closest point to the query?". The second intention is to find the answers to the question: "which points are close to the query?". And the third intention is to find the answers to the question: "which points have a large sum of values in some direction". The first and second questions are common in information retrieval while the third question expects more exploratory

answers.

Intuitively, if a data point is close to the query, it is close to the query in any direction. Based on this observation, we can approximate the locations of *candidates* as some areas that are remote from the query points by some small distances as shown in Figure 3-3. The original idea has been proposed by Kiyoki et al. [27] who applied this intuition for document retrieval problems.

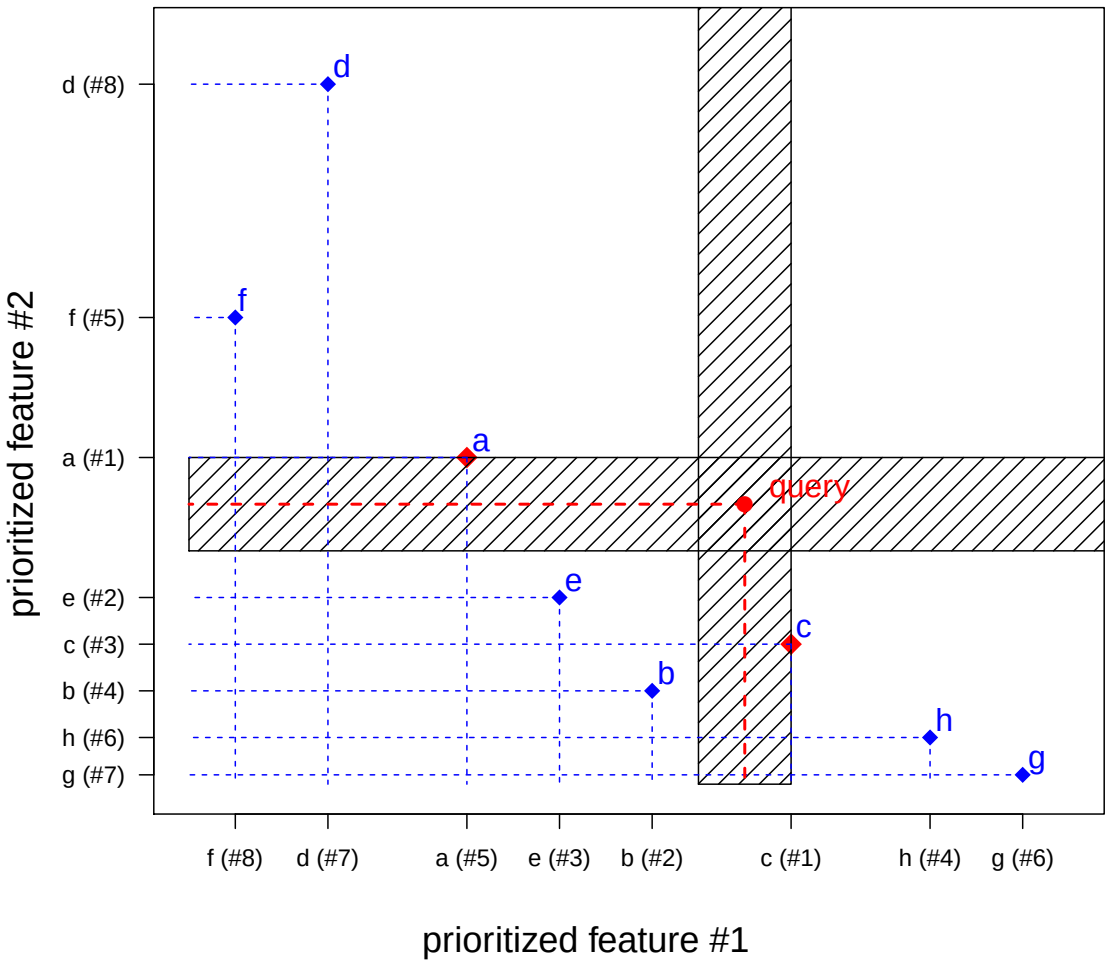


Figure 3-3: Approximating locations of first *similar* candidates.

This search for the very first candidates depends on the query point, not on the search space as shown in Figure 3-4. This idea is also known as "similarity heuristic" which has been studied in document retrieval [27, 94] and recently applied in image [28] and video [95], In this figure, a new priority ranking of features are defined

based on the position of the new query point. At this moment, the vertical feature is assumed to be more important than the horizontal feature. The search finds the locations of the first candidate, point "f", and then point "d".

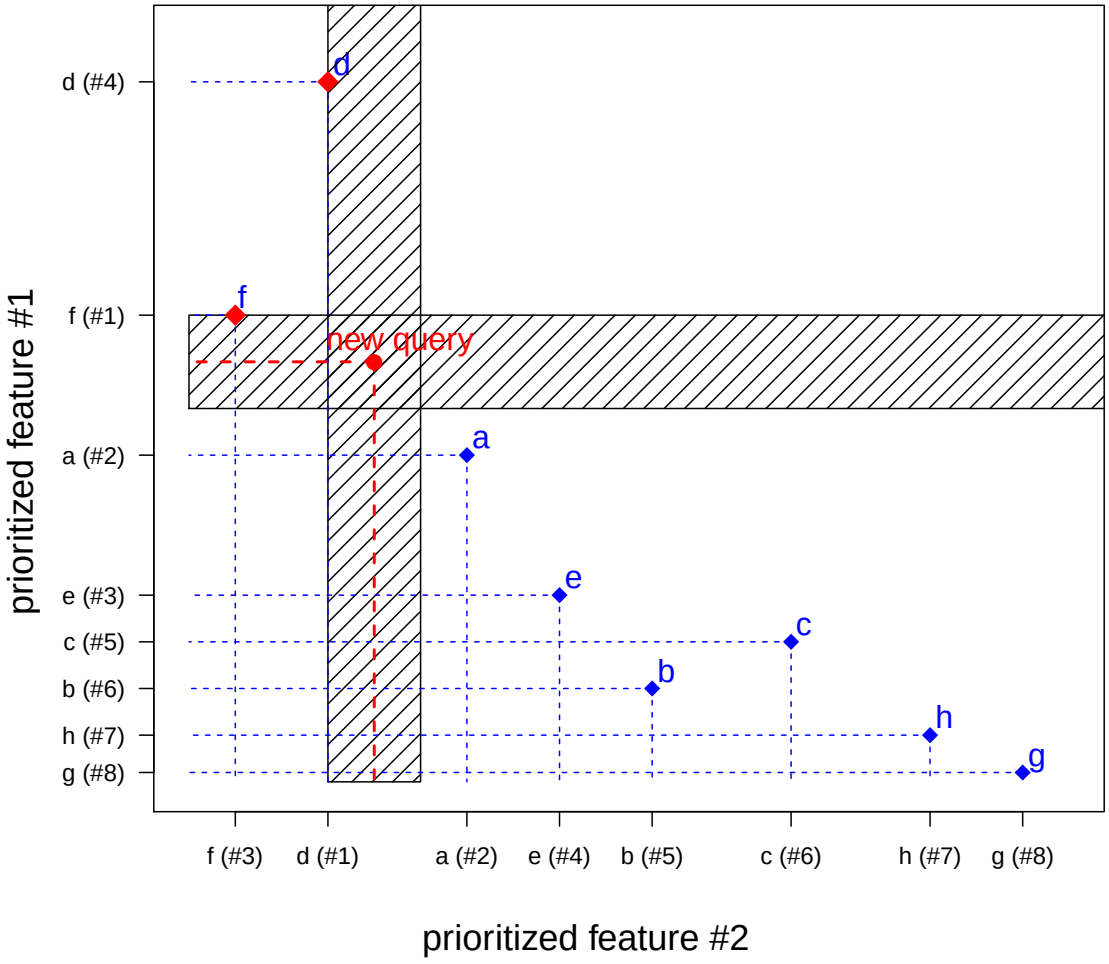


Figure 3-4: Location approximation depending on an input query

The effort to find the close points to the query continues from those starting locations. For the query given in 3-3, the continuing process is given in Figure 3-5. In this figure, the search continues from the first prioritized feature (horizontal axis) starting from point "c" as it is found to be closest to the query in this direction. On this direction, point "b" ("b (#2)") is found. And on the other direction, point "e" ("e (#2)") is found. It can be seen that this continuing looking for candidates has a purpose that is to avoid the error of missing the actually "close" point. In other

words, the error of a point is close to the query in one direction but in both directions, it returns to be remote. In this example, it is point "a" comparing to points "e" and "b".

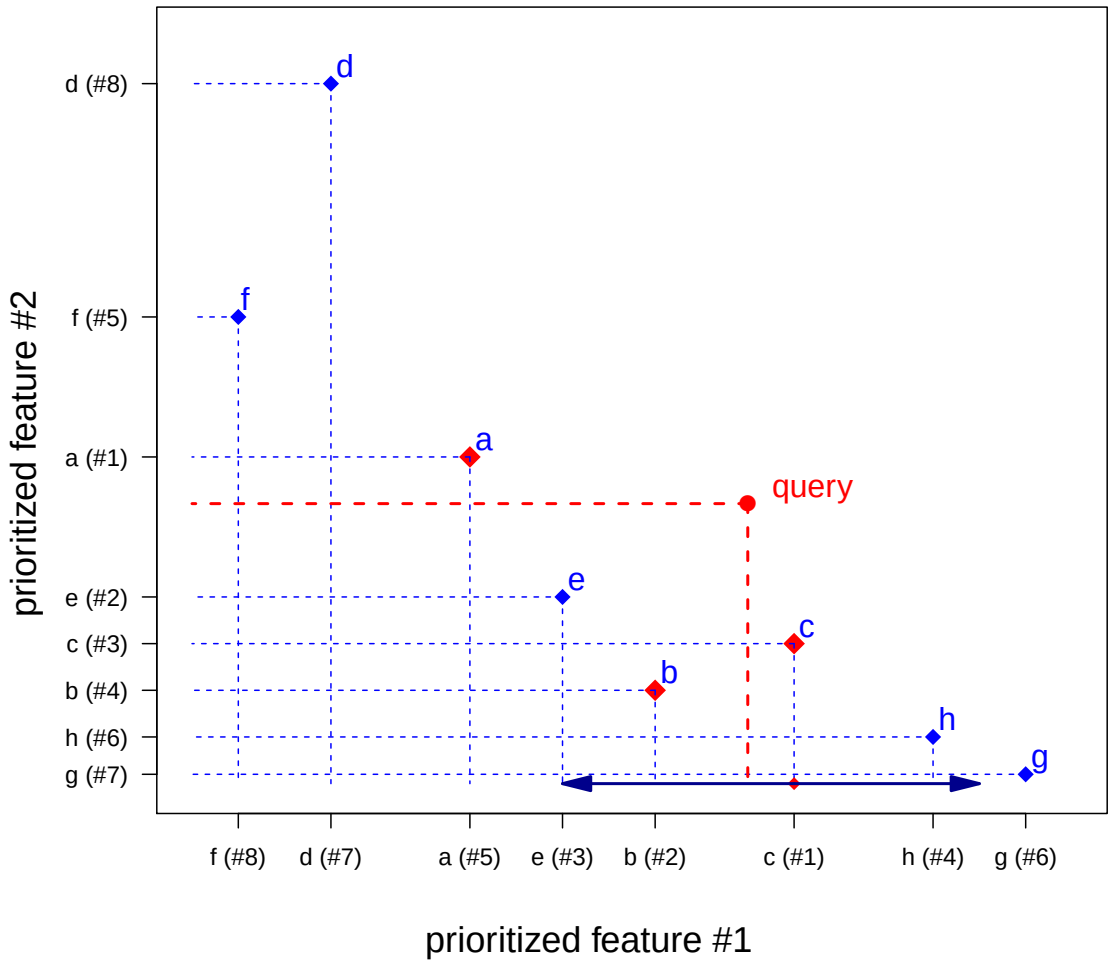


Figure 3-5: Expanding search area for new *similar* candidates.

A search process based on the intuitions in Figures 3-3 and fig:expand is named a "Combinatorial search" and described in Algorithm 5.2 in section 5.2.

Where can we be guaranteed to find the closest point without any error? If following the search process that we are doing, which is incrementally searching for candidates at each direction, there will be a moment we find at least one candidate that appears at all directions, and all found candidates make a covering rectangle. This rectangle is the guaranteed location that the closest point to the query will be

in. Figure 3-6 shows this intuition. By continue searching at the horizontal direction, we find points "c, b, e" where as at the same time, we find points "a, e" and realize that "e" appears in both candidate sets, meaning we have found a covering rectangle. All found candidates at the moment are "c, b, e, a" and we are guaranteed to find the closest point to the query among these candidates. The search process based on this intuition is named an "Exact-match search" and described in Algorithm 5.3 in section 5.3.

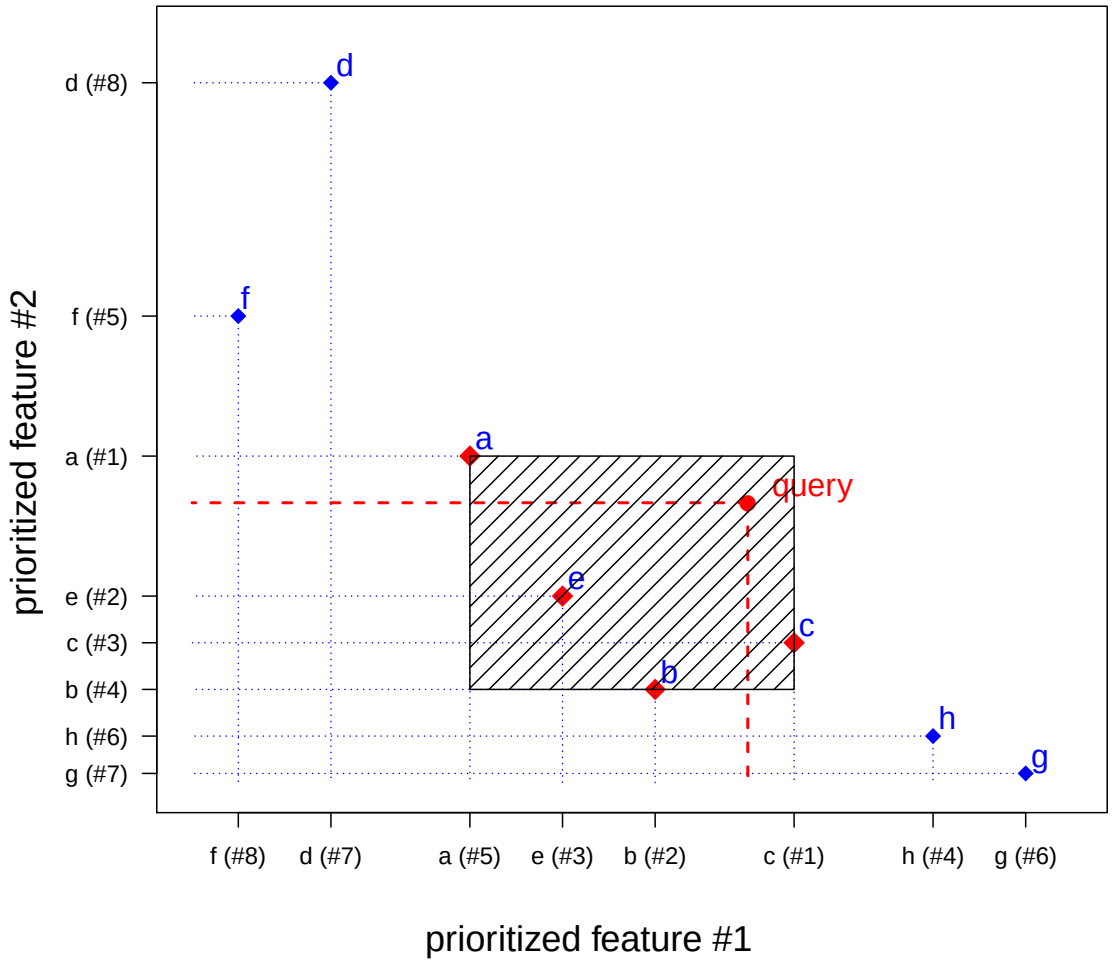


Figure 3-6: Determining area of *exact-match* candidates.

Let us consider the third question, "which points have a large sum of values in some direction?". The answers are expected by finding candidates starting from the points which have very large values in some desired directions. This intuition has

been originally applied for image data in the work of Miyagawa et al. [28]. Figure 3-7 shows the intuition of the search process.

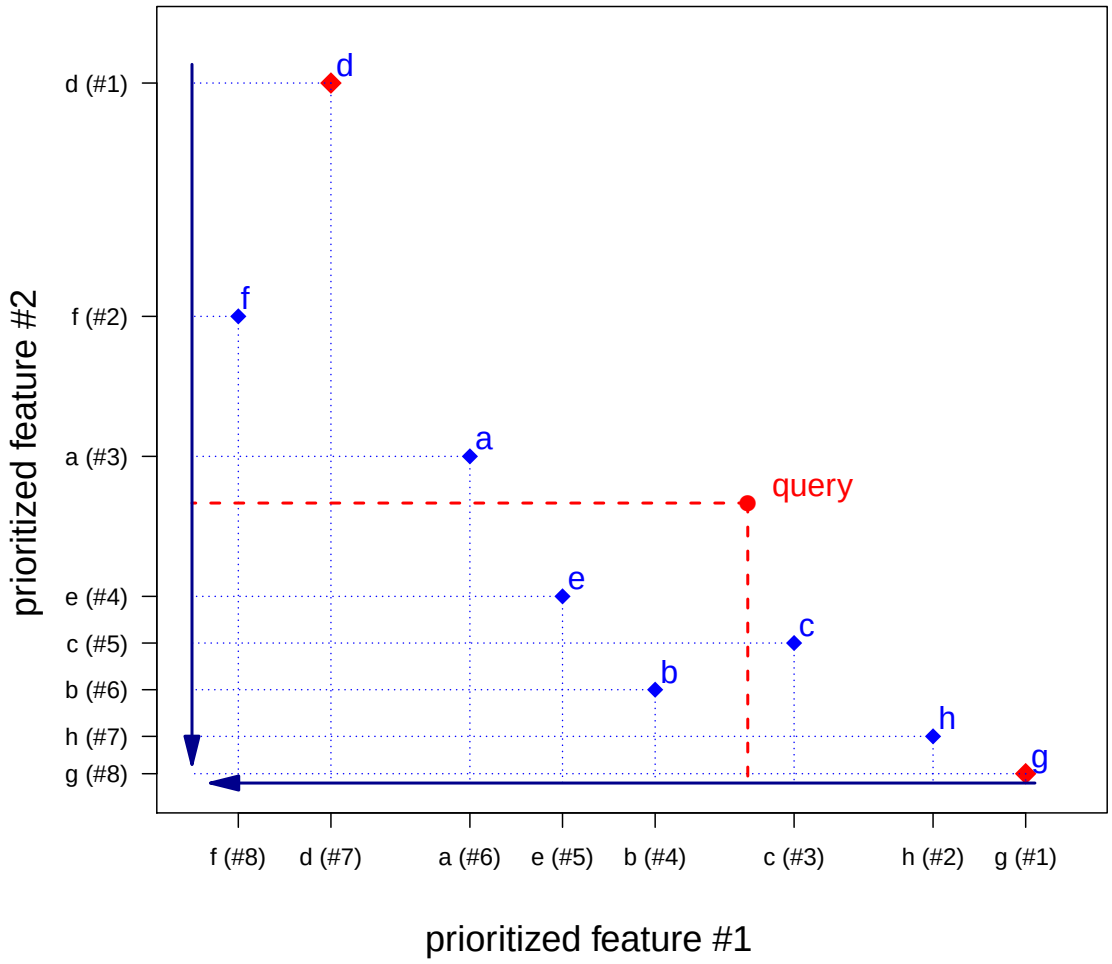


Figure 3-7: Approximating locations of first *dominant* candidates.

In Figure 3-7, we can see that we re-rank the data points on each direction based on a descending order of values. On the first prioritized feature direction, we find point "g" which has the highest value and on the second feature direction, we find the point "d". They are the starting points to continue looking for the candidates. It is important to note that this intuition works as a *greedy* strategy but it does not guarantee that the final order based on the sums of values in both directions. The sums of values in both directions will be done giving us the final order of candidate points. It is to be shown by experiments in section 7.5.6 that this intuition can

fail badly when the number of preferred features are increased. The search process based on this intuition is named a "Maxfirst search" and described in Algorithm 5.1 in section 5.1.

The above discussions have presented several geometric intuitions that motivate the design of a new indexing and search system with dynamic query creation for data in high dimensional space, which is the main goal of this thesis and will be described in more details in the next chapters. This thesis aims for an unified system for the context-based computing and heuristic search algorithms mainly based on the works of Kiyoki, Kitagawa, and Miyagawa [20, 21, 27, 28]. The proposed system is particularly designed for image and video data with rigorous detail in the indexing and search processes. Moreover, a third retrieval algorithm that quickly searches for exact-match candidates using the covering rectangle intuition (Figure 3-6) is newly proposed (Algorithm 5.3).

3.2 Generalized Multicontext-adaptive Pruning Search Mechanism

A multicontext-adaptive search method based on the geometric intuition discussed in the previous section is described in Algorithm 7.6 and its logic is shown in Figure 3-8.

The logic of this search method contains five steps: (1) reflecting content preference onto query vector (subspace selection), (2) prioritizing directions for searching, (3) initializing starting points of search, (4) iteratively finding matching candidates, and (5) ranking candidates and returning top results to users.

It is to note that, the candidates found on one direction will not have any actual *relevant score* comparing to the query until a distance function between its feature vector and the query's feature vector is applied and returns a real value as the *relevant score*. In other words, the candidates found on one direction only have some *local* properties that state it is *close* to the query, but not globally. Consequently, the

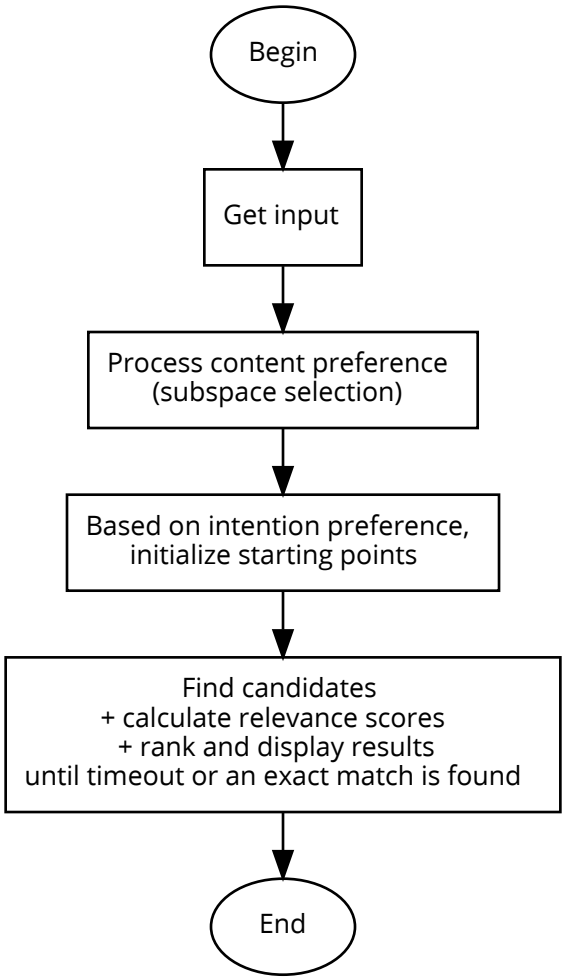


Figure 3-8: Generalized logic of the multicontext-adaptive search method.

fourth and fifth steps are always done right after each other every time a candidate is found. The definitions of several distance functions that will appear in this thesis are described in section 3.3.2. Figure 3-8 shows the general logic of the search method in which step 4 and 5 are integrated.

In Algorithm 7.6, the content preference is expressed as a vector p and reflected during computation in lines 1 and 15. The intention preference is processed at conditional branching in lines 4 and 5. The response time limit preference is controlled in the *while* loop from line 11.

The iterative finding candidates in each direction of prioritized features is repeated as in line 13. In order to speed up this process, we propose to index data using inverted lists in the next chapter.

Algorithm 3.1 Multicontext-adaptive pruning search mechanism

Input:

- A query vector $q \in \mathbb{R}_+^d$;
- A dataset $\mathbf{X} \subset \mathbb{R}_+^d$;
- A preference vector $p \in \mathbb{R}_+^d$ such that $p_i = 1$ if i-th feature is preferred and $p_i = 0$ otherwise;
- A preference of intention $\mathbb{I} \in \textit{dominant}, \textit{similar}, \textit{exact}$;
- Optional: Maximum number of priorities m, by default $m = 5$;
- Optional: Timelimit t, by default $t = 1(\textit{second})$ or *None* for *exact* matching;
- Optional: Number of results R, by default, $R = 20$.

Output: Top R ranked relevant $x \in \mathbf{X}$.

[*Step 1: Reflect content preference onto query*] Set $q \leftarrow q \odot p$ where $\odot$ is the elementwise multiplication or Hadamard product.
[*Step 2: Prioritize features*] A permutation function σ of the set $\{1, \ldots, d\}$ that sorts elements of vector q in a descending order: $\big(q_{\sigma(1)} \geq \ldots \geq q_{\sigma(d)}\big)$.
[*Step 3: Initialize starting points*] Accumulator vector $v \in \mathbb{R}^m$:
if $\mathbb{I} = \textit{dominant}$ **then**
    $v \leftarrow \big(\max_{x \in \mathbf{X}} \ x_{\sigma(1)}, \ldots, \max_{x \in \mathbf{X}} \ x_{\sigma(m)}\big)$.
else
    $v \leftarrow \big(q_{\sigma(1)}, \ldots, q_{\sigma(m)}\big)$.
end if
[*Step 4: Find candidates*]
List of candidates $\mathbf{C} \leftarrow \{\}$
while time limit t is not reached **do**
    for i **from** 1 **to** m **do**
        $c \leftarrow \arg\min_x \big|x_{\sigma(i)} - v_i\big|$ for all $x \in \mathbf{X}$ not yet in $\mathbf{C}$.
        $v_i \leftarrow c_{\sigma(i)}$
        $score = relevant(q, c \odot p)$. ▷ see sections 3.3.2 and 7.5.2.2
        if $\mathbb{I} = \textit{exact}$ and $score = 0.0$ **then**
            return c
        end if
        Update $\mathbf{C}$ with candidate c and $score$.
        Sort $\mathbf{C}$ by an appropriate order of scores. ▷ see section 7.5.2.2
    end for
end while
[*Step 5: Return top results*]
return top R of $\mathbf{C}$ to users.

3.3 Definitions of Similarity Calculation

This thesis uses the following definitions of similarity and distance methods that refer their definitions in the Encyclopedia of Distances [96].

3.3.1 Similarity and Distance

The Encyclopedia of Distances [96, p. 3] defines a **distance space** as follows:

> A **distance space** (X, d) is a set X equipped with a **distance** d. A function $d : X \times X \to \mathbb{R}$ is called a **distance** (or **dissimilarity** on X if, for all $x, y \in X$, it holds:
>
> 1. $d(x, y) \geq 0$ (*nonenegativity*);
> 2. $d(x, y) = d(y, x)$ (*symmetry*);
> 3. $d(x, x) = 0$ (*reflexivity*).

Equivalently, a **similarity** is defined as:

> Let X be a set. A function $s : X \times X \to \mathbb{R}$ is called a **similarity** on X if s is nonnegative, symmetric and the inequality
>
> $$s(x, y) \leq s(x, x)$$
>
> holds for all $x, y \in X$, with equality if and only if $x = y$.

In this thesis, a "similarity" function is used interchangeably with a "distance" function but both refer to a similarity function. This indicates a two similar objects have a smaller "similarity" score returned by a similarity function s.

Some transformations can be used to convert between a distance (dissimilarity) d and a similarity s function are introduced in [96, p. 4] such as: $d = 1 - s$, $d = \frac{1-s}{s}$, $d = \sqrt{1 - s}$, $d = \sqrt{2(1 - s^2}$, $d = \text{arccos}s$, $d = -\ln s$.

3.3.2 Similarity Functions

Given two records (e.g., two images or two video frames), represented by nonzero vectors $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ from $\mathbb{R}^n$, the similarity functions used in this thesis are defined as in following subsections. The definitions of similarity functions are extracted and based on their definitions in the Encyclopedia of Distances [96, pp. 325-331].

3.3.2.1 Bray-Curtis Similarity

The Bray-Curtis similarity, denoted as $BRAY\_CURTIS(x, y)$, is a similarity on $\mathbb{R}^n$ defined by

$$\frac{2}{n(\bar{x} + \bar{y})} \sum min\{x_i, y_j\} \tag{3.1}$$

where $\bar{u} = \frac{\sum u_i}{n}$.

3.3.2.2 Canberra Distance

The Canberra distance, denoted as $CANBERRA(x, y)$, is a distance on $\mathbb{R}^n$ defined by

$$\sum \frac{|x_i - y_i|}{|x_i| + |y_i|} \tag{3.2}$$

where $|a|$ denotes absolute value of $a \in \mathbb{R}$.

3.3.2.3 Symmetric χ^2-distance

The symmetric χ^2-distance (or chi-distance or chi-square distance), denoted as $CHI\_SQUARE(x, y)$, is a distance on $\mathbb{R}^n$ defined by

$$\sqrt{\sum \frac{\bar{x} + \bar{y}}{n(x_i + y_i)} (\frac{x_i}{\bar{x}} - \frac{y_i}{\bar{y}})^2} = \sqrt{\sum \frac{\bar{x} + \bar{y}}{n(\bar{x} \cdot \bar{y})^2} \cdot \frac{(x_i\bar{y} - y_i\bar{x})^2}{x_i + y_i}}. \tag{3.3}$$

3.3.2.4 Cosine Similarity

The Cosine similarity, denoted as $COSINE(x, y)$, is defined by

$$1 - \frac{x \cdot y}{\|x\| \, \|y\|}. \tag{3.4}$$

where $\|u\| = \sqrt{\sum u_i^2}$ is the magnitude of a vector u and $x \cdot y$ operation is the dot product between two vectors x and y.

3.3.2.5 Normalized l_p-distance

The power (p, r)-distance is a distance on $\mathbb{R}^n$ defined by

$$(\sum |x_i - y_i|^p)^{\frac{1}{r}}. \tag{3.5}$$

For $p = r \geq 1$, it is the l_p-metric, including the Euclidean, Manhattan, and Chebyshev metrics for $p = 2, 1$ and ∞, respectively. The distance functions are also respectively denoted as $L2(x, y)$, $L1(x, y)$, and $CHEBYSHEV(x, y)$.

3.4 Feature Vector Normalization

In this section, three functions used for normalizing a feature vector that are used in this thesis are introduced: $NORMALIZE, LINEAR\_SCALE$, and $EQUALIZE$. The $NORMALIZE$ function is used in general cases whereas the $LINEAR\_SCALE$ and $EQUALIZE$ functions are particularly designed and used in the imagination-based query creation method in section 6.3.3.

3.4.1 Frequency Vector Normalization

A frequency vector is a element-wise non-negative vector that has the sum of its elements equal to 1.0. In this thesis, when there is no description further for a feature vector, it is implicitly referred as a frequency vector.

A feature vector that represents content of a data can be extracted by several different methods and its scale also varies. This thesis assumes that each extracted vector is non-negative, meaning all elements of the vector are non-negative. And as an implicit rule, the extracted vector will be normalized into a frequency vector by using the following formula.

Given a feature vector $x = (x_1, \ldots, x_d) \in \mathbb{R}^d_+$ which is a non-zero, non-negative vector, meaning there is at least one element is non-zero, the following formula applies for each element x_i

$$x_i \leftarrow \frac{x_i}{\sum_{j=1}^{d} x_j}. \tag{3.6}$$

If the extracted vector is a not non-zero vector, which is possibly the result of subspace selection (e.g., adaptive subspace selection in section 6.3), the feature vector will be left unchanged.

A transformation procedure implementing the equation 3.6 for a feature vector x will be denoted as $NORMALIZE(x)$.

3.4.2 Linear Scaling

This normalization method will be used when combining feature vectors, particularly in an imagination-based search algorithm described in section 6.3.3. The purpose of this normalization is to scale the values of elements of a feature vector to a range $[a, b]$. By default, the interval $[a, b] = [0, 1]$.

Given a vector $x = (x_1, \ldots, x_d) \in \mathbb{R}^d_+$, with $max = \max_{1 \le j \le d} x_j$ and $min = \min_{1 \le j \le d} x_j$, the following equation applies for each element x_i:

$$x_i \leftarrow \frac{(b-a)(x_i - min)}{max - min} + a. \tag{3.7}$$

The procedure implementing this normalization for a feature vector x will be denoted as $LINEAR\_SCALE(x)$, or $LINEAR\_SCALE(x, a, b)$.

3.4.3 Power-law Transformation

The normalization based on a power-law transformation has a useful interpretation when applying to a feature vector, which is by applying a power-law transformation we intentionally change the preference of how the content will be adapted.

Given a vector $x = (x_1, \ldots, x_d) \in \mathbb{R}^d_+$, the following formula applies for each element x_i

$$x_i \leftarrow x_i^\gamma \tag{3.8}$$

where $\gamma \geq 0$.

In equation 3.8, the smaller the value of γ, the more "balanced" the values of elements of the vector x will be. In other words, this normalization "equalized" the the values of elements of x, resulting a meaning that there no preference especially set for any direction basis of $\mathbb{R}^d_+$. If γ is very big, the contrast between values of elements of x becomes bigger. There are two special cases:

- $\gamma = 0$, every $x_i \neq 0$ will have value 1, meaning all non-zero elements have the same priority.
- $\gamma = 1$, x is unchanged, meaning the proportions are preserved.

The procedure implementing this normalization for a feature vector x will be denoted as $EQUALIZE(x, \gamma)$.

Chapter 4

Database Construction

> He who has a why to live can bear almost any how.
>
> *Friedrich Nietzsche*

The previous chapter has described the intuitions of the multicontext-adaptive search method and its generalized search procedure. The search procedure contains two main steps which are (1) to select a subspace and (2) to find candidates with respect to content and intention preferences at each query time. This chapter introduces a disk-resident database model, named Bamboo Forest database model, which is specifically designed to facilitate those two steps in order to achieve a high performance of the proposed search method.

The idea of this database model is straightforward. Each dimension of the feature space will be indexed independently so that a context-dependent subspace can be selected quickly by choosing appropriate dimensions based on the content preferences. This strategy not only makes it easy to choose a subspace but also reduces the memory cost when loading a database since only needed indices will be open to read. Each dimension will be indexed as an inverted list, meaning each dimension contains a set of (v, rid) where rid is the identifier of a datum and v is its corresponding feature value in this dimension.

In order to quickly find local candidates in one prioritized dimension (search in-

tuitions in Figure 3-3 or Figure 7-7), the corresponding inverted list is sorted by the values of projected data on this dimension. In this case, the inverted lists are so called value-sorted invert lists. We will see in section 4.4 that the cost of finding the first candidate in one prioritized dimension can be done in a $O(\log N)$ or a $O(1)$ while the next candidates are always found incrementally if supposed that the cost of accessing to a file at a random position is $O(1)$.

4.1 Indexing Principles

Given a dataset of N records (e.g., images or video frames) $(x_1, \ldots, x_N)$ and a set of corresponding N feature vectors in d-dimensional space ($\mathbf{X} \in \mathbb{R}_+^d$), we have a matrix $\mathbf{M}$ with column vectors are feature vectors of N data in $\mathbf{X}$ as shown in Figure 4-1.

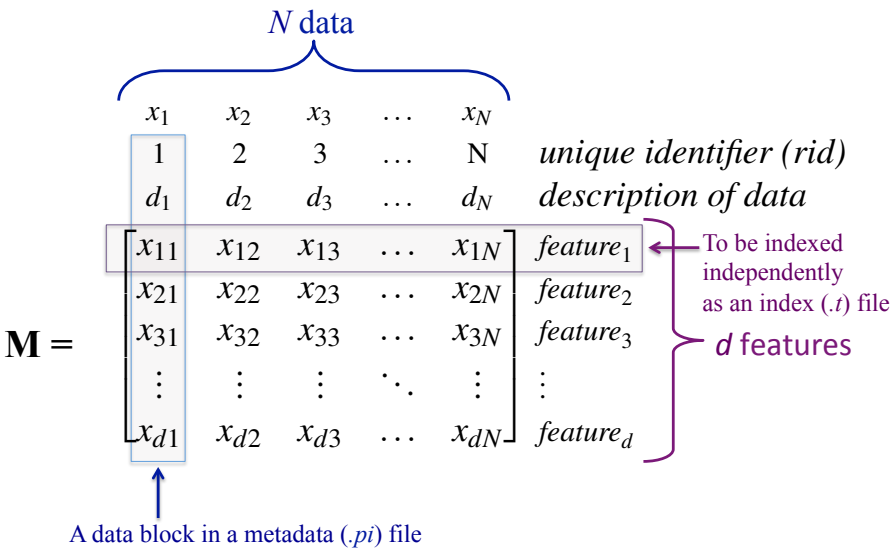


Figure 4-1: Indexing a dataset of N data in d-dimensional space.

In Figure 4-1, each record has an unique identifier (rid) which will be used as a primary key to index it in the database. Each record also has a description which can be text information about the record (e.g., path to the record in the file system or its URL in the Internet).

The dataset in Figure 4-1 will be indexed by the following rules:

1. each row vector $(x_{i1}, \ldots, x_{iN})$ of M is indexed independently as an inverted list,

2. each inverted list is a list of (rid_j, x_{ij}) and is sorted by **a descending order** of values $\{x_{ij}\}$,

3. all indexes are written to binary files on disk by data blocks $\{(rid_j, x_{ij})\}$ so that each can be accessed by random access,

4. all column-wise descriptive information and feature vectors are written to a metadata file for referencing when needed (i.e., when calculating the relevant scores, line 15 in Algorithm 7.6).

Since each index is a ordered independent inverted list, the database is named "Bamboo Forest" database while each index is metaphorically a *bamboo* tree, which is literally a tree mainly no branches but only sections. In the generalized search Algorithm 7.6, the search process finds one candidate on a direction at a time, however, in actual practice, it is suggested to obtain a number of candidates on the direction at a time. In other words, instead of *fetching* one piece of a bamboo tree at a time, we will *prune* (or *trim*) a section of it at a time. This is recommended especially for large datasets since for large datasets, since there are likely many different identifiers that have same values in an inverted list and one-by-one accessing to file is likely too expensive. The number of candidates on a *section* to be fetched out is set equal to 50 in the implementations and experiments in this thesis (chapters 5 and 7).

4.2 Bamboo Forest Database

4.2.1 File Naming and File Content Format

The database contains three types of files that are used to store inverted list and reference metadata: metadata files (named *.pi* files), feature reference files (named *.meta* files), and feature index files (named *.t* files). A demonstration of metadata files (*.pi* files) and index files (*.t* files) can be seen in Figure 4-1.

4.2.1.1 Metadata File (*.pi* file)

A metadata file follows the 4th indexing rule (section 4.1) and indexes the column-wise information of a dataset (see Figure 4-1).

The data structure of a *.pi* file is shown in Table 4.1. A *.pi* file stores each column $(rid, description, feature\_vector)$ as a data block and indexes them by their *rid*s as a primary key to look up a record. contains the identifiers of records (*rid*s), and their metadata including sources of data (e.g., file paths to data), and values of feature vectors. The filename format for this file type is "db_name.pi".

Table 4.1: Data structure of a metadata file (*.pi* file).

File extension	.pi
File type	binary

Structure of a data block	`{long rid;`
	`string description;`
	`float[d] feature_vector}`

Number of bytes of a data block	$8 + 60 \times 2 + d \times 4$

4.2.1.2 Feature Index File (*.t* file)

An index file follows the indexing rules number 1, 2, and 3 (section 4.1). An index file indexes the row-wise information of feature matrix **M** of a dataset (see Figure 4-1).

The data structure of a *.t* file is shown in Table 4.2. A *.t* file stores each value x_{ij} of the feature row $feature_i$ and the corresponding jth *rid* as a data block (rid, x_{ij}). Data blocks in a *.t* file are sorted by the values (x_{ij}) with an ascending order. The filename format for this file type is "db_name_featuretype_index.t ".

Table 4.2: Data structure of an index file (*.t* file).

File extension	.t
File type	binary
Structure of a data block	`{ float value;` `long rid }`
Number of bytes of a data block	$4 + 8$

4.2.1.3 Reference File (*.meta* file)

A reference file is optional and contains the reference information regarding a feature vector. It is a dictionary about feature types in a feature space. It indexes ranges of indices of a feature vector. An example of a dictionary of feature space created by several features types in section 7.3 is as in the following definition (Figure 4-2) using a Python dictionary data structure:

```
FINDEX = OrderedDict({
          "cedd": range(0, 144),
          "color": range(144, 207),
          "fcth": range(207, 399),
          "phog": range(399, 439),
          "jcd": range(439, 607),
          "gabor": range(607, 667)
})
```

Figure 4-2: Sample data structure of a reference file (*.meta* file) for a feature space defined in Python.

In the above definition, "cedd" features are from index 0 to index 143th; "color" features are from index 144 to index 206th, and so forth. The filename format for this file type is "db_name.meta".

4.2.2 Database Storage Structure

For a dataset of N records represented by d-dimensional feature space, a Bamboo Forest database consists of one metadata file (*.pi* file) and one reference file (*.meta* file) and d feature index files (*.t* files).

The database is stored in a directory (or a folder) of the file system of a computer. The metadata file and index files will be stored as binary files and written by data blocks so that they can be accessed using random access of the computer's file system. The reference file will be stored as plain text files. However, in order to access the dictionary of a reference file more efficiently, it is recommended to use a serialization method of a programming language to translate each dictionary to a byte stream that

can be stored as a binary file in the file system and reconstructed easily when needed [97]. In this thesis, the *pickle*[1] Python module will be used in experiments in chapters 6 and 7 as a serialization method for the reference files.

Figure 4-3 shows a sample tree view of a Bamboo Forest database directory. The root of the tree is the path to the database directory on storage disk. The directory contains a *.meta* file, a *.pi* file and many index *.t* files.

```
/Volumes/chupi/pidb/500k
| ---------- 500k.meta
| ---------- 500k.pi
| ---------- 500k_JCD_1.t
| ---------- 500k_JCD_2.t
| ---------- 500k_JCD_3.t
| ---------- 500k_JCD_4.t
| ---------- 500k_JCD_5.t
| ---------- 500k_JCD_6.t
| ---------- 500k_JCD_7.t
| ---------- 500k_JCD_8.t
| ---------- 500k_JCD_9.t
| ---------- 500k_CEDD_1.t
| ---------- 500k_CEDD_2.t
| ---------- 500k_CEDD_3.t
| ---------- 500k_CEDD_4.t
| ---------- 500k_CEDD_5.t
| ---------- 500k_CEDD_6.t
| ---------- 500k_CEDD_7.t
| ---------- 500k_CEDD_8.t
| ---------- 500k_CEDD_9.t
| ---------- 500k_FCTH_1.t
| ---------- 500k_FCTH_2.t
| ---------- 500k_FCTH_3.t
| ---------- 500k_FCTH_4.t
| ---------- 500k_FCTH_5.t
| ---------- 500k_FCTH_6.t
| ---------- 500k_FCTH_7.t
| ---------- 500k_FCTH_8.t
| ---------- 500k_FCTH_9.t
```

Figure 4-3: A tree-view directory structure of a Bamboo Forest database with a metadata file (*.pi*), a reference file (*.meta*), and some sample index files (*.t*).

[1] https://docs.python.org/2/library/pickle.html

4.3 Data Insertion

4.3.1 Single Insertion

When a datum is inserted to the database, its feature vector is extracted. Then, it will request for an unique identifier *rid* in the database. This *rid* is unique by setting it equal the maximum *rid* plus one. Its data block including this *rid*, its information and the feature vectors in bytes will be appended to the *.pi* file of the database.

In the next step, each element of the feature vector will be written at an appropriate position in the corresponding *.t* file. The position is found by a binary search algorithm so that when written to the index file, the orders of values are sorted. In the experiments, we use an ascending order.

4.3.2 Batch Insertion

Instead of inserting one by one data into a database, we can index all the dataset at once if we are given the whole dataset. In this case, an algorithm to generate unique identifiers for each datum will be applied. We can write the batch of metadata directly to a *.pi* file. As in Figure 4-1, each row vector will be extracted, combing with the *rid* row then sorted and write to a *.t* file. Moreover, a parallel indexing process can be applied for each feature index.

4.4 Big-O Complexity Analysis of Bamboo Forest Database

For indexing a dataset of N records in d-dimensional feature space using the indexing method described in the previous section 4.2, the Bamboo Forest database requires d files of inverted lists. This leads to the space complexity of $O(dN)$ of the database.

For indexing a new record to the database, it takes $O(1)$ to insert into the metadata *.pi* file and $O(\log N)$ to insert into each index *.t* file. In total, the complexity of insertion is $O(d \log N)$.

Table 4.3 summaries the complexity of the Bamboo Forest indexing method.

Table 4.3: Big-O complexity of the Bamboo Forest indexing method

Space complexity	$O(dN)$
Insertion complexity	$O(d \log N)$

Chapter 5

Search Algorithms on Bamboo Forest Database

> Everything is theoretically
> impossible, until it is done.
>
> *Robert A. Heinlein*

Given a Bamboo Forest database indexed by the indexing method proposed in chapter 4, we define further three search algorithms that work compatibly with the general search architecture described in chapter 3.

The first algorithm is named "Maxfirst" algorithm, which corresponds to the context where the *intention* preference is "dominant". The second search algorithm is named "Combinatorial" algorithm, which corresponds to the context where the *intention* preference is "similar". The third search algorithm is named "Exact-match" algorithm, which corresponds to the context where the *intention* preference is "exact".

Table 5.1 shows the classification of contexts in which each proposed search algorithm can be applied. The content preferences are introduced following the feature types of image data which are used in the experimental application system in chapter 7. They include "color", "shape", "texture" or combination of those features. However, these content preferences are not limited to these types. Any feature type that satisfies the scope of content discussed in section 1.3.2 is applicable to the search al-

gorithms. The combination of two or more feature types can be done using combining functions in Table 6.2 in section 6.3.3.

Table 5.1: Classification of applicable contexts of three proposed search algorithms.

Contextual preference		**Maxfirst search**	**Combinatorial search**	**Exact-match search**
Content	Color	Yes	Yes	Yes
	Shape	Yes	Yes	Yes
	Texture	Yes	Yes	Yes
	Combination	Yes	Yes	Yes
Intention	"dominant"	Yes	No	No
	"similar"	No	Yes	Yes
	"exact-match"	No	No	Yes
Response time	limited	Yes	Yes	No
	unlimited	Yes	Yes	Yes
	interruptible	Yes	Yes	No

It is also to note that, the "interruptible" preference regarding the response time limits is introduced in Table 5.1 refers to an interactive behavior of a user to abort the search process at anytime it is running. This interaction between users and search systems can provide an interesting usability in which the users of search systems can receive "better" results while the systems are running and decide at their will when to stop. Although, this is not shown in the pseudocodes of the proposed search algorithms in the following sections, it is possibly implemented by modifying the main *while* loop of each appropriate algorithm.

5.1 Maxfirst Search Algorithm

The pseudocode of Maxfirst search algorithm compatible with a Bamboo Forest database is described in Algorithm 5.1. This algorithm uses the search intuition in Figure 3-7. Note that in this algorithm, *relevant* calculation in line 15 of Algorithm 7.6 between a candidate's feature vector and query vector is calculated as the "sum" of its feature corresponding to prioritized features. Additionally, the ranking of candidates respect to a query is decided by an descending order of those sum scores.

Algorithm 5.1 Maxfirst Search Algorithm

Input:

- A query vector $q \in \mathbb{R}_+^d$;
- A Bamboo Forest database $\mathbf{B}$ that indexes a dataset $\mathbf{X} \subset \mathbb{R}_+^d$;
- A preference vector $p \in \mathbb{R}_+^d$ such that $p_i = 1$ if i-th feature is preferred and $p_i = 0$ otherwise;
- Optional: Maximum number of priorities m, by default $m = 5$;
- Optional: Time limit t, by default $t = 1(second)$;
- Optional: Number of results R, by default, $R = 20$.

Output: Top R ranked relevant $x \in \mathbf{X}$.

[*Step 1: Reflect content preference onto query*] Set $q \leftarrow q \odot p$ where $\odot$ is the elementwise multiplication or Hadamard product.
[*Step 2: Prioritize features*] A permutation function σ of the set $\{1, \ldots, d\}$ that sorts elements of vector q in a descending order: $\big(q_{\sigma(1)} \geq \ldots \geq q_{\sigma(d)}\big)$.
[*Step 3: Initialize starting points*] Set reading offset of $.t$ files in $\mathbf{B}$ corresponding to features $\sigma(1), ..., \sigma(m)$ is 0. ▷ beginning of file
[*Step 4: Find and rank candidates*]
List of candidates $\mathbf{C} \leftarrow \{\}$
while time limit t is not reached **do**
    for i **from** 1 **to** m **do**
        $S \leftarrow$ Get candidate identifiers from next lower section of a $.t$ file corresponding to feature $\sigma(i)$.
        Update reading offset of this $.t$ file.
        for each $c \in S$ **do**
            $u \leftarrow$ feature vector of candidate c from the $.pi$ file in $\mathbf{B}$.
            $score \leftarrow \sum_{i=1}^{m} u_{\sigma(i)}$.
            Update $\mathbf{C}$ with $(c, score)$.
        end for
        Sort $\mathbf{C}$ by a descending order of *scores*.
    end for
end while
[*Step 5: Return top results*]
return top R of $\mathbf{C}$ to users.

Algorithm 5.2 Combinatorial Search Algorithm

Input:

- A query vector $q \in \mathbb{R}_+^d$;
- A Bamboo Forest database $\mathbf{B}$ that indexes a dataset $\mathbf{X} \subset \mathbb{R}_+^d$;
- A preference vector $p \in \mathbb{R}_+^d$ such that $p_i = 1$ if i-th feature is preferred and $p_i = 0$ otherwise;
- Optional: Maximum number of priorities m, by default $m = 5$;
- Optional: Timelimit t, by default $t = 1(second)$;
- Optional: Number of results R, by default, $R = 20$.

Output: Top R ranked relevant $x \in \mathbf{X}$.

1: [*Step 1: Reflect content preference onto query*] Set $q \leftarrow q \odot p$ where $\odot$ is the elementwise multiplication or Hadamard product.
2: [*Step 2: Prioritize features*] A permutation function σ of the set $\{1, \ldots, d\}$ that sorts elements of vector q in a descending order: $\left(q_{\sigma(1)} \geq \ldots \geq q_{\sigma(d)}\right)$.
3: [*Step 3: Initialize starting points*]
4: **for** i **from** 1 **to** m **do**
5: Using binary search on a *.t* file in $\mathbf{B}$ corresponding to feature $\sigma(i)$ to find the offset of the data block which has the value closest to $q_{\sigma(i)}$ and set this offset as the current reading offset of the *.t* file.
6: **end for**
7: [*Step 4: Find and rank candidates*]
8: List of candidates $\mathbf{C} \leftarrow \{\}$
9: **while** time limit t is not reached **do**
10: **for** i **from** 1 **to** m **do**
11: $S \leftarrow$ Get candidate identifiers from next higher and lower section of the *.t* file corresponding to feature $\sigma(i)$.
12: Update current reading offset of this *.t* file.
13: **for each** $c \in S$ **do**
14: $u \leftarrow$ feature vector of candidate c from the *.pi* file in $\mathbf{B}$.
15: $score = COSINE(q, u \odot p)$. ▷ see sections 3.3.2, 3.3.2.4
16: Update $\mathbf{C}$ with $(c, score)$.
17: **end for**
18: Sort $\mathbf{C}$ by an ascending order of *scores*
19: **end for**
20: **end while**
21: [*Step 5: Return top results*]
22: **return** top R of $\mathbf{C}$ to users.

5.2 Combinarorial Search Algorithm

The pseudocode of Combinatorial search algorithm compatible with a Bamboo Forest database is described in Algorithm 5.2. This algorithm uses the search intuitions in Figures 3-3 and fig:expand. In this algorithm, the reading offsets of $.t$ files are kept at both directions: going higher and lower finding for $.s$ section files. Also in this algorithm, the similarity calculation is by default set to Cosine similarity, but it can always be reconfigured. The ranking of candidates respect to a query is decided by an ascending order of similarity scores.

5.3 Exact-match Search Algorithm

The pseudocode of Exact-match search algorithm compatible with a Bamboo Forest database is described in Algorithm 5.3. This algorithm uses the search intuition in Figure 3-6. In this algorithm, the reading offsets of $.t$ files are kept at both directions: going higher and lower finding for $.s$ section files like in the case of Combinatorial search algorithm. However, using this algorithm, response time is not controllable. The algorithm runs until it finds at least one candidate that is returned by an intersection operation of a list of sets of candidates from repeatedly expanding sections on prioritized $.t$ files. In this algorithm, the similarity calculation is also by default set to Cosine similarity, but it can always be reconfigured too. The ranking of candidates respect to a query is decided by an ascending order of similarity scores. But only one top result is returned to users.

This algorithm does not guarantee the just of the returned result is an exact match to a query if m is set other than the default value. The default value $m = \sum_{i=1}^{d} p_i$ indicates that all the prioritized features will be used to find the candidates. When m is set other than this value, however, the results are guaranteed with only some probability to be the right match. In section 7.5.3, the effects of setting m on the performance of the algorithm will be discussed in details.

Algorithm 5.3 Exact-match Search Algorithm

Input:

- A query vector $q \in \mathbb{R}_+^d$;
- A Bamboo Forest database $\mathbf{B}$ that indexes a dataset $\mathbf{X} \subset \mathbb{R}_+^d$;
- A preference vector $p \in \mathbb{R}_+^d$ such that $p_i = 1$ if i-th feature is preferred and $p_i = 0$ otherwise;
- Optional: Maximum number of priorities m, $m \geq 2$. By default, all positive features q_i with respect to a subspaced selected by p will be used, meaning $m = \sum_i^d \lceil q_i \rceil \times p_i$.

Output: Top R ranked relevant $x \in \mathbf{X}$.

1: [*Step 1: Reflect content preference onto query*] Set $q \leftarrow q \odot p$ where $\odot$ is the elementwise multiplication or Hadamard product.
2: [*Step 2: Prioritize features*] A permutation function σ of the set $\{1, \ldots, d\}$ that sorts elements of vector q in a descending order: $\left(q_{\sigma(1)} \geq \ldots \geq q_{\sigma(d)}\right)$.
3: [*Step 3: Initialize starting points*]
4: **for** i **from** 1 **to** m **do**
5: Using binary search on a *.t* file in $\mathbf{B}$ corresponding to feature $\sigma(i)$ to find the offset of the data block which has the value closest to $q_{\sigma(i)}$ and set this offset as the current reading offset for the *.t* file.
6: **end for**
7: [*Step 4: Find candidates*]
8: List of candidates $\mathbf{C} \leftarrow \{\}$
9: **while** $\mathbf{C}$ is empty **do**
10: Initialize m empty sets $T_i \leftarrow \{\}$ for all $i \in [1, \ldots, m]$
11: **for** i **from** 1 **to** m **do**
12: $S_i \leftarrow$ Get candidate identifiers from next higher and lower section of a *.t* file corresponding to feature $\sigma(i)$.
13: $T_i \leftarrow T_i \cup S_i$.
14: **end for**
15: $C \leftarrow \bigcap_{i=1}^m T_i$.
16: **end while**
17: [*Step 5: Calculate similarity*]
18: **for each** $c \in \mathbf{C}$ **do**
19: $u \leftarrow$ feature vector of candidate c from the *.pi* file in $\mathbf{B}$.
20: $score = COSINE(q, u \odot p)$. ▷ see sections 3.3.2, 3.3.2.4
21: Update $c \in \mathbf{C}$ with $(c, score)$
22: **end for**
23: [*Step 6: Sort candidates and return one top result*]
24: Sort $\mathbf{C}$ by an ascending order of *scores*
25: **return** top 1 of $\mathbf{C}$ to users.

5.4 Big-O Complexity Analysis of Search Algorithms

Three search algorithms have been introduced. This section will discuss their complexity using big-O notation.

Given a Bamboo Forest database **B** which indexes N data for d features (N d-dimension data) and given an input query vector, each proposed search algorithm firstly prioritizes the features in the query vector based on their values, then conducts the search process on m prioritized feature indices ($m \leq d$), which are feature indices which have largest values. Also assume that the cost to calculate relevance score, either $COSINE(x, y)$ or sum, and the cost of a random access to a position of the binary file are $O(1)$, the big-O complexity analysis of three proposed search algorithms are shown in Table 5.2.

Table 5.2: Big-O complexity of three proposed search algorithms

Algorithm	Complexity
Maxfirst search	$O(m + C)$
Comibnatorial search	$O(m \log N + C)$
Exact-match search	$O(m \log N + C)$

Maxfirst search algorithm For each feature index, the Maxfirst search algorithm (algorithm 5.1) obtains the first candidate at the first datab block in the corresponding $.t$ file. Since reading this block costs $O(1)$, the cost for obtaining m first candidates for m prioritized features are $O(m)$. The Maxfirst search algorithm continues to find candidate by sequentially read each data block from the starting position until the input time limit is reached. If C is the total number of candidates can be found during this time limit, the complexity of the algorithm is evaluated as $O(k + C)$. If the time limit is set to infinity (unlimited search time), C can be very large ($C \simeq N$), then the complexity of the algorithm can be considered as linear.

Combinatorial search algorithm For each feature index, the Combinatorial search algorithm (algorithm 5.2) needs $O(m \log N)$ to find the first candidate then sequentially reads the index file to the next candidates. If C also denotes the number of

candidates found during the input time limit, the complexity of the algorithm is $O(k \log N + C)$. As also noted in analysis of the Maxfirst search algorithm, if the time limit is large, C can be very large and the complexity of the algorithm becomes linear.

Exact-match search algorithm Similarly to the Combinatorial search algorithm, the Exact-match search algorithm (algorithm 5.3) also have $O(m \log N + C)$ as its complexity. However, the time limit can not be set in this algorithm, and C can be smaller than C of the Maxfirst search algorithm or of the Combinatorial search algorithm.

The big-O complexity of three proposed search algorithms are summarized in Table 5.2. In section 7.5.7 in chapter 7, this complexity analysis will be revisited and demonstrated by experimental results.

Chapter 6

Adaptive Color-based Image Search with Dynamic Query Creation

> Logic will get you from A to Z;
> imagination will get you everywhere.
>
> *Albert Einstein*

Color plays an important role in our world. "The concept of colour is one of the biggest concepts in our life. Every colour has different effects on our psychology..." [98]. The study of color and its effect as well as its applications on our life has been started very early such as color and its effect on psychological states of human including perception, impression, imagination, mood [21, 99, 98, 100, 101] or its applications on design, art and industry [102, 103]. In computer vision, particularly in content-based image retrieval (CBIR), color-based techniques are the most well-known and applicative. Despite the importance of color, most of the approaches in CBIR give up on the delicate variations of color in order to achieve fast extracting and compact indexing. However, when the users' contexts and preferences are valued higher than those technical performance, it obviously demands a systematic method to utilize the fine ranges in color spaces.

This thesis proposes such a system, which extracts and indexes refined color information of an image in a systematic way and then adapts to users' contexts and

imaginations to conduct relevant searches.

6.1 Image Datasets

The image dataset used for experiments in this chapter is the painting dataset[1] provided by the *Visual Geometry Group* at Department of Engineering Science, University of Oxford [104]. This dataset is a collection of works from the "Your painting" project at BBC UK, which contains 8,629 modern art paintings with several genres as shown in Table 6.1.

Table 6.1: Description of Crowley's painting dataset (from "Your paintings" project at BBC UK) [104].

	Train	**Validation**	**Test**	**Total**
Aero	74	13	113	**200**
Bird	319	72	414	**805**
Boat	862	222	1059	**2143**
Chair	493	140	569	**1202**
Cow	255	52	318	**625**
Dtable	485	130	586	**1201**
Dog	483	113	549	**1145**
Horse	656	127	710	**1493**
Sheep	270	76	405	**751**
Train	130	35	164	**329**
Total	3463	865	4301	**8629**

Copyright information: `https://artuk.org/footer/copyright-notice-15`.
Download link: `http://www.robots.ox.ac.uk/~vgg/data/paintings/`.

6.2 Color Space Sampling and Color Feature Extraction

A hierarchical indexing model of HSV color space is proposed. The HSV color space consists of hue 'H', intensity value 'V', and saturation 'S' [105] which is often represented as a hexacone in three dimensional space. Hue is described with the words

[1]Thanks Dr. Elliot J. Crowley for giving me the copyright information.

that we normally think of as describing colors: red, purple, blue, etc. and also a term describing a dimension of colors we readily experience when we look at colors. Intensity value refres to how light or dark a color is. Saturation refers to the dominance or purity of hue in the color. A common HSV color space defines the values of H, S, V are in range of (0, 360), (0, 100), and (1, 100), respectively.

Based on this nature of the HSV color space, the sampling method will walk around the hue wheel, go according to the outer edge and move toward the center of the wheel. During this sampling path, it can pick up chromatic colors with pure hue values which are more vivid than the colors with the same hue values but closer to the central vertical axis. If the sampling pointer walks through the central axis of the wheel, achromatic colors (grayscale) colors are picked up. In this thesis, only chromatic colors are treated.

Figure 6-1 shows the illustration of the sampling process. At level 0 of the tree, it is the HSV color space (a hexacone). At level 1, it is the common color names. These are the most seven common color names in English language [98] and undoubtedly in many other languages. At next levels, the saturation and intensity value are sampled continuously.

This thesis focuses on the chromatic colors and features the hue wheel more than saturation and intensity value, so that the sampling method will sample the hue wheel, saturation and intensity value into 180, 5, and 5 values returning total 180*5*5 = 4500 colors. This set of colors will be used to extract a color feature vector of an image. The process of extracting a color feature vector, which is also known as a color histogram is described in Algorithm 6.1. This algorithm is based on the idea of color indexing which has been proposed earlier by [106] and become a conventional method for extracting color histograms of color images in the literature. Swain and Ballard [106, p. 13] defines the process to obtain a color histogram as follows: "Given a discrete color space defined by some color axes (e.g., red, green, blue), the color histogram is obtained by discretizing the image colors and counting the number of times each discrete color occurs in the image array." Using this algorithm and the above sampled color set, the extracted color feature vector has 4500 elements.

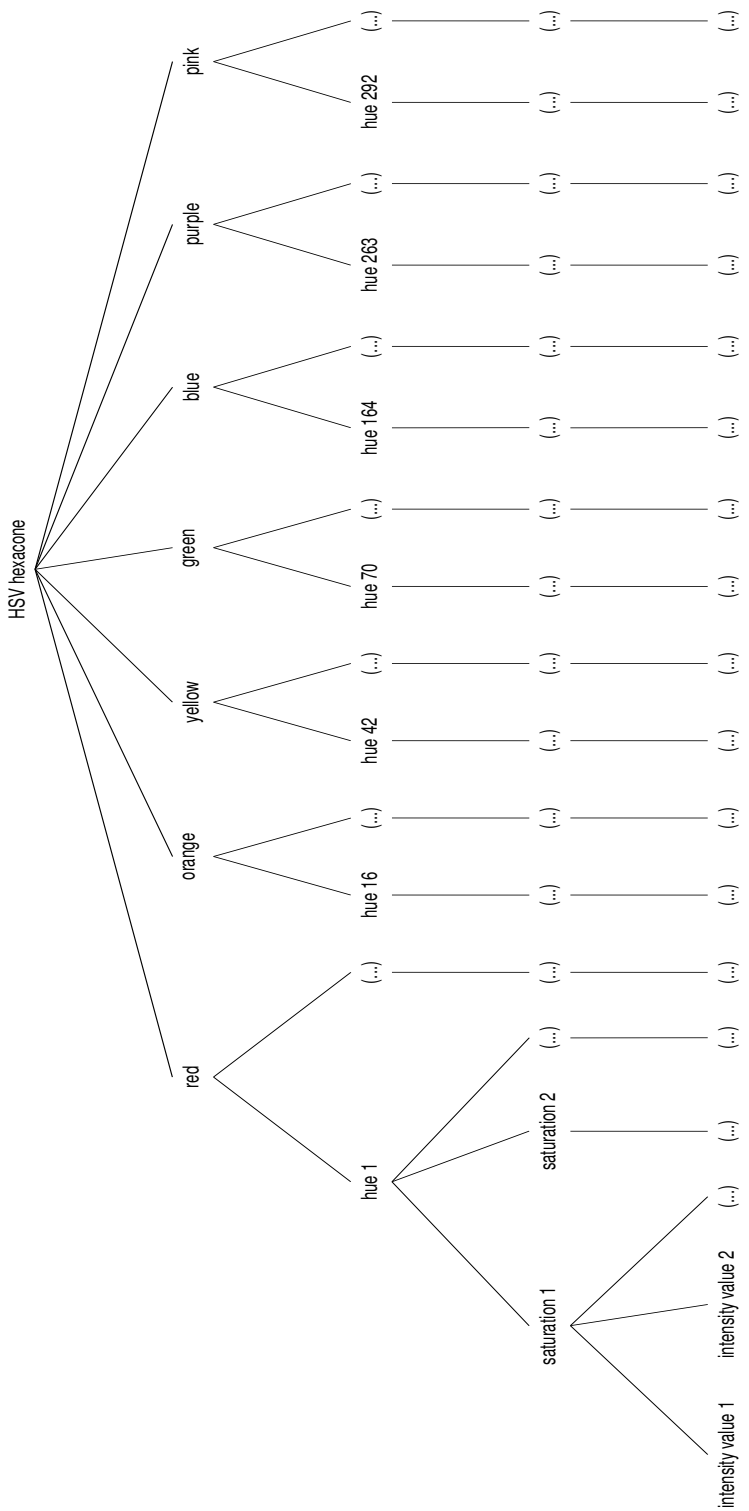


Figure 6-1: Illustration of the tree-shaped hierarchical model of HSV color space sampling method for feature indexing. The tree has three non-root levels: name of basic colors, hues, saturations, and intensity values.

Algorithm 6.1 Conventional Color Histogram Extraction Algorithm

Input:

- A RGB color image I;
- A list of n colors $C = [c_1, c_2, \ldots, c_n] \subset \mathbb{R}^3$ where c_i is a color in a color space **C**.

Output: A n-element frequency feature vector (i.e., histogram) $h \in \mathbb{R}^n_+$.

1: [*Initialize a list to store the histogram*] $h \leftarrow []$.
2: **for each** pixel p in the image I **do**
3: [*Project pixel to the color space*] $cp \leftarrow$ color of p in the color space **C**.
4: [*Find index of a color which is closest to the pixel*]
$i \leftarrow \arg\min_i L1(c_i, cp)$. ▷ Equation 3.5 in section 3.3.2.5
5: $h[i] \leftarrow h[i] + 1$.
6: **end for**
7: [*Normalize the histogram*]
$h \leftarrow NORMALIZE(h)$. ▷ Equation 3.6 in section 3.3.2.5
8: **return** h.

It is fistly to note that there are also other methods besides the method in Algorithm 6.1 for fast color histogram extraction in the literature. Those methods apply several techniques such as "fuzzy color histogram" [107], or "integral histogram" [108]. Finally, there are also existing libraries for calculating color histograms of image data such as OpenCV[2] in Python, C, C++, and Java (the *calcHist* function), or JFeatureLib in Java [3].

6.3 Adaptive Query Creation

6.3.1 Context-based Color Subspace Selection

Content preferences as contextual preferences will be treated in this section. Each preference will be expressed in a color name, which is a combination of a word describing the vibrancy of preferred colors ("vivid" or "light" or "" (void)) and a basic color name from a set of basic color names at level 1 of the tree in Figure 6-1: "red", "orange", "yellow", "green", "blue", "purple", and "pink". This preference is denoted as

[2]http://opencv.org/
[3]https://jfeaturelib.googlecode.com

$color = (vibrancy, colorname)$.

Given a content preference *color* and a feature vector q, Algorithm 6.2 calculates the adapted feature vector q'. Note that the index of a vector starts from 0. In this algorithm (lines 2-8), the range of hue values for color names are defined in a non-uniform way. This is based on a psychologocal result[4]. The range of "green" or "blue" colors are larger than of "purple" or "pink".

Moreover, it is argued that lower intensity valued and lower saturated colors are treated perceivably as achromatic colors in human eyes for a given hue. Algorithm 6.2 reflects this in the settings of *vibrancy* ("vivid" or "light" or nothing) of colors as from line 10 to line 18. For "vivid" colors, only high saturations and high intensity values are chosen since they are more *pure* to the corresponding hues. For "light" colors, low saturations and high intensity values are chosen. Very low intensity values are ignored since the corresponding colors would be very dark (black) if the saturations are high, or very bright (white) if the saturations are low. If there is no setting for *vibrancy*, the colors are chosen as combinations of both vibrant values. In this case, very low intensity values are ignored.

6.3.2 Adaptive Color-based Query Creation

Given a dataset of images, their features vectors are extracted based on the color set defined by the sampling method described in section 6.2. Each feature vector is a vector in 4500-dimensional feature space. At query time, given an input image and a content preference *color*, Algorithm 6.2 computes and returns the top relevant images from the dataset to users. In principle, this algorithm selects a subspace of feature space ($\mathbb{R}_+^{4500}$) and then applies similarity calculation on that subspace. Semantically, it reflects the idea of using preferred subspaces while ignoring unrelated outer spaces in order to make concise calculation. From a technical point of view, this selection of subspace not only provides the ability to deal with delicate contents and dynamic contexts, but also alleviates the "curse of dimensionality" in high-dimensional space ($\mathbb{R}_+^{4500}$ in this case).

[4]https://vasilis.nl/nerd/code/human-colours/tests/hue-en-gb.php

Algorithm 6.2 Adaptive Color Subspace Selection Algorithm

Input:

- A feature vector $q \in \mathbb{R}_+^{4500}$;
- A content preference $color = (vibrancy, colorname)$.

Output: Adapted featured vector $q' \in \mathbb{R}_+^{4500}$.
Procedure: $ADAPT(q, color)$

[*Dictionary of hue indices*]
Hues['red'] $\leftarrow [1, \ldots, 7, 160, \ldots, 179]$
Hues['orange'] $\leftarrow [8, \ldots, 20]$
Hues['yellow'] $\leftarrow [21, \ldots, 34]$
Hues['green'] $\leftarrow [35, \ldots, 81]$
Hues['blue'] $\leftarrow [82, \ldots, 130]$
Hues['purple'] $\leftarrow [131, \ldots, 145]$
Hues['pink'] $\leftarrow [146, \ldots, 159]$
[*Indices of saturations and values*]
if *vibrancy* = "vivid" **then**
    saturations $\leftarrow [3, 4]$
    values $\leftarrow [2, 3, 4]$
else if *vibrancy* = "light" **then**
    saturations $\leftarrow [1, 2]$
    values $\leftarrow [3, 4]$
else
    saturations $\leftarrow [1, 2, 3, 4]$
    values $\leftarrow [1, 2, 3, 4]$
end if
[*Generate content preference vector* $p \in \mathbb{R}_+^{4500}$]
indices $\leftarrow \{\}$
for h **in** Hues[*colorname*] **do**
    for s **in** saturations **do**
        for v **in** values **do**
            $i \leftarrow h \times 25 + s \times 5 + v$
            indices $\leftarrow$ *indices* $\cup \{i\}$
        end for
    end for
end for
$v \leftarrow [v_1, \ldots, v_{4500}]$ s.t $v_i = 1.0$ if $i \in$ *indices*
[*Return adapted feature vector*]
$q' \leftarrow q \odot p$ where $\odot$ is the elementwise multiplication or Hadamard product.
return q'

Algorithm 6.3 Adaptive Color-based Query Creation and Search Algorithm

Input:

- A feature vector $q \in \mathbb{R}_{+}^{4500}$;
- A dataset $\mathbf{X} \subset \mathbb{R}_{+}^{4500}$;
- A content preference $color = (vibrancy, colorname)$;
- Optional: Number of results R, by default, $R = 20$.

Output: Top R ranked relevant $x \in \mathbf{X}$.

[*Compute adapted feature vector*]
$q' \leftarrow ADAPT(q, color)$. ▷ Algorithm 6.2
$q' \leftarrow NORMALIZE(q')$ ▷ Equation 3.6 in section 3.4.2
[*Calculate similarity*]
List of targets $\mathbf{C} \leftarrow \{\}$
for $x \in \mathbf{X}$ **do**
    $x' \leftarrow ADAPT(x, color)$
    $x' \leftarrow NORMALIZE(x')$
    $score = similarity(q', x')$
    Update $\mathbf{C}$ with $(x, score)$.
end for
Sort $\mathbf{C}$ by an ascending order of scores.
return Top R of $\mathbf{C}$

In this algorithm, the *similarity* calculation can implement any of similarity functions described in section 3.3.2. However, the choice of a similarity calculation can affect the performance of the search algorithm. Experimental results in section 6.4 will show the cases.

6.3.3 Imagination-based Query Creation

A user's imagination can be thought as a combination of many existing pieces of visual information in which each piece can be extracted from a picture by specifying the content preference in the picture. Using the proposed adaptive color subspace selection function described in section 6.3.1, an imagination-based query creation method is introduced. This method extends the earlier proposed methods that have been published in [43, 44, 48].

The idea of designing an imagination-based query creation method is to apply

some binary functions to feature vectors of input images to construct a combined feature vector that can express the "imagination" of a user when searching for relevant images. Many previous researches have proposed such combining functions for creating search queries for image data [41, 42, 45, 46].

In this thesis, a set of functions that extend functionality of query processing of the adaptive color-based search algorithm in section 6.3.2 is introduced. Given a set of input images with corresponding feature vectors $\{f_i\}$ (section 6.2) and a set of respective content preferences $\{color_i\}$, we have a set $\mathbf{X} = \{(f_i, color_i) \mid f_i \in \mathbb{R}_+^{4500}, color_i = (vibrancy_i, colorname_i)\}$.

Applying the adaptive function in Algorithm 6.2 to each pair $(f_i, color_i) \in \mathbf{X}$, we obtain a set $\mathbf{X}' = \{f_i'\}$. The following binary functions are used to combine those vectors into one vector. Each binary function takes two vectors to produce a new vector with the same number of dimensions.

1. $PLUS(x, y)$

   This function produces an element-wise sum of two vectors x and y. The produced vector represents an imaginative image which contains all characteristic features of the two images which correspond to x and y.

$$PLUS(x, y) = (x_1 + y_1, x_2 + y_2, \ldots, x_d + y_d) \tag{6.1}$$

2. $INTERSECTION(x, y)$

   This function produces a vector of element-wise minimum values from two vectors x and y. The produced vector represents an imaginative image which contains characteristic features that exist in both two images of vectors x and y.

$$INTERSECTION(x, y) = (min(x_1, y_1), min(x_2, y_2), \ldots, (x_d, y_d)) \tag{6.2}$$

3. $UNION(x, y)$

   This function produces a vector of element-wise maximum values from two

vectors x and y. The produced vector represents an imaginative image which contains characteristic features that exist in either of the two images of vectors x and y.

$$UNION(x, y) = (max(x_1, y_1), max(x_2, y_2), \ldots, max(x_d, y_d)) \tag{6.3}$$

4. $MINUS(x, y)$

   This function produces an element-wise minus of two vectors x and y. The produced vector represents an imaginative image which contains reduced characteristic features of the image of x by them of the image of y.

$$MINUS(x, y) = (max(0, x_1 - y_1), max(0, x_2 - y_2), \ldots, max(0, x_d - y_d)) \tag{6.4}$$

5. $DIFFERENCE(x, y)$

   This function produces a vector of element-wise contrasting values from two vectors x and y. The produced vector represents an imaginative image which contains characteristic features that exist in the image of x but not in the image of y.

$$DIFFERENCE(x, y) = (x_1 \times \phi(y_1), x_2 \times \phi(y_2), \ldots, x_d \times \phi(y_d)) \tag{6.5}$$

where $\phi(a)$ is a function on real numbers, $\phi(a) = \begin{cases} 0 & \text{if } a = 0 \\ 1 & \text{if } a > 0 \end{cases}$.

It is to note that a vector produced by a binary function introduced above is in general not a frequency vector (section 3.4.1). It is recommended to apply the $NORMALIZE$ function (equation 3.6) to the produced vector before using it to combine it with other vectors. However, it also depends on the desired semantic of the combining operation.

Besides the above five binary functions, two unary functions $LINEAR\_SCALE$ and $EQUALIZE$ (sections 3.4.2 and 3.4.3) can be applied for a feature vector in

order to express an imagination of the preferred search results.

Table 6.2 lists the applicable functions to express users' imagination at query time and their properties. In this table, the "commutative" property of a binary function $\oplus(x, y)$ indicates that the function satisfies $\oplus(x, y) = \oplus(y, x)$, meaning the order of input vectors does not affect the produced vector. The "associative" property of a binary function $\oplus(x, y)$ indicates that the function satisfies $\oplus(x, \oplus(y, z)) = \oplus(\oplus(x, y), z)$, meaning the order of applying the functions to a set of input vectors does not affect the produced vector. These properties give a guide to precisely express orders of inputs and functions in order to obtain the desired results.

It is also to note that, regardless of the theoretical algebra, the addition and multiplication of floating point numbers are not necessarily associative due to "roundoff errors" [109].

Table 6.2: Imagination-based query creation functions and their properties.

Query manipulation function	Type	Properties
$PLUS$	binary	Commutative, associative*
$INTERSECTION$	binary	Commutative, associative
$UNION$	binary	Commutative, associative
$MINUS$	binary	Non-commutative, non-associative
$DIFFERENCE$	binary	Non-commutative, associative*
$NORMALIZE$	unary	
$LINEAR\_SCALE$	unary	
$EQUALIZE$	unary	

Note for (*): this property of a function holds in theory but does not necessarily hold in practice when applying for floating-point numbers (refer to [109]).

6.4 Experiments

In this section, several experimental studies using the adaptive color-based and imagination-based query creation functions described in section 6.3 are discussed using the Crowley's painting dataset (see Table 6.1) in section 6.1.

6.4.1 System Implementation and Experiment Setting

6.4.1.1 System Implementation

The image search system is implemented using using Python programming language version 2.7.12 on MacBook Air computer with specifications as follows: 1.7 GHz Intel Core i7, 8 GB 1600 MHz DDR3 with OXS Yosemite operating system. The detailed description and performance benchmarks of this computer is in Table 7.2 in section 7.2.2. The Python interface of OpenCV library framework is used to pre-process images and extract color feature vectors.

Two widely used color-based feature extraction methods are also implemented in order to compare with the proposed adaptive color feature extraction and indexing method. The first method uses a set of 130 basic colors sampled from the Munsell color system to represent an image. The descriptions for these colors can be found in [101]. The second method uses a set of $7 \times 3 \times 3 = 63$ colors sampled from the HSV color space. These method applies Algorithm 6.1 to extract the feature vectors of image data in 130- and 63-dimensional feature spaces, respectively.

Seven distance functions described in section 3.3.2 are also implemented in order to study the effectiveness of a combination of a feature extraction and similarity calculation methods.

6.4.1.2 Experiment Setting

To study the effectiveness of the adaptive color-based query creation method, for each *colorname* consisted of a basic color *color* among 7 colors, and a level of *vibrancy* (3 levels), five images are randomly chosen as input to the system. For each input, one of seven different distance functions will be used when calculating the similarity scores. As a result, for this setting, we have a total of $7 \times 3 \times 5 \times 7 = 735$ queries.

To compare the proposed extraction method to other methods which use lesser number of dimensions (proposed adaptive feature ($d = 4500$), Munsell color feature ($d = 130$), and HSV color feature $d = 63$), seven images are randomly chosen as input to the system. Also, seven similarity distances are used for each input. Consequently,

for this setting, we have a total of $3 \times 7 \times 7 = 147$ queries.

In summary, we have total $735 + 147 = 882$ queries and the top 20 results of each query will be evaluated manually to assess the precision of the results. A brute force algorithm, which is similar to Algorithm 7.6 in section 7.5.2.2 is used to search for the most relevant images given a query from the image dataset. The precision is defined as the percentage of correct results among 20 results returned by the search algorithm.

6.4.2 Adaptive Color-based Search

6.4.2.1 Precision Statistics

Figure 6-2 shows the average precision of different combinations of features and distance functions. In this figure, we can see that overall precision of the proposed adaptive color-based image search method performs better than other "non-adaptive" color-based image search methods. It is also seen that the average precisions depend on the distance functions as expected but Chi square and Cosine distances yield overall higher precisions. Comparing to three non-adaptive search results including the one returned by using the whole 4500 color features, adaptive search returns reasonable high precisions. It is noteworthy that the Munsell color based method uses a relative low number of colors ($d = 130$) but returns the lowest performance.

In Figure 6-2, we also see the performance of the adaptive color-based search method drops slightly for queries of "purple" and "pink" color names. This can be explained by two reasons. The first reason is that the target dataset has a few number of images with "purple" and "pink" colors. The second reason is based on the intrinsic distinguishability of "purple" to "blue" colors and of "pink" to "red" colors. Since the precisions are manually checked, the appearance of a color on actual images can slightly changed depending on which colors it is with. To solve this problem, a further investigation on the sampling of hue, saturation and intensity values in Algorithm 6.2 is required and planned as a future work.

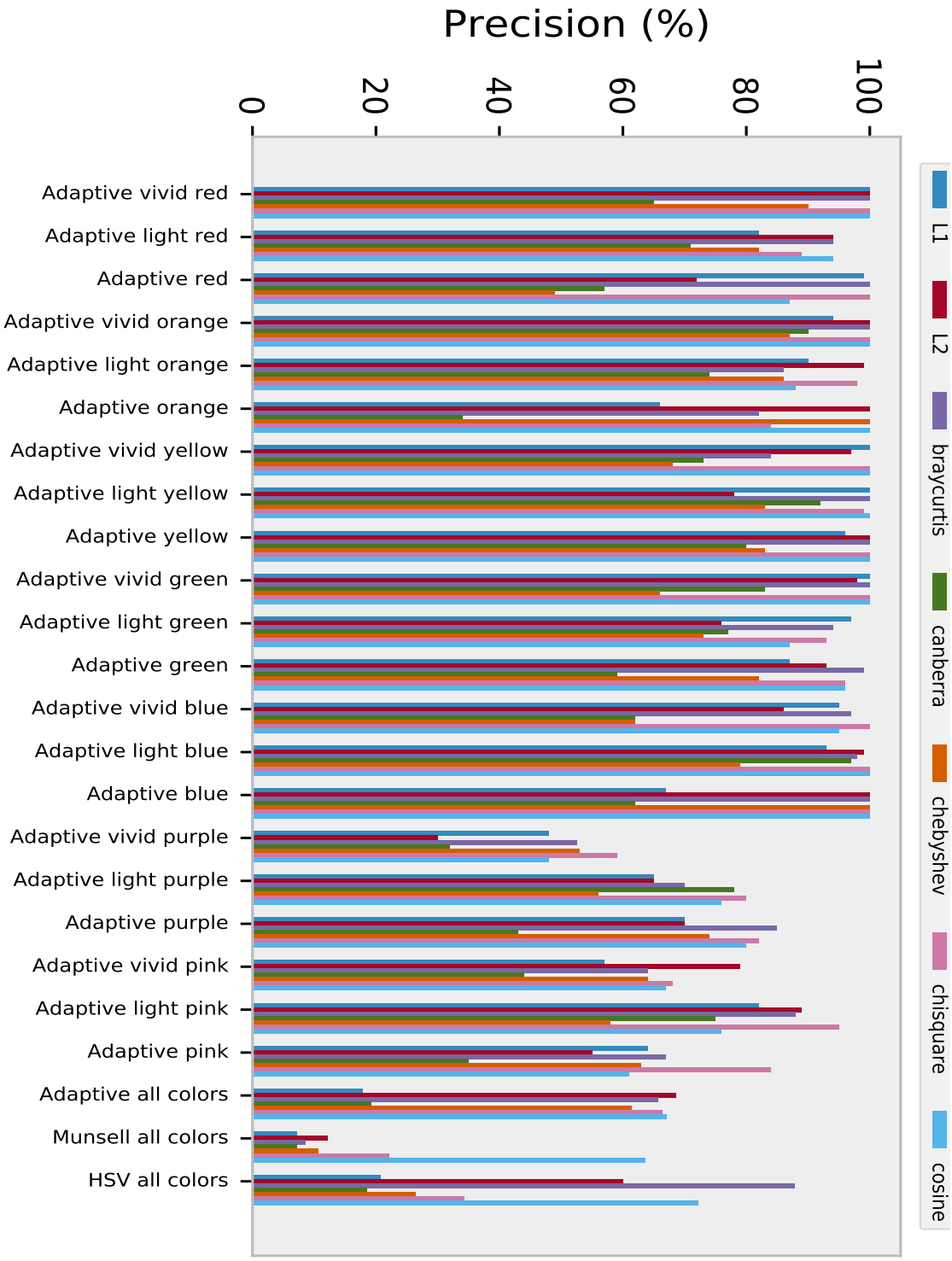


Figure 6-2: Precision of adaptive color-based image search methods comparing to other methods with different combination of color feature and similarity functions. The proposed adaptive color-based image search method returns relative higher precisions. The Chi square and Cosine distances obtain overall better performance.

6.4.2.2 Sample Search Results using Adaptive Color-based Image Search

This section shows some sample search results using the adaptive color-based search method on the Crowley's painting dataset (section 6.1) as in Figures 6-3 to 6-7 to demonstrate the effectiveness of the system when adapting to different content preferences of users. These figures demonstrates two different adaptive searching scenarios using color names. Figures 6-3 to 6-5 show the search results of queries with a same input image but different vibrancy levels of a same *color* to express the content preferences. Figures 6-6 and 6-7 show the search results of queries with a same input image but different *colors* as content preferences.

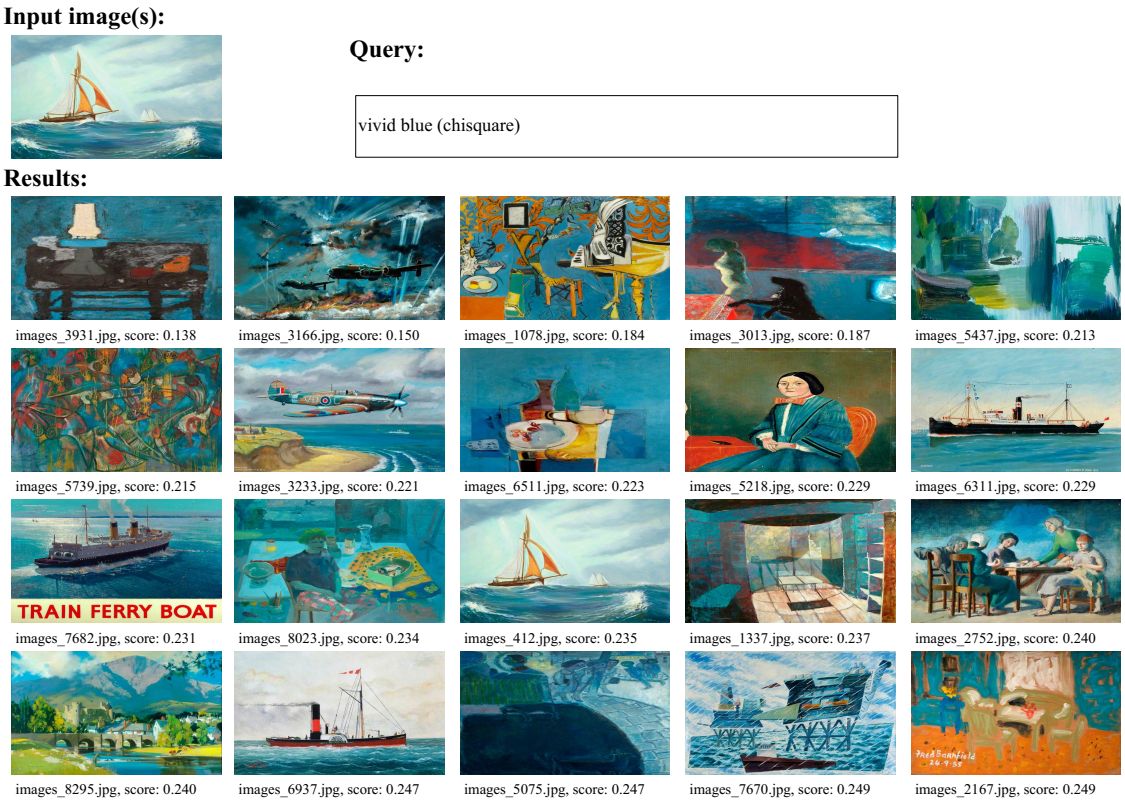


Figure 6-3: Sample search results of the adaptive color-based search method with one input image, a input keyword "vivid blue" and using Chi-square similarity. Images are from the Crowley's painting dataset (more detail in section 6.1). Copyright information: `https://artuk.org/footer/copyright-notice-15`.

Figures 6-3, 6-4, and 6-5 show the sample results using the proposed adaptive color-based search method for queries with a same input image but different keywords

to express the content preferences which are the different vibrancy levels of a same color "blue". The use of an input image aids a usefulness to express "colors like in this image", since the colors that human can perceive vary greatly with detailed subtleness but are often difficult to express without an example.

In Figure 6-3, the input keyword "vivid blue" expresses a content preference of "vivid blue colors like in this image". The search results are images from the painting dataset which contain similar vivid blue colors. Comparing to the search results in Figure 6-4 with the input keyword "light blue", the images returned from the search algorithm are greatly different and reflected the content preferences in each case. When only inputting color keyword without specifying the vibrancy, the search process returns images with colors that contain similar colors regardless of the vibrancy as shown in Figure 6-5.

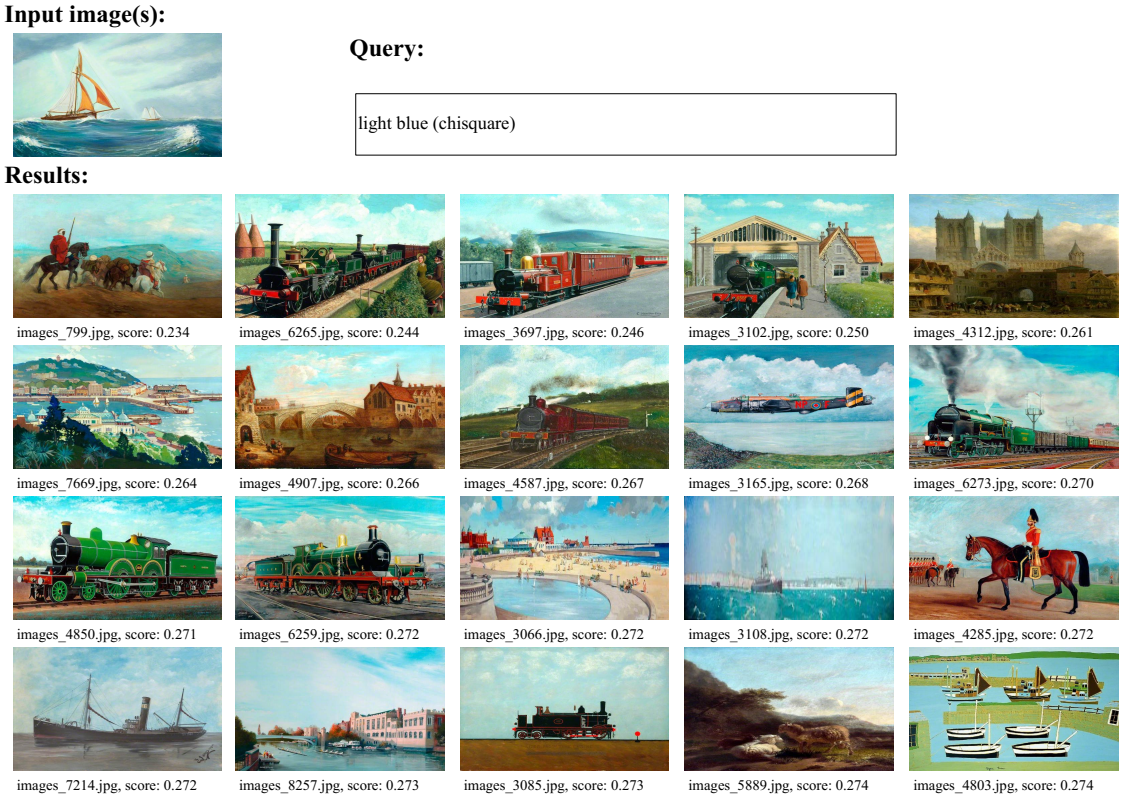


Figure 6-4: Sample search results of the adaptive color-based search method with the same input image in Figure 6-3, a input keyword "light blue" and using Chi-square similarity. Images are from the Crowley's painting dataset (more detail in section 6.1). Copyright information: `https://artuk.org/footer/copyright-notice-15`.

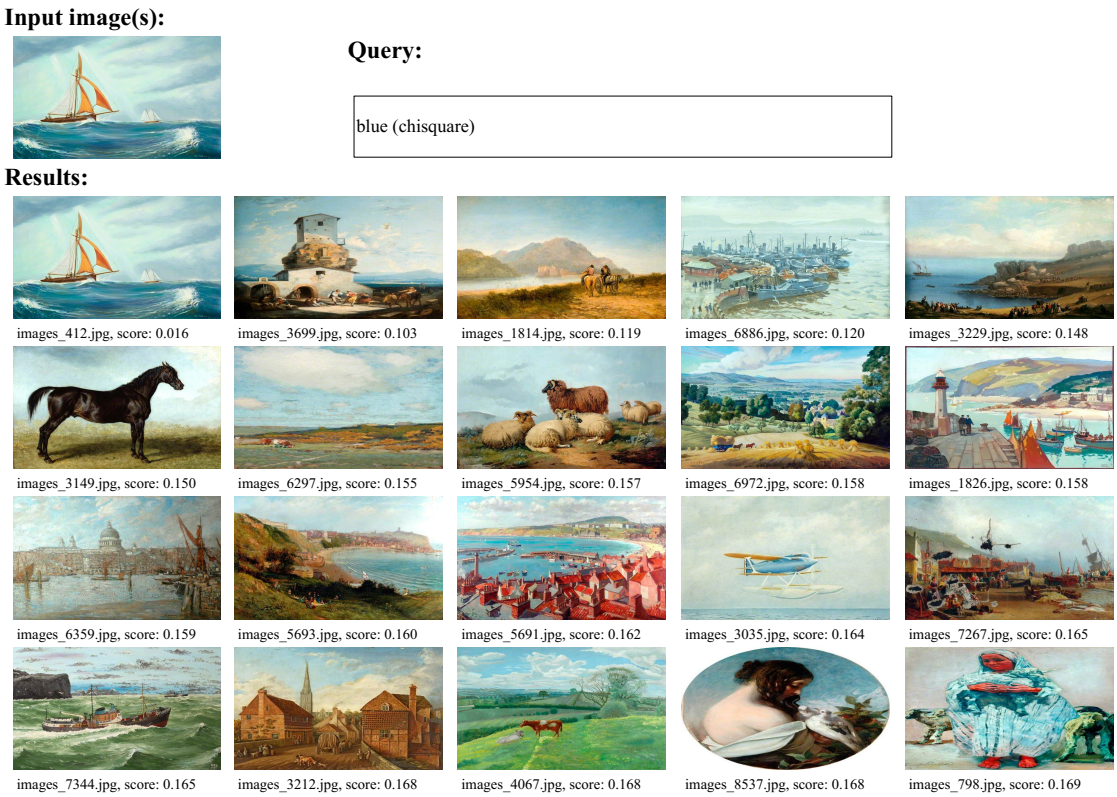


Figure 6-5: Sample search results of the adaptive color-based search method with the same input image in Figure 6-3, a input keyword "blue" and using Chi-square similarity. Images are from the Crowley's painting dataset (more detail in section 6.1). Copyright information: `https://artuk.org/footer/copyright-notice-15`.

Figures 6-3, 6-4, and 6-5 have shown the usefulness of using the adaptive color-based query creation to explore an image dataset. The search results demonstrate the adapted subspace according to each query to retrieve appropriate results for various levels of a same color.

Figures 6-6 and 6-7 show other search results the proposed adaptive color-based search method for two queries of two different colors. The query in Figure 6-6 expresses the content preference of "red" colors in the input image, while the query in Figure 6-7 expresses the content preference of "yellow" colors in the input image.

In Figure 6-6, it can be seen that the exact-match image for the input image is returned not at the first rank. This is due to an effect of the application of the subspace selection method. In the selected subspace created by the content preference "red", meaning the subpace of "red" colors, each image of the image dataset has a different

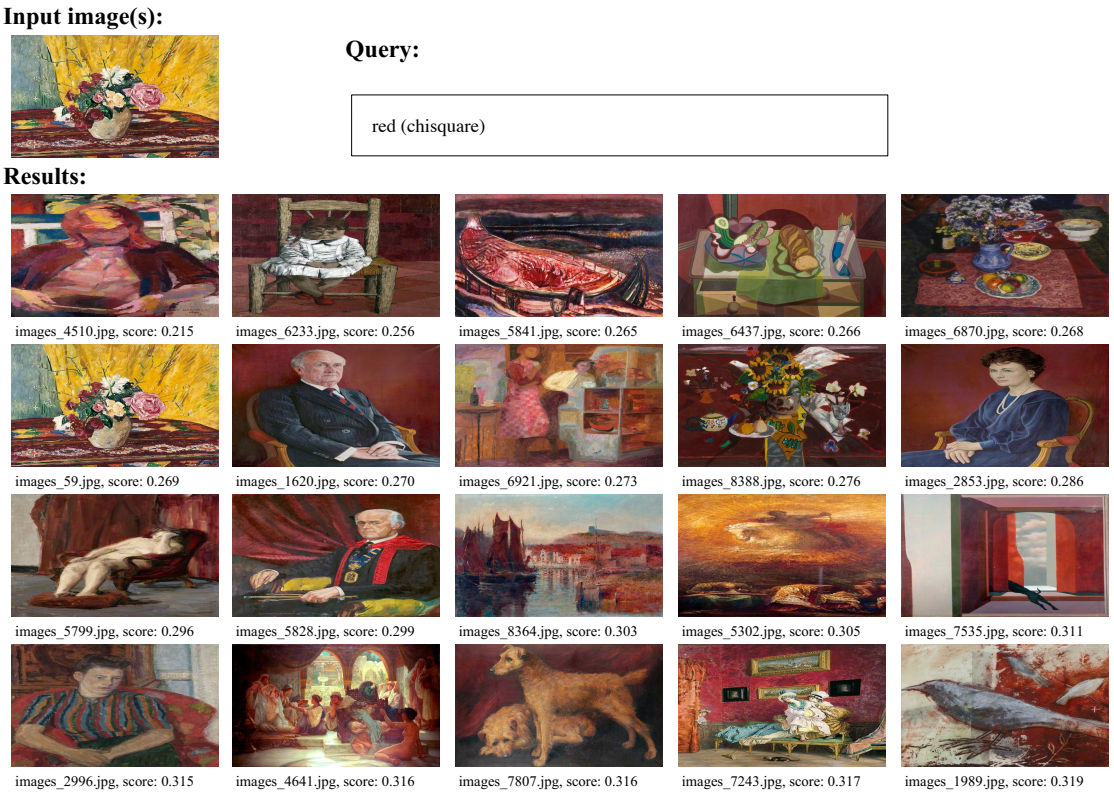


Figure 6-6: Sample search results of the adaptive color-based search method with one input image, a input keyword "red" and using Chi-square similarity. Images are from the Crowley's painting dataset (more detail in section 6.1). Copyright information: `https://artuk.org/footer/copyright-notice-15`.

relative distance to the query image comparing to its original distance to the query image in the original feature space. This is intuitively verifiable by the search results in Figure 6-6. The returned images are ranked by the relevance scores of "redness" of their contents (based on the Chi square distance function).

In Figure 6-7, the first ranked result is the same as the query image (based on the Chi square distance calculation). This indicates that even in the selected subspace created by "yellow" colors, the closest image in the dataset to the input query image is itself.

Those above observations have intuitively demonstrated the behaviors subspace selection method and how it can affect the search results. Moreover, it has been shown to be effective and useful for adapting the contextual preferences of users into queries.

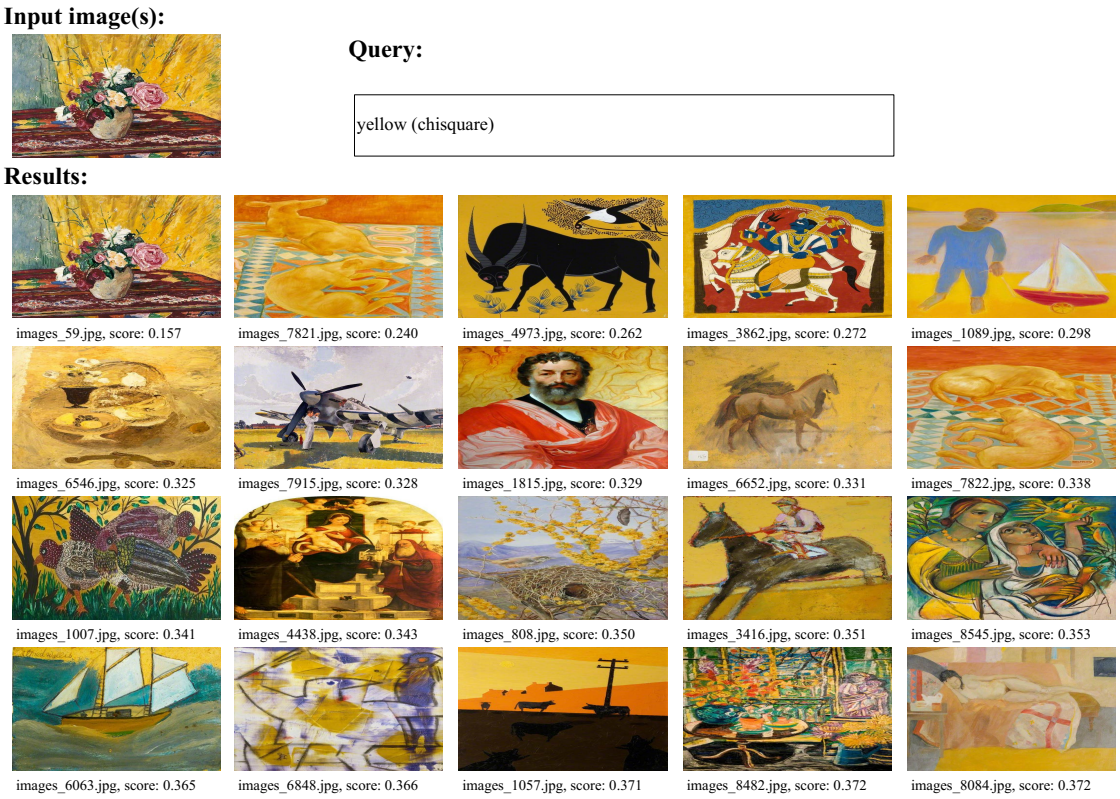


Figure 6-7: Sample search results of the adaptive color-based search method with the same input image in Figure 6-6, a input keyword "yellow" and using Chi-square similarity. Images are from the Crowley's painting dataset (more detail in section 6.1). Copyright information: `https://artuk.org/footer/copyright-notice-15`.

6.4.3 Imagination-based Image Search

Section 6.4.2 has shown quantitative experiments of the adaptive color-based query creation method for searching for relevant images from an image dataset. The queries used in those experiments contain only one input image and a keyword that specifies the content preference. However, it is possible to combine several images to express more refined content preferences, which are called imaginations. The functions used to combine input images have been described in section 6.3.3. This section shows an example of how to use a combining function to express a content preference for search.

Figure 6-8 shows sample search results of an imagination-based query that is created using the adaptive color-base query creation method (section 6.3) and *PLUS*

combining function (section 6.3.3). The query contains two input images and a input keyword "vivid orange PLUS vivid blue". The keyword expresses a content preference of the user and based on the keyword, the query is interpreted as "searching for images with *vivid orange* colors like in the first image **and** *vivid blue* colors like in the second image" using a $PLUS$ operation.

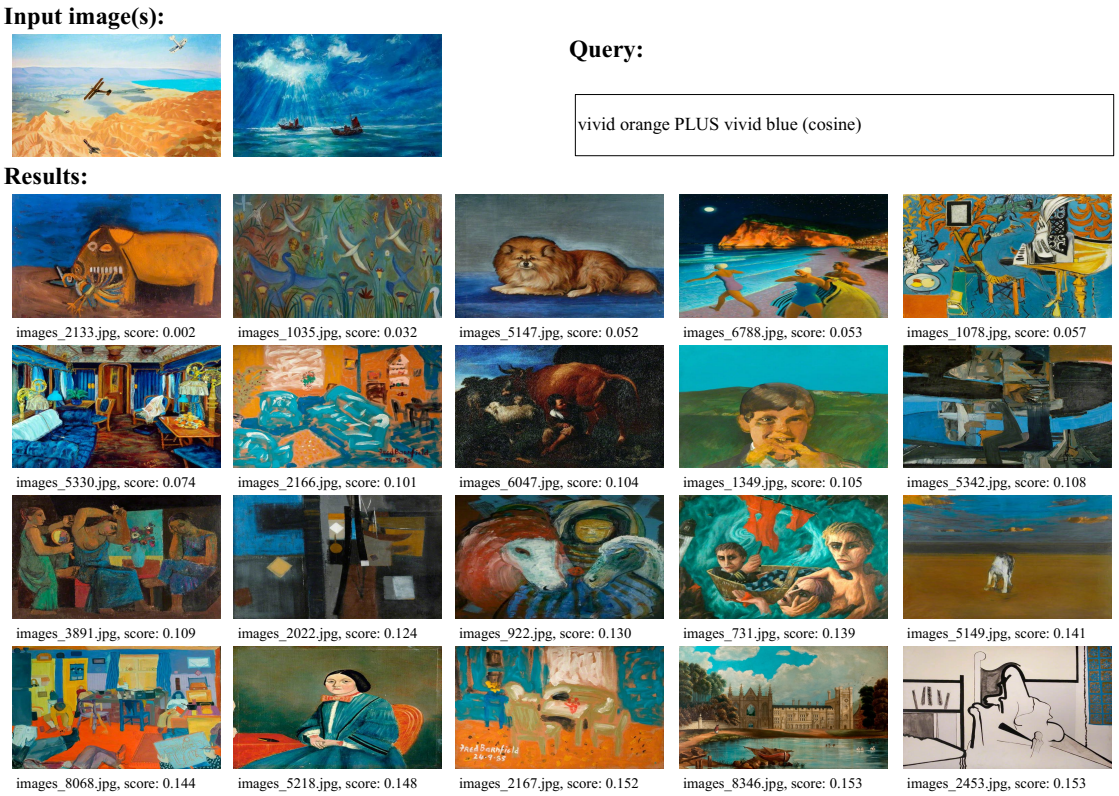


Figure 6-8: Sample search results of an imagination-based query created by two input images with corresponding content preferences "vivid orange" and "vivid blue" using $PLUS$ operation. Images are from the Crowley's painting dataset (more detail in section 6.1). Copyright information: `https://artuk.org/footer/copyright-notice-15`.

6.5 Discussion

It is firstly to note that the experiments in this chapter focuses on the performance of the query creation proposed in section 6.3 while implementing a brute force search algorithm for retrieving relevant images from an image dataset. The brute force

search process follows an exhaustive search strategy and is similar to Algorithm 7.6 in section 7.5.2.2. It is because the purpose of the experiments in this chapter is to assess the effectiveness of the proposing color feature indexing method, it is more convincing to use a such exhaustive search strategy. For that reason, the performance measured by how fast to complete a search query hasn't been concerned. However, the system is not limited with this search strategy. On the contrary, it is naturally compatible with the adaptive pruning search mechanism introduced in section 3.2.

Secondly, quantitative experiments on the statistical effectiveness of the query creation method using combining functions described in section 6.3.3 are reserved as future works. However, an interested audience can find useful related information in the previously published works [41, 42, 43, 44, 45, 46, 47, 48].

Finally, the adaptive color-based query creation method is not limited by using *color names*. One algorithm that is similar to the proposed algorithm in Algorithm 6.2 is applicable based on the logic of the proposed query creation method. For example, in section 8.1.3, an application that indexes the color spaces differently using different keywords ("hot" and "cold") is described.

Chapter 7

Large-Scale Frame-wise Video Navigation System

The true method of knowledge is experiment.

William Blake

7.1 Video Datasets

There are two video datasets used in this thesis: "TRECVid 2015", and "Movie dataset" whose details are shown in Table 7.1.

"TRECVid 2015" : The TRECVid 2015 dataset is the dataset provided by the National Institute of Standards and Technology, U.S. that is a well-known dataset for multimedia research[1]. This dataset includes "master shot boundary reference for semantic indexing test data" and will be used mainly to evaluate the proposing scene detection algorithm.

"Movie dataset" : The movie dataset is a personal collection of 2.14TB movie DVDs including various genres but mainly of animation, drama, and documentary.

[1]Download link: http://trecvid.nist.gov/trecvid.data.html#tv15

Table 7.1: Description of two video datasets: TRECVid 2015 and Movie dataset

	TRECVid 2015	**Movie dataset**
Number of videos	6870	806
Total duration	568 hours	597 hours
Total frames	2,047,413	2,150,428
Total scene frames		485,337
Compress rate		22.60%
Storage size	141GB	2.14TB

The total view length is 597 hours for 806 movies meaning the averaged length of a movie is 45 minutes. This dataset is used to demonstrate the feasibility of using the proposed architecture to index and search for personal collections of video data. It will be used mainly to evaluate the performance of the indexing and search architecture. The implementation to get the information regarding number of scene frames and compress rate for this dataset will be discussed in section 7.5.1 while the detail of the scene detection algorithm will be discussed in section 7.4.

7.2 System Architecture

7.2.1 Overall Architecture

The overall system architecture is shown in Figure 7-1. It includes six modules which are (1) data collection, (2) frame extraction, (3) feature extraction, (4) indexing, (5) search, and (6) ranking and display. The data collection module collect image and video data from many resources which can be either from personal collections of photos, videos or from the Internet such as from Youtube (video data) or Flickr (image data) and many others.

1. **Data collection**

   This module either collects video urls from the Internet, e.g., from YouTube or converts videos in local hard disks into MPEG-4 format. It's noteworthy that crawling videos from the Internet is restricted by copyrights, thus the system does not download them to store into local disks, but only store the

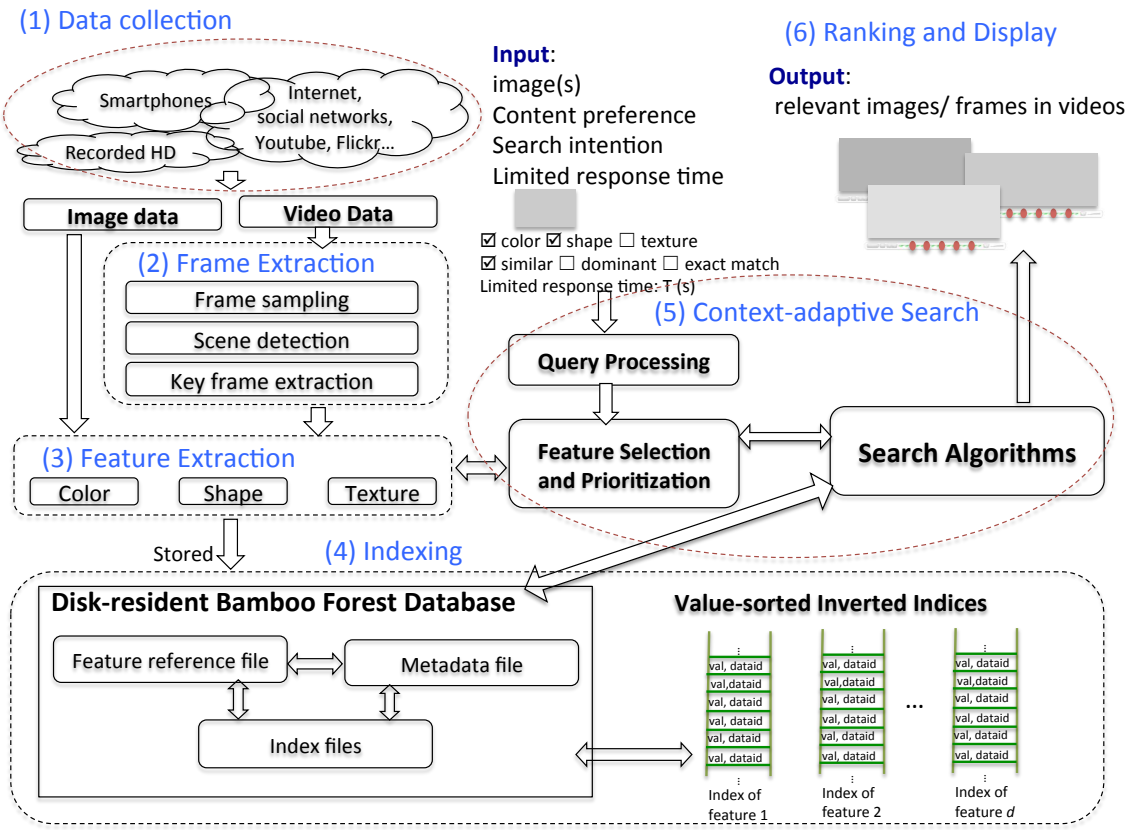


Figure 7-1: Overall architecture of multicontext-adaptive query creation and search system.

streaming urls for later video analysis and after that remove them from the system. Moreover, the videos in the Internet might be unpredictably deleted but once they are already indexed into the database, they will be used as other videos meaning some results of search can be unable to trace. The current version of the system does not support a function to eliminate such videos and their related information such as associated frames and their metadata information from an indexed database.

2. **Frame extraction**

   The frame extraction module works only with video data and its purpose is to reduce the number of frames of a video to be indexed. This module contains three main steps: for each video record, it samples the video frames from the

beginning by a fixed time (by default, $\delta t = 1second$) and applies a threshold-based scene detection algorithm to detect scene frames in the video. Finally, the thumbnails of the detected scene frames are extracted from the video and temporally saved as representative frames of the video to be indexed. The details of this module will be discussed in section 7.4.

3. **Feature extraction**

   This module uses existing feature extraction methods in literature to extract feature vectors of low-level features of data such as color, shape, texture. References to the methods that are applicable are introduced in section 1.3 and concretely in section 7.3. Algorithm 6.1 in section 6.2 is also applicable to extract color feature vectors of data.

4. **Indexing**

   The Bamboo Forest indexing module is integrated with a database manager sub-module that manages the filesystem of the database. This module creates and stores indexes for a dataset and the corresponding metadata related to the dataset and the feature space. The data structure and file structure of a Bamboo Forest database are described in chapter 4.

5. **Search**

   The Search module includes three sub-modules: "query processing", "Feature Selection and Prioritization", and "Context-adaptive Search". The query processing sub-module interprets input from users into variables of search algorithms. It is coupled with the "Feature Selection and Prioritization", which is a sub-module that reflects the content and intention preferences in order to select an appropriate search algorithm. This works as a search planner for the next sub-module. The context-adaptive search sub-module has three algorithms as introduced in chapter 5 implemented and this will find high-matching-possibility candidates and return those to the next module "Ranking and Display".

6. **Ranking and Display**

   This module receives candidates from the search module and makes a ranking based on relevance scores and displays the results to users. For video data, each candidate is a video frame but when being displayed to users, the whole video with the replay time is marked at the time of the video frame is shown to users. By this, the users can (1) know when the relevant frames are in videos, and (2) replay and watch videos from the scenes they are looking for. This module ranks the relevant results depending on a ranking preference specified by a user, such as ranking by relevance scores or by timestamps of data.

7.2.2 System Implementation

The video navigation system with three algorithms described in chapter 5 are implemented using Python programming language version 2.7.12[2]. This system will be studied in two computers: a desktop and a laptop.

The desktop computer, which is a lab server, has a faster processor and larger memory but databases are stored on its hard disk drive (HDD). Its configurations are: CentOS Linux release 7.1.1503 (Core), 32 CPU Intel(R) Xeon(R) CPU E5-2680 0 @ 2.70GHz. The laptop computer is a general purpose computer, which is a MacBook Air 1.7 GHz Intel Core i7, 8 GB 1600 MHz DDR3 with OXS Yosemite operating system. On this laptop, the databases are stored on an external SSD disk (Samsung 850 EVO 500GB), connecting to the laptop via a USB 3.0 cable.

Table 7.2 shows benchmarks of these two computers for their CPU and I/O performance. For benchmarking the performance, the following suites (libraries) are used:

1. Intel®Math Kernel Library (MKL) Benchmarks - Optimized LINPACK benchmark[3]: This benchmark, followed the introduction in the website, "solves a dense ($real * 8$) system of linear equations ($Ax = b$), measures the amount of time it takes to factor and solve the system, converts that time into a performance

[2]https://www.python.org/

[3]https://software.intel.com/en-us/articles/intel-mkl-benchmarks-suite

Table 7.2: CPU and I/O performance benchmarks of two computers used in implementation.

Computer	**Laptop**	**Desktop**	**Benchmark suite**
Computer name	MacBook Air	Linux Server	
Operating system	OSX Yosemite 10.10.2	CentOS 7.1.1503	
Processor name	Intel®Core i7	Intel®Xeon E5-2680 (8 cores)	
CPU model	1.7 GHz	2.7 GHz	
Number of CPUs	1	2	
Memory	8 GB 1600 MHz DDR3	16 sticks * 16 GB 1600 MHz DDR3	
GFlops	47.57	274.19	Intel®MKL Linpack
Whetstone	1923.1 MIPS	2222.2 MIPS	Whetstone
Pystone	69206.4 pystones/second	94339.6 pystones/second	Pystone
Storage disk	Samsung SSD 850 EVO 500GB	HDD WDC WD20EZRX-00D 2TB	
Random read IOPS	3267	97	Flexible I/O Tester (FIO)
Random write IOPS	3286	97	FIO
Random read	13069 KB/s	391 KB/s	FIO
Random write	13146 KB/s	390 KB/s	FIO

rate, and tests the results for accuracy. The generalization is in the number of equations (N) it can solve." This benchmark is used to measure the Intel processor systems, equivalently to FLOPS (floating-point operations per second) benchmarks. The results are in GFlops. A GFlop (gigaflops) is a billion FLOPS.

2. Whetstone[4]: the Whetstone benchmark is another synthetic benchmark for evaluating the floating-point arithmetic performance of computers [110].

[4]http://www.netlib.org/benchmark/whetstone.c

3. Pystone[5]: The Pystone benchmark is equivalent to the Dhrystone benchmark, which measures the performance of computers for integer and string operations. The Pystone "measures the number of pystones (iterations) per second that can be run over the machine. A higher number of pystones per second indicates a better performance of a Python interpreter" [111].

4. Flexible I/O Tester (FIO)[6]: This thesis uses FIO tool, which is a workload generator, to benchmark the I/O performance of computer storage devices (HDD and SSD drives). Using the FIO tool to define a workload of creating a 1GB text file and test random read and write, the results return are the number of input/output per second (IOPS), and number of KB per second (KB/s).

7.3 Feature Extraction

Three kinds of basic features represent low-level semantic information of a frame are color, shape, and texture are extracted. There are many methods to extract those features alone, and all of them have strong quality to be applied in common or specific domains of content-based information retrieval field. The focus of this research is not to propose a new method for feature extraction, therefore existing extraction methods in literature are used in this thesis. Nevertheless, It is noteworthy that the choice choosing such methods is limited by the scope of content to be applicable as discussed in 1.3. If this requirement is satisfied, any feature extraction method would be suitable to my proposing indexing and search architecture.

7.3.1 Color

A HSB color histogram is extracted for each frame by splitting HSB color space in a non-uniform way. For example, to give more detail, hue can be split finer while saturation is not. Good results have been reported with 7x2x2 and 7x3x3 splits [112]. In this thesis, I use 7x3x3 splitting scheme that creates 63 descriptors for color feature.

[5]https://pybenchmarks.org/u64q/performance.php?test=pystone
[6]http://freecode.com/projects/fio

7.3.2 Shape

Shape feature can be statistically represented using a spatial pyramid kernel [113]. The feature is captured based on the spatial distribution of edges and formulated as a "Pyramid of Histograms of Orientation Gradients (PHOG)" vector representation. Each image is divided into a sequence of increasingly finer spatial grids by repeatedly doubling the number of divisions in each axis direction. The pyramid at level $l = L$ has 4^L grids and if use K orientations to generate histograms, the PHOG descriptor of the entire image is a vector with dimensionality $K \sum_{l=0}^{L} 4^l$. More details of generating PHOG descriptors are discussed in [113].

This thesis uses $l = 1$ (the pyramid has two levels: a root level ($l = 0$) and a level $l = 1$) and use 8 orientations $K = 8$ so that PHOG descriptor vector for shape feature has 40 elements in total.

7.3.3 Texture

Texture features of video frames are extracted using Gabor wavelet descriptors early introduced in [114, 115] and implemented recently in well known libraries such as Lire library [116] and JFeaturelib [112]. This method uses a multiresolution representation based on Gabor filters that are considered as orientation and scale tunable edge and line (bar) detectors because the statistics of these micro features in a given region are often used to characterize the underlying texture information [115]. This thesis uses five scales and six orientations, and consequently total 60 Gabor filters to extract texture features. The details of Gabor filters and its characteristics are discussed widely in literature that an interested audience might want to investigate such as [114, 115, 117]

7.3.4 Integrated Descriptors

Generally above features (color, shape and texture) are not used alone but used together in order to present images in more details. Additionally, the combinations of those features might be at querying time as discussed in section 6.3.3. At one time,

one person may be interested in finding results with similar color feature, whereas at another time, she may be interested in similarity in both color and shape, etc. To prepare such changing contexts, this thesis used combined features of color, shape and texture that are already normalized and integrated.

7.3.4.1 FCTH: Integrated descriptor of color and texture features

A fuzzy color and texture histogram is introduced by Savvas et al. in [118] known as FCTH feature. The method used a two-input fuzzy system to generate 24-bin color histogram, then used Haar wavelet transformation for fixed 8 regions to export texture elements, and consequently total 8 x 24 = 192-bin FCTH feature is extracted as a packed feature of both color and texture features. This feature is chosen because of its robustness to deformations, and noise. Ones might want to investigate [118] for detail.

7.3.4.2 CEDD: Integrated descriptor of color and edge directivity features

A compact descriptor of both color and edge directivity are introduced in [119]. Similar to FCTH feature, the method used a two-input fuzzy system to generate 24-bin color histogram, then applied to a set of 6 texture filters which contains 5 digital filters of MPEG-7 edge histogram descriptor: vertical, horizontal, 45-degree diagonal, 135-degree diagonal and non-directional edges. It is to note that the sixth filter is for filtering *no edge region*. Consequently, the CEDD histogram includes 6 x 24 = 144 elements.

7.3.4.3 JCD: Integrated descriptors of CEDD and FCTH

The JCD descriptor is a joint descriptor joining CEDD and FCTH descriptors [120]. The joint descriptor includes 168 elements in total. The detailed method and implementation are reported in [112, 120].

7.4 Scene Detection

7.4.1 Two Scene Detection Algorithms

Scene detection which is also known as scene segmentation is very significant in summarizing content of videos data and is engaged in many research works such as [121, 122, 123]. The most straightforward and conventional approach is using a similarity threshold to detect changes between sequential video frames. The threshold is either fixed or dynamically decided based on a sliding window through the sequence of video frames [124]. In case of a fixed threshold, it can be chosen by an automatic threshold finding algorithm either using histogram differences [125], entropy [126], or the Otsu method [127] (from [124]).

In this thesis, an automatic threshold finding method is introduced. This method will be apply to two scene detection algorithms: basic scene detection algorithm which uses one threshold, and advanced scene detection algorithm which uses two thresholds. The method contains a main training step to automatically learn the most effective thresholds for different combinations of feature extraction and similarity distance functions.

7.4.1.1 Basic Threshold-based Scene Detection Algorithm

The basic threshold-based scene detection algorithm detects a new scene simply by comparing to previous scene and keeps only the starting frame in the consecutive frames of a scene as a representative frame for the scene. The pseudocode of the algorithm can be found in Algorithm 7.1. In this algorithm, the frame at 0 second (the beginning of the video) is the first presentative frame of a video by default. A frame is said to be on new scene if it is less similar to the current frame by a fixed threshold.

7.4.1.2 Advanced Two-thresholds-based Scene Detection Algorithm

To reduce a penalty of missing a scene, an advanced scene detection algorithm using two thresholds instead of one fixed threshold is introduced. The algorithm is based

Algorithm 7.1 Basic Threshold-based Scene Detection Algorithm

Input:

- A sequence of frames of a video;
- Similarity threshold DT.

Output: A set of frames detected as representative scene frames.

```
1: [Initialize result set] S ← {}
2: [Initialize] Current scene frame ← first frame.
3: while there is still a frame in video sequence do
4:     if similarity(Current scene frame, frame) > DT then
5:         [New scene is detected] S ← S ∪ {frame}.
6:         Current scene frame ← frame.
7:     end if
8:     [Move to next frame] frame ← next frame in the sequence.
9: end while
10: return S.
```

on an observation that the temporal distance between two frames effects the decision of whether the two frames are from two distinguishing scenes. In other words, two frames separated in a long time window in a video might be in different scenes even they are very much similar to each other. This observation also compensates to the fact that the similarity metrics based on low-level features can mistakenly represent two different frames by the same feature vectors.

The pseudocode 7.2 describes this two-thresholds-based scene detection algorithm. In this algorithm, two similarity thresholds are required as input: one is called a *direct threshold* (DT) and another one is called a *conditional threshold* (CT) with $DT > CT$. DT is regardless to the temporal distance (*time window length* (PT)) of two frames, while CT provides a delay step to check the distance in time whether is greater than PT.

Implementing the above two algorithms 7.1 and 7.2 requires a specific implementation of the *similarity* calculation function between two frames (line 4). As already shown in the experiments in section 6.4.2, the performance of of the similarity calculation function depends on the choice of features that are used to represent a frame (which is equivalent to an image), and a similarity metric (or distance metric). In the

Algorithm 7.2 Advanced Scene Detection Algorithm with Two Thresholds

Input:

- A sequence of frames of a video;
- Two similarity thresholds CT and DT where $CT < DT$;
- Optional: a time window length PT, by default $PT = 5$.

Output: A set of frames detected as representative scene frames.

```
1: [Initialize result set] S ← {}
2: [Initialize] Current scene frame ← first frame.
3: while there is still a frame in video sequence do
4:     if similarity(current scene frame, frame) > DT then
5:         [New scene is detected] S ← S ∪ {frame}.
6:         Current scene frame ← frame.
7:     else
8:         if Temporal distance between frame and current scene frame > PT
               AND similarity(current scene frame, frame) > CT then
9:             [New scene is detected] S ← S ∪ {frame}.
10:            Current scene frame ← frame.
11:        end if
12:    end if
13:    [Move to next frame] frame ← next frame in the sequence.
14: end while
15: return S.
```

next section, a training method to automatically learn the most effective combination of features and similarity metrics will be introduced.

7.4.2 Threshold Learning for Scene Detection Algorithms

As discussed in the previous section, a specific choice of *similarity* calculation function is required when implementing either Algorithm 7.1 or Algorithm 7.2. This function containing a combination of choices of a feature type and a distance metric greatly affects the efficiency of the detection algorithm. If the feature in use has a discriminating power large enough, it would efficiently help us to distinguish a frame from others. And the distance measure varying in their units undoubtedly affects how we set a threshold for "how much is similar".

The important task to find which combination is the most effective is named

"feature engineering" with respect to the original idea which is widely in machine learning field [128]. The goal of feature engineering can be summarized into two main points:

- To promote discriminability of features that represent video frames (i.e., feature selection)
- To control signal-to-noise sensitivity (i.e., measure selection)

The problems of feature selection and measure selection are widely studied among literature such as [9], [10], [129], [130], and so forth. The researchers focus on the behaviors of data objects in high dimensional representation of objects, and the correlation between features. However, the feature and measure selection problems seem to be studied independently. On one hand, a work on feature selection would choose a specific distance measure and use it to understand how features are related to each other, like of [129]. On the other hand, a work on behaviors of distance functions would select a dataset of several specific distributions [9, 10]. It is observed that the feature and measure chosen interact with each other. And the study of them would benefit the understanding and selection properly specified for each data at hand. In this section, a training method which uses relative contrasts to learn the most effective combination of feature and distance function is introduced.

7.4.2.1 Relative Contrast

A relative contrast is the contrast of the maximum and minimum distances to a data point from all data points in the dataset and is defined as:

$$\mathfrak{C}_k^{(N)} = \frac{Dmax_k - Dmin_k}{Dmin_k} \tag{7.1}$$

In which $\mathfrak{C}_k^{(N)}$ is the relative contrast from a data point to N data points in k-dimensional feature representation, and $Dmax_k$ and $Dmin_k$ are the farthest and nearest distance of the N points to the query point, respectively using measure D as the distance measure method. The value of $\mathfrak{C}$ has an interesting property regarding

the meaningfulness of choosing the distance metric D that have been discussed in [9, 10]. The low value of $\mathfrak{C}$ "makes the proximity query meaningless and unstable because there is a poor discrimination between the nearest and furthest neighbor" [9].

7.4.2.2 Threshold Learning Algorithm

This section describes a threshold learning algorithm as shown in Algorithm 7.3 based on the idea of relative contrasts in the previous section.

Algorithm 7.3 Threshold Learning using Distance-based Relative Contrast for a Scene Detection Algorithm

---

Input:

- A scene detection algorithm (either Algorithm 7.1 or 7.2);
- A sequence of N frames of a video;
- A feature extraction function F for video frames;
- A similarity distance function D for each pair of video frames;
- A scaling factor $s = [s_1]$ (if using Algorithm 7.1) or $s = [s_1, s_2]$ (if using Algorithm 7.2) with $0 < s_i \leq 1.0$ and $s_1 \leq s_2$.

Output: Measures of the algorithm performance using *compress_rate* and *recall*.

1: [*Step 1: Initialize feature space*] Using function F to extract feature vectors of N video frames .
2: [*Step 2: Calculate distance matrix*] Calculating distances using function D between each pair of N feature vectors in step 1 to create a $N \times N$ square matrix $\mathbb{A}$.
3: [*Step 3: Get maximum distance*] $Dmax \leftarrow$ largest value in distance matrix $\mathbb{A}$.
4: [*Step 4: Get minimum distance*] $Dmin \leftarrow$ smallest non-zero value in distance matrix $\mathbb{A}$.
5: [*Step 5: Calculate similarity threshold(s)*] $T \leftarrow s * (Dmax - Dmin)$.
6: [*Step 6: Detect scene frames*] Applying scene detection algorithm for the sequence of video frames using threshold(s) in T and a *similarity* method constructed by functions F and D to get a set of scene frames S.
7: [*Step 7: Measure compress rate*] $compress\_rate \leftarrow \frac{|S|}{N}$.
8: [*Step 8: Measure recall*] $recall \leftarrow \frac{S'}{G}$ where S' is the number of scenes in S that belongs to different scenes, and G is the number of different scenes in the video (known ground truth information relating to the video).
9: **return** $(compress\_rate, recall)$.

Algorithm 7.3 takes an input including a video sequence, a feature extraction function, a similarity distance function, and a scaling factor to learn one threshold value for the basic scene detection Algorithm 7.1 and two threshold values for the advanced scene detection Algorithm 7.2. The scaling factor s has one positive value if it is used with Algorithm 7.1 or two positive values if it is used with Algorithm 7.2. Since $Dmax$ and $Dmin$ are respectively thelargest and smallest distance between a pair among all pairs of video frames in a video sequence, this scaling factor s, as used in line 5 in Algorithm 7.3, indicates a probably useful proportion of relative differences between frames to detect different frames out of the video sequence. In other words, $s * (Dmax - Dmin)$ can be used to detect the scenes in a video sequence.

After learning the threshold value(s), Algorithm 7.3 evaluates the scene detection algorithm by comparing to the "ground truth" information regarding scenes in the video[7]. Two measures are used to evaluate the performance of a scene detection algorithm: $compress\_rate$ and $recall$. The $compress\_rate$ measure refers to the ratio of frames are detected as scene frames. It is defined as:

$$compress\_rate = \frac{\text{total number of detected scene frames}}{\text{total number of frames}} \tag{7.2}$$

The $recall$ measure expresses the quality of the detected scene frames, which is the ratio of successfully detected scene frames to the number of scenes. It is defined as:

$$recall = \frac{\text{total number of detected scene frames belonging to different scenes}}{\text{total number of scenes in the video}} \tag{7.3}$$

Obviously, a low $compress\_rate$ value and a high $recall$ value are preferred. Therefore, a balancing measure that combines these two measures are defined:

$$balanced\_recall = (recall - compress\_rate) * recall \tag{7.4}$$

A high value of $balanced\_recall$ denotes a high performance of the scene detection

[7]The ground truth information is assumably available with the video dataset. In the experiments, the "master shot boundary ground truth" of the TRECVid dataset will be used.

aglorithm.

7.4.2.3 Automatic Threshold Selection Algorithm

This section introduces an automatic threshold selection algorithm for the scene detection algorithms. This algorithm will iteratively evaluate a scene detection algorithm by applying it with different feature extraction functions and distance functions and learning the most effective scaling factor s for the scene detection algorithm for a combination of a feature extraction function and a distance function. The algorithm is described in Algorithm 7.4.

Algorithm 7.4 initializes several parameter settings including a list of random videos, a list of feature extraction functions, a list of distance functions, and a list of scaling factors that will be used for training the scene detection algorithm. This algorithm returns the most effective scaling factor for each combination of a feature extraction function and a distance function. These combinations will be used in a testing process for remaining videos of the dataset as described in the next section.

7.4.3 Performance Evaluation of Scene Detection Algorithms

In this section, the performance of the two scene detection algorithms introduced in section 7.4 will be studied by using the TRECVid dataset that is described in Table 7.1 in section 7.1.

Algorithm 7.4 will be used with 200 random videos from the dataset to obtain a trained scaling factor for each scene detection algorithm. Interestingly, the scaling factors are returned as the same for all combinations of feature extraction and distance functions, which are 0.1 and (0.05, 0.1) for Algorithm 7.1 and 7.2, respectively.

As a testing process, Algorithm 7.3 will be repeatedly applied with the scaling factors found above to each combination of feature extraction and distance functions and each scene detection algorithm. Figure 7-2 shows the performance in terms of the average $compress\_rate$ and $recall$ of two scene detection algorithms. In this figure, two sub-figures, Figure 7-2a shows the performance of Algorithm 7.1 and Figure 7-2b

Algorithm 7.4 Automatic Threshold Selection for a Scene Detection Algorithm

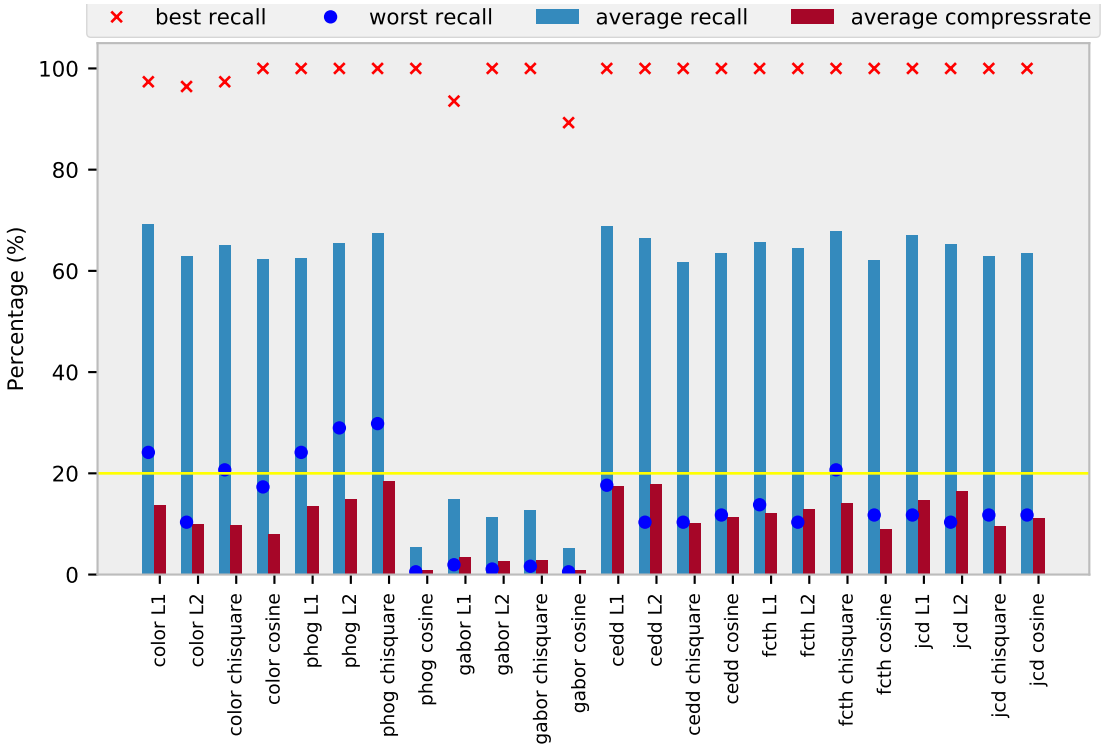
Input: A scene detection algorithm (algorithm 7.1 or 7.2).

Output: The most effective scaling factor for pair of feature and distance functions for setting threshold(s).

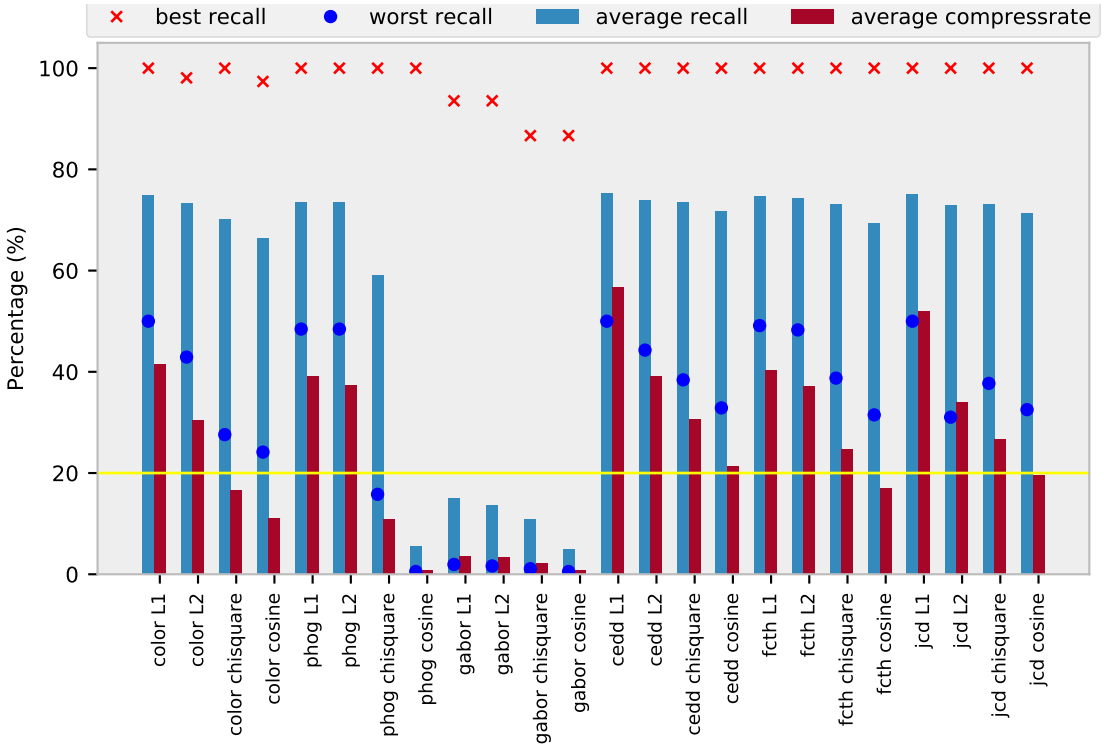
1: [*Initialize a list of training videos*] $VS \leftarrow$ a list of random videos.
2: [*Initialize a list of feature extraction functions*]
$FS \leftarrow [Color, CEDD, Gabor, PHOG, FCTH, JCD]$. ▷ See section 7.3
3: [*Initialize a list of distance functions*]
$DS \leftarrow [L1, L2, COSINE, CHI\_SQUARE]$. ▷ See section 3.3.2
4: [*Initialize a list of scaling factors*]
5: **if** algorithm is basic Algorithm 7.1 **then**
6: $SS \leftarrow [0.05, 0.1, 0.2, 0.3, 0.4, 0.5]$.
7: **else**
8: $SS \leftarrow [(0.05, 0.1), (0.1, 0.2), (0.3, 0.4), (0.4, 0.5)]$.
9: **end if**
10: [*Initialize training dictionary*] $B \leftarrow \{\}$
11: **for each** feature extraction function $F \in FS$ **do**
12: **for each** distance function $D \in DS$ **do**
13: **for each** scaling factor $s \in SS$ **do**
14: **for each** video $v \in VS$ **do**
15: [*Learn*] Learning *compress_rate* and *recall* of the input scene detection algorithm using Algorithm 7.3 with parameters (v, F, D, s).
16: [*Combine measures*]
$balanced\_recall \leftarrow (recall - compress\_rate) * recall$.
17: [*Store result*] $B[(F, D)][s].append(balanced\_recall)$.
18: **end for**
19: **end for**
20: **end for**
21: **end for**
22: **return** (F, D, s) values so that the corresponding list $B[(F, D)][s]$ of s has the largest average *balanced_recall* value for each key (F, D).

shows the performance of Algorithm 7.2.

Overall, both *recall*, *compress_rate*, *recall* at the worst case measurements of Algorithm 7.1 based on one threshold are lower than of Algorithm 7.2 based on two thresholds. While low *compress_rate* is desirable, it's more desired that the *recall* and *recall* at the worst case are reasonably high. By this criterion, the scene detection algorithm, Algorithm 7.2 based on two thresholds seems to be more applicable.



(a) Performance of the basic scene detection Algorithm 7.1 (scaling factor $s = (0.1)$)



(b) Performance of the advanced scene detection Algorithm 7.2 (scaling factor $s = (0.05, 0.1)$)

Figure 7-2: Performance of two threshold-based scene detection algorithms with different settings of feature extraction and distance functions.

Also in these figures, we can see the poor performance of some combination of feature extraction and distance functions: "Gabor" (texture) feature and Euclidean, Manhattan, Cosine and Chi-square distance, and "PHOG" (shape) feature and Cosine distance. Several explanation for the poor performance returned by texture feature can be suggested as: (1) the original data was compressed very much causing poor quality video, (2) the videos were old data in which texture features are not distinctively changing between frames.

Finally, the "CEDD" feature combined with $COSINE$ distance function using two-thresholds-based scene detection algorithm is chosen as the best combination for applying to video data. It has comparative high *recall*, even in worst-case scenarios and has reasonably low $compress\_rate$. This setting will be applied for detecting key frames of the Movie dataset, which will be used intensively in section 7.5 to study the performance of the proposed Bamboo Forest database model and search algorithms.

7.5 Experiments on Frame-wise Video Search System

7.5.1 Video Indexing

Using the experimental results and observations in section 7.4, the advanced scene detection Algorithm 7.2 is used to extract scene frames for each video in the movie dataset (described in 7.1), with the *similarity* function is constructed using a combination of $CEDD$ feature extraction function and Cosine distance function. The threshold values for Algorithm 7.2 are learnt by using Algorithm 7.3 with scaling factor $s = (0.05, 0.1)$. The number of extracted scene frames and compress rate are already shown in Table 7.1.

The detected scene frames for the movie dataset will be divided into six subsets which include 10,000 frames, 50,000 frames, 100,000 frames, 200,000 frames, 300,000 frames, and 485,000 frames and construct the respective database for each using the indexing method as introduced in chapter 4. The six databases are named "10k", "50k",

"100k", "200k", "300k", and "500k" respectively. Table 7.3 shows the six databases and their file counts and on-disk storage size. Without applying any compression technique on inverted indices, the size of database is relatively large. However, it is assumed that that can be ignored at present time. Firstly, it is because the price of disk memory is cheap comparing to working memory. Secondly, it's because given a query, there is only a small part of this database will be loaded into memory for a search process.

Table 7.3: Six databases constructed from the Movie dataset and their file counts and storage size.

Database	# **.pi file**	# **.meta file**	# **.t files**	**Size of database**
10k	1	1	667	105M
50k	1	1	667	517M
100k	1	1	667	1.0G
200k	1	1	667	2.0G
300k	1	1	667	3.0G
500k	1	1	667	4.9G

7.5.2 Setting of Baseline and Test Process for Search Algorithms

7.5.2.1 Setting of Test Process

The test process for each search algorithm proposed in chapter 5 including Maxfirst search, Combinatorial search, and Exact-match search algorithms are shown in procedure 7.5. Each test process will be conducted on two computers whose benchmarks can be found in Table 7.2.

In this test process, each algorithm will be evaluated for different database with increasing size and different settings of its parameters. The baseline for the evaluation is the search results by a brute force search algorithm (algorithm 7.6) and the measure for the performance is the R-precision which will be described in the following sections.

Algorithm 7.5 Test Process for a Search Algorithm

Input: A search algorithm

Output: A dictionary of search performance with different keys as setting parameters.

1: [*Step 1: Reboot the test system*]Reboot the computer.
2: [*Step 2: Initialize a list of databases*]
$DBS \leftarrow [10k, 50k, 100k, 200k, 300k, 500k]$.
3: [*Step 3: Initialize a list of feature types*]
$FS \leftarrow [Color, CEDD, Gabor, PHOG, FCTH, JCD]$. ▷ See section 7.3
4: [*Step 4: Initialize a list of search time limits*]
5: **if** search algorithm is NOT *Exact-match search algorithm* **then**
6: $TS \leftarrow [0.1, 0.5, 1, 2, 3, 5]$.
7: **else**
8: $TS \leftarrow []$.
9: **end if**
10: [*Step 5: Initialize result dictionary*] $\mathbb{D} \leftarrow \{\}$
11: **for each** $db \in DBS$ **do**
12: [*Step 6: Generate random queries*] $QS \leftarrow 50$ random rid in the database.
13: [*Step 7: Repeatedly evaluate search performance*]
14: **for each** $f \in FS$ **do**
15: **for each** $t \in TS$ **do**
16: **for each** $q \in QS$ **do**
17: Applying the search algorithm with input parameters (q, t, f, db) to get the result list RL.
18: Calculating the R-precision p (equation 7.5) by comparing RL with the results returned by the brute force search algorithm (algorithm 7.6) with input (q, f, db).
19: $\mathbb{D}[db][t].append(p)$.
20: **end for**
21: **end for**
22: **end for**
23: **end for**
24: **return** $\mathbb{D}$.

It is to note that in the test process 7.5, several feature types FS is used for each test. For each feature type $f \in FS$, it can be converted to a preference vector using the reference dictionary in a *.meta* file in the corresponding database as described in section 4.2.1.3.

Since a Bamboo Forest database is stored on disk, it is needed to reboot the computer system that implements the search algorithms every time conducting a

performance test. For more precise tests, some other memory cleansing techniques can be applied.

7.5.2.2 Baseline Setting

The top 20 results returned by a brute force algorithm will be used as the baseline for a query search. The brute force algorithm used in this thesis is designed to be compatible with the Bamboo Forest database model as shown in Algorithm 7.6.

Algorithm 7.6 Brute Force Search Algorithm

Input:

- A query vector $q \in \mathbb{R}_+^d$;
- A Bamboo Forest database $\mathbb{B}$ that indexes a dataset $\mathbf{X} \subset \mathbb{R}_+^d$;
- A preference vector $p \in \mathbb{R}_+^d$ such that $p_i = 1$ if i-th feature is preferred and $p_i = 0$ otherwise.

Output: Top R ranked relevant $x \in \mathbf{X}$.

1: [*Step 1: Reflect content preference onto query*] Set $q \leftarrow q \odot p$ where $\odot$ is the elementwise multiplication or Hadamard product.
2: [*Step 2: Repeatedly calculating relevance scores*]
3: List of results $\mathbf{C} \leftarrow \{\}$
4: **for each** *datablock* in *.pi* file of database $\mathbb{B}$ **do**
5: Target vector $x \leftarrow$ feature vector from *datablock*.
6: $score = relevant(q, c \odot p)$
7: Update $\mathbf{C}$ with $(x, score)$
8: Sort $\mathbf{C}$ by a descending order of scores.
9: **end for**
10: **return** Top 20 of $\mathbf{C}$.

In Algorithm 7.6, when searching for "similar" or "exact", the relevant similarity will be calculated by the Cosine similarity. In this case, the candidates will be ranked by an ascending order of scores. But when search for "dominant", the sum of corresponding prioritized features are calculated as relevance scores and the ranks of candidates will be on a descending order of those scores.

For each database, a list of 50 frames are randomly selected as input images. For each input image, and each feature extraction function for six feature types (section

7.3), the brute force algorithm is applied to obtain the top search results. In total, we have $50 * 6 = 300$ queries for each database.

7.5.2.3 R-precision Criterion for Measuring Search Performance

The 'R-precision' will be used to measure the quality of a search algorithm. The definition of R-precision is as follows: "For a given query topic Q, R-precision is the precision at R, where R is the number of relevant data for Q. In other words, if there are r relevant frames among the top-R retrieved frames, then R-precision is $\frac{r}{R}$" [131].

In the test process (Algorithm 7.5), if the top 20 search results, BL (where $|BL|= R = 20$) of the brute force algorithm given a query are used as a baseline, the testing search algorithm returns a list RL of 20 results then:

$$R\text{-precision} = \frac{|BL \cap RL|}{|BL|} * 100(\%) \tag{7.5}$$

It is to note that the R-precision is highly correlated to the well-known mean average precision (MAP) as discussed by Christopher et. al [132].

7.5.3 Running Time and Confidence of Finding an Exact Match

In this experiment, we examine the running time of our Exact-match algorithm at different settings of database size and number of prioritized features used ($\#features = \{2, 5, 10, all\}$). Since the default number of result of the Exact-match algorithm is only one ($R = 1$), instead of using R-precision, an "average confidence" measure, which is defined as a measure of the percentage when the algorithm returns a right exact match, will be used.

The results are shown in Figure 7-3. In this figure, we can see the correlation between the average search time and the data size and the correlation between the number of prioritized features and the average time and average confidence. It is not surprising that when all features are used, the Exact-match search algorithm returns a result with 100% confidence that it is the exact match. Although the search time is high and increases proportionally to the data size, it is reasonably low (about 0.15

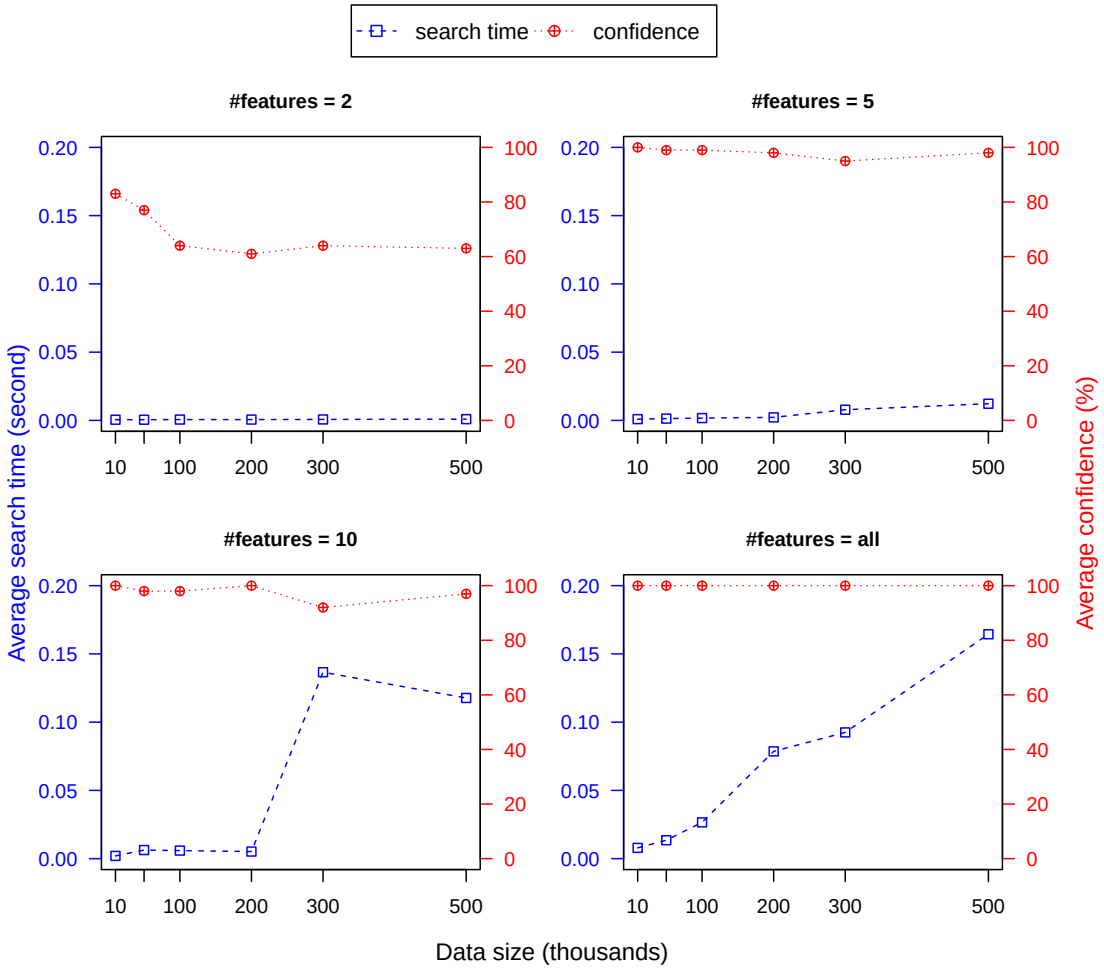


Figure 7-3: Performance of the proposed Exact match search algorithm by different data sizes and settings of number of prioritized features.

second).

Table 7.4 shows the average number of non-zero features of a query which is used as prioritized features at the default search mode ($m = \sum_i^d \lceil q_i \rceil \times p_i$ in Algorithm 5.3). In this case, m is the number of features with positive values for each query. In other words, these values of m are the ones that guarantee returning exact match with confidence of 100%. It is to see in this table, as d increases, m decreases, meaning the number of positive features decreases as the number of dimensions increases.

On the other hand, we can reduce the average search time by using a smaller number of prioritized features (let m denote this number) although if applying this,

Table 7.4: Average number of prioritized features m at default search mode.

Feature type	PHOG	Gabor	Color	CEDD	JCD	FCTH
Number of features (d)	40	60	63	144	168	192
Average m (10k databse)	40	60	34.84	28.68	33.24	16.52
Average m (50k databse)	40	60	36.24	28.42	34.1	17.14
Average m (100k databse)	40	60	36.56	28.64	33.52	18.02
Average m (200k databse)	40	60	36.14	29.4	34.98	17.76
Average m (300k databse)	40	60	31.54	26.44	30.42	15
Average m (500k databse)	39.2	58.8	33.22	26.56	30.38	15.12

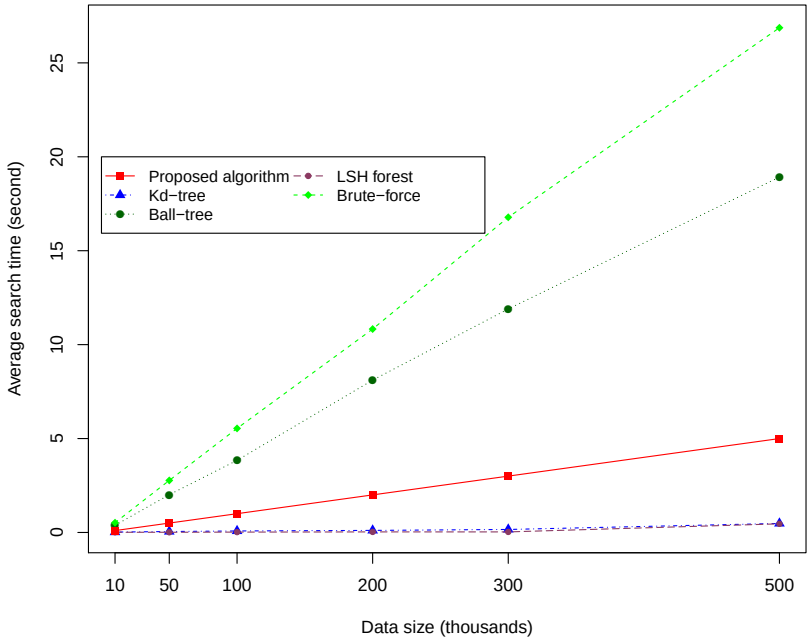
the returned result can only be guaranteed to be the right match with some probability. When $m = 2$, which is the lower bound of m in the Exact-macth algorithm (algorithm 5.3), we see the average search time is very low and stable regardless of data size. However, the average confidence is only about 60%.

As m increases, the average search time increases but also the the average confidence. When $m = 5$ and $m = 10$, we see that the average confidences are almost the same (slightly higher in the case $m = 5$) but the average search time are quite different, especially when the size of data is large (the number of data is greater than 300,000).

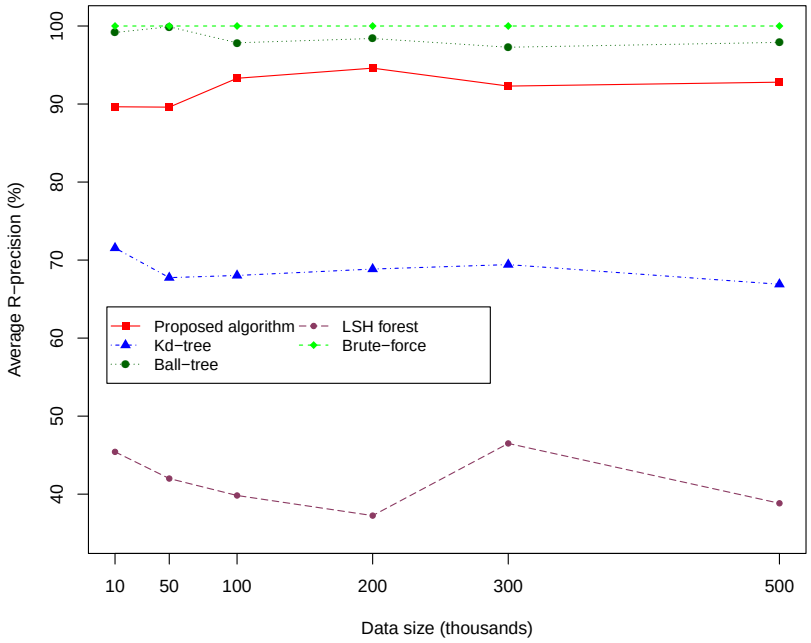
7.5.4 Comparative Search Time and Precision versus Data Size

In this second experiment, we compare the average search time for one query and the average R-precision of the proposed Combinatorial search algorithm with the brute force algorithm, and two spatial tree algorithms (KD-tree [133] and Ball-tree [79]) and a hashing algorithm (Local sensitive hashing (LSH) forest [134]) for different sizes of data[8]. The result is shown in Figure 7-4.

[8](1) KD-tree, ball-tree and LSH forest algorithms are implemented using *scikit learn nearest neighbors* modules. Source: http://scikit-learn.org/stable/modules/neighbors.html. (2) For large data sets, the maximum recursion limit of a tree algorithm can be exceeded, therefore set it before running to 10,000.



(a) Comparative average search time



(b) Comparative average R-precision

Figure 7-4: Comparative average search time and R-precision of the Combinatorial search algorithm to other search algorithms with increasing size of data.

In Figure 7-4a, the KD-tree and LSH forest algorithms response remarkably quick (less than 0.5 second for all databases), however, their precisions as in Figure 7-4b are also noticeably poor: less than 50% (LSH forest) or about 70% (KD-tree). On the contrary, Ball-tree algorithm obtains very high precision (Figure 7-4b) but responses comparatively as the brute force algorithm (Figure 7-4a).

In figures 7-4a and 7-4b, we can see the proposed Combinatorial search algorithm responses with reasonably high precision and short average search time. Although the average search time proportionally increases as the size of data increases, comparing to Ball-tree and brute force algorithms, this search time is fairly acceptable. It is to note that the response time of the Combinatorial search algorithm can be controlled by setting a time limit for it. In Figure 7-4a, the average search time is set so that the corresponding precision in Figure 7-4b is above 90%.

7.5.5 Precision of Combinatorial Search Algorithm

The third experiment examines the performance of our proposed Combinatorial search algorithm running with different settings of number of prioritized features (m) and response time limits and database size. The results are shown in Figure 7-5. In this figure, the overall precision drops as the size of database increases but gradually obtains reasonable precision as the limit time increases.

By comparing the average precision by different settings of m, we can see that a larger m does not return better precisions. It is because when m is large, the algorithm have to look for the candidates from more number of feature indexes, which can be useful until a some value of m. As we can see in Figure 7-5, when increasing m from 5 to 10, we obtain some better precisions for large databases ("300k" and "500k") but not when m is larger than 15.

Apparently the performance of the Combinatorial search algorithm is affected by the I/O access speed as shown in Figure 7-6. In this figure, the queries are set with $m = 10$ and done on two computers as described in section 7.2.2. The details of benchmarks for I/O performance of the two computers are in Table 7.2 in section 7.2.2. Figure 7-6 shows that the search algorithm is likely effected by the I/O access

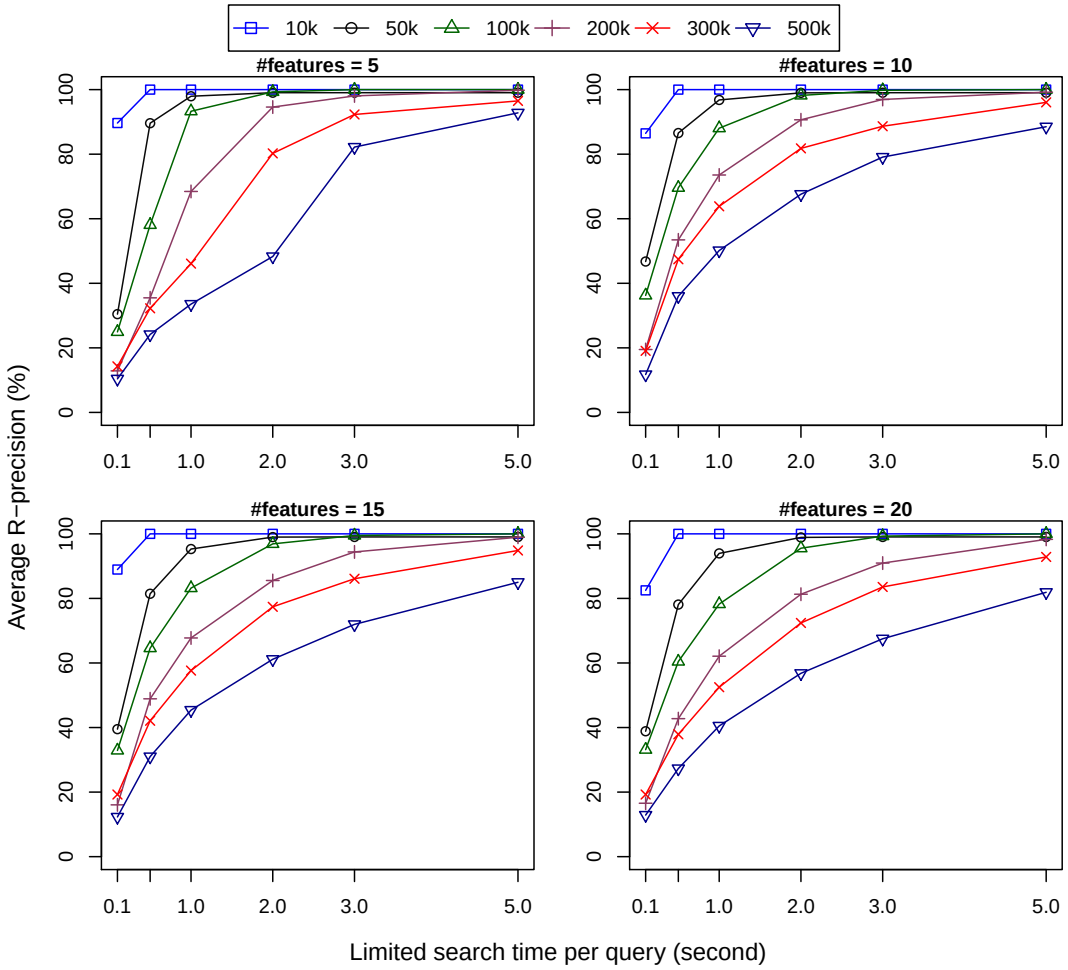


Figure 7-5: Precision of the proposed Combinatorial search algorithm by different response time limits, number of prioritized features, and data size.

speeds to the Bamboo Forest databases stored on storage devices. Averagely, the performance on the laptop computer which configures the databases on a SSD device yields about 10% higher in average R-precision. Based on the benchmarks of the two computers in Table 7.2, in which the desktop computer has nearly six times in terms of CPU power (GFlops) than the laptop, however, its random read/write performances are about 33 times less than of the laptop, it is reasonable to conclude that the performance of the Combinatorial search algorithm can be improved by implementing the databases on faster I/O storage devices.

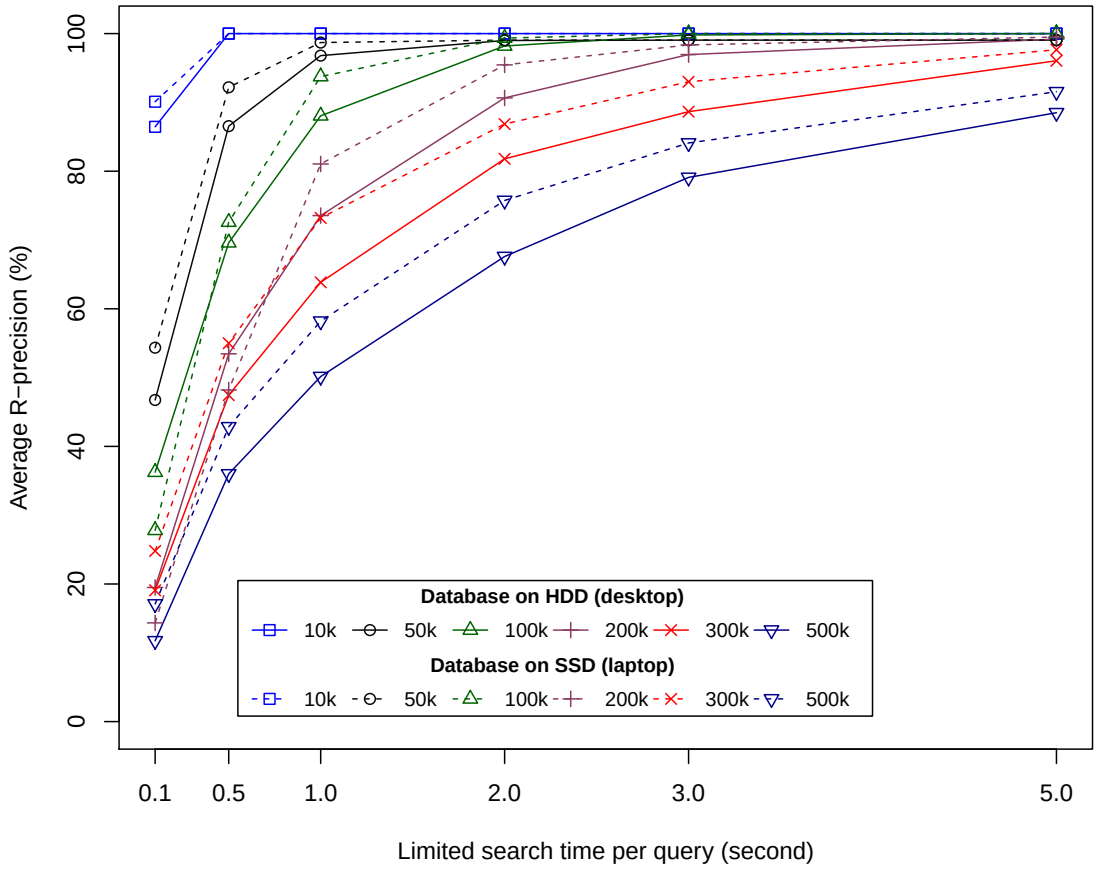


Figure 7-6: Performance of the proposed Combinatorial search algorithm implemented on two computers with different CPU and IO-access speeds.

7.5.6 Precision of Maxfirst Search Algorithm

In this experiment, the performance of the proposed Maxfirst search algorithm will be examined by setting its parameters with various values of response time limits, numbers of prioritized features, and size of data indexed in Bamboo forest databases. The response time limits are 0.1, 0.5, 1.0, 2.0, 3. 0, and 5.0 seconds. The number of prioritized features (m) is one from the set ($\#features = \{2, 5, 10, 15\}$). The databases are as described in Table 7.3.

The result using R-precision measurement in this experiment is shown in Figure 7-7. In this figure, the proposed Maxfirst search algorithm performs more stable and

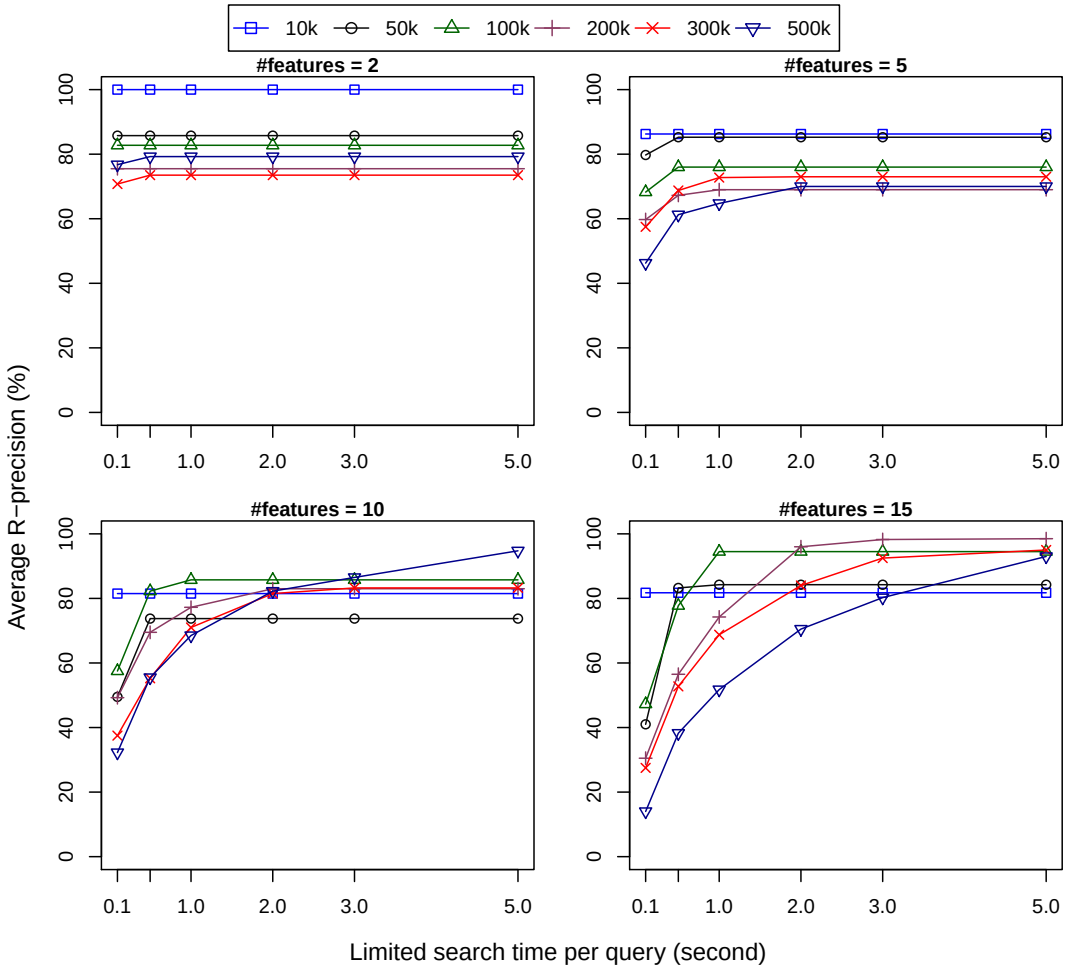


Figure 7-7: Average precision of the proposed Maxfirst search algorithm by different settings of number of prioritized features, data size, and response time limits.

efficient when the number of prioritized features (m) is small or the limited response time is large. As m increases, the average precisions decrease and apparently correlate to the size of data and the limited response time. This can be explained by the sparse distribution of the values of features when the number of features are large, meaning the "dominant" characteristics are lost and shared by many dimensions.

When m is small, increasing time limit does not necessarily increase the precision unless the time limit is set to infinite and in this case the Maxfirst search algorithm works equivalently as a brute force algorithm. However, when m is large, increasing time limit can increase the precision up to some limits.

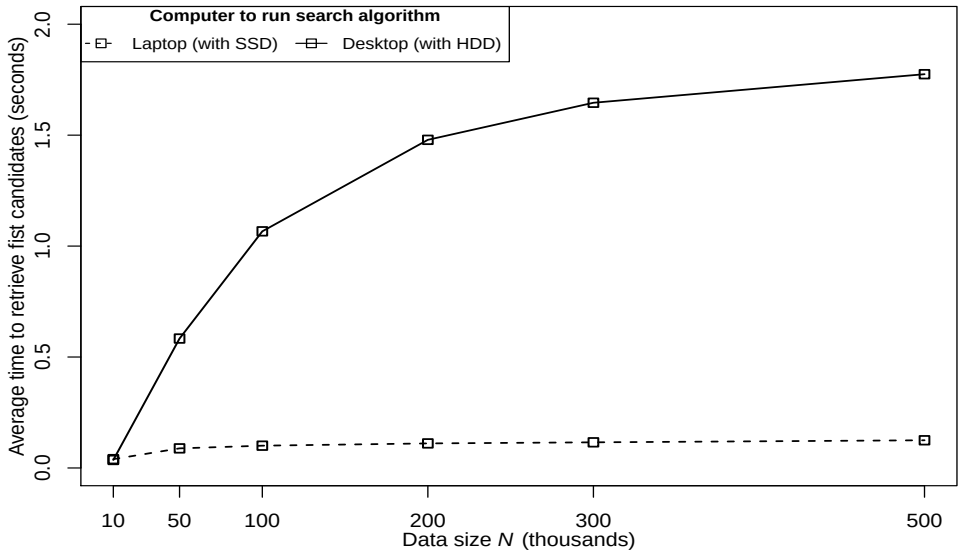
7.5.7 Revisiting Complexity of Search Algorithms

In this section, we revisit the theoretical complexity analysis of three search algorithms which is discussed in section 5.4 of chapter 5 by empirical experiments. The average number of retrieved candidates for a setting of a fixed number prioritized features ($m = 5$), a feature type (*Color*) and varying data size and search time limits will be used as the representative values for complexity.

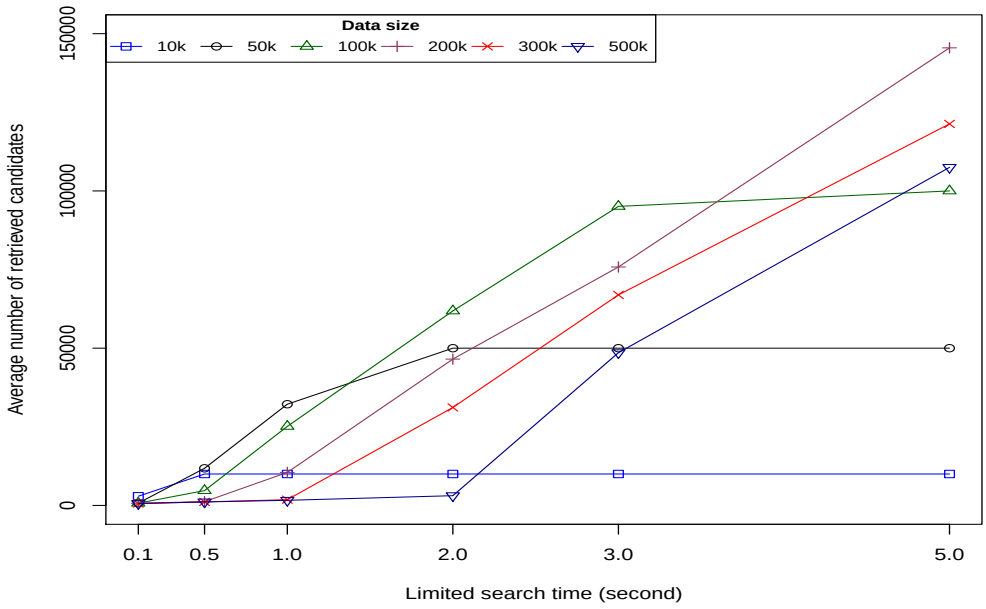
Figure 7-8 shows the results for the Combinatorial search algorithm. There are two subfigures in this figure: a subfigure, Figure 7-8a shows the average time to retrieve first candidates by increasing data size and a subfigure, Figure 7-8b shows the average number of retrieved candidates on each database of different data size by different time limits.

Figure 7-8a has two curves that describe the time measured on the laptop and desktop computers. Both curves indicate a log-like behavior that confirms the complexity $O(m \log N)$ to retrieve first candidates as discussed in section 5.4.

Figure 7-8b shows three important points: (1) "flat" heads for small time limits indicates the cost to open the feature index files and find the first candidates, (2) the linearly proportional to the time limits at the middle, and (3) the "flat" tails again, which indicate all records in each index file are already retrieved. Also in this figure, we can see the "flat" head becomes longer as the size of data increases. This is due to the $m \log N$ factor in the complexity $O(m \log N + C)$. Moreover, the line of a database with a small size is higher in the plot comparing to other databases with bigger size databases. This can be explained as the number of C retrieved for a fixed time limit descreases due to $m \log N$ factor in the complexity $O(m \log N + C)$.

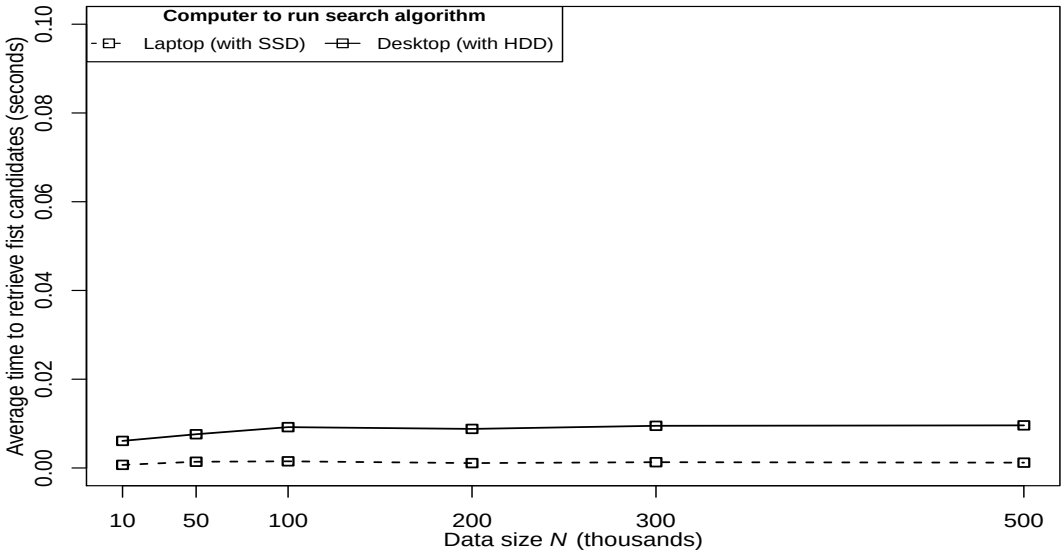


(a) Confirming $O(m \log N)$ complexity of Combinatorial search algorithm to retrieve first candidates.

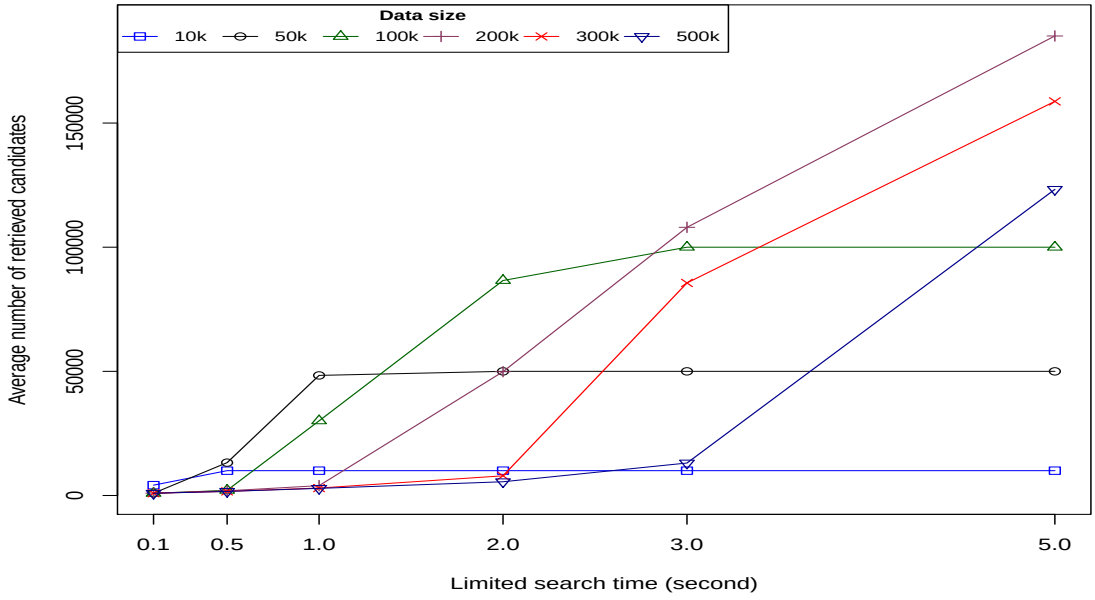


(b) Complexity of the proposed Combinatorial search algorithm based on average retrieved candidates: "flat heads" due to $O(m \log N)$, linear area C by increasing response time limits, and saturated "flat tails" due to data size. (Run on the desktop computer.)

Figure 7-8: Confirming $O(m \log N + C)$ complexity of Combinatorial search algorithm.



(a) Complexity of Maxfirst search algorithm to retrieve first candidates ($O(m)$).



(b) Complexity of Maxfirst search algorithm based on average retrieved candidates: "flat heads" due to $O(m)$, linear area (corresponding to C) by increasing response time limits, and saturated "flat tails" due to data size. (Run on the desktop computer.)

Figure 7-9: Confirming $O(m + C)$ complexity of the proposed Maxfirst search algorithm.

Similarly, Figure 7-9 shows the confirming results for the proposed Maxfirst search algorithm. There are two subfigures in this figure: a subfigure, Figure 7-9a shows the average time to retrieve first candidates by increasing data size and a subfigure, Figure 7-9b shows the average number of retrieved candidates on each database of different data size by different time limits.

Figure 7-9a has two lines that describe the time measured on the laptop and desktop computers. Both lines indicate a linear-time behavior that confirms the complexity $O(m)$ factor in the $O(m+C)$ of the Maxfirst search algorithm to retrieve first candidates as discussed in section 5.4.

Similar to Figure 7-8b, Figure 7-9b shows three important points: (1) "flat" heads for small time limits indicates the cost to open the feature index files and find the first candidates, (2) the linearly proportional to the time limits at the middle, and (3) the "flat" tails again, which indicate all records in each index file are already retrieved. However, the most noteworthy attribute in this figure comparing to Figure 7-8b is the heads of the lines. They are not as "flat" as in Figure 7-8b which corresponds to the Combinatorial search algorithm. However, there is still a dramatical changes in each line in Figure 7-9b. It is likely due to the *page caching* of the Linux system on which the experiment was tested. As described in Algorithm 7.5, rebooting the computer was done only once before running the program for this experiment but not everytime the program tries to access the database. As a consequence, it is likely that the "sudden changes" at result lines in Figure 7-9b were because the corresponding database files have been cached mostly to the memory. The size of databases are in Table 7.3. A more accurate experiment regarding this "phenomenon" is left as a future work. Nonetheless, the results in Figure 7-9 reasonably confirm a complexity $O(m+C)$ of the proposed Maxfirst search algorithm.

Figure 7-10 shows both the average number of retrieved candidates (y-axis) and average search time (z-axis) for different sizes of data. In this experiment, the default search mode for configuring the number of prioritized features ($m = \sum_{i,\lceil q_i \rceil=1}^{d} q_i \times p_i$ in Algorithm 5.3) is used. As shown in Table 7.4, the average value of m is 34.

In Figure 7-10, we can see the complexity of the Exact-match search algorithm

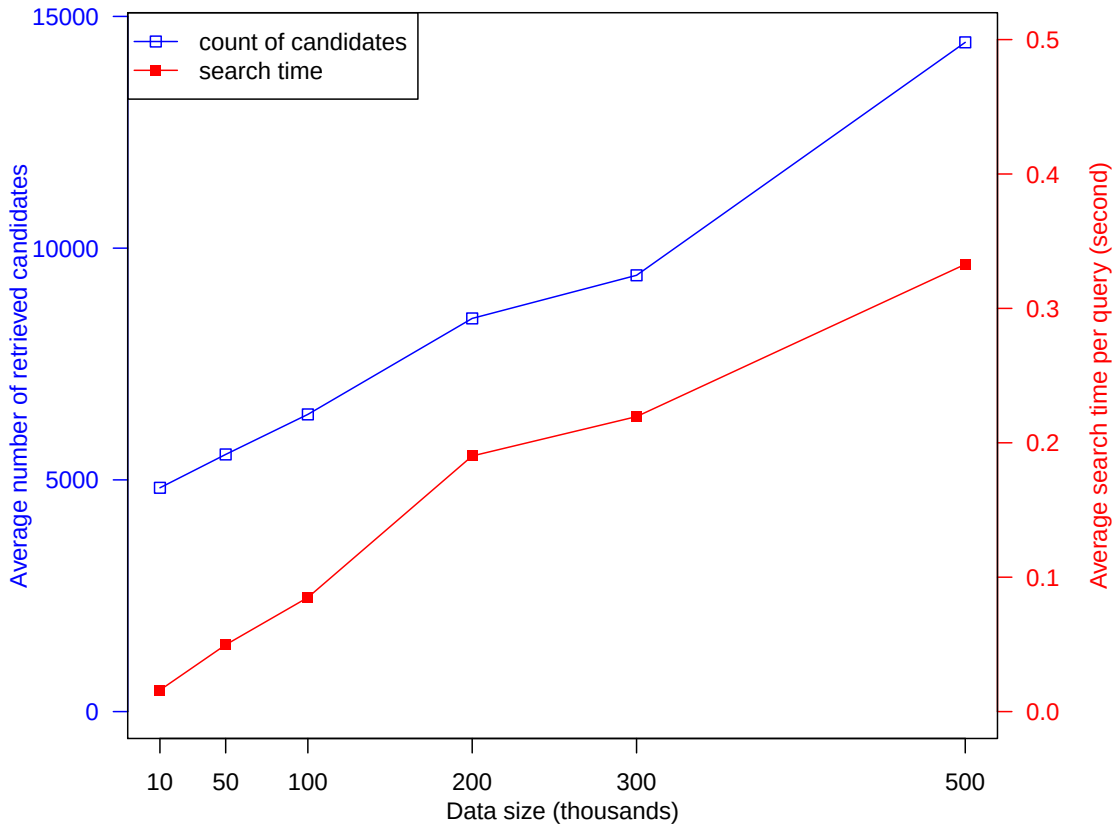


Figure 7-10: Confirming $O(m \log N + C)$ complexity of the proposed Exact-match search algorithm using average number of retrieved candidates (complexity analysis in section 5.4).

depends on the data size (N) in a *log*-like relation. This figure also shows that the number of candidates needed to be retrieved to induce an exact match for the query increases as the data size increases. This is referred to the factor C in the complexity analysis of the algorithm of $O(m \log N + C)$ as shown in Table 5.2.

Comparing the average number of retrieved candidates in figures 7-8b and 7-9b to Figure 7-10, we see that the number of retrieved candidates to find exact-match candidates is greatly less. This indicates the factor C in the Maxfirst search and Combinatorial search algorithms affect the complexity more than in the Exact-match

search algorithm.

Based on the above experimental results on complexity of the proposed search algorithms, we can see that there exists some lower bounds for response time limits due to configuration and file access and some values at which the search algorithms gain a steep "linear" performance. These values are shown in Figure 7-8b and Figure 7-9b.

7.5.8 Discussion

The previous sections have discussed the performance evaluation of the proposed search algorithms by several settings of their parameters. It is not surprising that in all three search algorithms, increasing the time limits gains better precisions. However, it is interesting to notice that the choice of the number of prioritized features (m) is critical in order to avoid redundant computations while aiming for higher performance.

7.5.8.1 Interesting Choices of m

In the case of exact match finding (using the Exact-match search algorithm), if the goal is to find the guaranteed right answer (100% *confidence*), then m should not be chosen less than the number of features that the content preference indicates. In other words, all dimensions in the selected subspace will be used. However, if the goal is to find the right answer with some (high) probability and in a short time, then m can be set for a value not very low or high. Figure 7-3 shows $m = 5$ is a reasonable setting.

In the same way for similar matches finding (k-NN problem) using the Combinatorial search algorithm, a low or high value set for m does not gain the higher performance. Figure 7-5 shows $m = 10$ is a reasonable setting.

On the contrary, the Maxfirst search algorithm performs stably when m is low and when m is high, it can only gain higher precision with higher settings for time limits.

7.5.8.2 Analysis of Bad Search Cases

This discussion focuses on the question "When do the proposed algorithm suffer?". A typical bad case is intuitively shown in Figure 7-11.

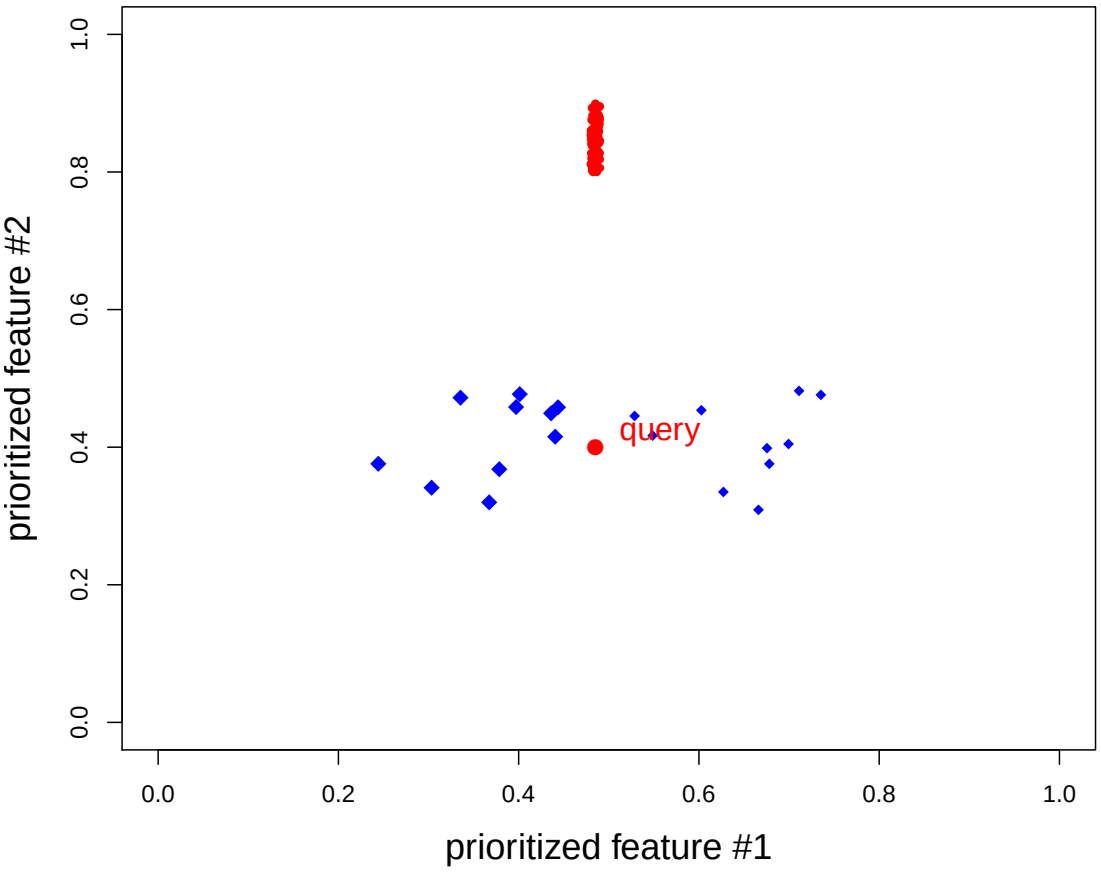


Figure 7-11: A typical bad case of the proposed heuristic search strategy in which the searching process wastes time at "local" candidates.

Figure 7-11 shows a query point (the big red point) and many other target data points (red and blue points) in two-dimensional space. The query point has x-axis value larger than y-axis value, therefore the x-axis is prioritized higher than the the y-axis. A search process based on a heuristic search strategy described in section 3.1 will start checking the candidates which has x-axis value close to the x-axis value of the query point. In this case, those candidates are the red points. Intuitively,

the actually distances from the query to those data points are larger than to some blue data points. In other words, the search process wastes time at some "local" candidates.

This kind of bad cases is supposed to happen in several special situations: (1) the size of dataset increases, (2) there are two many duplicated data in the dataset, or (3) there are non-discernible features used for representing data.

The degree of a proposed search algorithm suffering from this "local candidate" phenomenon depends on the distribution of data on the search space. However, it is likely that the Maxfirst search algorithm will suffer more than other two algorithms.

7.5.8.3 Comments on Indexing Time and Size of Databases

The final comment is regarding to the indexing time and size of indexed databases. In section 7.5.4, only the average search time per query of each algorithm was used to compare the performance of the proposed Combinatorial search with other search algorithms. It is to note that the brute force algorithm performs similarity directly on the data without any pre-indexed database. KD-tree, Ball-tree and LSH forest search algorithm index the data into their defined data structures (in-memory database) and query for results on those database. The indexing time for each algorithm for each size of data was not counted in the search time. Likewise, the proposed search algorithm works with the pre-indexed disk-resident databases and the index time was not taken into account. The storage size of the databases have also not yet concerned.

Chapter 8

Discussion and Conclusion

> Intelligence does not always define wisdom, but adaptability to change does.
>
> *Debasish Mridha*

There has never been like the world we are now living in. Everything is changing so fast, so diversely. "Preferences" of users have become essential factors in computing models and are put at the center of the design for new technologies as well as new services. As a consequence, the "adaptability" has gained many attentions in research fields as an important evaluation measure of a computing system.

This thesis addresses the dynamic adaptability of a multimedia search system to the contexts of its users by proposing a new query creation and search system. The system aims to give an integrated solution for three-pillar search problem which includes (1) the emergence of multimedia data to many large scales, (2) the varying contexts of users at query time, and (3) the computing complexity. By explicitly defining three distinctive contextual aspects of user preferences, the proposed system is a yet straightforward but powerful indexing and search system for multimedia data.

The dynamic contexts vary from the imaginations of the users expressed in the queries to their expectations to the results returned by the search system including their desire to control the response time of each search. In such contexts, the pruning

search mechanism shows its attractive adaptability. For example, to search by dominant features of the input image, the Maxfirst search algorithm is the most proper due to its intrinsic characteristic when searching for candidates. On other hand, the Combinatorial search algorithm targets for similars searching with capacity to control the response time. The Exact-match search algorithm using an ingenious idea of covering rectangles to find perfectly matched candidates. It has been shown by experiments in the previous chapter that those changing expectations of querying and searching can be done effectively without changing the structure of database, meaning on a same indexed database. This is the essential difference to other approaches to the adaptability and high performance of a search system.

The next sections will discuss the applicability of the multicontext-adaptive query creation and search system and conclude this thesis by some major research findings.

8.1 Applicability of the Multicontext-adaptive Query Creation and Search System

The proposed multicontext-adaptive query creation and search system can be used as a general content-based multimedia search system. However, some representatively specific applications can be designed such as an imagination-based image search application, a frame-wise video navigation application, and a video monitoring and analyzing application. Table 8.1 describes some representative use cases of the proposed query creation and search system.

8.1.1 Imagination-based Image Search Application

The imagination-based image search application is the most suitable to the task of exploring visual art archives where colors are greatly important in conveying ideas, impression, and emotions. By using the hierarchical color sampling method discussed in section 6.2, it is possible to index the complex information of hue, saturation, and intensity values of colors into a systematic structure that can be used for later querying

Table 8.1: Some representative use cases of the multicontext-adaptive query creation and search system.

Representative application	**Representative task**	**Sample users**	**Applicable methods**
Imagination-based image search	Exploring visual art space by colors	Art students, art professionals	Adaptive color-based query creation functions (section 6.3), adaptive search algorithms (chapter 5)
Frame-wise video navigation	Navigating to relevant frame locations in videos	General video search users, film editors	Combinatorial search, Exact-match search algorithms (sections 5.2-5.3)
Surveillance video monitoring and analyzing	Monitoring and alerting environmental situations	Building managers, environmentalists	Adaptive color-based query creation functions (section 6.3), Maxfirst search algorithm (section 5.1)

using colornames in natural languages. The adaptive color-based query creation in section 6.3.2 supports a query of both a keyword and an input image to discover the colorful space of visual art. This combination of words and input images is useful since the shade of colors are often greatly delicate and hard to express or recall when searching for images. Instead of searching for a general image with a "blue" color which is not refined enough, using an input image which has the desired colors and inputing a keyword "light blue" suggesting an imagination of the result images that contain *these* "light blue" color.

Furthermore, the supporting functions for combining subspaces of adapted colors which are described in section 6.3.3 are rather advantageous. By using them, it is possible to dynamically express the imagination of the result images which are helpful when finding new visual information in the visual space. For an art student or professional, this can also helpful in finding a useful combination of colors to create

a painting palette for practice.

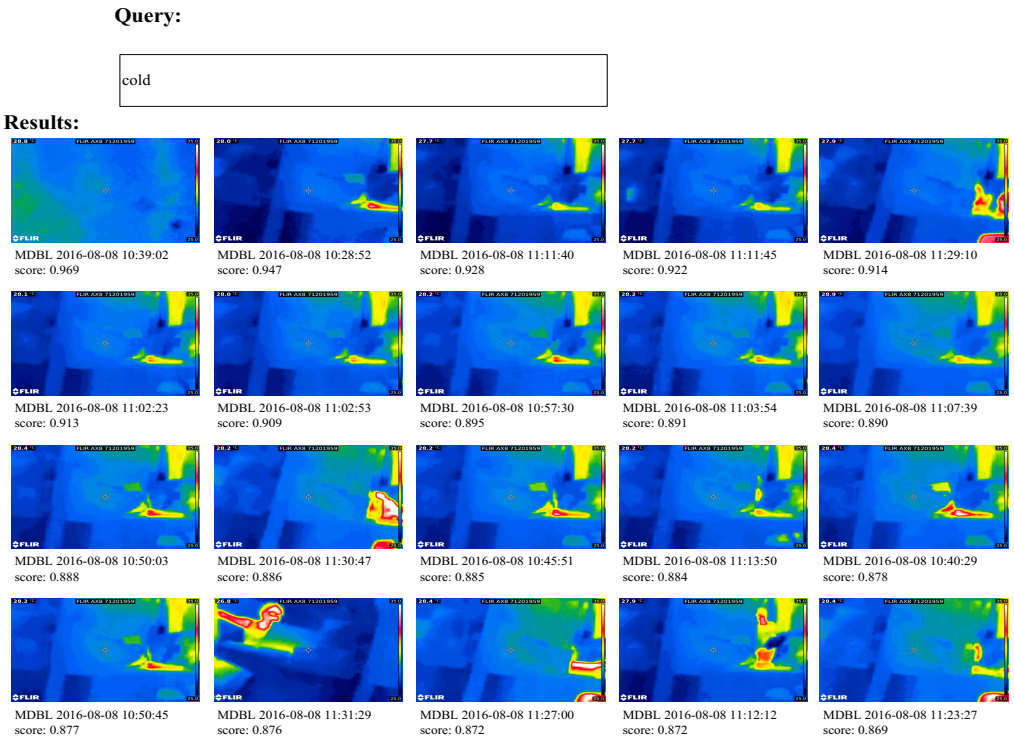
8.1.2 Frame-wise Video Navigation Application

This application is used for the representative task: navigating users to relevant frame locations in long duration videos which are similar to the objects of interest in the input image. It is obviously useful to general end-users of a video search system since such a system is not yet existing in practice. Moreover, it can also be thought helpful for film editors when they need to find relevant materials from a list of videos for demonstration. In such scenarios, the users can input an image which contains the interested objects regarding their color, shape, or texture to find similar objects that appear somewhere in the videos. The proposed Combinatorial and Exact-match search algorithms (Algorithms 5.2 and 5.3) are the most suitable to find the answers in these search scenarios.

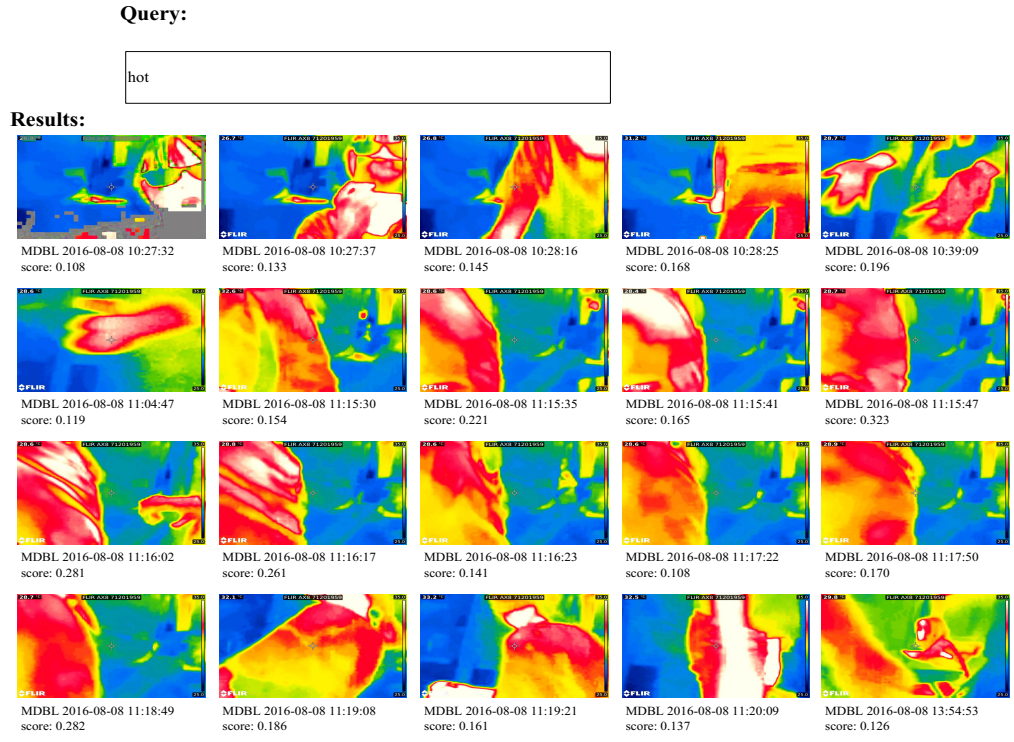
8.1.3 Video Monitoring and Analyzing Application

The Maxfirst search algorithm (Algorithm 5.1) presents a new type of search for information. Unlike conventional search problem which is often searching for "similar", the Maxfirst search algorithm conducts a search for substances which has "dominant" features. This kind of retrieval tasks emerges representatively in monitoring and analyzing tasks such as video surveillance.

A manager of a building, for example, wants to check when and where the room temperature is coolest or hottest. She or he can use the adaptive color indexing in section 6.2 to index the thermal colors of video frames with several levels from "cool" to "hot", then apply the Maxfirst search algorithm to find the relevant information. This representative task is demonstrated in Figure 8-1. In this figure, subfigure 8-1a shows when the room is "cold" sorted by the coldness of the room temperature represented by the proportion of more vivid blue colors. Comparatively, subfigure 8-1b shows when the room is the "hot" by a order of timeline.



(a) Navigating to relevant frames and sorting by relevance scores to the input context "cold"



(b) Navigating to relevant frames and sorting by timestamps regarding the input context "hot"

Figure 8-1: A demonstration of an environmental monitoring application with context-adaptive querying.

8.2 Summary of Research Findings

The following list summarizes the major findings in this thesis:

- **Context-dependent Query Formulation**: The experiments on the context-dependent image search system using color features show that by indexing color features of image data systematically so that they can be appropriately selected to construct subspaces for search algorithms, the "curse of high dimensionality" can be alleviated. Moreover, it can yield more powerful consequences: (1) preserving delicate contents of image data, (2) achieving higher precisions, and (3) providing capability to combine and manipulate subspaces to reflect users' content preferences that are expressed in either natural language words or imagination.

- **Multicontext-adaptive Search Pruning Mechanism**: The proposed search mechanism that including three search algorithms is studied intensively using the frame-wise video navigation system. The experiments show the effectiveness of the heuristic search strategies based on value-sorted inverted indices in the Bamboo Forest database model. The search algorithms can obtain comparatively high precisions in a relative low response time comparing to other state-of-the-art methods including tree-based and hashing methods. The experiments also show the scalability of the system for large-scale datasets. Beyond that achievement, it is shown the advantages of using independent indices to construct disk-resident database in providing more dynamic controls over users' contextual preferences including content, intention, and response time.

8.3 Open Questions in Future Work

The previous chapters in this thesis have described the essence of the multicontext-adaptive query creation and search system but there are still many adaptations, improvements, and tests have been left for the future. Future work concerns more refined design implementation of the proposed indexing and search algorithms to yield

more efficiency and deeper analysis of applications of the proposed system. At the same time, some new questions are to be asked based on the work in this thesis, which suggest some future research directions.

The following sections introduce some ideas.

8.3.1 Improvement of Indexing and Search Methods

The current design of the Bamboo Forest database model has not yet supported deletion of records from an indexed database. Moreover, some further studies such (1) index compression in order to reduce the storage size of a Bamboo Forest database and (2) fast access to index including memory buffering or page caching are expected.

Regarding search algorithms, it is a potential direction to investigate a parallel search process on each prioritized feature when finding candidates. This is supposed to improve the performance of the search algorithms (or *is it not?*). In addition, a further work on how to avoid wasting time in local candidates as discussed in section 7.5.8.2 would be both interesting and important.

8.3.2 New Questions

Is there any new type of "contexts" of users that a search engine can (or should) adapt to? Can the proposed search mechanism adapt to such contexts? And how?

This thesis defines and uses three types of contexts of users to organize an adaptive search process: content, intention, and response time. But an eager and curious audience would doubt if there is more. The answer is, obviously, yes. A simple example of content-related context can be: "I want to search for video frames similar to this image in the last *10 days*."[1]

The question is yet so simple and reasonably useful but also challenging to the current indexing and search methods to get a desired performance. It reveals a range of problems: context-based filtering. The intuition for this type of problems is shown in Figure 8-2.

[1]Thanks Professor Yoshiyasu Takefuji for raising this question.

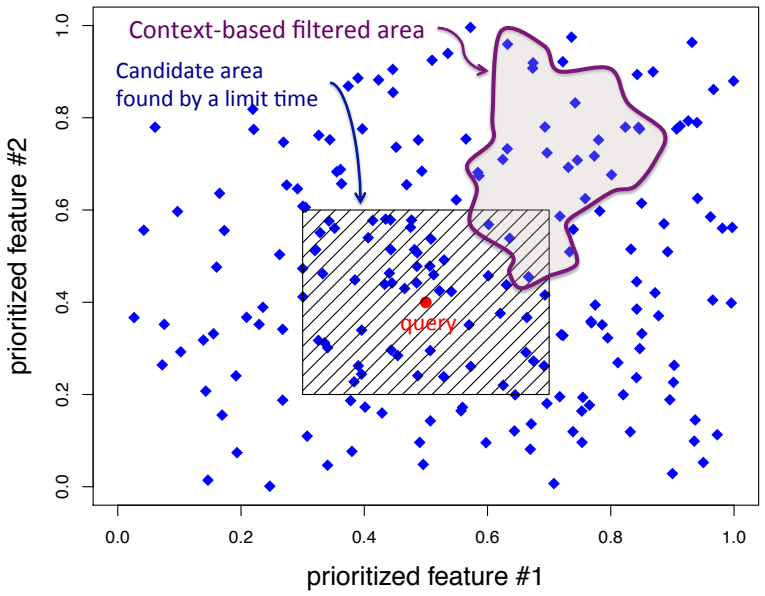


Figure 8-2: New questions in future work: (1) how to express more complex contexts (shown as the "context-based filtered area") and (2) how to quickly find candidates with respect to this filter (quickly find the intersecting area without checking the whole "candidate area".)

In Figure 8-2, a new context (e.g., "in the last 10 days") is treated as a filter to limit the desired candidates into a complicated polygon, while the search mechanism proposed in this thesis (chapters 3 and 5) expands the area of candidates based on orthogonal dimensions (a rectangle). How can we quickly find the candidates in the intersected area without wasting time checking unrelated candidates? One potential solution for this question is to organize the indices of the current Bamboo Forest database model using a spatial partition methods if given a new well-defined context.

Final comment It is an exciting future to discover the answers to those questions and to explore the new territory of dynamic context-based query creation and search methods.

Bibliography

[1] Jean Clottes. *What is Paleolithic Art?: Cave Paintings and the Dawn of Human Creativity*. University of Chicago Press, 2016.

[2] Mary Meeker. 2016 internet trends. Code conference, Kleiner Perkins Caufield and Byers (KPCB), June 2016.

[3] Andrei Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, September 2002.

[4] Donald Owen Case. *Looking for information: A survey of research on information seeking, needs and behavior*. Emerald Group Publishing, 2012.

[5] Paul André, Edward Cutrell, Desney S Tan, and Greg Smith. Designing novel image search interfaces by understanding unique characteristics and usage. In *IFIP Conference on Human-Computer Interaction*, pages 340–353. Springer, 2009.

[6] Amanda Spink and Bernard J Jansen. *Web search: Public searching of the Web*, volume 6. Springer Science & Business Media, 2006.

[7] Alan Hanjalic, Christoph Kofler, and Martha Larson. Intent and its discontents: the user at the wheel of the online video search engine. In *Proceedings of the 20th ACM international conference on Multimedia*, pages 1239–1248. ACM, 2012.

[8] Hanwang Zhang, Zheng-Jun Zha, Yang Yang, Shuicheng Yan, Yue Gao, and Tat-Seng Chua. Attribute-augmented semantic hierarchy: Towards a unified framework for content-based image retrieval. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 11(1s):21, 2014.

[9] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. On the surprising behavior of distance metrics in high dimensional space. In *International Conference on Database Theory*, pages 420–434. Springer, 2001.

[10] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is nearest neighbor meaningful? In *International conference on database theory*, pages 217–235. Springer, 1999.

[11] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, volume 98, pages 194–205, 1998.

[12] Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[13] Julien Jacques and Cristian Preda. Functional data clustering: a survey. *Advances in Data Analysis and Classification*, 8(3):231–255, 2014.

[14] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *International Symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer, 1999.

[15] Alan C Bovik. *Handbook of image and video processing*. Access Online via Elsevier, 2010.

[16] Steve Krug. *Don't make me think: A common sense approach to web usability*. New Riders, 2nd edition edition, 2005.

[17] Richard W. DeVaul. Introduction to wearable computing.

[18] Patrick Brézillon. Context in artificial intelligence: I. a survey of the literature. *Computers and artificial intelligence*, 18:321–340, 1999.

[19] Varol Akman and Mehmet Surav. Steps toward formalizing context. *AI magazine*, 17(3):55, 1996.

[20] Takashi Kitagawa and Yasushi Kiyoki. A mathematical model of meaning and its application to multidatabase systems. In *Proceedings RIDE-IMS '93: Third International Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems*, pages 130–135, Apr 1993.

[21] Yasushi Kiyoki, Takashi Kitagawa, and Takanari Hayama. A metadatabase system for semantic image search by a mathematical model of meaning. *ACM Sigmod Record*, 23(4):34–41, 1994.

[22] Beng Chin Ooi, Ken J McDonell, and Ron Sacks-Davis. Spatial kd-tree: An indexing mechanism for spatial databases. In *IEEE COMPSAC*, volume 87, page 85, 1987.

[23] Kaushik Chakrabarti and Sharad Mehrotra. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *VLDB*, pages 89–100, 2000.

[24] Akrivi Vlachou, Christos Doulkeridis, and Yannis Kotidis. Angle-based space partitioning for efficient parallel skyline computation. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 227–238. ACM, 2008.

[25] Lawrence Cayton. Fast nearest neighbor retrieval for bregman divergences. In *Proceedings of the 25th international conference on Machine learning*, pages 112–119. ACM, 2008.

[26] 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*, pages 950–961. VLDB Endowment, 2007.

[27] Yasushi Kiyoki, Takashi Kitagawa, and Takayuki Miyahara. A fast algorithm of semantic associative search for databases and knowledge bases. In *Information modelling and knowledge bases VII*, pages 44–58. IOS Press, 1996.

[28] Akiko Miyagawa, Yasushi Kiyoki, Takayuki Miyahara, and Takashi Kitagawa. A fast semantic associative search algorithm for image databases. *Transactions*, 41:1–10, 2000.

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

[30] Christoph Kofler, Martha Larson, and Alan Hanjalic. User intent in multimedia search: A survey of the state of the art and future challenges. *ACM Comput. Surv.*, 49(2):36:1–36:37, August 2016.

[31] Mathias Lux, Christoph Kofler, and Oge Marques. A classification scheme for user intentions in image search. In *CHI'10 Extended Abstracts on Human Factors in Computing Systems*, pages 3913–3918. ACM, 2010.

[32] Marian Kogler, Mathias Lux, and Oge Marques. Adaptive visual information retrieval by changing visual vocabulary sizes in context of user intentions. In *Multimedia on the Web (MMWeb), 2011 Workshop on*, pages 40–42. IEEE, 2011.

[33] Yong Rui, Thomas S Huang, Michael Ortega, and Sharad Mehrotra. Relevance feedback: a power tool for interactive content-based image retrieval. *Circuits and Systems for Video Technology, IEEE Transactions on*, 8(5):644–655, 1998.

[34] Thomas S Huang and Xiang Sean Zhou. Image retrieval with relevance feedback: From heuristic weight adjustment to optimal learning methods. In *Image Processing, 2001. Proceedings. 2001 International Conference on*, volume 3, pages 2–5. IEEE, 2001.

[35] David McG Squire, Wolfgang Müller, Henning Müller, and Jilali Raki. Content-based query of image databases, inspirations from text retrieval: inverted files, frequency-based weights and relevance feedback. In *Pattern Recognition Letters*, pages 143–149. Citeseer, 1999.

[36] Roman Kreuzer, Michael Springmann, Ihab Al Kabary, and Heiko Schuldt. An interactive paper and digital pen interface for query-by-sketch image retrieval. In *European Conference on Information Retrieval*, pages 317–328. Springer, 2012.

[37] B Szántó, P Pozsegovics, Z Vámossy, and Sz Sergyan. Sketch4match–content-based image retrieval system using sketches. In *Applied Machine Intelligence and Informatics (SAMI), 2011 IEEE 9th International Symposium on*, pages 183–188. IEEE, 2011.

[38] Eugenio Di Sciascio, G Mingolla, and Marina Mongiello. Content-based image retrieval over the web using query by sketch and relevance feedback. In *International Conference on Advances in Visual Information Systems*, pages 123–130. Springer, 1999.

[39] Carlton W Niblack, Ron Barber, Will Equitz, Myron D Flickner, Eduardo H Glasman, Dragutin Petkovic, Peter Yanker, Christos Faloutsos, and Gabriel Taubin. Qbic project: querying images by content, using color, texture, and shape. In *IS&T/SPIE's Symposium on Electronic Imaging: Science and Technology*, pages 173–187. International Society for Optics and Photonics, 1993.

[40] Ying Liu, Dengsheng Zhang, Guojun Lu, and Wei-Ying Ma. A survey of content-based image retrieval with high-level semantics. *Pattern recognition*, 40(1):262–282, 2007.

[41] Shiori Sasaki, Yoshiko Itabashi, Yasushi Kiyoki, and Xing Chen. An image-query creation method for representing impression by color-based combination of multiple images. *Information Modelling and Knowledge Bases XX*, pages 105–112, 2009.

[42] Yasuhiro Hayashi, Yasushi Kiyoki, and Xing Chen. An image-query creation method for expressing user's intentions by combining multiple images. In *Information Modelling and Knowledge Bases XXI*, volume 206, pages 188 – 207. IOS Press, 2010.

[43] Diep Thi Ngoc Nguyen, Shiori Sasaki, and Yasushi Kiyoki. Imagination-based image search system with dynamic query creation and its application. *The 13th IASTED International Conference on Software Engineering and Applications SEA*, November 2010.

[44] Diep Thi Ngoc Nguyen, Shiori Sasaki, and Yasushi Kiyoki. 5dpicmap: Imagination-based image search system with spatiotemporal analyzers. *IADIS International Conference, e-Society 2011*, March 2011.

[45] Yasuhiro Hayashi, Yasushi Kiyoki, and Xing Chen. A combined image-query creation method for expressing users intentions with shape and color features in multiple digital images. In *Information Modelling and Knowledge Bases XXII*, pages 258–277. IOS Press, 2011.

[46] Ali Ridho Barakbah and Yasushi Kiyoki. An emotion-oriented image search system with cluster based similarity measurement using pillar-kmeans algorithm. In *Proceedings of the 2011 conference on Information Modelling and Knowledge Bases XXII*, pages 117–136. IOS Press, 2011.

[47] Yasushi Kiyoki, Shiori Sasaki, Nhung Trang Nguyen, and Diep Thi Ngoc Nguyen. Cross-cultural multimedia computing with impression-based semantic spaces. In *Conceptual Modelling and Its Theoretical Foundations*, pages 316–328. Springer, 2012.

[48] Diep Thi Ngoc Nguyen and Yasushi Kiyoki. An imagination-based query creation method for image retrieval. In *Information Modelling and Knowledge Bases XXVI*, Frontiers in Artificial Intelligence and Applications. IOS Press, 2013.

[49] Gonzalo Vaca-Castano and Mubarak Shah. Semantic image search from multiple query images. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 887–890. ACM, 2015.

[50] Simily Joseph and Kannan Balakrishnan. Multi-query content based image retrieval system using local binary patterns. *International Journal of Computer Applications*, 17(7):1–5, 2011.

[51] Diep Thi Ngoc Nguyen, Shiori Sasaki, and Yasushi Kiyoki. Imagination-based travel designing system with 5d world picmap. In *The 1st Indonesian-Japanese Conference on Knowledge Creation and Intelligent Computing (KCIC)*, 2012.

[52] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2169–2178. IEEE, 2006.

[53] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[54] John D Lafferty and Larry Wasserman. Challenges in statistical machine learning. *Statistica Sinica*, 16:307, 2006.

[55] Eamonn Keogh and Abdullah Mueen. Curse of dimensionality. In *Encyclopedia of Machine Learning*, pages 257–258. Springer, 2011.

[56] Jerome H Friedman. On bias, variance, 0/1 loss, and the curse of dimensionality. *Data mining and knowledge discovery*, 1(1):55–77, 1997.

[57] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[58] Brian Hayes. An adventure in the nth dimension. *American Scientist*, 99(6):442–446, 2011.

[59] Miloš Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. Hubs in space: Popular nearest neighbors in high-dimensional data. *Journal of Machine Learning Research*, 11(Sep):2487–2531, 2010.

[60] Hans-Peter Kriegel, Arthur Zimek, et al. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 444–452. ACM, 2008.

[61] Michael E Houle, Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Can shared-neighbor distances defeat the curse of dimensionality? In *International Conference on Scientific and Statistical Database Management*, pages 482–500. Springer, 2010.

[62] Charles D Feustel and Linda G Shapiro. The nearest neighbor problem in an abstract metric space. *Pattern Recognition Letters*, 1(2):125–128, 1982.

[63] Peter N Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, volume 93, pages 311–21, 1993.

[64] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[65] Jonathon S Hare, Sina Samangooei, David P Dupplaw, and Paul H Lewis. Imageterrier: an extensible platform for scalable high-performance image retrieval. In *Proceedings of the 2nd ACM International Conference on Multimedia Retrieval*, page 40. ACM, 2012.

[66] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[67] Herwig Lejsek, Friorik Ásmundsson, B Th Jónsson, and Laurent Amsaleg. Nv-tree: An efficient disk-based index for approximate search in very large high-dimensional collections. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(5):869–883, 2009.

[68] Diana Moise, Denis Shestakov, Gylfi Gudmundsson, and Laurent Amsaleg. Indexing and searching 100m images with map-reduce. In *Proceedings of the 3rd ACM conference on International conference on multimedia retrieval*, pages 17–24. ACM, 2013.

[69] Hervé Jégou, Florent Perronnin, Matthijs Douze, Cordelia Schmid, et al. Aggregating local image descriptors into compact codes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(9):1704–1716, 2012.

[70] Yan-Tao Zheng, Ming Zhao, Yang Song, Hartwig Adam, Ulrich Buddemeier, Alessandro Bissacco, Fernando Brucher, Tat-Seng Chua, and Hartmut Neven. Tour the world: building a web-scale landmark recognition engine. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1085–1092. IEEE, 2009.

[71] Herwig Lejsek, Björn Jónsson, and Laurent Amsaleg. Nv-tree: nearest neighbors at the billion scale. In *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, page 54. ACM, 2011.

[72] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. Searching in one billion vectors: re-rank with source coding. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 861–864. IEEE, 2011.

[73] Matthijs Douze, Hervé Jégou, Harsimrat Sandhawalia, Laurent Amsaleg, and Cordelia Schmid. Evaluation of gist descriptors for web-scale image search. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, page 19. ACM, 2009.

[74] Michal Batko, Fabrizio Falchi, Claudio Lucchese, David Novak, Raffaele Perego, Fausto Rabitti, Jan Sedmidubsky, and Pavel Zezula. Building a web-scale image similarity search system. *Multimedia Tools and Applications*, 47(3):599–629, 2010.

[75] Xirong Li, Le Chen, Lei Zhang, Fuzong Lin, and Wei-Ying Ma. Image annotation by large-scale content-based image retrieval. In *Proceedings of the 14th annual ACM international conference on Multimedia*, pages 607–610. ACM, 2006.

[76] Gylfi Gudmundsson, Björn Jónsson, and Laurent Amsaleg. A large-scale performance study of cluster-based high-dimensional indexing. In *Proceedings of the international workshop on Very-large-scale multimedia corpus, mining and retrieval*, pages 31–36. ACM, 2010.

[77] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[78] Ting Liu, Charles Rosenberg, and Henry A Rowley. Clustering billions of images with large scale nearest neighbor search. In *Applications of Computer Vision, 2007. WACV'07. IEEE Workshop on*, pages 28–28. IEEE, 2007.

[79] Stephen M Omohundro. *Five balltree construction algorithms*. International Computer Science Institute Berkeley, 1989.

[80] Antonin Guttman. *R-trees: a dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[81] Lakshmi Balasubramanian and M Sugumaran. A state-of-art in r-tree variants for spatial indexing. *International Journal of Computer Applications*, 42(20):35–41, 2012.

[82] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, June 1998.

[83] Leonard Brown and Le Gruenwald. Tree-based indexes for image data. *Journal of Visual Communication and Image Representation*, 9(4):300–313, 1998.

[84] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

[85] Monika Jain and SK Singh. A survey on: content based image retrieval systems using clustering techniques for large data sets. *International Journal of Managing Information Technology*, 3(4):23, 2011.

[86] Flavio Chierichetti, Alessandro Panconesi, Prabhakar Raghavan, Mauro Sozio, Alessandro Tiberi, and Eli Upfal. Finding near neighbors through cluster pruning. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 103–112. ACM, 2007.

[87] Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1470–1477. IEEE, 2003.

[88] David M Chen, Sam S Tsai, Vijay Chandrasekhar, Gabriel Takacs, Ramakrishna Vedantham, Radek Grzeszczuk, and Bernd Girod. Inverted index compression for scalable image matching. In *DCC*, page 525, 2010.

[89] Michael Persin, Justin Zobel, and Ron Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *JASIS*, 47(10):749–764, 1996.

[90] Gonzalo Navarro and Simon J Puglisi. Dual-sorted inverted lists. In *International Symposium on String Processing and Information Retrieval*, pages 309–321. Springer, 2010.

[91] Hao Yan, Shuai Ding, and Torsten Suel. Inverted index compression and query processing with optimized document ordering. In *Proceedings of the 18th international conference on World wide web*, pages 401–410. ACM, 2009.

[92] Roberto Konow, Gonzalo Navarro, Charles LA Clarke, and Alejandro López-Ortíz. Faster and smaller inverted indices with treaps. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 193–202. ACM, 2013.

[93] Vo Ngoc Anh and Alistair Moffat. Pruned query evaluation using pre-computed impacts. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 372–379. ACM, 2006.

[94] Ellen M. Voorhees and Donna K. Harman. *TREC: Experiment and Evaluation in Information Retrieval (Digital Libraries and Electronic Publishing)*. The MIT Press, 2005.

[95] Diep Thi Ngoc Nguyen and Yasushi Kiyoki. An adaptive search path traverse for large-scale video frame retrieval. In *Information Modelling and Knowledge Bases XXVI*, Frontiers in Artificial Intelligence and Applications, pages 324–342, 2014.

[96] Michel Marie Deza and Elena Deza. Encyclopedia of distances. In *Encyclopedia of Distances*, pages 1–583. Springer, 2009.

[97] Kazuaki Maeda. Comparative survey of object serialization techniques and the programming supports. *Journal of Communication and Computer*, 9(8):920–928, 2012.

[98] Yagmur Unal. The effect of colour on human body and psychology. *International Journal of Life Sciences Research*, Vol. 3, Issue 4:pp. 126–128, October-December 2015.

[99] Takashi Kitagawa, Takafumi Nakanishi, and Yasushi Kiyoki. An implementation method of automatic metadata extraction method for music data and its application to semantic associative search. *Systems and Computers in Japan*, 35(6):59–78, 2004.

[100] Johannes Itten. *The Elements of Color: A Treatise on the Color System Based on His Book The Art of Color*. Chapman & Hall, 1970.

[101] Shigenobu Kobayashi. *Color image scale*. Kodansha Amer Incorporated, 1991.

[102] Masataka Tokumaru, Noriaki Muranaka, and Shigeru Imanishi. Color design support system considering color harmony. In *Fuzzy Systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE International Conference on*, volume 1, pages 378–383. IEEE, 2002.

[103] Lauren I Labrecque and George R Milne. Exciting red and competent blue: the importance of color in marketing. *Journal of the Academy of Marketing Science*, 40(5):711–727, 2012.

[104] Elliot J. Crowley and Andrew Zisserman. In search of art. In *Workshop on Computer Vision for Art Analysis, ECCV*, 2014.

[105] Alvy Ray Smith. Color gamut transform pairs. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '78, pages 12–19, New York, NY, USA, 1978. ACM.

[106] Michael J Swain and Dana H Ballard. Color indexing. *International journal of computer vision*, 7(1):11–32, 1991.

[107] Ju Han and Kai-Kuang Ma. Fuzzy color histogram and its use in color image retrieval. *IEEE Transactions on image Processing*, 11(8):944–952, 2002.

[108] Fatih Porikli. Integral histogram: A fast way to extract histograms in cartesian spaces. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 829–836. IEEE, 2005.

[109] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, 23(1):5–48, 1991.

[110] Sam Harbaugh and John A Forakis. Timing studies using a synthetic whetstone benchmark. *ACM SIGAda Ada Letters*, 4(2):23–34, 1984.

[111] Bassem El Zant and Maurice Gagnaire. Performance evaluation of cloud service providers. In *Information and Communication Technology Research (ICTRC), 2015 International Conference on*, pages 302–305. IEEE, 2015.

[112] Franz Graf. Jfeaturelib – a free java library containing feature descriptors and detectors, https://jfeaturelib.googlecode.com, 2012.

[113] Anna Bosch, Andrew Zisserman, and Xavier Munoz. Representing shape with a spatial pyramid kernel. In *Proceedings of the 6th ACM international conference on Image and video retrieval*, pages 401–408. ACM, 2007.

[114] Anil K Jain and Farshid Farrokhnia. Unsupervised texture segmentation using gabor filters. *Pattern recognition*, 24(12):1167–1186, 1991.

[115] Bangalore S Manjunath and Wei-Ying Ma. Texture features for browsing and retrieval of image data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(8):837–842, 1996.

[116] Mathias Lux and Savvas A Chatzichristofis. Lire: lucene image retrieval: an extensible java cbir library. In *Proceedings of the 16th ACM international conference on Multimedia*, pages 1085–1088. ACM, 2008.

[117] J-K Kamarainen, Ville Kyrki, and Heikki Kalviainen. Invariance properties of gabor filter-based features-overview and applications. *Image Processing, IEEE Transactions on*, 15(5):1088–1099, 2006.

[118] Savvas A Chatzichristofis and Yiannis S Boutalis. Fcth: Fuzzy color and texture histogram-a low level feature for accurate image retrieval. In *Image Analysis for Multimedia Interactive Services, 2008. WIAMIS'08. Ninth International Workshop on*, pages 191–196. IEEE, 2008.

[119] Savvas A Chatzichristofis and Yiannis S Boutalis. Cedd: color and edge directivity descriptor: a compact descriptor for image indexing and retrieval. In *Computer Vision Systems*, pages 312–322. Springer, 2008.

[120] Savvas Chatzichristofis, Mathias Lux, and Yiannis Boutalis. Selection of the proper compact composite descriptor for improving content based image retrieval. In *Proc. of the 6th IASTED International Conference*, volume 134643, page 064, 2009.

[121] Behzad Shahraray. Scene change detection and content-based sampling of video sequences. In *Proc. SPIE 2419, Digital Video Compression: Algorithms and Technologies 1995*, volume 2419, pages 2–13, 1995.

[122] John R Kender and Boon-Lock Yeo. Video scene segmentation via continuous video coherence. In *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*, pages 367–373. IEEE, 1998.

[123] Chong-Wah Ngo, Yu-Fei Ma, and Hong-Jiang Zhang. Video summarization and scene detection by graph modeling. *Circuits and Systems for Video Technology, IEEE Transactions on*, 15(2):296–305, 2005.

[124] Anastasios Dimou, Olivia Nemethova, and Markus Rupp. Scene change detection for h.264 using dynamic threshold techniques. In *Proceedings of 5th EURASIP Conference on Speech and Image Processing, Multimedia Communications and Service*, 2005.

[125] Xinying Wang and Zhengke Weng. Scene abrupt change detection. In *Electrical and Computer Engineering, 2000 Canadian Conference on*, volume 2, pages 880–883. IEEE, 2000.

[126] Kin-Wai Sze, Kin-Man Lam, and Guoping Qiu. Scene cut detection using the colored pattern appearance model. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, volume 2, pages II–1017. IEEE, 2003.

[127] Prasanna K Sahoo, SAKC Soltani, and Andrew KC Wong. A survey of thresholding techniques. *Computer vision, graphics, and image processing*, 41(2):233–260, 1988.

[128] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[129] Thomas Deselaers, Daniel Keysers, and Hermann Ney. Features for image retrieval: A quantitative comparison. In *Joint Pattern Recognition Symposium*, pages 228–236. Springer, 2004.

[130] Jan Puzicha, Joachim M Buhmann, Yossi Rubner, and Carlo Tomasi. Empirical evaluation of dissimilarity measures for color and texture. In *Computer*

Vision, 1999. The Proceedings of the Seventh IEEE International Conference on, volume 2, pages 1165–1172. IEEE, 1999.

[131] Nick Craswell. *Encyclopedia of Database Systems, R-precision*, pages 2453–2453. Springer US, Boston, MA, 2009.

[132] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. Evaluation in information retrieval. *Introduction to information retrieval*, pages 151–175, 2008.

[133] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[134] 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*, pages 651–660. ACM, 2005.