Université de Cergy-Pontoise

École Doctorale Sciences et Ingénierie

Local-Descriptor Matching for Image Identification Systems

Thèse de Doctorat

Groupe IX — Section 61 du Conseil National des Universités
Sciences et Technologies de l'Information et de la Communication

Eduardo Alves do Valle Jr.

Soutenue le 12 juin 2008 à Cergy-Pontoise

devant le jury composé par

M Laurent AMSALEG, *examinateur*
M Arnaldo ARAUJO, *examinateur*
M Pierre BONTON, *président*
M Matthieu CORD, *co-encadrant*
M Carl FRELICOT, *rapporteur*
Mme Marie-Hélène MASSON, *rapporteur*
Mme Sylvie PHILIPP-FOLIGUET, *directrice de thèse*

Cite this work as:

> Eduardo Valle. Local-Descriptor Matching for Image Identification Systems. Ph.D. Thesis. École Doctorale Sciences et Ingénierie, Université de Cergy Pontoise. Cergy-Pontoise, France. June, 2008. 181 p.

BibTeX:

This revision — 081020081

Table of Contents

To the memory of my
dear friend Josiane:
gone too soon, sorely missed.

Acknowledgements

The completion of this work was only possible due to the support of many people and institutions, which I would like to thank.

First and foremost, I thank my supervisors, Prof. Matthieu Cord and Prof. Sylvie Philipp-Foliguet for their continual support, guidance and inspiration. Modern scientific education being, in many ways, a continuation of the apprenticeship tradition, the choice of one's masters is one of the most critical variables to success, and I could not be happier with mine.

I thank also the members of my Examination Board: their suggestions and corrections greatly contributed for the final text and our exchange of ideas significantly enrichened my understanding of my own work.

I am grateful for the ETIS labs for receiving me and hosting me for the duration of this work. I thank the TI support team of ETIS and ENSEA, and in particular, Michel Jordan, Michel Leclerc and Laurent Protois, for their kind and helpful assistance in all matters of hardware, software and networking. Also, I thank the LIP6 labs for providing convenient facilities to finish the thesis when I moved to Paris, in the final three months.

I thank the C2RMF — Centre de Recherche et Restauration des Musées de France — for the discussions which lead to the choice of the subject of this thesis, and also, for letting me make my first prospective experiments with their conservation/restoration image database.

I am grateful for the Arquivo Público Mineiro for giving me access to their digital collection of historic pictures. I also acknowledge Directmedia Publishing for very generously putting their Yorck Project in public domain, and the Wikimedia Foundation for making it available through the Wikimedia Commons website.

During most of the duration of this work, I received a scholarship from CAPES (the Brazilian government agency that supports research and post-graduate education) through a CAPES/COFECUB project. Without this funding, this thesis simply would not have happened. I thank the support and help of CAPES and in, particular, of Joana d'Arc Gonçalves, Jussara Prado, Maria de Almeida and Nancy Santos. I've also received occasional funding from the European Project MUSCLE. As always, I have received considerable funding from my parents and much indirect material support from friends and family.

For most part of my scientific education, Prof. Arnaldo Araújo has been providing me with guidance and opportunities, and this thesis was no exception. As the journey comes to an end, I would like to acknowledge my profound debt of gratitude.

I am grateful for the people, both in France and in Brazil, who helped me to get me back on my feet and complete the thesis. In the French side, I would like to stress my gratitude to Inez

Machado-Salim, of the Maison du Brésil, and to all the team of René Dubos, in Pontoise. In Brazil, I am enormously indebted to Dr. José Pereira.

My family has always been a strong source of support and inspiration. I would like to thank my mother, father and sisters for sharing with me the material and emotional costs and risks involved in such a long (10+ years since high school!) preparation to pursue a career in research.

I warmly thank the friends I have made in France, whose teachings I consider an integral part of my education. I cannot thank enough Guy Flechter, Frédéric Precioso and Lucile Sassatelli for their friendship, support, generosity and willingness to help, without which this work would have never seen the light of day. I would also like to thank by name Arnaud Blanchard, Marlena Bueno, Safin Gaznai, Lionel Guibert, André Monteil, Alain de Rengervé and Laurent Sereys.

Last, but not least, I thank my friends in Brazil for their continued love and companionship, even if mediated by 10 000 km of optical fibre.

Notations and Conventions

Generalities:

A	a set is denoted by an upper case Latin letter
a_i	a generic element from a set may be denoted by the corresponding lower case Latin letter and an index
$\#A$	the cardinality of set A
$\mathbb{R}$	the set of real numbers
$\mathbb{R}^+$	the set of non-negative real numbers
$[a;b]$	real interval from a to b, inclusive
$]a;b]$	real interval from a, exclusive, to b, inclusive
$\{a,\ldots,b\}$	integer interval from a to b, inclusive
$\lceil x \rceil$	ceiling: the smallest integer number greater than x
$\lfloor x \rfloor$	floor: the largest integer number lesser than x
$\log x$	logarithm of x on base 2
$\ln x$	logarithm of x on base e
$\mathcal{O}(\ )$	big-O asymptotic upper bound
$\neg$	negation: logical or bitwise “not”
$\wedge$	conjunction: logical or bitwise “and”
$\vee$	disjunction: logical or bitwise “or”
$\oplus$	exclusive disjunction: logical or bitwise “exclusive-or”
$x \circlearrowleft y$	logical rotate x to the left by y bits
$x \circlearrowright y$	logical rotate x to the right by y bits
$\Pr[\mathrm{A}] = a$	the probability of the event A happening is a
$\mathrm{E}(\ )$	the expected value function

Image identification:

I	the set of images in the database
O	the set of identified original images, $O \subset I$
T	the set of transformations from images to images
q	the query image

kNN search:

D	a domain space for data points
d	the dimensionality of D
B	a dataset of data points such as $B \subset D$
n	number of data points in the database, $\#B$
q	the query data point, $q \in D$
Δ	a dissimilarity function $\Delta : D \times D \to \mathbb{R}$
ε	the relative error in the approximate search
α	the probability of missing a correct answer in the approximate search
S	a solution set to the kNN search
s_i	the i^{th} true nearest neighbour
S'	a solution set to the approximate kNN search
s'_i	the i^{th} proposed (possibly approximate) nearest neighbour
k	number of neighbours to search, $\#S, \#S'$

Pseudo-code:

x ← y	attribution: makes the value of x equal to y
a[]	an array
a[i]	the i^{th} element of the array a
`{ comment }`	non-executing comment
`code`	this indicates operations specified in “low-level”
`pseudo-code`	this indicates operations specified in “high-level”, where the specific implementation is purposely omitted for the sake of clarity, brevity or generality

Number notation:

We adopt the English version of the SI notation for numbers, with dots separating the decimals from the unit, and a space separating the thousands, e.g., 2^{20} = 1 048 576; 1÷4 = 0.25.

Binary prefixes:

When measuring data, we adopt the recommendation of the International Electrotechnical Commission [IEC 2005] for binary prefixes, to avoid ambiguity with the decimal prefixes used by the SI (International System of Units).

Symbol	**Name**	**Quantity of data**
B	byte	8 bits
KiB	kibibyte	2^{10} = 1 024 bytes
MiB	mebibyte	2^{20} = 1 048 576 bytes
GiB	gibibyte	2^{30} = 1 073 741 824 bytes
TiB	tebibyte	2^{40} = 1 099 511 627 776 bytes

Chapter 1

Introduction

1.1 Motivation

In the latest years, a paradigm shift took place on documental preservation, an activity that has traditionally faced a severe compromise between the conservation of the artefacts, and the widespread access to them. Digital technology has offered the possibility to break this compromise, by supplying to the users high quality copies, while preserving the originals from unnecessary manipulation. At the very least, digital tools could now be used to ease the search operations, relieving the collections from the "hunting expeditions", which exposed the documents to excessive and avoidable wear.

Suddenly, the activities of conservation and access, which were previously treated as separated and even conflicting goals, now formed an indissoluble cooperation:

> "In the world of paper and film, preservation and access are separate but related activities. It is possible to fulfill a preservation need of a manuscript collection, for example, without solving the collection's access problems. Similarly, access to scholarly materials can be guaranteed for a very long period of time with[out] taking concrete preservation action. Modern preservation management strategies, however, posit that preservation action is taken on an item so that it may be used. (…) *Preservation in the digital world puts to rest any lingering notion that preservation and access are separable activities.*" [Conway 1996, emphasis added]

Therefore, the development of an efficient strategy of information retrieval is *sine qua non* for the digitisation of a collection to be considered a preservation initiative.

It is interesting to note that this realignment of preservation and access meets the idea that documental memory has a social role beyond the simple accumulation of artefacts. Already in 1939, the archivist Robert Binkley defended that "the objective of archival policy in a democratic country cannot be the mere saving of paper, it must be nothing less than the enriching of the complete historical consciousness of the people as a whole" [Binkley 1939].

However, the application of digital technology to cultural collections brings challenges in the same proportion as it gives promises. In a previous work [Valle 2003], we have explored the

serious issues of digital preservation: the fragility of media, the Babel of standards, the short cycles of obsolescence, the dependency on several layers of components to make the raw sequence of bits intelligible; all making digital data extremely difficult to preserve in the long term (50–100 years and more). This sobering reality has made digital technology more a supporting actor than a protagonist in documental conservation, but, since a crushing majority of documents are now created digitally, there is a pressing need to solve those problems.

If on matters of conservation, digital technology is still overshadowed by more traditional (and safer) approaches like microfilming, on matters of access it reigns absolutely. Archivists have watched with astonishment that painstakingly crafted search instruments often loose favour to brute-force Google-like search engines. And digital technology goes beyond identifying the correct documents: it can immediately retrieve digital surrogates, even to remote sites, without the need to fetch the original from the deposit — which is sufficient for 99% of the users.

This best-case scenario only happens when the preoccupation with access permeates all the application of digital technology to the institution. In conventional archives, the physical structure of the documents (boxes, folders, envelopes, bindings) warrants a baseline organisation. But digital documents have no material form, and easily degenerate in an amorphous mass of badly classified, badly indexed data, where it is impossible to find any useful information whatsoever.

Iconographic collections have their own challenges, both in terms of preservation and access. In what concerns the latter, most of the problems come from the fact that the choice of the indexing terms (whether for the composition of a traditional, hand crafted, search instrument, whether for the exploitation in a search engine) are all but obvious.

A search engine capable of retrieving images accordingly to high-level semantic concepts (like “images of flowers”, “images portraying the River Seine”) is one of the greatest lures of digital technology. However, though several advances have been obtained recently [Cord 2008, pp. 115–287; Smeulders 2000], a general solution to the problem has been elusive, and remains an ambition of several fields of science, including Machine Learning, Image Processing, and Computer Vision.

But content-based techniques are not limited to high-level semantics. Several other applications are sought by cultural institutions, ranging from the most transcendental (stylistic classification, identification of forgeries) to the most earthbound (classifying craquelure patterns, detecting wood panel reinforcements).

Our choice, image identification, is probably the path of the middle, as on one hand it does not have the complexities and fuzziness of semantic (or stylistic, for that matter) search, but on the other hand, it is not completely deprived of conceptual issues. The questions “what is a copy”, “what is a derived image” are less obvious than they appear at first glance — including juridically, where deciding about what is acceptable as “fair use”, “substantial appropriation” and “derivative work” is a matter of bitter argument.

1.2 Objectives

The immediate objective of this work is the study of image identification systems, with a certain bias towards cultural institutions needs, even though those systems can be used in other organisations, like press agencies, stock image vendors, etc.

We started by focusing on the architectural choices, in order to understand how the time was spent in the different operations, how the hardware technology affected the systems, and how we could ameliorate their performance. The architecture that most attracted our attention provided the most precise results, but presented performance penalties, and our major task became to conquer this disadvantage.

That, in turn, influenced our emphasis on one of the subcomponents of the architecture. To explain the challenge in a nutshell, all image identification systems use the concept of descriptor in order to establish the similarity between the images. The architecture we have chosen is based on the simultaneous use of a large amount of high-dimensional descriptors per image, and to query a single image, each one of those descriptors has to be matched. The match itself is made using the k nearest neighbours search (a technique which also finds important applications outside the universe of image identification). Our main objective became improving the performance of k nearest neighbours search in order to use the chosen architecture without harsh efficiency penalties.

We also kept in mind the need to empirically evaluate our techniques in the context of cultural institutions, and for this purpose, we obtained an extract of the digital reproductions of the photographic collection of Arquivo Público Mineiro (the Archives of the State of Minas Gerais, Brazil), on which we performed most of our experiments.

1.3 Contributions of the thesis

The subject of this thesis is the improvement of the efficiency of local descriptor-based image systems, while retaining their efficacy, in the context of the huge image databases characteristic of cultural institutions.

The main contribution of the thesis is the creation of three new approaches for the efficient matching of high-dimensional local descriptors, through k nearest neighbours search. One of those techniques is adequate to moderate to large databases, and is very fast and precise. The others, while maintaining good performance, allow for huge databases.

Another contribution is the empirical evaluation of those techniques against state-of-the-art methods for local descriptor matching, and the development of datasets, ground truths and tools to evaluate nearest neighbours search techniques.

Finally, there is the empirical comparison of our system architecture against a real-world architecture, and the participation in the competition ImagEVAL.

We dare to believe that our review on the state of the art of kNN search is a non-negligible contribution, as it focuses on the approaches of practical interest for large high-dimensional databases.

1.4 Organisation of the text

This thesis is organised in six chapters, including this introduction.

In the second chapter, **we formalise the problem of image identification and explore the architecture of image identification systems based on local descriptors**, analysing the performance challenges of those systems. We also provide a brief overview of the literature on image identification systems.

In the third chapter, we review **the state of the art on the high-dimensional *k*-nearest neighbours search**, used to match the local descriptors. We discuss as well a few alternative techniques to perform the matching.

In the fourth chapter, we propose **three approaches for nearest neighbours search**. One, the **3-way tree,** adds redundancy to a classical method, the *KD-tree*, allowing a very fast and precise matching for moderate to large databases. The two others, **projection KD-forests** and **multicurves**, project the data onto multiple moderate-dimensional subspaces to allow quick and accurate matching for huge databases.

In the fifth chapter, **we present our experiments, with the comparison between our local descriptors matching methods and a few state of the art methods**. We describe in detail the design of the experiments and the results.

Finally, in the sixth chapter, **we present our conclusions**.

Chapter 2

Architecture of Image Identification Systems

In this chapter, we define the problem of image identification and discover its challenges, its applications and the proposed solutions to solve it. We examine some alternatives to image identification systems, and why, in our opinion, they fail to address the problem adequately.

After discussing content-based image retrieval in general, we present a brief overview of existing image identification systems, concentrating on those that, like ours, make use of local descriptors.

Finally, we dissect the structure of systems based in points of interest and local descriptors. We want to understand how the hardware technology affects their performance and what we can do to accelerate them.

Response time is an important usability consideration for a search system. A fast system provides a more comfortable interaction, inviting the exploration of alternative query formulations, which ultimately increases the quality of the results and the interest of the user.

However, the emphasis on quick response times depends on the intended application. In some contexts, while still a desirable property, it is of secondary importance. A user searching for an image in a large archive, for example, will be willing to wait for several minutes in order to get a correct answer, since the back-up strategy — manual search — may take entire days. Contrast this to copyright enforcement, where each search operations must take at most a few seconds, since a large number of queries has to be posed to determine the intersection between two databases.

2.1 What is image identification

The task of *image identification* or *copy detection* consists in taking a given query image and finding out the original from where it possibly derives, together with any relevant metadata, such as titles, authors, copyright information, etc.

More formally [Joly 2005, § 1.2], if we have a set of images I, a set of transformations T, and a query image q, we want to find the set of possible original images from which q derives, $O = \{o_i\} \subseteq I$, such that:

$$\exists t \in T : o_i = t(q) \qquad \textbf{Eq. 2.1}$$

Though in many cases it is expected that the set O will have exactly one element, it may also be empty (in the case that no image from the set can I originate q) or it can have many elements (in the case that several images are able to originate q through different transformations).

The set of transformations may include:

- **Geometric transforms**: mainly affine transformations, like translations, rotations and scale changes. Sometimes, a small amount of non-affine distortion may be present, mainly trapezoidal and spherical distortions from the photographic reproduction;
- **Photometric** and **colorimetric transforms**: changes in brightness, contrast, saturation or colour balance;
- **Selection** and **occlusions**: the image may have been cropped to select just a detail; it may also present labels, stamps, annotations, censor bars, etc.;
- **Noise** of several kinds: compression artefacts (from lossy compression like), electronic noise from cameras and scanners, "salt and pepper" noise from dust, etc.;
- **Analogical/digital conversion**: the initial acquisition of the digital image, as well as reprinting and rescanning operations may be a source of important distortion, including intrinsic quantisation effects of digitisation, halftoning methods used for printing and moiré patterns;
- Any **combination** of those.

The large variety of transformation types and intensities makes the task very challenging (Figure 2.1).

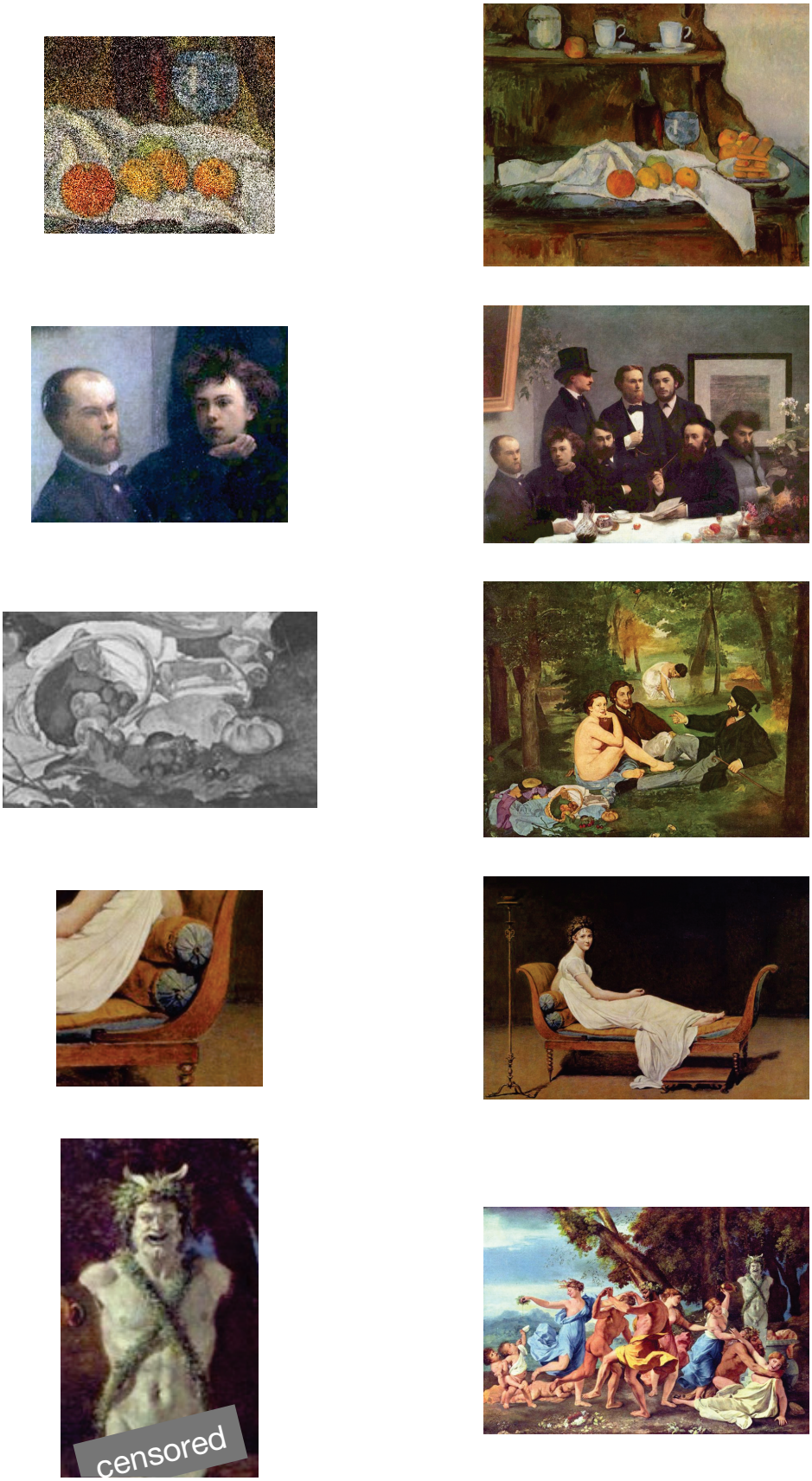


Figure 2.1: Image identification—we should be able to match the query images (left) to their originals in the database (right), even after intense transformations are applied on the query images: cropping (all images), noise (1st), colour balance change (2nd), conversion to greyscale (3rd), geometric reflection (4th) and occlusion (5th)

2.2 Applications

The meaning of a document depends on its context, and the lack of metadata reduces greatly its usefulness. This leads to the need of reliable ways of retrieving all information concerning a document, when only the visual evidence is available.

Image identification has several important applications, which we will describe in this section.

2.2.1 Source identification in image repositories

Institutions possessing large sets of images, like museums, archives and press agencies, are often asked to perform the identification of images from newspaper clippings, books, thesis and even postcards, where the references are missing, too summary, outdated or incorrect. Sometimes the only source reference available is a copyright notice.

In those cases, the visual document itself is the only reliable evidence for the identification of the original and the retrieval of its metadata. But because the collection consists of hundreds of thousands of items, manual search is infeasible. An operator with good knowledge on the collection may use conventional search tools to narrow the search and retrieve the desired image, but even so, hundreds of possibilities may remain and the whole process may take hours, and even days. The simple question "which image is this" becomes very difficult to answer, frustrating the users.

Armitage and Enser identify the need of checking the identification, attribution and provenance of items as one of the major types of inquires users present to libraries and archives, in what concerns image collections [Armitage 1997][P.G.B. Enser, personal communication, January 27, 2006].

2.2.2 Recovery of accidental image / metadata separation

A variant of the situation described above is the separation between the image and its metadata, within the boundaries of an institution, at one of the steps of a complex workflow. In this case, quality control will identify images with lacking or incorrect metadata, and proceed to rectify the situation.

A concrete example is the image database of a conservation/restoration laboratory. The study of a single artwork will generate many images, including general views, close-up of details, pictures taken under different illumination conditions, etc. Those images are only useful in the context of a dossier of analysis. If one of them gets lost and is found isolated, it is important to be able to retrieve its dossier.

2.2.3 Traceability of image reproduction and copyright enforcement

One of the most prominent applications of image identification is the tracking of the reproduction of visual documents, either to reveal the historic of exploitation of a given document, or to enforce the copyright.

Traceability allows to uncover the exploitation of an image, which may be interesting both for understanding the usage of the collection and to establish links between different collections.

Unauthorised reproduction is a serious issue for authors and holders, especially with the availability of digital tools and the Internet, which make it very easy to create and disseminate high-quality copies of protected material.

Photography agencies, for example, make their profit by licensing the use of images. If they believe someone is using their material without paying the due fees, they can hire an expert to determine if the suspect images come indeed from their database. This process can be greatly streamlined if the software can automatically detect which are the most probably misused images and associate them to their most probable originals.

2.3 Alternatives to image identification systems

2.3.1 Textual annotation

Traditional image retrieval systems are based on textual annotations, either added manually by a human operator or automatically harvested from the context of the image (e.g., captions and text around the image in the page).

A user can use his knowledge about the image in order to search those textual annotations. For example, facing the query image in Figure 2.2, the user can use the keywords “Paris”, “Eiffel Tower”, “World’s Fair”, “July 1888” or a combination of those, until the desired original is found.

Figure 2.2: Example of query image

There are, however, serious limitations in using textual search to identify images:

- Manual annotation is very labour intensive, and text harvesting depends on the availability of contextual text;
- The annotations may not be available, may be incorrect, or may be too summary;
- The available annotations may not match the users' knowledge about the picture (e.g., the database indexes the subjects of the pictures, the users only know the name of the photographer);
- The users may not be knowledgeable enough about the image in order to provide textual descriptors. For example, they may want to find the original of a portrait precisely in order to know whose portrait is that;
- Synonyms, homonyms, polysemes, specialisation and generalisation also make textual search more difficult. The users may search for "bat" and not find the desired image, because it was indexed as "animal", "*Chiroptera*", or "*Acerodon jubatus*". Instead they may fruitless browse through hundreds of images of clubs and bludgeons, which have nothing to do with their intents.

2.3.2 Watermarking

Another alternative, especially in the context of copyright enforcement, is the use of watermarks. This technique allows the embedding of a certain amount of information in the visual mesh of the image, normally almost imperceptibly to the human eye. This may be used to include a unique identification in each image.

Though theoretically very interesting, this technique faces many practical challenges:

- The watermarks are not robust to strong image transformations;
- An operator aware of the presence of the watermark and its mechanism can take measures to remove it or change it;
- The watermarks always distort the image a little, and for a given method, there is a trade-off between the robustness and the degree of distortion;
- The watermarks only work for images reproduced *after* their adoption, all the copies circulating before they were adopted will remain unidentifiable;
- Most analogical sources cannot use the technique, for example, a picture in a museum which can be photographed outside the context of the digital image distribution.

2.3.3 General-purpose content-based image retrieval systems

Content-based image retrieval systems (CBIR) [Smeulders 2000], even if not specifically conceived for image identification, may be a powerful ally, since they rely only on the visual aspect of the documents, needing not keywords, annotations or watermarks, which can be unavailable, lost or removed. The image on hand may be used as a query in a search for others with similar distribution of colours, textures, shapes, etc.

The major disadvantage of those systems, in comparison to those specific for image identification, consists in their criterion of similarity, which is often inadequate to retrieve the correct images. For example, a system based on colour distributions will not be able to match an original to a greyscale reproduction. Generally speaking, semantic-oriented CBIR systems are not as well suited for image identification as dedicated systems, because their matching criteria do not take in account the kinds of distortions which are most commonly expected in this specific task.

2.4 Image identification systems

2.4.1 General discussion

Image identification systems are a specialisation of CBIR systems, and share with them many characteristics. Both use the concept of *descriptor* in order to establish the *similarity* between the images, instead of trying to compare directly the raw visual contents.

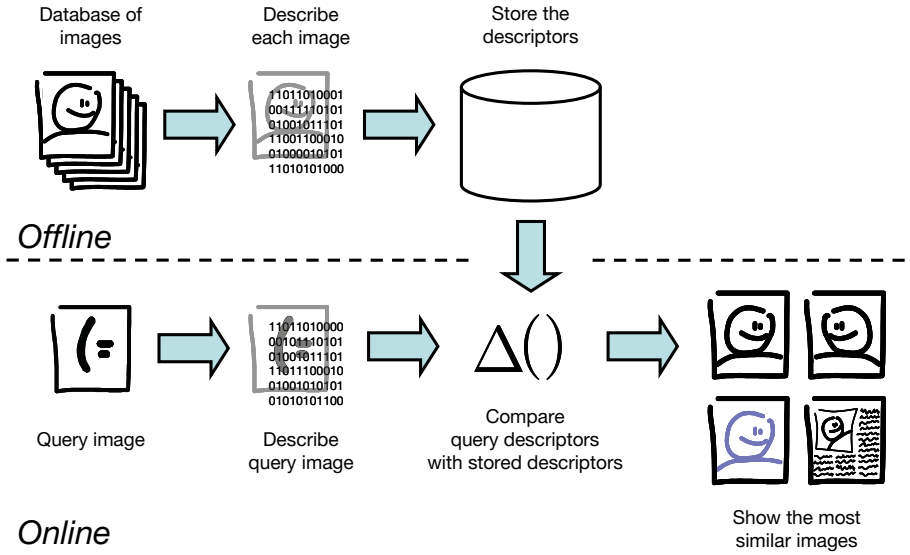


Figure 2.3: A CBIR system is based on a measure of dissimilarity between image descriptors. The images corresponding to the lowest values of dissimilarity are presented to the user

The two main difficulties faced by every CBIR system are:

- How to synthesise the information contained in the image in a descriptor (or set of descriptors) which can be used to faithfully represent the desired visual, aesthetical or semantical characteristics which the system will be later looking for;
- How to create a dissimilarity criterion between descriptors (or set of descriptors), which correctly emulates the user mental model of (visual, aesthetical or semantical) similarity.

The same desiderata apply for the descriptor and the dissimilarity: as little cost as possible; and a delicate balance between analytic and synthetic powers, in order to be able to set apart the images that we (using the metal model) consider extraneous, but put together those that we consider fitting.

Incidentally, it is interesting to remark that the burthen of obtaining this balance may be shared by the descriptor and the dissimilarity in radically different proportions, accordingly to the design choices of the system. To mention two extremes, some systems choose to have very simple descriptors, and use complex dissimilarity functions capable to recognise the patterns [Chang 2003; Cord 2006]; others put most of the "intelligence" on the descriptors themselves, and use very simple dissimilarity criterions [Mindru 2004].

The ultimate challenge is answering very high-level user queries (general semantic categories such as "images from Europe", fuzzy aesthetic criterions such as "paintings I like"). Compared to those, the task of image identification is much simpler, for two reasons:

- It is possible, at least in theory, to conceive a relatively simple model of the categories we consider similar (by using the formulation of § 2.1);
- It is possible to establish with relative ease an objective criterion of success or failure.

In spite of that (or perhaps because of that), the expectation of performance is much higher for image identification systems than for high-level CBIR systems, and there reposes the greatest challenge of those systems.

Performance is a value which can be measured in two orthogonal dimensions: efficacy and efficiency. *Efficacy* is the ability to accomplish a task: a system which satisfies this requirement is said to be *effective*. *Efficiency* is the ability to use as little resources as possible.

In search systems, efficacy is traditionally measured in terms of *precision* (what fraction of the system answers are relevant), *recall* (what fraction of the relevant answers in the database the system retrieved) and some derivative metrics, like the *MAP* (mean average precision). The most important efficiency concern, from the users' point of view, is the search time, but the index construction time and the space occupied are also important for very large databases. We discuss the choice of metrics more in-depth in the discussion of the experimental protocols (§§ 5.1.1.3, 5.1.1.4, 5.2.1 and 5.3.1).

The need to boost the efficiency makes unappealing the pairwise comparison between the query image and all images in the database (by the bias of their descriptors). The problem with this approach is that the search time grows linearly with the size of the database, while we would prefer a sublinear (logarithmic, if possible) growth.

This is possible with the use of an index, a particular organization of the descriptors which allows discarding a large fraction of the database from the analysis. In order to decide both the organisation of the data, and which data to consider during the search, the index must take into account the nature of the dissimilarity. In general, an exact index (i.e., one which guarantees the

same results as a sequential, pairwise comparison) is only possible for very simple dissimilarities, and if using complex criterions, we are forced to accept approximate results.

We hinted, but not developed, on the subject of the images being described either by one descriptor or a set of descriptors. In the former case, when a single descriptor must capture the entire information of the image, we say it is a *global descriptor*. In the latter case, the descriptors are associated to different features of the image (regions, edges or small patches around points of interest) and are called *local descriptors*.

Creating a dissimilarity between two sets of descriptors is not an easy task, and making it efficiently is even less so. Most systems based on local descriptors greatly simplify the matter by adopting as criterion the plain vote count: each individual query descriptor is matched with its most similar individual descriptor in the database (using a simple distance, like the Euclidean distance). Then we count how many matches each image on the database received and use this as a criterion of similarity.

2.4.2 System reviews

We have not the presumption that the following is an exhaustive panorama of existing image identification systems. Those few systems were selected either because we took inspiration in them (like ARTISTE [Lewis 2004], which was our first contact with image identification as a concrete concern of cultural institutions) or because their architectural choices are similar to ours.

The application of CBIR systems on cultural institutions and specifically artwork databases has been envisaged as one of the most exciting promises of digital technology. Still very incipient, it already attracted the attention of the influential JISC committee: “despite its current limitations, CBIR is a fast-developing technology with considerable potential, and one that should be exploited where appropriate” [Eakins 1999]. Their study identified different levels of CBIR, depending on the degree of abstraction of the concepts being queried: level one covered visual features (“images with a read ball over a big blue region”); level two concerned object or object-category search (“images of Eiffel Tower”, “images of buildings”); level three comprised the search for abstract, non-material, categories (“images of people eating”, “representations of suffering”). Taking into account the state of the art at the time, only level-one queries were deemed attainable.

An interesting study by Ward et al. showed that to be valuable a system does not necessarily have to provide remarkable precision [Ward 2001]. In their assessment of a CBIR system for the London Guildhall Library and Art Gallery, including 181 respondents, though only 40% of the users indicated that they had found what they wanted through the CBIR system, over 65% affirmed that the results were useful and nearly 80% said they wanted to use the system again. This indicates that the impact of factors like novelty and amusement should not be ignored.

ARTISTE was one of the earliest real-world systems to provide image identification facilities in the context of artwork image databases [Lewis 2004]. The system, developed to assist cross col-

lection content and metadata-based search, is the result of an international cooperation of European cultural institutions, and was funded by the European Union.

In terms of image identification, ARTISTE provides two different methods: one for the matching of small fragments of very high-resolution images, and another for the retrieval of very low-quality queries which have been transmitted by fax to the institution.

The retrieval on an image using a fragment as query was motivated by the process of documentation of artworks during conservation/restoration process, when multiple shootings of the piece are taken, using different equipments, at different moments of the process and by different institutions. Those pictures vary enormously in their coverage of the piece, some of them portraying the entire artwork, and others reproducing very small details.

To be able to match those details to the painting, ARTISTE proposed a hierarchical version of the colour coherence vector (CCV) descriptor [Pass 1997], which is itself an enhancement over the simple colour histograms. The CCV can be viewed as a double colour histogram, where the frequency of each colour is accounted in two separated entries: one for large uniform regions and one for small, insulated regions. The signature, originally global, was adapted to allow the retrieval of an extract of an image, in a scheme that applies CCV to hierarchical subregions of the image, and was named Multiscale-CCV [Lewis 2004, § III, pp. 303–306].

We found out empirically (§ 5.3.2) that this approach has some serious shortcomings: first, it is very sensitive to colorimetric deformations. Second, it is even more sensitive than the colour histogram for deformations which disturb the coherence of the colours (like halftoning). Finally, it assumes that the fragments will have approximately the same aspect ratio as the regions of interest analysed.

The authors evaluated the method in a database of about 1 000 images of paintings, extracting the queries from the originals and subjecting them to different levels of distortion (blurring, Gaussian noise, resizing). They typically found the correct image among the 20 first answers in about 95% of the cases, except for the noise distortion where the success rate was of 75%.

The query by low-quality images (QLBI) was motivated by the need to answer the question "do you have this piece in your collection?" when the query image is a much deteriorated representation of the original, like a photocopy or a fax [Fauzi 2002; Lewis 2004, § III, pp. 306–310].

In those cases, they suggested considering the query as a binary (pure black-and-white) image — even if some colour or greyscale remains — resample it to a 64×64 pixel thumbnail, and compare the resulting pixels directly with those of the images in the database. The percentage of similar pixels is used of a criterion of similarity. They called this "slow QBLI".

Since the linear of comparison of those $64^2 = 4\,096$ bits for every image in the database may be too time consuming, they propose using the Pyramid Wavelet Transform (PWT) to compute a 19-dimensional feature vector, to be compared using the Euclidean distance. They compute the PWT on binarised and resampled versions of the images (to the size of 256×256 pixels). To take into account the photometric distortions, 99 descriptors are computed per image, each corresponding to a different threshold of binarisation adjusted to result in the equivalent percentage of black pixels (from 1 to 99%). They called this version "fast QLBI".

The evaluation was made in a database of 1 058 images of both 2D and 3D objects. The queries were faxes of 20 of those images. In all those 20 queries the good answer was among the top 5 in the rank in all cases.

Though QLBI showed good results even for radical colorimetric/photometric distortions, it is not robust for most other distortions (especially occlusion and cropping). Another difficulty of the method is the "delimitation" of the query image when no obvious margins exist, which affects all posterior computations.

The earliest trace of the use of a vote-counting algorithm for recognition in computer vision appears in Hough's method for line identification [Hough 1962], later generalised for arbitrary shapes [Ballard 1981].

The essential elements of the architecture we adopt appeared in Schmid et al. [Schmid 1997]: the use of points of interest, local descriptors computed around those points, a dissimilarity criterion based on a vote-counting algorithm, and a step of consistency checking on the matches before the final vote-count and ranking of the results.

Developing on those ideas, Lowe proposed an efficient scheme to find points of interest using differences of Gaussians of the image, and then a very robust descriptor computed on a patch around those points, based on histograms of the gradient, in a scheme he called scale invariant feature transform (SIFT) [Lowe 1999]. He also used a vote-counting algorithm and a consistency checking. Derivatives of SIFT ensued, including on the PCA-SIFT (SIFT with reduced dimensionality using a Principal Component Analysis, [Ke 2004a]) and GLOH (SIFT computed over a log-polar location grid, [Mikolajczyk 2005]).

Ke et al. [Ke 2004b] proposed a system for image identification (which they call "near-duplicate detection and sub-image retrieval") based on PCA-SIFT, defending that local descriptors are an ideal solution for the problem, as they are invariant to several geometric and photometric transformations, and to significant occlusion and cropping. To match the descriptors, they proposed a modified version of locality-sensitive hashing (§ 3.7, [Indyk 1998]), adapted from Gionis et al. [Gionis 1999] to perform better on disks. After matching the points, they eliminate the geometrically inconsistent matches using a RANSAC (random sample consensus) algorithm with an affine transformation model [Fischler 1981].

The system was evaluated on a database of 6 261 images of fine art. From those, 150 were selected as queries and subjected to 40 distortions, including colorimetric, photometric and geometric transformations. They added the transformed queries to the originals, obtaining a final test database of 12 111 images. Using the query images to retrieve the 40 best matches, they obtained a recall of 99.85% and a precision of 100%.

They performed a second set of experiments, on a database of 15 582 photographs. They choosed 314 among them and applied 10 more intense transforms, including, this time, shear-

ing (for which PCA-SIFT is not invariant). Querying for the 10 best matches, they obtained a recall of 96.78% and a precision of 96.12%.

The Eff² Project (which stands for *eff*icient and *eff*ective content-based image search) is an ongoing cooperation between the IRISA laboratory and the Reykjavik University. They, too, defend the use of local descriptors, arguing that traditional global descriptors schemes fail to meet the robustness requirements of image identification.

In [Lejsek 2005] the authors proposed an approach based on 24-dimensional local descriptors, which were computed around points of interest (the details are given in [Amsaleg 2001]). To match the points, they proposed the PvS index (§ 3.9). The criterion of similarity for the images was a simple count of matches, without geometric consistency [Grétarsdóttir 2005].

The system was evaluated on a collection of 29 077 press photos. From those, 18 queries were randomly chosen and subjected to distortions, including cropping, rotation and lossy compression with JPEG, in different intensities. They always obtained the original image ranked first for the cropping and rotations, but not for the JPEG compression (they did not report, though, which was the average rank obtained for those queries).

In an earlier prototype, described in [Grétarsdóttir 2005], the authors suggested that user interaction with intermediary results could be useful to reduce the time to obtain a meaningful search. In this early version they still used a simple sequential search to match the descriptors.

A still earlier work developed on IRISA [Berrani 2004], used the same 24-dimensional descriptors, with an indexing scheme with probabilistic control over the quality of the results, described in [Berrani 2002].

The PvS index was recently improved and rechristened NV-tree [Lejsek 2008]. In this new incarnation, it was tested with a database of almost 180 million SIFT descriptors.

The Eff² Project is among earliest to propose solutions for image detection in very large image databases, being also a pioneer in the study of the matching of local descriptors for huge databases of 10^7–10^8 descriptors, and more.

A system for copy detection on a *video* database was proposed by Joly in [Joly 2005]. The system uses composed descriptors which sample 4 different positions along a diagonal in space-time. On each of those positions, a 5-dimensional local descriptor is computed, using a slightly modified version of the Gaussian local jet, which results in a 20-dimensional descriptor vector. The "root" positions of those vectors are found using the Harris points of interest, made stable by the use of a Hermite filter, on the *key frames* of the video sequences (an invariant method to locate the key frames is also proposed).

The descriptors are matched using a probabilistic approach, which estimates what are the data pages on the index most susceptible of having good candidates. This allows an intelligent pruning of the pages to examine, which accelerates the search (see § 3.12.2). For efficiency reasons,

they proposed to build the indexing structure entirely in main memory [Joly 2005, pp. 79–119].

After the points are matched, the space-time consistency is verified, using a model which takes into account only translations and scale changes (since, the author defends, other geometric deformations are very unusual in the context of video copy detection). The model fitting is done by an M-estimator (Tukey's Biweight).

The method of probabilistic search obtains great improvement in efficiency (a speed-up of 11 to 78 over the traditional criterion of pruning the pages, depending on the precision desired).

The whole system was evaluated in databases of video sequences from the French *Institute National de l'Audiovisuel*, containing typically 3 500 (and up to 10 000) hours of video. The queries were short video extracts from the database (up to 20 seconds), submitted to several distortions like vertical shift (a combination of translation and cropping), scale changes, gamma correction and Gaussian noise. The whole system shows a very good compromise of efficiency *versus* efficacy, attaining a success rate of up to 87.4% for 10-second video sequences and up to 97.6% for 20-second video sequences, for a querying time of less than 10 ms.

2.5 Performance issues

2.5.1 Indexes

If the data has no organisation whatsoever, the only way to perform the query is by sequential comparison. This leads to a linear growth of querying time versus the database size, which itself leads to prohibitively long times for large databases. Therefore, we need to structure the data, in order to explore only the portions which are likely to contain descriptors similar to a given query. This is obtained through an index.

Generally speaking, an index is a data structure which provides faster search operations. The idea is to trade-off the time spent building the index (and the space it occupies) for faster search operations. There are many index designs, with different compromises between speed-up (how much they improve the search performance), space overhead (how much space the index occupies), set-up time (how much time does it take to build the index) and dynamicity (how well the index supports updates).

Abstractly, an index may be seen as a black box which, given the query, determines which subsets of the database must be examined in order to perform the search. Those subsets are named *data pages* (Figure 2.4).

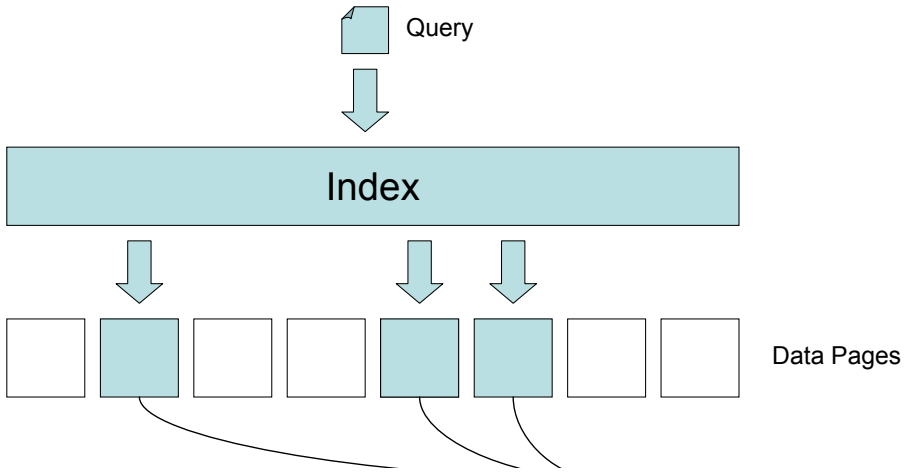


Figure 2.4: Given a query, an index will select the data pages which need to be explored in order to find a match

The most important measure of index performance is the *selectivity*, which indicates the average fraction of the data which has to be explored when answering a query. Since there is a certain overhead involving in querying the index, the selectivity must be high enough so that this overhead is compensated by the time saved by the fraction of the points which need not to be examined.

Conversely, we can measure the resulting cost by associating it to the number of records we have to explore. This notion appears often in our discussions, so we take the time to define it:

Definition 2.1 — Probe depth:

- *Probe depth* is the number of records (data points) we explore.

■

Traditional indexing structures do not store the data. Instead, they store pointers to the location of the actual records. This is done under the assumption that selectivity will be so high, that it will compensate the overhead of performing the indirect, out-of-sequence, fetches for the data.

Unfortunately, this assumption does not hold for high-dimensional descriptors, where, even with the best indexes, we still should expect to explore from a few hundred to a few thousand database descriptors *for each query descriptor*. Because of the high costs associated to random access (which we will discuss in detail in the next section), this solution in unfeasible for any high performance system.

Therefore, one of our basic assumptions is that the indexes store all data necessary for the posterior steps of the system and no further accesses will be necessary to fetch other information.

Indexes can vary on their level of complexity, and can sometimes be composed of several layers of components. In our discussions we find useful to define the following concept:

Definition 2.2 — Subindex:

- A *subindex* is a component of an index which is an index itself. The composed index can query its subindexes and use the partial results to compile its final answer.

■

2.5.2 Disk access

Except for relatively small databases, the main memory will not be large enough to hold all the descriptors, and we will be forced to use magnetic hard disks.

The hard disk is a non-volatile secondary storage device composed of a set of platters mounted on a common spindle. The platters are covered in a magnetic media, which allows the storage of the data, and spin very fast. To read and write the data, small electromagnetic heads (one for each platter surface) are mounted at the end of an arm which can move across the surface of the disks. The disks spin so fast that they create a cushion of air making the heads float above their surface. That way, there is no actual contact between the heads and the surface, greatly reducing friction and wear. Current workstation hard disks can store between 80 GB and 1 TB* of information.

What concern us here are the performance issues of accessing data on hard disks. The first thing that should be noticed is that disks are several orders of magnitude slower than the main memory. Second, though hard disks provide random access at much lower cost than tapes or optical disks, for example, sequential access is still typically one order of magnitude faster. The reason for the latter phenomenon is the need of relocating the i/o magnetic heads to the right track and waiting until the disk turns to the suitable block (Figure 2.5).

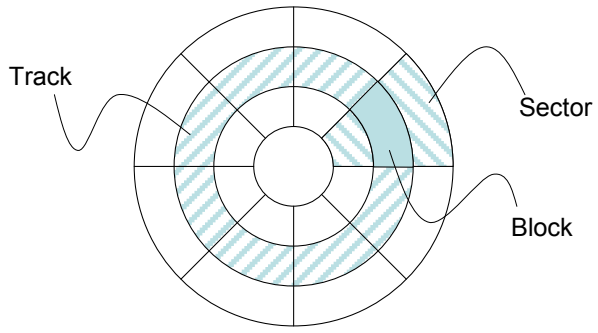


Figure 2.5: The organization of a platter in a hard disk. Additionally the corresponding tracks in all platters form a *cylinder*

Thus, the three times involved in an i/o disk operation are:

- **Seek time**: is the time for the i/o head to move for the right track, which depends on how far the head destination is from its origin.
- **Rotational delay or latency time**: is the time it takes for the right sector to come under the i/o head. Hard disks spinning at a constant angular velocity, this does not depend on the track.
- **Transfer time**: is the time it takes to transfer the data, which depends of several factors, including the rotational speed, the interface between the disk and the host, whether (and how) the sectors on the disk are interleaved, and naturally, the amount of data transferred. Often this parameter is expressed as its reciprocal, or the **disk transfer rate**, which is measured as the quantity of data transferred per unit of time. Because angular velocity is constant for hard disks, linear velocity depends on track, and this can affect the transfer time.

* For marketing reasons, disk drive vendors insist in measure data in powers of 10 (e.g., 1 KB = 10^3 bytes) which is often a source of ambiguity.

Different equipments can vary widely on those parameters. For the sake of comparison we explore three typical cases (Table 2.1).

	Seagate Cheetah 15K.5 300 GB	Western Digital WD7500AAKS 750 GB	Hitachi Microdrive 3K8
Main application	Enterprise servers	Desktop computers	Mobile devices
Capacity (in GiB)	279.40	698.49	7.45
Velocity (in RPM)	15 000	7 200	3 600
Form factor (in inches)	3.5	3.5	1.0
Seek time (for reading, average, in ms)	3.5	8.9	12.0
Latency time (average, in ms)	2.0	4.2	8.3
Transfer rate (sustained, average, in MiB/s)	92.2	75.3	7.2
Opportunity cost of a random access (in bytes read)	507 105	986 958	146 381
(in records read)	1 878	3 655	542

Table 2.1: Operational parameters for typical hard disks in three application domains

Assuming the average case, in the time needed to make a single random access, we could read hundreds, even thousands of descriptors. Even taking into account the cost of processing those descriptors, the relative cost of random access stays very high.

The consequence is clear: any algorithm, to perform well on disks, has to minimise the amount of random accesses.

It is perhaps worthy noting that even for primary memory, the hypothesis of random and sequential access costing about the same is false. This trend, which tends to aggravate in the next years, is caused by the disparity between the speed of the CPU and that of DRAM*, a phenomenon which is called *the memory wall* [McKee 2004].

In current computer architectures, the only way to conquer this disparity is by using fast cache memories, whose access times are compatible if the CPU. As the gap between memory and processing technologies widen, caching schemes have become progressively more sophisticated, including sometimes two or three layers of caches, each smaller and faster than the one below it, composing a *memory hierarchy*.

The problem is, caching mechanisms only work under the assumption of locality of reference (the principle that the probability of accessing a datum grows in proportion to its proximity to previously accessed data), and are defeated by random access. Therefore, if an algorithm access a large amount of data, in an essentially unpredictable order, its execution time will be bounded by DRAM, and not by CPU, which means, for a typical PC architecture, running several orders of magnitude more slowly.

* Dynamic RAM, the kind of memory which stores the bits on capacitors and requires constant refreshing

2.6 Synopsis

Most of our architectural choices were motivated by the main application we had in mind: image identification for very large image databases on cultural institutions. In this context, the primary emphasis is on the efficacy of the system, since every failure will lead to a lengthy and labour-intensive manual search. With that in mind, we have based our system on local descriptors computed around points of interest, since they have been shown to be so robust.

The principle behind the system is, for every image in the database, to identify a large number of PoI (Points of Interest), compute a local descriptor around those points, and store the descriptors in an index. This is done only once and forms what is named the *offline phase*.

When the index is ready, the user can present a query to the system. The same process of PoI extraction and local descriptor computation will be applied to the query. Then, using the index, each query descriptor will be matched to a small number of descriptors in the database (the most similar). Each matched descriptor gives a vote to its associate image. To make the process more reliable, we apply a geometric consistency step, and eliminate all the extraneous matches. We recount the votes (considering this time, only the consistent ones) and present the ranked results to the user. This whole process, which may be repeated as many times as the queries the user wants to perform, is named the *online phase.*

The separation between the *online* and *offline* phases is more conceptual than concrete, all depending on the structure of the descriptor index. If it is a static index, then the separation is perceived concretely by the user, because in order to add new images to the database, the index has to be completely rebuilt. However, on a dynamic index, images can be inserted or removed during or between queries, without further ado.

The whole process is illustrated on Figure 2.6.

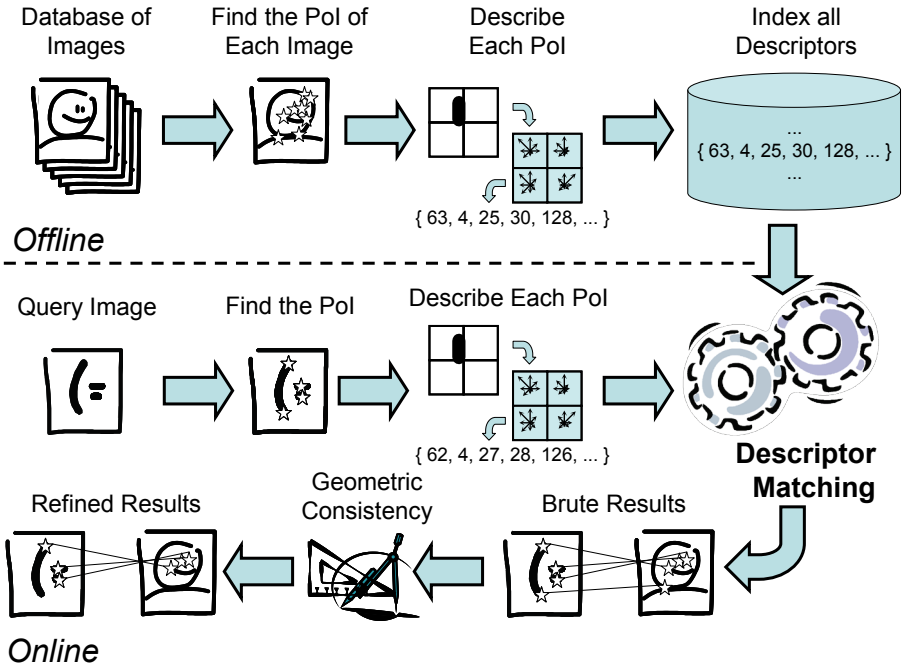


Figure 2.6: Image identification using PoI

It is easy to see why the method is so robust. Because the descriptors are many, if some get lost due to occlusions or cropping, enough will remain to guarantee good results. Even if some de-

scriptors are matched incorrectly, giving votes for the wrong images, only a correctly identified image will receive a significant amount of votes. Partial modifications of the image may disrupt some descriptors, but since they are computed around a small patch, enough can survive so to guarantee a match.

Unfortunately, the multiplicity of descriptors brings also a performance penalty, since hundreds, even thousands of matches must be found in order to identify a single image. The matter is made worse if the descriptors are high-dimensional, because of the infamous effect of the *curse of dimensionality* (which we discuss in § 3.2), each match operation becomes very expensive.

The essential quality of PoI is conveying more uniqueness than the other points of the image. In that way, the same points can be located, even if the image suffers distortions. This is the major criterion of quality for PoI, and is named *repeatability*. Once a point is detected, a descriptor must be generated for indexing purposes, usually analysing only a small patch around the point. If the method is to be robust, the descriptor must be invariant under the concerned deformations. The criterion of quality for descriptors is the *ability to match correctly*, i.e., using the appropriate dissimilarity function, the descriptors corresponding to the same points in distorted images should match, even in the presence of a great number of other points.

For our system, we have chosen SIFT, which is one of the most robust methods under similarity transformations (translations, scale changes and rotations).

The SIFT PoI are local extremes (minima and maxima) in a scale-space composed of differences of Gaussians of progressively larger standard deviations (the choice of the difference of Gaussians is motivated by the fact it approximates the Laplacian function, which performs very well in the composition of a scale-space).

The descriptors are based on a histogram of the gradients around the point. To achieve rotation invariance, the patch is rotated towards the most frequent direction of the gradient. To achieve invariance to affine and non-affine luminance changes, several normalisations and thresholds are applied.

The SIFT algorithm is complex, with the application of several thresholds, quite a few empirical parameter choices, and some ad hoc optimisations. A detailed description is given in [Lowe 2004].

SIFT has showed excellent performance in comparative studies (see for example [Mikolajczyk 2005]), but its power of discrimination may have undesirable collateral effects in some contexts. The most obvious drawback is when we are interested in recognising a category, situation in which it is advantageous to moderate this over-discrimination [Szumilas 2008]. In addition, the capacity to match images that share even small visual features, like a logotype, leads to false positives, which in the context of copyright enforcement or of large video databases (see [Joly 2005, pp. 161–162]) can quickly become a problem. But for metadata recovery in cultural institutions the relative cost of a handful of false positives is insignificant compared to that of a false negative.

SIFT has, of course, an additional disadvantage: its dimensionality, of 128, is perhaps the highest among the commonly used local descriptors.

The choice of SIFT, motivated, as we said, by its excellent performances in image matching, had an important influence on our later explorations. Because it is a very high-dimensional descriptor, this meant we would have to investigate a way either to reduce its dimensionality (like in PCA-SIFT, [Ke 2004a]) or to advance the state of the art of indexing mechanisms. We were somewhat sceptical on the former option, since it has been reasonably established by now that the performance of indexing schemes depends more on the "intrinsic" dimensionality of the data than on the dimensionality of the space in which they are embedded (see § 3.2.6). Moreover, the only way to take into account the non-linear and local correlations, which respond for the complex structure of this kind of data, is to perform lengthy and computationally expensive pre-processing, which we wished to avoid.

Therefore, our attention focused on the challenging problem of indexing high-dimensional descriptors. There are a handful of ways of performing descriptor matching in the literature, but similarity search (also known as *k*-nearest neighbours search or simply kNN search) is probably the most popular. Our decision to adopt it was somewhat biased by the fact it is an important problem on its own right, with many valuable applications.

It is well established in the database community that exact kNN search is unfeasible for dimensionalities much higher than ten: we are forced to accept approximate solutions [Weber 1998]. Moreover, as dimensionality grows the compromise between the precision of the results and the time spent on search (i.e., between efficacy and efficiency) becomes increasingly severe.

In final analysis, this compromise adds another layer of complexity in the evaluation of the descriptors: in order to have discriminating power, we may agree to increase the dimensionality; but this means accepting a more inaccurate matching, which will reduce the discrimination.

But by then *alea jacta erat*, and we were convinced that kNN search could be precise enough so that the greater dimensionality of SIFT would still pay.

2.7 Conclusion

In this chapter we have defined the problem of image identification and presented some of its possible solutions. After asserting the advantages of image identification systems, we have discussed their architecture, presented some existing systems, and discussed their performance issues.

Finally we have presented the architecture of our own system, our choices and their rationale, based on the following hypothesis:

- Local-descriptor based systems have a substantial efficacy advantage over global-descriptor based systems, but also pay a considerable efficiency penalty;
- This penalty is mostly due to the myriad of descriptor matching operations, which are already expensive when taken individually;

- Advancing the state of the art of matching techniques (through kNN search) is more fruitful than trying to reduce the descriptors, since the inherent dimensionality is very complex (and computationally expensive) to take into account;
- At the dimensionality of SIFT we cannot hope to make exact kNN search; it has to be approximate. Our objective is to obtain the best compromise between efficacy and efficiency;
- For kNN search to be of practical use, the constraints of computer architecture have to be considered; concretely, we have to minimise the number of random accesses to the data;
- "Hidden" compromises of the technique often prevail in its practical adoption: in our case, implementation complexity, index creation times, and whether or not the index is dynamic (admitting insertions and deletions without significant performance degradation).

In short, the heart of our problem becomes to solve the kNN search for a very large number of points, in a very high-dimensional vector space, respecting the reality of technology.

Therefore, in the next chapter we enter the universe of kNN search, its challenges, and some very interesting existing solutions from the literature. In the chapter which immediately follows, we propose our three new techniques.

Chapter 3

State of the Art on Methods for Nearest Neighbours Search

In chapter 2, we have seen that the time spent matching the descriptors is critical for the efficiency of the image identification system. In this chapter we will introduce the *k nearest neighbours search* (also known as *kNN search* or *similarity search*), which is the most used technique to do this matching.

Unfortunately, due to a phenomenon known as *curse of dimensionality*, the time needed to perform the kNN search grows exponentially with the dimensionality of the descriptor. While no single method is known to perform well in every instance of the problem, many algorithms exist to accelerate the search, normally through an approximation of the correct answer.

Because there are hundreds of methods and variants described in the literature, we refrain from providing an account of each one of them. Instead we think it is more useful to identify the essential mechanism of those methods, and understand how they are affected by the increase of descriptor dimensionality. We describe in detail a small selection of methods, which either are directly related to our own developments, or are particularly fitted to contexts similar to ours.

After stating what kNN search is, and how it is affected by dimensionality, we analyse the design decisions involved in the creation of a method to perform it. We then describe in-depth a few selected methods.

At the end of this chapter, we briefly discuss some alternatives to the kNN search for matching the descriptors.

3.1 What is the *k* nearest neighbours search

The *k nearest neighbours search*, also known as *kNN search* or *similarity search* consists in finding the *k* elements which are the most similar to a given query element. Or, more formally:

Definition 3.1 — kNN search:

- Given a *d*-dimensional domain space *D*;
- and a base set *B* with the elements $b_1, b_2, \ldots, b_n \in D$;
- and a query $q \in D$;
- and a dissimilarity function $\Delta : D \times D \rightarrow \mathbb{R}$;
- the kNN search for the *k* elements most similar to *q* consists in finding a set $S = \{s_1, s_2, \ldots, s_k\}$ where

$$\forall s_i \in S \; \forall b_j \in B - S, \quad \Delta(q, s_i) \leq \Delta(q, b_j) . \qquad \textbf{Eq. 3.1}$$

■

An obvious solution is the sequential search, where each element of the base is compared to the query, and the *k* most similar are kept. Unfortunately, this brute-force solution is acceptable only for small bases, being unfeasible in our context.

Because (as we are going to see) it is so difficult to perform the exact kNN search in high-dimensional spaces, often we are willing to accept approximate solutions. It is useful, in those cases, to use the following definition:

Definition 3.2 — Nearby kNN search, or (1+ε)-kNN search::

- Given *D*, *B*, *q*, Δ and *S* as in Definition 2.1;
- and a relative error tolerance $\varepsilon \in \mathbb{R}^+$;
- consider that *r* is the distance between the query and the furthest of the true *k* nearest neighbours, i.e.,

$$r = \max_j \Delta[(q, s_j)] ;$$

- the (1+ε)-kNN search for the *k* elements most similar to *q* consists in finding a set $S' = \{s'_1, s'_2, \ldots, s'_k\}$ where

$$\forall s'_i \in S', \quad \Delta(q, s'_i) \leq (1+\varepsilon) \times r . \qquad \textbf{Eq. 3.2}$$

■

The $(1+\varepsilon)$ factor is often called *factor of approximation*, and indicates that the answers in the solution set S' are within a *relative error* ε of the true answers. It is possible to give a geometric interpretation to the relative error: if the minimum bounding sphere centred on *q* and containing *S* has a radius *r*, then the minimum bounding sphere centred on *q* and containing S' has a radius of at most $(1+\varepsilon)r$ (Figure 3.1).

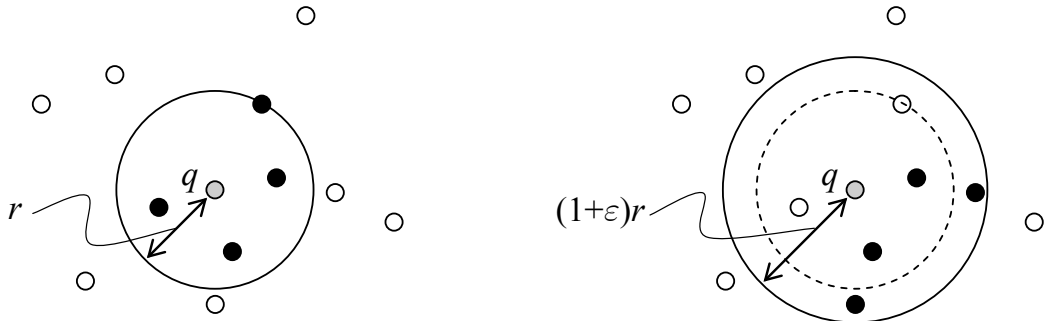


Figure 3.1: Geometric interpretation of a nearby 4-NN search. Left: exact search; right: nearby search. The query is the central point q; the black points compose the solution set

Another less common way to define the approximate search is found in [Berrani 2002], which considers the probability of missing some of the points of the exact solution:

Definition 3.3 — Probabilistic kNN search, or (1-α)-probable-kNN search:

- Given a D, B, q, Δ and S as in Definition 2.1;
- and a probability of missing a point $\alpha \in \mathbb{R}^+$;
- the (1-α)-probable-kNN search for the k elements most similar to q consists in finding a set $S'' = \{s''_1, s''_2, \ldots, s''_k\}$ where

$$\Pr[S'' \neq S] \leq \alpha . \qquad \textbf{Eq. 3.3}$$

■

When considering approximate solutions, the nearby kNN search (Definition 3.2) is useful if we are less interested in the identity of the points than in finding a "close enough" solution. For example, if the query is "find the nearest restroom", finding a solution at the distance 55 m is just as good as the true solution at 50 m. However, if we are — as in descriptor matching — interested in *identifying* the points, probabilistic kNN describes the degree of approximation more faithfully. Unfortunately, it is much easier to design algorithms for the nearby than for the probabilistic search, and therefore the latter definition is seldom adopted.

3.2 The curse of dimensionality

The efficiency of similarity search methods depends greatly on the dimensionality of the elements. While the search time can be made to grow only logarithmically to the size of the base, it will grow exponentially to the dimensionality of the elements.

For moderate dimensionality (between 2 and 10), most methods will perform efficiently enough to allow an exact or almost exact solution in reasonable time. For higher dimensionalities, it will be necessary to use approximate methods to trade-off precision for efficiency. For

very high dimensionalities (more than 30) this trade-off will become progressively more severe, resulting in serious penalties in precision in order to obtain acceptable efficiency.

The reason for dimensionality affecting the performance of indexing is related to several phenomena, some of which are very counter-intuitive and defy the geometrical common sense we have from our everyday experience with 2 or 3-dimensional spaces.

The classical account for this problem was given by Bellman, who coined the expression "curse of dimensionality", observing that the partition of solution spaces in optimisation problems is inefficient for highly dimensional problems [Bellman 1961]. The effects of this "curse" on spatial indexes may be found in [Böhm 2001, § 1.2] and [Weber 1998, § 2.3].

3.2.1 The number of partitions grows exponentially

Indexes work by partitioning the base set of elements into manageable-sized pages, letting the search algorithm operate on a limited number of those pages. The partitioning is done considering the distribution of the elements in the domain space, in order to be able to retrieve, later, only the pages nearest to the query.

This is a problem for high-dimensional spaces, because the number of possible partitions grows exponentially with the number of dimensions. Consider, for example, that we divided each dimension in two halves, and inspect only the half nearest to the query. This will generate 2^d partitions, which, as dimensionality grows, may greatly exceed the number of points in the base, resulting in a useless index.

3.2.2 Volume concentrates on the borders

A surprising characteristic of high-dimensional spaces is the concentration of volume* on the borders of the common geometric figures. Consider, for example, the formula for the volume of respectively, a hypercube and a hypersphere:

$$V_{cube} = r^d \qquad V_{sphere} = \frac{\pi^{d/2}}{\Gamma\left(d/2+1\right)} r^d \qquad \textbf{Eq. 3.4}^{\dagger}$$

where r is either the side of the hypercube or the radius of the hypersphere. Consider now the ratio between the volume of the figure with side/radius $r+\varepsilon$ to the one of side/radius r:

$$ratio = \frac{(r+\varepsilon)^d}{r^d} \qquad \lim_{d\to\infty} \frac{(r+\varepsilon)^d}{r^d} = \infty \qquad \textbf{Eq. 3.5}$$

* Rigorously speaking, the term "volume" applies only to 3D spaces, the generalised notion is known as *content*.

† Γ is the *gamma function*, a generalization of the factorial for the complex numbers. If z is a positive integer, then we have $\Gamma(z) = (z-1)!$, otherwise the function "fills in" the values by analytical continuation.

This means that in higher dimensions, enlarging slightly a figure increase so much its volume, that the volume of the original figure, in comparison, is negligible. The effect grows very quickly with dimensionality, as the graph in Figure 3.2 shows.

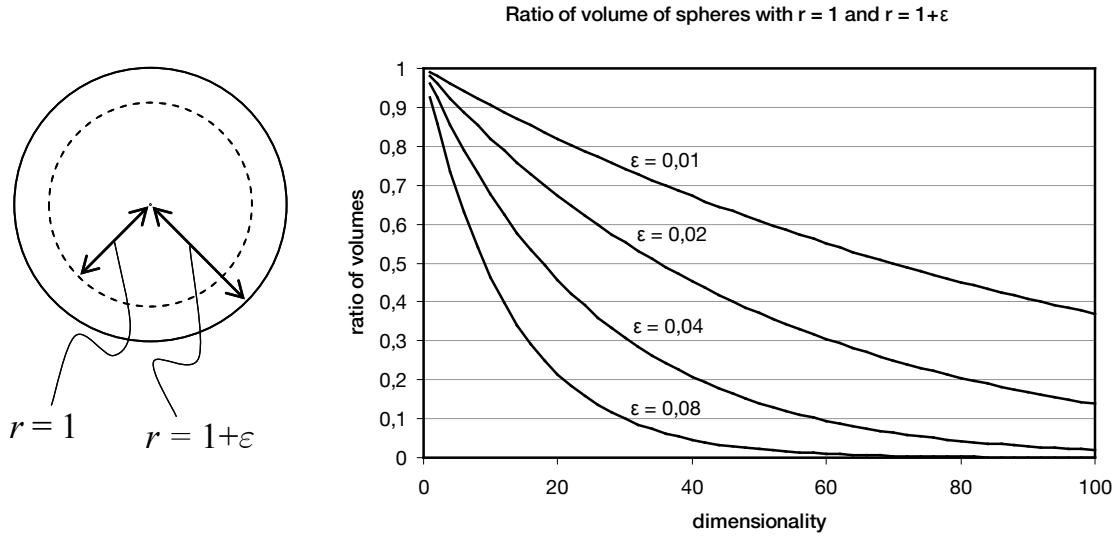


Figure 3.2: In high dimensionalities, volume concentrates on the borders

This phenomenon bears serious consequences for nearest neighbours search. For example, if one is performing approximate $(1+\varepsilon)$-kNN search, it is easy to see that, as the dimensionality grows, even very small ε will have a great impact on the region of space considered.

3.2.3 Space seems progressively emptier

As dimensionality grows, the space seems to be progressively emptier. Suppose the points are uniformly distributed on the space, on the interval $[0; 1]^d$. Because of the previous result, the probability of finding a point in an arbitrary hypercube of side r, where $r < 1$, tends to zero, as the dimensionality tends to infinity.

Another way to express the same phenomenon is to consider the number of points needed to sustain a constant density as dimensionality grows. As the global volume of the space grows exponentially, so does the size of the base set.

In practice, the sparseness of data prevents a meaningful clusterisation of the points, which hinders the working of indexes.

3.2.4 Proximity to boundaries becomes the rule

A serious difficulty in high-dimensional spaces is the fact that the number of neighbours in a given region grows exponentially. There is a high probability of the query being near to several boundaries. This is problematic, because it makes it necessary to check those bordering regions for candidate neighbours.

3.2.5 Distances become indistinct

Supposing again that the data are uniformly distributed and non-correlated, one can prove that in high-dimensional spaces, the points become progressively equidistant. This phenomenon defies the very usefulness of the concept of nearest neighbours, because the solution becomes extremely sensitive to small perturbations.

What is meant, is that, for very high dimensionalities, the difference of the distances between the nearest and the furthest points to the query becomes so small, that if slightly perturbed, the points may very well exchange places in the list (Figure 3.3).

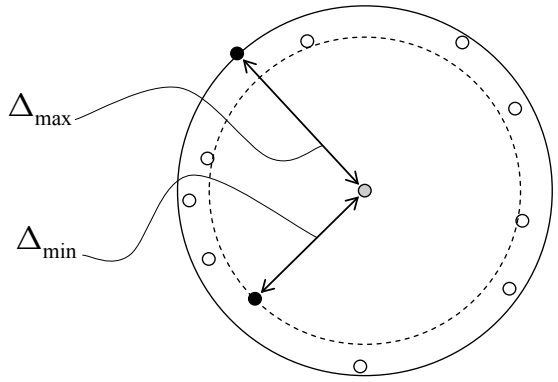


Figure 3.3: In high-dimensional spaces, distances become indistinct [Moënne-Loccoz 2005]

One way to escape this effect, and be able to perform the kNN search in a meaningful way, is to assume that the base set is clustered and that the query follows the same distribution as the base set. One can also use specially designed distance functions (§ 3.3.5.2), which preserve their discriminating power in high-dimensional spaces.

3.2.6 Which dimensionality?

When considering the dimensionality of the data, the more obvious one — the embedding dimensionality (the dimensionality of the space in which the data are immersed) — may not be the most indicative of the difficulty in indexing the data. The complex internal structure of the data may make a relatively high-dimensional database easier to process than another of lower dimensionality, but whose points are uniformly distributed.

Though linear techniques like the PCA may help to reveal the effective dimensionality of the data, almost always the interrelationships of the data are much more complex than simple linear correlations.

A more effective approach is the analysis using fractal geometry. In a nutshell, the idea is that uniform, uncorrelated data (which are the most difficult to index) have a “fractal dimension” which approaches their embedding dimension. In contrast, strongly correlated data, like a straight line or a smooth curve, will have just one “fractal dimension”. Partially correlated data should have a “fractal dimension” between those two extremes.

The notion of fractal dimension is, however, ambiguous, and can be defined in different ways. Faloutsos et al. propose defining it as the Hausdorff-Besicovitch dimension, approximated through the “box count” dimension and using it to predict the difficulty in indexing the data

[Faloutsos 1994]. This dimension belongs to a family of dimensions which are calculated by partitioning the space in a grid of hypercubes of side r. Then we sum the fraction of points in each box (raised to a non-negative number p, which can be used to fine-tune the measurement, see below). This sum can be expressed as:

$$S_p = \sum_i \left(\frac{n_i}{N} \right)^p . \qquad \textbf{Eq. 3.6}$$

where n_i is the number of points which fall into the i^{th} hypercube and N is the total number of points in the database. We repeat this sum for various values of r and then plot a graph with logarithmic axis (Figure 3.4). Eliminating the flat areas of the graph and doing a linear interpolation on the plot, the absolute value of the inclination of the line found will give the value of the dimension d_p.

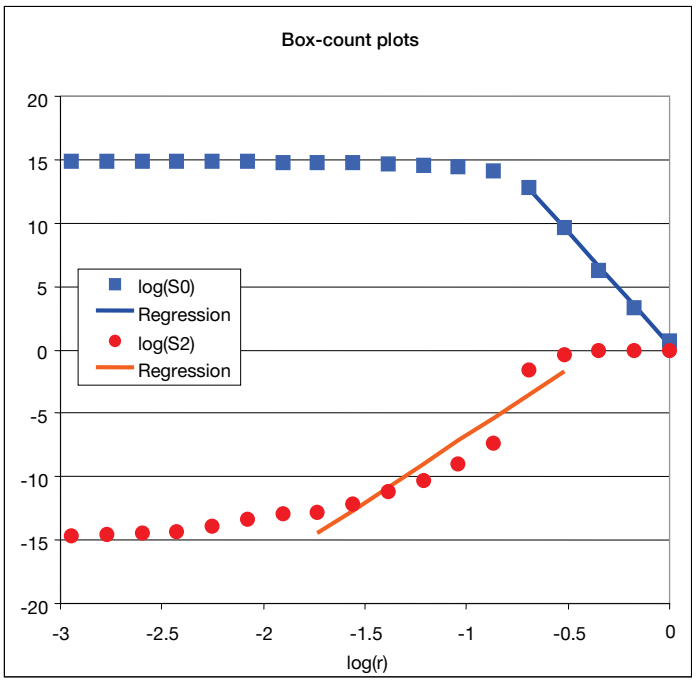


Figure 3.4: Finding the fractal "box-count" dimensionalities of a database — for different values of r, the values of S_0 and S_2 are calculated. A log–log graph is plotted, and the non-flat areas are fitted in a straight line. The absolute value of the inclination of those lines gives, respectively, d_0 and d_2

In [Faloutsos 1994] d_0 (also known as Hausdorff or "box-count" dimension) is used to predict the difficulty in indexing the data, while in [Belussi 1995] d_2 (which the authors name the "correlation" fractal dimension) is preferred. For mathematical fractals, the two values should be strictly equal.

In real-world datasets, the similarity of the two dimensions may indicate the actual degree of self-similarity of the dataset (or conversely, if they are too much discrepant, this may indicate that the dataset is not self-similar and that the fractal analysis might not be appropriate).

3.3 Approaches to the solution

The literature on kNN methods is prolific, counting dozens of methods and hundreds of variants. While being an evidence of active and productive interest of the scientific community on the topic, this abundance makes almost impossible to explore exhaustively the state of the art.

Our objective, therefore, is not to describe (or even list) every important method, but instead:

- To understand the challenges and the design choices one faces when creating a kNN search method;
- To understand the great "families" of solutions;
- To provide a more in-depth description of the methods closely related to our work, either because they served as inspiration for our own methods (chapter 4) or because they were designed to tackle the problem in contexts similar to ours.

First, let us examine some design choices.

3.3.1 Index structure

We have already discussed some properties of indexing in § 2.5.1.

- *One-dimensional indexes*: since most conventional indexes require the existence of a total order on the data, many methods work by mapping the d-dimensional points into one-dimensional *keys* for which such an order can be devised. This allows to sorting the data and then using an efficient method for the search. The induced total order must capture, as well as possible, the spatial proximity of the points.

  Many efficient data structures exist for the indexing of one-dimensional data. In secondary memory, one of the most used is the B-tree and its variations, like the B^+-tree [Bayer 1972].

- *Multidimensional trees*: the large majority of methods use some form of multidimensional tree. Since in multidimensional spaces there is no total order, the search operations on those trees rely on different strategies than the common sorted trees. Usually, *pruning* techniques are used to discard the branches where the answers are impossible (or unlikely) to be found. Some kind of prioritisation can also be use, to traverse first the branches more likely to contain the answers.

- *Hashing* is a very efficient technique which allows direct access to the data, without the need of traversing a decision structure (like we have to do in trees). A *hash function* computes directly the location of the data from the indexing key [Fredman 1984].

  In a perfect scenario the hash function would be a bijection between the positions in the index and the set of inserted keys. In practice, the index has to deal with *collisions*, which occur when the hash function associates the same position to different keys. Also, because of the trade-off between occupancy and collisions, there is almost always some degree of sparseness in the index.

3.3.2 Static x dynamic indexes

Some methods are completely static, meaning that the base set must be completely known before the construction of the index begins. Those methods do not admit alterations on the base set once the index has been constructed. If data are added or removed, the index must be completely rebuilt.

In the other extreme, a few methods are completely dynamic, allowing for easy incremental insertion/deletion of data without penalties on the performance of the index.

The large majority of methods fall between those extremes, allowing alterations on the base set, but withstanding a progressive loss of performance. Some methods provide an optional operation of *reindexing*, which may be called from time to time, in order to improve the performance after a certain amount of alterations. Deletion is normally more easily supported than insertion, by simply marking the deleted entries.

There are techniques to dynamise an otherwise static index. One of those is the *logarithmic method*, which consists in maintaining a set of $\log_2 n/m$ indexes, where m is the number of points the primary memory buffer can hold. The indexes on disk have the increasing capacities of m, $2m$, $4m$, …, $2^i m$. When inserting a point, one checks if it fits in the buffer, and, in this case, insertion is done incrementally. Otherwise, one checks for the first empty index i in the disk and bulk loads it with the contents of the buffer and of the indexes $0 \ldots i–1$, emptying them in the process (Figure 3.5).

Though one individual insertion may become expensive (in the worst cases a completely new index will have to be added to the series with all data stored so far), the *average* cost of the insertion is quite acceptable and may, in fact, be cheaper than performing incremental management of the index after every insertion.

In order to do a search, one has to inquire all the indexes (including the buffer) and combine the results.

Note that the indexes do not have to be implemented homogeneously: adequate data structures can be chosen accordingly to their sizes and location (in particular, the memory buffer may use a different technique than the disk indexes). For a detailed treatment of this technique and related others, see [Overmars 1983]. One spatial index method which uses this technique is the BKD-tree [Procopiuc 2003].

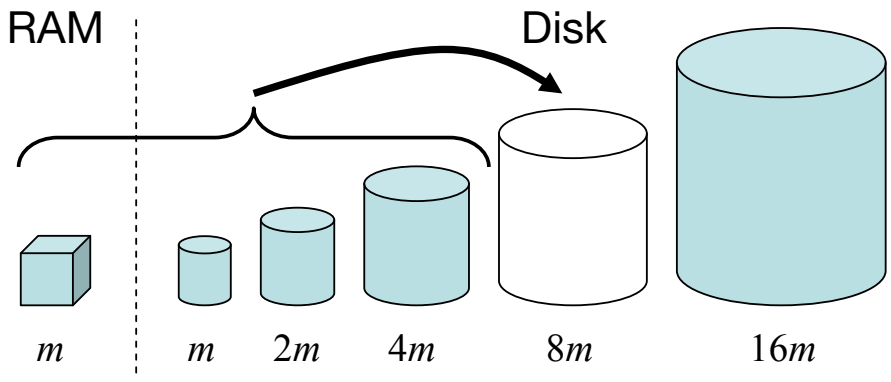


Figure 3.5: The logarithmic method allows transforming a static index into a dynamic one. Here, to make room in the buffer, we will bulk load the index 8*m* with the contents of the buffer and all the lesser indexes. Remark that at all times the disk indexes are either completely empty or completely full

3.3.3 Random-access, locality of reference and disk-friendliness

Many methods are based on the hypothesis of cheap random access to the data. Those methods are useful only when implemented in primary memory, for disks, while allowing random access, charge a considerable penalty for the relocation of the read/write heads, as we have seen in § 2.5.2. Because primary memory is of limited size, those methods can be used only for relatively small base sets.

Methods which reduce the number of random accesses and explore the data mainly sequentially are easier to implement efficiently on disk. Methods which, in addition, provide the separation of data in fixed sized chunks, are even more adequate to disk storage, where data are stored in *blocks*.

3.3.4 Data domains

The design of the method must consider the nature of the data, in order to explore certain properties of the spaces in which they are embedded. The choices are:

- Vector spaces;
- Metric spaces;
- Non-metric spaces.

The vast majority of methods assume the data belong to a d-dimensional vector space, with finite d. Those spaces are isomorphic to $\mathbb{R}^d$, which allows the use of geometric reasoning in the solution (though one must avoid relying too much on the intuitions built upon experience on 2D and 3D spaces, which may prove false in high-dimensional spaces).

Because requiring a vector space limit the data domains we can treat, some methods try to make less stringent demands. A whole family of methods was created which uses only the *distances* between the points, and no other geometric property. Those are the methods based on *metric spaces*, which are more general than those which require vector spaces, since vector spaces can be always be turned into metric spaces (using an appropriate distance function).

Definition 3.4 — Distance, metric space:

- A function $\Delta : M \times M \to \mathbb{R}$ is a *distance function* iff:
  (1) The distances are *non-negative*, i.e., $\forall x, y \in M,\ \Delta(x,y) \geq 0$;
  (2) The *identicals are indiscernible* and vice-versa, i.e., $\forall x, y \in M,\ x = y \Leftrightarrow \Delta(x,y) = 0$;
  (3) The distances are *symmetrical*, i.e., $\forall x, y \in M,\ \Delta(x,y) = \Delta(y,x)$;
  (4) *Triangle inequality* holds, i.e., $\forall x, y, z \in M,\ \Delta(x,z) \leq \Delta(x,y) + \Delta(y,z)$.
- A set M together with a distance function $\Delta : M \times M \to \mathbb{R}$ is a *metric space*.

■

Not all (1)–(4) properties are needed by all methods. We can always substitute (2) by the weaker property $\forall x \in M,\ \Delta(x,x) = 0$ (making the space *pseudometric*). The weaker condition suffices because for the entire class $B_a = \{x \in B \mid \Delta(x,a) = 0\}$ of points in the base in-

discernible from a, we can insert in the index the unique proxy a and point it to the other elements in the class.

Methods which forgo (3), (4), or both are typically named *non-metric*. In those cases, it is customary to use the more general name of "dissimilarity" for the function Δ, instead of "distance", which traditionally evokes metric spaces.

3.3.5 Dissimilarities

Methods vary by the kinds of dissimilarities they allow. For example, a method for vector spaces can be very specific and allow only the Euclidean or the Manhattan distance (Definition 3.5). It can be less strict and accept any distance based on p-norm. Or even less strict and allow any metric distance. If it is really general, it can accept a dissimilarity, asking only for very general properties.

3.3.5.1 Norm-based distances

On vector spaces, the most commonly found distances are based on the p-norms.

Definition 3.5 — p-norms, p-norm distances

- On the vector space $\mathbb{R}^d$, the **p-norm** of a vector x, whose components are $x_1, \ldots, x_i, \ldots, x_d$ is:

$$\|x\|_p = \left(\sum_{i=1}^{d} |x_i|^p \right)^{1/p} \qquad \textbf{Eq. 3.7}$$

- The **maximum norm**, also known as infinity-norm is:

$$\|x\|_\infty = \lim_{p \to \infty} \|x\|_p = \max\left(|x_i|\right) \qquad \textbf{Eq. 3.8}$$

- The **zero-norm** is defined by analogy, despite the fact it is not a vector norm *stricto sensu*[*]:

$$\|x\|_0 = \sum_{i=1}^{d} |x_i|^0 \text{, if we define } 0^0 = 0 \qquad \textbf{Eq. 3.9}$$

- The **p-norm distance** between vectors x and y, also known as **Minkowski distance** of order p, or $\mathbf{L^p}$ distance is defined as:

$$\|x - y\|_p \qquad \textbf{Eq. 3.10}$$

- The zero-norm distance is also known as **Hamming distance**, and counts how many different components the two vectors have;
- The 1-norm distance is also known as **Manhattan distance**, rectilinear distance or city-block distance. It is the simple sum of the absolute differences of the components;

[*] In particular, the zero-norm is violates *homogeneity*, which establishes that $\forall a \in \mathbb{R}, \forall x \in \mathbb{R}^d,\ \|ax\| = a\|x\|$.

- The 2-norm distance is also known as **Euclidean distance**, and is the one which corresponds the most to our everyday idea of distance in 2D and 3D spaces;
- The maximum norm distance is also known as **Chebychev distance**, and corresponds to the maximum absolute difference between the components.

■

If the dimensions are not homogeneous, which happens often when they are measured along different scales, the p-norm distances are not adequate. Consider, for an exaggerate example, a 2-dimensional vector where the first component is measured in millimetres and the second in kilometres. A simple Euclidean distance will not represent the reality of the data.

For a more realistic example, consider another 2-dimensionsional set where the data are pairs <*height*, *weight*> collected from people in a medical setting. The geometry of the space where we measure height in centimetres and weight in kilograms will be different from the one where we use inches and pounds, respectively. In particular, the kNN search may give completely different results in the two sets, even if they actually represent the same reality. To avoid this kind of anomaly, it is useful to define another class of distances.

Definition 3.6 — Weighted p-norms, weighted p-norm distances:

- On the vector space $\mathbb{R}^d$, the **weighted p-norm** of a vector x is:

$$\|x\|_{w,p} = \left(\sum_{i=1}^{d} w_i |x_i|^p \right)^{1/p} \qquad \textbf{Eq. 3.11}$$

- Where $w \in \left(\mathbb{R}^+\right)^d$ is the *weight* of the norm;
- Distance, as usual, is defined as the norm of the difference.

■

Using the definition above, one can take the components of the weights as the standard deviations of the components of the data, making the distance scale-invariant.

A problem left unsolved by the weighted p-norm distance is the correlation of data. Consider again our database of patient heights and weights. If we want our function distance to separate the slender from the stout, we have to consider that weight is, in part, a function of height.

Correlation is somewhat related to the problem of scaling, if we consider that in both cases we want all dimensions to have equal importance. In our example, the height artificially overcomes the weight, because the former is somewhat "embedded" in the latter. In order to avoid this, we want to design a distance that takes correlation into account.

Definition 3.7 — Mahalanobis distance:

- On the vector space $\mathbb{R}^d$, the **Mahalanobis distance** between vectors x and y is:

$$D_{\mathcal{M}}(x,y) = \sqrt{(x-y)^{\mathrm{T}} \Sigma^{-1} (x-y)} \qquad \textbf{Eq. 3.12}$$

- Where Σ is the *covariance matrix* of the data and x^{T} is the transpose of vector x.

■

The Mahalanobis distance will correct the effects of linear correlation. It belongs to the larger class of *generalised Euclidean distances*, which use arbitrary matrices in the place of the inverse of the covariant matrix.

It is interesting to note that, in a vector space, each one of those distance functions is associated to a certain geometry. Consider $\mathbb{R}^3$ and take the locus of the points at unitary distance from the origin. For the Euclidean distance, the locus is a sphere; for the Manhattan distance, it is an octahedron with all the vertices on the axes of the coordinates; for the Chebychev distance, it is a cube with edges parallel to the axes of the coordinates, etc. (Figure 3.6).

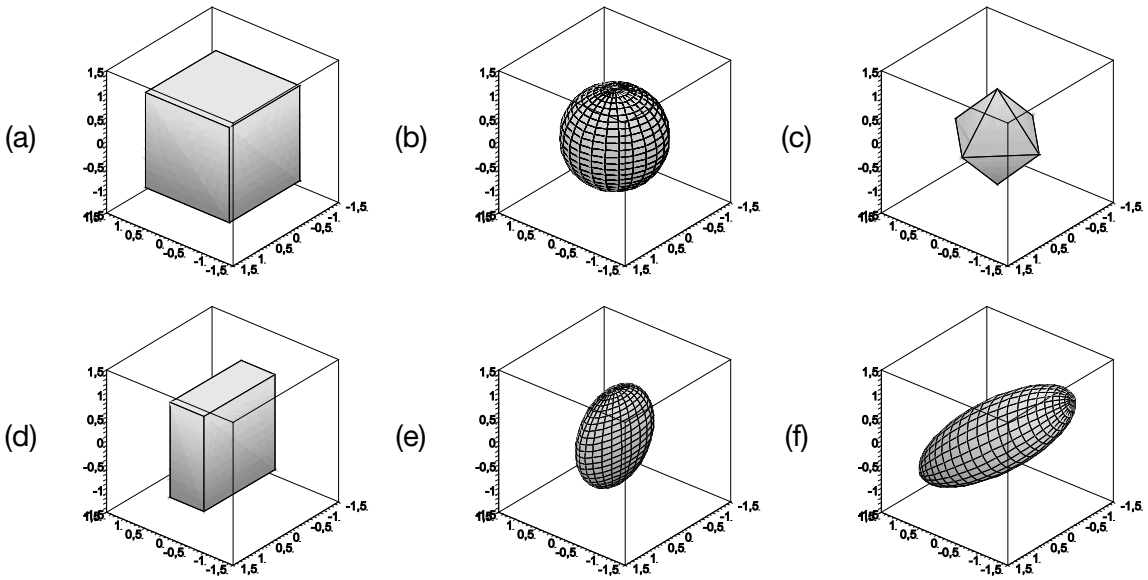


Figure 3.6: The locus induced by some distances. a) Chebychev; b) Euclidean; c) Manhattan; d) Weighted Chebychev; e) Weighted Euclidean; f) Mahalanobis

Two notes of warning. First, the Mahalanobis distance takes into account only *linear correlation*. It is not adequate to correct the effects of non-linear correlation. (Again, the example of heights and weights is good, because the correlation between the two is actually non-linear.*)

Second, care must be taken when applying a distance which normalises and decorrelates the data. In those cases, one must ensure that the transformed function is indeed more meaningful, which may not be true. Facing a 2D dataset where the standard deviation of the first dimension is 20 times greater than that of the second dimension, we may be eager to normalise it. But if it is a table of latitudes and longitudes of cities in Chile, this will not result in a more accurate representation of reality. For a less extreme example, consider colour data, where normalisation may sever the connection between the distance function and human perception of the data.

3.3.5.2 Designing distances for high-dimensional spaces

The worst drawback of the distances studied so far is the implicit assumption that the distortion between two matching points (i.e., the ones which should be at small distances from each other) is composed of small fluctuations in many coordinates. Contrarily to this premise, we

* That is the reason why the BMI (*body mass index*) is computed to the square of the height.

often observe that matching points may present large differences in just a few coordinates, while all the others agree almost perfectly.

Take the p-norm distances, for example: they tend to be dominated by the largest component differences, even when the other components are very similar. For many datasets, this characteristic makes them a poor choice.

For high-dimensional spaces, Aggarwal proposes the design of specially conceived distance functions, with several desirable properties, and presents, as an example of their potential benefits, a simple distance function based on the raw application of his ideas [Aggarwal 2001].

He defends that the averaging effect of most distances leads to a lack of contrast, because two points, even when they match, are unlikely to be similar in all dimensions. The distance must be able to take in consideration the general agreement between the points, in terms of the number of dimensions in which the points are similar, in order to obtain more contrasting values.

In addition, when the distance treats the coordinates uniformly, it fails to take into account the statistic significance of each coordinate. Consider, for example, text retrieval applications, which have hugely dimensional spaces (often one dimension per possible research term). In those contexts, most of the coordinates of any point is zero, meaning that the research term is absent from the document. The distance function, however, takes the non-zero coordinates much more into account, since the co-occurrence of a term is much more indicative of similarity than the co-absence.

Therefore, instead of ignoring (or removing) the internal correlations of the data, the ideal distance for high-dimensional data should use them to improve its significance.

Of course, the distance algorithm should also be efficiently in terms of time and space requirements; otherwise its computation will become a bottleneck.

3.3.6 Families of methods

We can make a tentative classification of kNN search methods into a few large families, though it is sometimes difficult to separate the boundaries:

- *Space partition*: this approach divides the space into a certain number of partitions, normally hyperrectangles. Data distribution is taken into account by partitioning more finely the densely populated regions of the space. The objective is to limit the exploration to the few partitions of the space which are nearer to the data.

  Historically, one of the first (and most successful) space partition methods is the KD-tree, which we discuss in § 3.5 [Bentley 1975; Friedman 1976]. Another early example is the quad-tree [Finkel 1974].

- *Data partition*: this approach divides the *data* into partitions, which can be overlapping in the space. Again, the objective is to limit the exploration to a few partitions, the ones most probable to contain the answer set.

  The archetypical data partition method is the R-tree, which groups the data into (possibly overlapping) bounding hyperrectangles [Guttman 1984]. Many variants ensued.

The R*-tree [Beckmann 1990] has an optimised splitting strategy. The SS-tree [White 1996] uses bounding hyperspheres instead of bounding hyperrectangles. The SR-tree [Katayama 1997] uses both. The TV-tree [Lin 1994] uses only a subset of the dimensions at each partitioning decision. The X-tree [Berchtold 1996] allows controlling the degree of overlapping between the partitions.

Most data partition methods suffer from the difficulty in choosing a good partitioning policy. The ideal partitions are more or less uniform in number of points (to facilitate storage), have a small volume, and do not overlap (to optimise selectivity). However, if the method tries too hard to optimise the partitioning, the cost of building the index may become prohibitive. Though we do not rule out the possibility of this problem being solved in the future, we decided, for the purposes of this thesis, not to further investigate this family.

- *Clustering*: if space and data partitioning are *top-down* approaches, which work by splitting the data in smaller portions, this approach works *bottom-up*, progressively aggregating the data. The final result is much similar to data partitioning — though the methods to separate the data are different, the goals are the same: obtaining groups which are compact (small volume) and well separated (distant from each other).

  Examples of method which work by clustering are DBIN [Bennett 1999], CLINDEX [Li 2002] and BIRCH [Zhang 1996].

  Even more than data partitioning, clustering is expensive to compute, which puts a limit on the size of the databases one can expect to build in a reasonable time. We did not further explore this family.

- *Metric*: this family of methods uses only the distances between the points, and is not limited to vector spaces. They separate the data into groups of points which are near to each other and use the triangle inequality to prune the groups which need not to be considered for a particular query.

  In many cases those methods are used with distances which are expensive to compute (e.g., string editing distances). One of the objectives is to use approximations to minimise the need of actually computing the distance.

  In metric spaces, the notion of *dimensionality* may not always be useful to characterise the difficulty of indexing the data. Instead, the distribution of the expected pairwise distances is used. The more the distances are alike, the more the database will be difficult to index [Chávez 2001].

  A well-known metric method is the M-tree [Ciaccia 1997]. The slim-tree [Traina Jr. 2000] is an enhanced variant.

  Some methods relax the constraints on the distance function, admitting dissimilarities instead. Though those methods are referred to as *non-metric* in the literature, *partially metric* would be a more adequate description. They vary in the axioms they relax (see, for a typical example, [Goh 2002]).

  Metric (and partially metric) methods are another family for which index construction is very expensive, though they form an elegant theoretical framework. We did not further explore this family.

- *Data approximation*: those methods work by quantising the data and considering only a few bits per dimension. If the data are uniformly distributed and uncorrelated, this technique works very well, since selectivity will actually *improve* as dimensionality grows.

  However, it is seldom the case that data are uniform and uncorrelated, and the pre-treatment necessary to make them so (PCA, kernel-PCA) may render those methods unusable for very large databases. Still, if the data have a low level of clustering, those methods may be very useful.

  The first method to use data approximation was the VA-file, described in § 3.6 [Weber 1997]. The method was applied on data pre-treated with PCA [Ferhatosmanoglu 2000] and kernel-PCA [Heisterkamp 2003]. An interesting extension which uses local polar coordinates in addition to the higher-significant bits is the LPC-file [Cha 2002].

- *Projection and hashing*: those methods project the data into lower-dimensional spaces. The nearest neighbours are sought in the projected spaces and the partial results are aggregated to produce the final result. The scheme can be as simple as projecting the data onto a straight line, or as elaborate as using especially conceived space-filling curves. If hashing is used, the position in the index can be directly obtained from the keys.

  A very early method based on projection was proposed by Friedman where he used a single projection to filter the points which were obviously out of reach [Friedman 1975].

  More recent examples of methods based on this technique are the LSH (see § 3.7, [Indyk 1998]), the MEDRANK (§ 3.8, [Fagin 2003]) and the whole family of methods based on space-filling curves (§ 3.11).

The methods we will describe in detail, further in this chapter, come from the space partitioning, data approximation and projection/hashing families, which have been proved successful in tackling very large databases.

3.3.7 Complementary techniques

These complementary techniques can be applied in conjunction to any of the previously seen methods in order to enhance their performance.

- *Randomization*: the principle of randomization is to avoid the worst case of the algorithms, by using aleatory perturbations which make this worst case very improbable.

  In tree indexes, for example, the choice of the ramification to follow is quite critical in the upper levels (near the root): if the wrong path is chosen, the algorithm will spend a long time backtracking (or, most probably, will be interrupted before, resulting in large approximation errors). In other words, the early choices are critical.

  One way to avoid those problematic cases is to have multiple trees with different random partitionings, and to traverse them in parallel. Even if in some of the trees we make poor early choices, chances are that in at least one of the trees a good path will be selected.

  Random trees have been used extensively for classification [Geurts 2006; Lepetit 2005]. Specifically for kNN search in the context of descriptor matching, a random variant of the KD-tree has been proposed [Silpa-Anan 2005].

- *Redundant covering*: methods based on partitioning (both spatial and data) can be relaxed to allow redundant covering, where data points may appear in multiple nodes of the index. The advantage is that for complex data distributions, the intersection between clusters may be duplicated, in order to avoid the need to visit multiple nodes in the index. The impact may be considerable, when we keep in mind the cost of random access (§ 2.5.2, § 3.3.3).

3.3.8 Precision control

Because exact kNN search is too expensive for the databases we want to treat (more than 30 dimensions, millions of items), we have to admit an approximate solution.

As we have seen in § 3.1, this "approximation" may be defined either by the notion that we find points not so far from the true answer (nearby kNN search), or by the notion that we have a certain probability of finding the true answer (probabilistic kNN search). The large majority of the methods available use the concept of nearby kNN. To our knowledge, the only methods with probabilistic control of the precision are the ones proposed in [Berrani 2002].

Not all methods are able to guarantee that the desired level of approximation will be actually delivered. The methods may be classified as:

- *Enforced approximation*, *informed approximation*, *unknown approximation*: if the method has enforced approximation, the users are able to specify the level of approximation they want, i.e., the ε in (1+ε)-kNN search or the α in (1-α)-probable-kNN search. The method will guarantee that the results are within those levels of approximation.

  Alternatively, the method, being unable to enforce the approximation, may estimate it at the end of the search. Typically, those methods will return the answer set and the estimated value of ε, which means that the answers given are no more than a factor of (1+ε) of the true answers.

  Most methods, however, have no precision control whatsoever.

- *Index-wise* or *search-wise* precision: in most methods, the level of approximation has to be decided *a priori*, during the construction of the index, and will be the same for all searches. Other methods are very flexible, allowing the selection of any degree of precision during the search.

  Some methods are midway, with a set of approximation levels being specified during the construction of the index, and the desired level selected at the search operation.

3.4 Reviews

As we mentioned before, there is a very abundant literature on kNN search, which attests both the importance of the subject and the persistent lack of efficient general solutions. The reader

may appreciate the plentitude of approaches in Figure 3.7, which presents a genealogy of a fraction of the methods and variations available up to 1996.

An early and very complete review of the field may be found in [Samet 1990], which treats the general subject of spatial data structures. Other in-depth reviews can be found in [Gaede 1998] and [Böhm 2001], the latter being concerned with the integration of spatial indexes in relational databases. A short but recent review can be found in [Moënne-Loccoz 2005].

A very complete review concerned exclusively with metric methods may be found in [Chávez 2001].

An interesting review, focused on the theoretical issues of kNN search may be found in [Indyk 2004], where the author is not interested in listing or classifying methods, but in compiling the knowledge on the behaviour of the kNN search and related problems under the lens of theoretic algorithm analysis.

Here, we focus on a few methods which are of practical interest for our context. Each method (with a few closely related variants) is studied in a separate subsection. Table 3.1summarises the studied methods.

Method	Type of Search	Storage	Implem. Difficulty	Static/Dynamic
KD-Tree	Exact search; Nearby, by best-bin-first	RAM; Disk, if leaf access is minimised	Simple	Mostly static, admits a few changes before reindexing is needed
VA-File	Exact search; Nearby search possible in principle, but never tested	Disk (not advantageous for RAM)	Simple	Admits changes, but does not adapt to them
LSH	Combination of probabilistic and nearby, with a bounded probability of complete failure (no answers at all)	RAM (too many random accesses for disk)	Complex	Dynamic
MEDRANK/ OMEDRANK	Combination of probabilistic and nearby search	Designed for disk, but works well on RAM	Simple	Dynamic
PvS, NV-tree	Combination of probabilistic and nearby search	Designed for disk, but works well on RAM	Complex	Admits changes but requires periodic repartitioning
Pyramid trees	Exact search Nearby search possible in principle, but never tested	Disk or RAM indifferently	Complex	Dynamic
Space-filling curve methods	Combination of probabilistic and nearby search	Disk or RAM indifferently	Varies, but normally easy	Dynamic

Table 3.1: Summary comparison of the reviewed methods

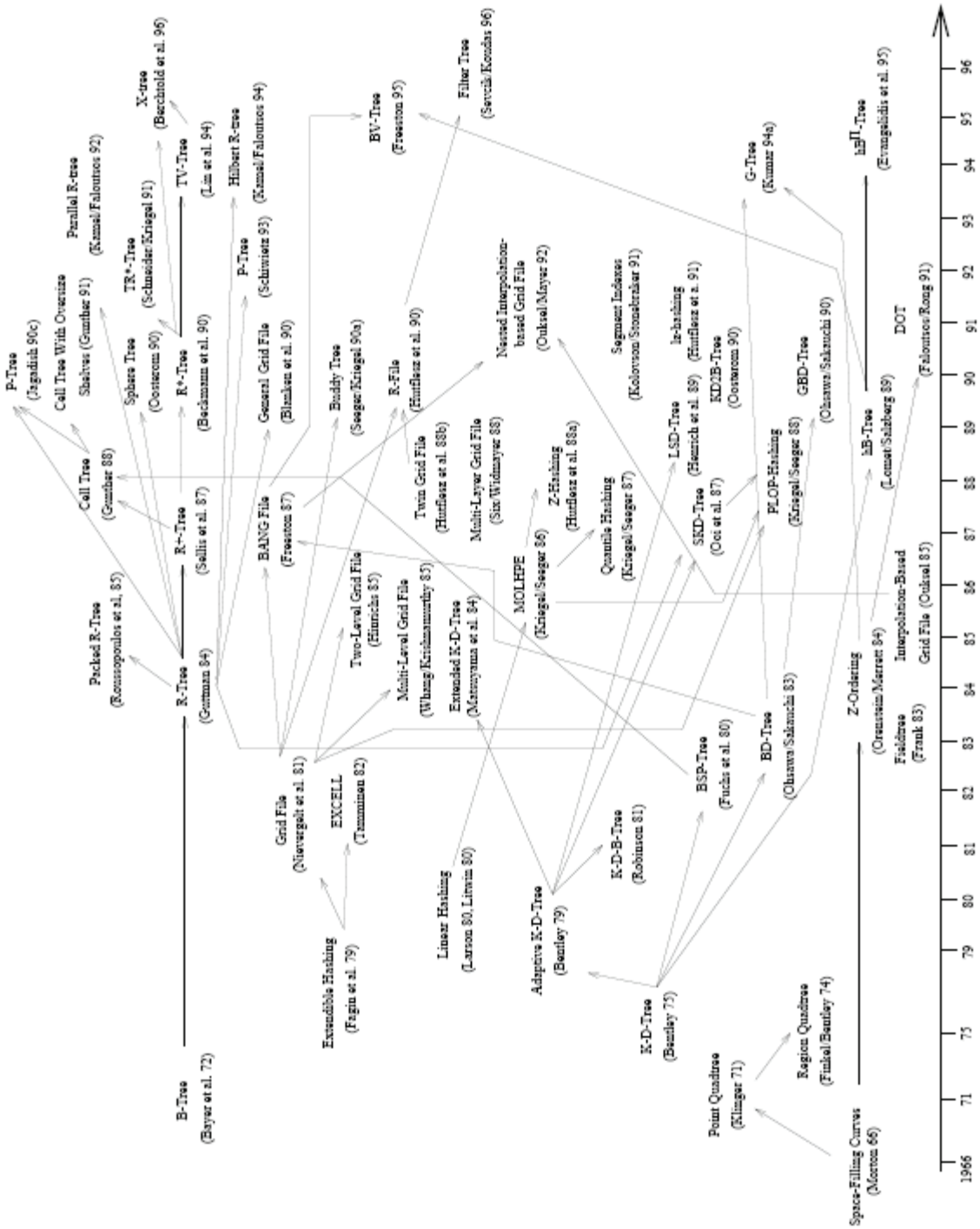


Figure 3.7: A genealogy of kNN methods up to 1996 [Gaede 1998, reproduced with permission]. All references in the figure are *apud* [Gaede 1998]

3.5 Study of case: the KD-tree

The KD-tree (***k*-d**imensional **tree***) was described for the first time in [Bentley 1975] and used for kNN search in [Friedman 1976]. The KD-tree is essentially a generalisation of the binary tree, where each node divides the space among a different dimension.

To build a KD-tree from a dataset B in the space D, we start by choosing the dimension to be split, normally the dimension where the data are the most spread. The point of splitting must then be chosen by selecting an element as the *pivot* of the splitting. A node is created with the pivot, the left subtree is created recursively with all elements lesser than or equal to the pivot and the right subtree is created with all elements greater than the pivot. If the number of elements in a subtree is smaller than a threshold, a leaf containing all remaining elements is created: this is called a *bucket* (Figure 3.8).

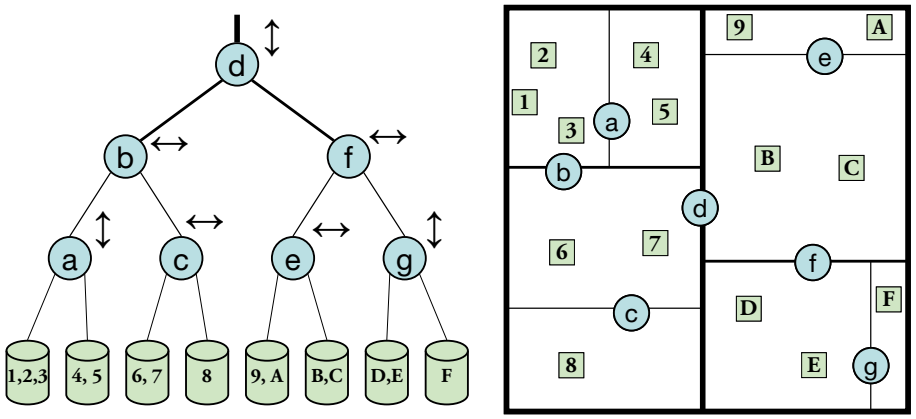


Figure 3.8: A 2-dimensional KD-tree and its associated space. The arrows beside the nodes indicate the direction of the splitting

In order to perform the nearest neighbours search in a KD-tree, we start by finding the bucket whose associated region contains the query. Once we reach it, we simply find the nearest elements sequentially, keeping the distances found. If those distances are smaller than the distance between the query and any other regions of the space, we can stop. Otherwise, we explore the subtrees corresponding to other regions of the space in order to check if there are nearer elements. The search stops when no region can possibly contain an element nearer than the elements already found.

* It is an unfortunate nomenclature tradition that the "k" in kNN and in KD-tree mean different things, respectively, the number of neighbours and the number of dimensions.

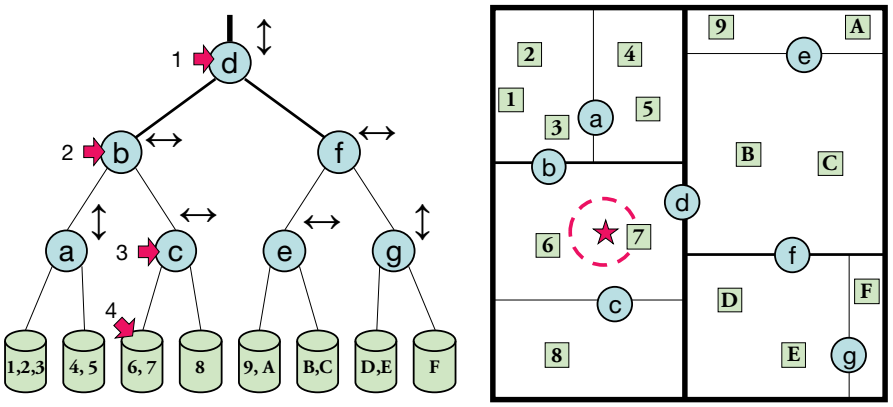


Figure 3.9: A particularly uneventful search operation, the numbers at the side of the thick arrows indicate the order in which the nodes are visited

The steps of a search operation are illustrated in Figure 3.9. Note, that at each branch we choose the region to which the query belongs. Once we reach a bucket, we compare the query sequentially to all elements. We then trace the distance from the query to the nearest element and verify that no other region can contain a better candidate. This indicates that the search can stop.

A more difficult search operation is illustrated in Figure 3.10. Once we reach the first bucket and determine the nearest candidate "b", we can trace the distance circle. But as it intersects many other regions, we are forced to check them. The order in which the regions are visited is related to the topology of the tree, so the sister bucket "9, A" is visited before the exploration continues on the left branch of the root "d".

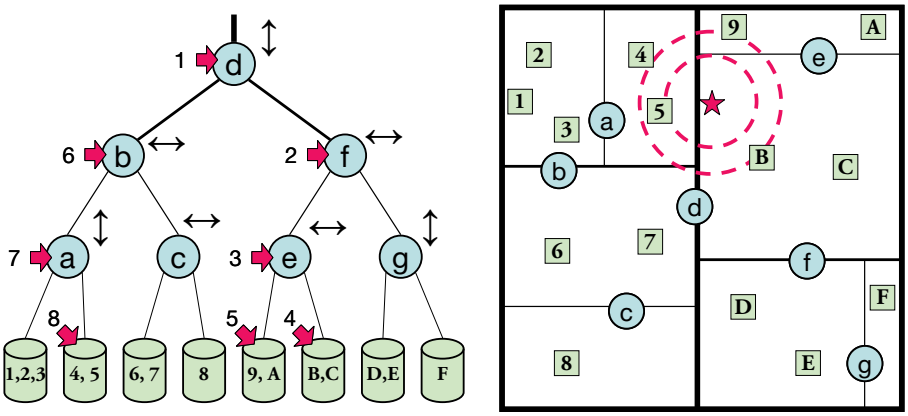


Figure 3.10: A more troublesome search, forcing the exploration of many branches

One of the reasons this second search is more problematic is that the query falls too close to the boundary of a cell. Whenever this happens, the circle traced around the query will intersect other cells, forcing us to visit them, in order to check for better candidates. The trouble is that, as dimensionality grows, chances are we will always fall close to several boundaries. Eventually the method will become prohibitively costly, as we are forced to visit a large proportion the tree.

One way to adapt the KD-tree to higher dimensionalities is to make it approximate. The strategy consists in exploring first the subtrees which have the corresponding regions nearer to the query, and stopping the search after a few regions have been explored. This technique is known as *best-bin-first* [Beis 1997] or *priority KD-tree search* [Arya 1996], and allows a trade-off be-

tween performance and precision. We are not guaranteed to always find the correct answers, but for moderate dimensionalities this method gives good results.

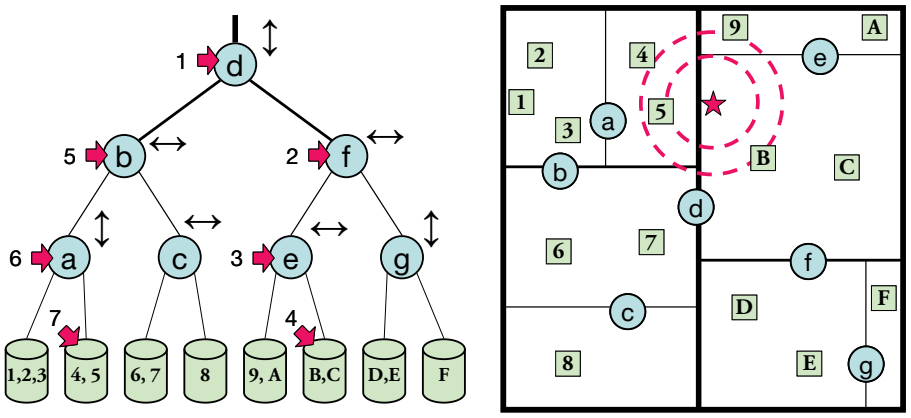


Figure 3.11: The best-bin-first strategy explores preferentially the most promising buckets

An example of best-bin-first search is shown in Figure 3.11. Note that, contrarily to Figure 3.10, we explore the left branch of "d" before the left branch of "e", because the query is closer to the region associated to the former. This way, we are able to avoid exploring one of the buckets and still get the correct answer.

Despite of its simplicity, the KD-tree has been widely used for kNN search and other spatial indexing operations, and many variants have been proposed to extend it.

The performance of the KD-tree depends a lot on the criteria used to choose the splitting dimension and the pivot. Early examples simply used a round-robin scheme to choose the dimension (accordingly to the height of the node on the tree) and the pivot as the median in that dimension. Current approaches try to choose the dimension where the data are more spread. Moore advocates that instead of using the median element, the pivot should be the element nearest to the average of the extremes of the dimension, to avoid creating regions which are thin hyperrectangles (approximately equilateral regions lead to better performance). Since this can create very unbalanced trees, a threshold can be used on the maximum allowed height after which one reverts to the median rule [Moore 1991].

The KDB-tree [Robinson 1981] is an adaptation which tries to make the KD-tree more dynamic and disk-friendly. It is an adaptation of the KD-tree inspired on the B-tree. The BKD-tree is a further extension which uses the logarithmic method to amortise the cost of insertions and to obtain a better occupation of the KDB-tree nodes by bulk loading the tree (see § 3.3.2).

Another variation is the BSP-tree [Fuchs 1980], which uses arbitrary plans, not necessarily parallel to the coordinate axis, to separate the subspaces. Though theoretically this flexibility allows for better separation of the data, the cost of the complex arithmetic does not compensate its benefits.

The variations LSD-tree and LSD^h-tree [Henrich 1989; Henrich 1998] were specially conceived for storage on disks. The LSD^h stores, in each region, the minimum bounding rectangle which contains the points in that region. This improves the performance by allowing ruling out more regions when searching the tree.

More recently, a randomisation scheme has been proposed for the KD-tree, where several trees are used with a somewhat random factor in the choice of the splitting dimension [Silpa-Anan

2005]. The idea is to avoid the situation where we have a fixed region that performs very poorly for some queries (see § 3.3.7).

The KD-tree (and its variants) requires a vector space, but does not require the dissimilarity function to obey triangle inequality. The dissimilarity must, though, be *monotonic on every component*, which means that:

$$\text{if } \forall i \neq j, |a_i - b_i| = |c_i - d_i| \text{ and } |a_j - b_j| < |c_j - d_j| \text{ then}$$
$$\Delta(a,b) < \Delta(c,d) \qquad \textbf{Eq. 3.13}$$

where a_i is the i[th] component of vector a, and so on.

The original KD-tree was conceived as a static data structure: further insertions and deletions are possible, but they will progressively degrade the performance of the index, as they are not taken into account in the region of the space. The KDB-tree tries to be more dynamic, but each insertion can be very costly, and the occupancy rate of the buckets can become quite low. The BKD-tree, which uses bulk-loading, is more promising, but, though the average, amortised, insertion is cheaper than in the KDB-tree, one particular insertion can be very expensive (see § 3.3.2, [Procopiuc 2003]).

All the implementations of the KD-tree suffer from the fact that many buckets must be read from the disk in order to obtain acceptable approximation. As the order in which the bucket will be read is unpredictable, a random i/o operation must be performed for each one of them. In § 4.1 we propose a more disk-friendly extension to the KD-tree, which makes at most one random i/o operation *per search*.

The KD-tree can be adapted to perform the $(1+\varepsilon)$-kNN search using the best-bin-first strategy. It can both enforce the approximation (measuring the ratio between the next nearest region to explore and the nearest candidate found so far and stopping when this ratio reaches $1+\varepsilon$) and inform it (stopping using any criteria — e.g., after a fixed number of buckets has been visited — and then computing the ratio). Different search operations can use different precisions, without the need of changing the index. For the random KD-trees, however, the precision control is not possible, because each individual tree is shallowly explored, and the next nearest region is not indicative of how thoroughly the space has been explored.

3.6 Study of case: the VA-file

The VA-file (**v**ector **a**pproximation file) [Weber 1997] divides the space into a grid of hyperrectangles by taking only a small number of bits for each dimension.

In order to build the VA-file, each dimension is assigned a small number of bits m_i and then partitioned in a way that, preferentially, populates equally each partition. Once the partitions are determined they are numbered and remain constant for the rest of the lifetime of the index. This will divide the space into 2^m hyperrectangular cells, where $m = \Sigma m_i$. The key for each point can then be computed and stored in the VA-File in a simple sequence, without any sort-

ing (Figure 3.12). The limits of the partitioning of each dimension should also be stored. Depending on how many bits are chosen to represent each dimension, the VA-file should be considerably smaller than the file with the points (normally 4 to 8 times smaller).

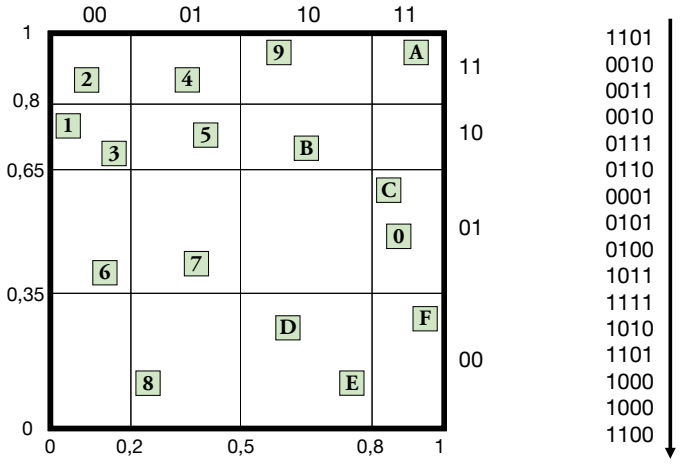


Figure 3.12: Building a 2D VA-File. Right: the partitioning of the space; left: the sequence of keys

In order to perform the kNN-search, first, in the *filtering step*, the entire file with the point approximations is sequentially scanned. Using the cell boundaries, the upper and lower bounds of the distance between the points and the query may be computed and used to eliminate the points which clearly do not belong to the nearest neighbours set. The rule is simple: if the lower bound of the distance of a prospective candidate is greater than the upper bound of the current *k*th best candidate, obviously this prospective candidate cannot be among the *k* nearest neighbours (Figure 3.13).

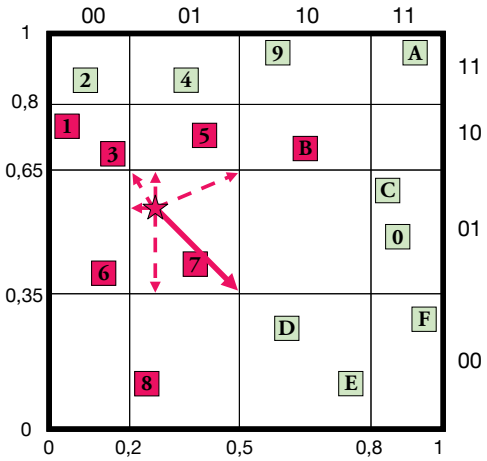


Figure 3.13: The filtering step during the search in a VA-file. The star represents the query, the thick arrow is the upper bound on the distance to best candidate so far, and the dashed arrows are the lower bounds on the distance to the candidates that cannot be discarded by the filtering rule

The filtering step will ideally filter out most points, keeping only a small subset of candidates. Now, we must access each one of the candidates and compute the exact distance to query, keeping the best *k* candidates. Not all candidates must be visited, though, only those whose lower bound distance is lower than the distance of the *k*th best already visited candidate.

The VA-file may be considered an accelerated sequential search. The filtering step is a complete sequential scan of all the points, but, since only a few bits per dimension are kept, the number

of blocks read from the disk is a fraction of those occupied by the point database. The access to each individual candidate is made by a random i/o read operation, which is expensive, so it is important to have a high selectivity in the filtering step, otherwise there will be no performance gains.

The VA-file was designed having in mind vector data using a p-norm distance. It is easy to see, however, that any metric distance will work.

Since the VA-file is simply an unordered sequence of approximations, insertions can be handled quite easily, by appending the new points at the end of the file. Deletions could be handled by marking the removed entries, though this could require an extra bit per entry.

The size of the VA-file is critical for the performance: it should be the smallest possible, since it will have to be completely scanned. For this reason, it is implied that the points appear in the same order in the VA-file and in the database: this prevents the need of storing pointers in the VA-file.

Once determined, the partitioning of the dimensions will not change. Thus, if the data distribution changes significatively during the lifetime of the index, performance may degrade. A technique of bulk-loading, like the logarithmic method may be used to solve this problem (see § 3.3.2).

The method, as described so far, is exact, in the sense that the neighbours found can be confidently considered the best. The method could be easily transformed into an approximate $(1+\varepsilon)$-kNN search, by adapting the filtering rules to reject any candidate whose lower bound distance is up to a $1+\varepsilon$ factor of the upper bound distance of the best candidate so far. It would be interesting to see how much the selectivity would improve with the increase of the approximation factor.

The VA$^+$-file [Ferhatosmanoglu 2000] is a variant in which the data are pre-processed using a principal component analysis. This allows reducing the dimensionality, and making the data more uniformly distributed. However, the pre-processing cost may hinder its use on very large databases.

The LPC-file [Cha 2002] is another variant which adds the local polar coordinates to the highest-significant bits on the index file. Two additional values are stored for each point: the angle between the point and the cell diagonal, and the distance between the point and the lower left corner of the cell. This will increase the size of the index file, but allow better selectivity.

The landmark file [Böhm 2000] sorts the points in the index file in a way that only a fraction of the file has to be scanned. The search operations are faster, but the index thus constructed is completely static.

3.7 Study of case: LSH

A *hash function* is a function that maps a large domain set into a small range set, in a way which the range values can be used to "identify" the data. It finds applications in cryptography, data storage (*hash tables*), error detection and correction and string searching.

The idea behind LSH (**l**ocality-**s**ensitive **h**ashing) is simple: to use hash functions which tend to have the same value when the points are close to each other and different values otherwise. The technique was introduced in [Indyk 1998] and improved in [Gionis 1999].

The implementation details of the method are complex. The steps are:

1. Devise a family of hash functions H, which obey certain locality-preserving properties when mapping the points to the identification keys;
2. Generate another family of hash functions G which are m-tuples of functions in H. Select randomly ℓ functions $g_1, g_2, \ldots, g_\ell$ from family G;
3. Set parameters m and ℓ to guarantee that, with high probability, points which have equal values in at least one of the g_i functions are near from each other and conversely;
4. Build ℓ hash tables (one for each g_i) with bin capacity c;
5. Use those tables to solve the (δ, ε)-neighbour problem (defined below);
6. Use the solution of the (δ, ε)-neighbour to solve the $(1+\varepsilon)$-kNN problem.

Figure 3.14 may help to visualise this chain of processes.

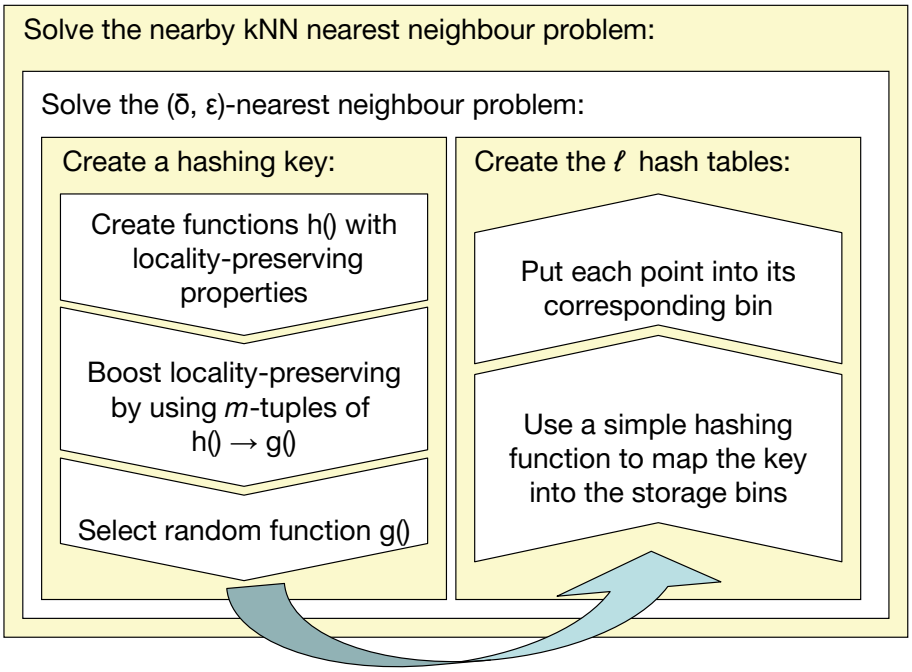


Figure 3.14: Solving the kNN search using LSH

3.7.1 Reducing the $(1+\varepsilon)$-kNN search to the (δ, ε)-neighbour search

The technique depends on the existence of an efficient way to reduce the $(1+\varepsilon)$-kNN search to the simpler (δ, ε)-neighbour search, also known as ε-PLEB (**p**oint **l**ocation in **e**qual **b**alls).

Definition 3.8 —The (δ, ε)-neighbour search:

- Given a domain D, a dataset B, a query q, and a dissimilarity function Δ as in Definition 2.1;
- The (δ, ε)-neighbour search will have one of the following results:
  (1) If there is a point in B less distant than δ from q, then the search returns YES and a point less distant than $(1+\varepsilon)\delta$ from q, i.e.:

$$\exists b \in B : \Delta(q,b) \leq \delta \rightarrow \text{returns (yes, } b') : \Delta(q,b') \leq (1+\varepsilon)\delta \,.$$

  (2) If all points in B are more distant than $(1+\varepsilon)\delta$ from q, then the search returns NO, i.e.:

$$\nexists b \in B : \Delta(q,b) \leq (1+\varepsilon)\delta \rightarrow \text{returns (no)} \,.$$

  (3) Otherwise (i.e., if there are points in the base whose distance to q is between δ and $(1+\varepsilon)\delta$, then the search either returns YES and one of this points, either returns NO.

■

In [Indyk 1998, § 3], the following simple reduction from the (1+ε)-1NN search to the (δ, ε)-neighbour search is offered:

1. Compute the smallest and the largest distance between any two points, respectively $r_{\min}$ and $r_{\max}$;
2. Compute a set of distances L such that $L = \left\{ r_{\min}, r_{\min}\left(1+\varepsilon\right), r_{\min}\left(1+\varepsilon\right)^2, \ldots, r_{\max} \right\}$;
3. Find the minimal element $\delta_{\min}$ of L such that ($\delta_{\min}$, ε)-neighbour search for q returns YES;
4. The answer to the (1+ε)-1NN search is the answer to the ($\delta_{\min}$, ε)-neighbour search.

This reduction has a small query time overhead of $\mathcal{O}\left(\log\log_{1+\varepsilon}\left(r_{\max}/r_{\min}\right)\right)$, because $\#L = \log_{1+\varepsilon} r_{\max}/r_{\min}$ and $\delta_{\min}$ can be determined by a binary search which performs $\mathcal{O}(\log \#L)$ (δ, ε)-neighbour search operations. However, the space overhead is of $\mathcal{O}\left(\log_{1+\varepsilon}\left(r_{\max}/r_{\min}\right)\right)$, because we have to build one index for each distance in L, in order to be able to solve efficiently all the possible (δ, ε)-neighbour searches. So, this reduction is only useful for moderate values of $r_{\max}/r_{\min}$.

A reduction with more constrained overheads is devised in [Indyk 1998, § 3], using a structure named ring-cover tree.

Gionis et al. argue that if the distribution of distances $q \in D, b \in B, \Delta(q,b)$ is more or less independent from the likely queries q, one can put a single value r in the set L and still obtain good results [Gionis 1999]. Note that doing so voids the assurance of the approximation error being (1+ε). It also induces the possibility of the search returning no points (if all points in the base are more distant than r to the query). In their experiments, this happens in less than 10% of the queries.

It is worth noting that all proofs presented in the original papers concern the search for the *first* (approximate) nearest neighbour. In [Gionis 1999], the proposed algorithm works for the general nearby kNN search (i.e., for values of $k > 1$), but no proof is offered that the approximation errors are guaranteed for the neighbours other than the first.

3.7.2 Creating the hash functions

Another premise of the methods is the possibility of building a family of locality-preserving hash functions.

Definition 3.9 —A family of locality preserving hash functions:

- Given a domain D and a dissimilarity function Δ as in Definition 2.1;
- A set of hash values V;
- The family of functions $H = \{\mathrm{h} : D \to V\}$ is *locality-preserving* or (δ, ε, p_1, p_2)-sensitive for Δ iff (given $\delta, \varepsilon \in \mathbb{R}_*^+$, $p_1 > p_2 \in [0;1]$) for all $d_1, d_2 \in D$:
  (1) $\Delta(d_1, d_2) \leq \delta \Rightarrow \Pr[h(d_1) = h(d_2)] \geq p_1$;
  (2) $\Delta(d_1, d_2) \geq (1+\varepsilon)\delta \Rightarrow \Pr[h(d_1) = h(d_2)] \leq p_2$.

■

To build an index based on those functions, one boosts their locality-sensitiveness by creating another family of functions $G = \{\mathrm{g} : D \to V^m\}$ such that each g associates a point x from the domain to a m-tuple $\mathrm{g}(x) = (\mathrm{h}_1(x), \mathrm{h}_2(x), \ldots, \mathrm{h}_m(x)), \mathrm{h}_i \in H$. One then chooses ℓ functions $\mathrm{g}_1(x), \mathrm{g}_2(x), \ldots, \mathrm{g}_\ell(x)$ at random from G. In practice, to build each one of the g_i, one can simply draw at random, with uniform probability, m functions from F with reposition. One will then create ℓ hash tables, one for each g_i, and insert the database points in them.

To perform the (δ, ε)-neighbour search for the query q, one retrieves the bins associated to each hash function $\mathrm{g}_i(x)$ on the corresponding hash table.

The parameters m and ℓ are chosen to guarantee that each one of the following two properties holds with a given probability:

1. Each point in the base less distant than δ from the query, will share with the query at least one of the ℓ keys, i.e.:

$$\exists b \in B : \Delta(q,b) \leq \delta \Rightarrow \exists i \in \{1, \ldots, \ell\} : g_i(q) = g_i(b) \qquad \textbf{Eq. 3.14}$$

2. The number of points in the base more distant than $(1+\varepsilon)\delta$ from the query and sharing with the query at least one of the ℓ keys is less than $\kappa\ell$ (for some arbitrary $\kappa \geq 2$), i.e.:

$$X = \{b \in B : \Delta(q,b) > (1+\varepsilon)\delta\}, Y_i = \{b \in B : \mathrm{g}_i(q) = \mathrm{g}_i(b)\} \Rightarrow$$
$$\sum_{i=1}^{\ell} \#(X \cap Y_i(q)) < \kappa\ell \qquad \textbf{Eq. 3.15}$$

Gionis et al. [Gionis 1999] demonstrate that, for a (δ, ε, p_1, p_2)-sensitive family H, and for $\kappa = 4$, property 1 above will hold with probability at least $1 - e^{-1} \cong 0,6321$, and property 2 will hold with probability at least 0,5, if the parameters are set at:

$$\ell = \left(\frac{n}{c}\right)^{\frac{\ln p1}{\ln p2}} \text{ and } m = \log_{1/p_2}(n/c) \qquad \textbf{Eq. 3.16}$$

However, in their experiments, they choose both values empirically, by testing a range of values on the dataset and choosing the optimum. For a database of 20.000 items and dimensionalities varying between 8 and 54, they take m = 700 and $\ell \in [1..10]$.

Indyk and Motwani propose a family H of hashing functions for indexing a Hamming space $\{0,1\}^d$ using the Hamming distance. They also propose another family for indexing the metric space composed of the power set of a finite set, using the *set resemblance measure*[*].

Gionis et al. [Gionis 1999] explain how the metric space $\{0,\ldots,max\}^d$ with the Manhattan distance can be embedded into a Hamming space of dimensionality $max \times d$, allowing the use of the Hamming hashing functions for indexing $\{0,\ldots,max\}^d$.

Datar et al. [Datar 2004] use p-stable distributions to create a family H of hashing functions for the space $\mathbb{R}^d$ using any p-norm distance with $p \in]0;2]$. Using this family, one can avoid having to embed the data into a Hamming space.

Since the underlying data structure is the hash table, this technique allows for efficient insertion and deletion of data. The hash table also allows for a fast disk-implementation, because each search on the table needs only one random-access operation.

The approximation is at the same time geometric and probabilistic, since the method will return with probability $1-\alpha$ the $(1+\varepsilon)$-nearest neighbours of the query. Using the implementation proposed by Gionis et al. the method fails on non-negligible fraction of the queries (between 1 and 10%). On those cases, the search returns with an empty set of points.

3.8 Study of case: MEDRANK and OMEDRANK

Apparently, Kleinberg was the first to propose the use of projections on random straight lines to perform approximate kNN search, developing a careful theoretical analysis to prove the bounds on pre-processing time, query time and approximation error [Kleinberg 1997].

A practical algorithm was proposed by Fagin et al., using *rank aggregation*, which is a way to combine the results obtained on each line to compute the answer to the kNN search [Fagin 2003].

3.8.1 Rank aggregation

Rank aggregation is a classic problem of voting systems, and consists of taking m different rankings of n candidates (given by different *voters*) and aggregating them in a single ranking, which must be representative of the global voters' preferences.

The aggregating rule should ideally satisfy certain criterions like the *Condorcet criterion* (if there is a candidate who, when compared in turn with each of the others, is preferred over the other, it should be ranked first), in order to produce expected results, where the global ranking cannot be "hijacked" by a few misbehaved voters.

---

[*] The resemblance measure between sets X and Y is defined as $\frac{\#(X \cap Y)}{\#(X \cup Y)}$.

Rank aggregation can be used to perform the kNN search efficiently in high-dimensional spaces. The idea is to consider each individual dimension as a voter, which will generate an independent ranking of the descriptors in the database. Those rankings will then be aggregated and the best k candidates will be kept.

Fagin et al. [Fagin 2003] consider that the rank aggregation is not only a heuristic for the standard kNN search, but it has also some advantages on its own. Where the elements do not fit any metric space because the features are categorical rather than numerical, or the numeric units are incompatible, the notion of rank aggregation may be more natural than that of nearest neighbours.

Kemeny [Kemeny 1959] proposed an aggregation mechanism which satisfies many desirable properties, including the Condorcet criterion [Young 1978]. Basically, given a set of input rankings, it produces the global ranking that minimizes the number of inverted pairs with the input rankings.

Or, more formally:

- Given a set of input rankings R_1, R_2, …, R_m;
- And the *Kendall tau distance* $K(X, Y)$ which measures the number of pairs (u, v) of candidates that are inverted on the two rankings (u is ahead of v on X and behind on Y, or vice-versa);
- The *Kendall-optimal aggregation* is the output ranking $\boldsymbol{G}$ that minimizes the sum:

$$\sum_{i=1}^{m} \mathrm{K}(G, R_i)$$

Producing a Kendall-optimal aggregation is, however, an NP-complete problem, for as few as 4 input rankings and over. The alternative is to produce a *Footrule-optimal aggregation*, i.e., one that minimizes the sum of the differences of the ranks.

Formally:

- If we define $R(j)$ to be the rank of the object j in the ranking R;
- The *Footrule-distance* between G and R is defined as:

$$\Phi(G, R) = \sum_{j=1}^{n} |G(j) - R(j)|$$

- The Footrule optimal aggregation minimizes the sum:

$$\sum_{i=1}^{m} \Phi(G, R_i) = \sum_{i=1}^{m} \sum_{j=1}^{n} |G(j) - R_i(j)|$$

It is known that the total Footrule distance is within a factor of two of the Kendall tau distance, i.e.:

$$\mathrm{K}(G, R) \leq \Phi(G, R) \leq 2\mathrm{K}(G, R)$$

which means that the Footrule optimal aggregation is a good heuristic approximation for the Kendall tau aggregation.

The Footrule optimal aggregation may be, in many cases, given by the median of the ranks. It is easy to see that if $M(j)$ = median of R_1, R_2, …, R_m for candidate j, and if the medians are all different, the ranking given by the median of the input rankings is a Footrule optimal aggrega-

tion. Even if the medians are not all different, the principle that the median rank optimizes a notion close to the Footrule optimal aggregation remains valid [Dwork 2001; Fagin 2003].

3.8.2 Performing kNN search with rank aggregation

The MEDRANK algorithm is a clever way of finding out the top candidates of a Footrule-optimal aggregation without having to examine all the input data. In fact, the amount of data examined in each voter should be small when compared to the size of the database.

To do the kNN search with MEDRANK, one starts by projecting all data in a small number of random straight lines. A sorted list is created for each line, and the data are sorted accordingly to the position on the line. This pre-processing is done offline, once.

In order to answer a query, the MEDRANK will project it in the same straight lines and find the elements nearest to the projections of the query in the sorted lists. Two cursors *up* and *down* will be created in each list. At each iteration of the algorithm, the cursor pointing to the element nearest to the query (in the line) is chosen, and it is added to a table. If the element is already in the table, its vote count is incremented.

The first element to have a number of votes equivalent to more than half the lines, is considered the *first nearest neighbour*. The search continues until k elements have been thus voted.

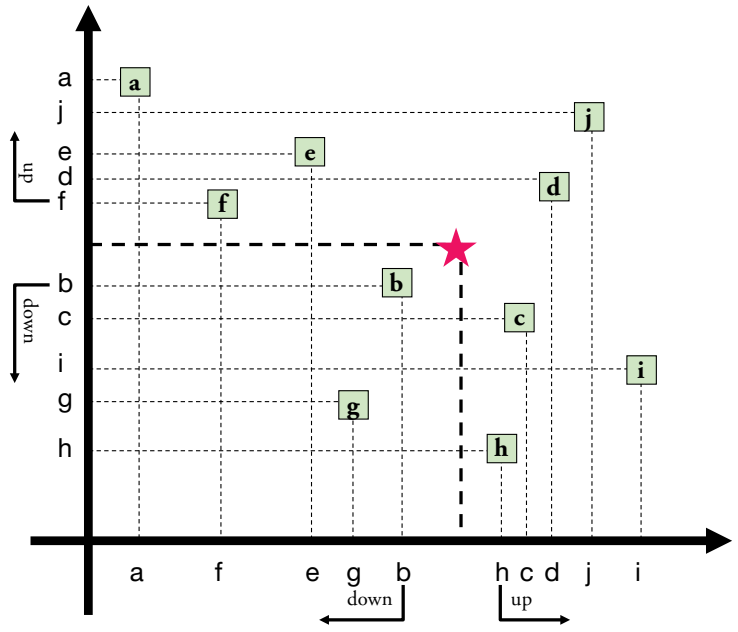


Figure 3.15: kNN search using MEDRANK, the star represents the query and the figures in the squares, the elements of the database. On each axis the cursors *up* and *down*, keep track of the next elements

We illustrate the MEDRANK algorithm in Figure 3.15. After all the database elements (the letters in the circles) have been projected on the two random lines (the two thick arrows), we are ready to process the query (represented by the star). The first element to be added in the horizontal line is "g" (for it is the nearest to the query in the line) then in the vertical line, "a" is added. Back to the horizontal line, "a" is added. Now "a" has two votes, i.e., more than half the number of lines. It is thus declared the nearest neighbour. The process continues on the vertical

line, where "e" is added. Then "b" on horizontal line, "c" on vertical line, "c" again on horizontal line ("c" is declared the second nearest neighbour), "b" on vertical line ("b" is declared the third nearest neighbour) and so on…

A small variation of the algorithm, named OMEDRANK, adds both elements in the cursors *up* and *down* in the same iteration, instead of adding the one which is the nearest to the query. This is to save the CPU time spent on comparisons. The complete description of the algorithms may be found in [Fagin 2003].

The MEDRANK method presents several advantages over other kNN methods, mainly:

- The *sorted list* is the mainly used data structure, and it can be handled quite graciously by the B-tree, resulting in a disk-storage friendly method;
- Because of this, insertions and deletions can be handled easily;
- The offline pre-processing time is reasonable, consisting just of sorting operations;
- All access to the data in the sorted lists are sequential, resulting in savings in disk operations, where random access is much more expensive;
- The actual distance function does not have to be computed, resulting in savings in CPU time.

Unfortunately, MEDRANK (and OMEDRANK) does not show good results for large databases in very high dimensionalities [Lejsek 2005]. A simple observation shows that for those databases, the probability that every element will be near to the query in at least one of the dimensions is close to 1, making the probe depth prohibitively elevated.

In particular, if each dimension assumes a limited set of discrete values (as is often the case), with great probability, each element will be exactly equal to the query in at least one of the dimensions, making the order into which the elements will be chosen highly aleatory, ruining the efficacy of the method.

In the original paper by Fagin et al. MEDRANK and OMEDRANK are used with the Euclidean distance. The use of other dissimilarities depends on how representative the distance on a straight line projection is for that particular dissimilarity in the space.

The choice of the number of projections determines the quality of the results. The approximation is at the same time geometric and probabilistic. Fagin et al, using a previous lemma by Kleinberg, prove that if the number of projections is $\mathcal{O}(\varepsilon^{-2}\log n)$, then their method will find the (1-ε) approximate nearest neighbour with probability at least $1-1/n$ [Fagin 2003, § 2.1.2; Kleinberg 1997, § 2]. This result, however, is only of theoretical interest, because it does not inform about the hidden constants.

In practice, though there is an obvious correlation between the number of projections and the quality of the results, this method does not provide precision control for the approximation.

A system based on OMEDRANK was used in an image-retrieval system for a database of conservation/restoration reproductions [Philipp-Foliguet 2006].

3.9 Study of case: PvS

In order to solve the problems of MEDRANK, a method has been proposed which uses multiple simultaneous straight lines to index the elements. This method, named PvS (**p**rojection versus **s**egmentation) keeps the elements together only if they are close to each other in several straight lines *at the same time* [Lejsek 2005; Lejsek 2006]. Since the probability of this happening without the elements being actually close to each other is low, the scheme avoids the two main problems of MEDRANK:

- Voting for the element $\boldsymbol{x}$ before the element $\boldsymbol{y}$ in a line, when $\boldsymbol{y}$ is actually closer to the query;
- Having huge probe depths, because every element in the database gets to be voted by at least one of the dimensions.

PvS works by projecting all the elements in a random line, and then dividing the data into segments. Those segments are then projected into *another* line. The process may be repeated until the number of elements in each segment is manageable both in terms of searching and of disk storage. In order to avoid boundary effects, overlapping segments are created in each step of the process. The complete set of segments composes a tree which is used as a voter. Several voters are necessary to implement the scheme.

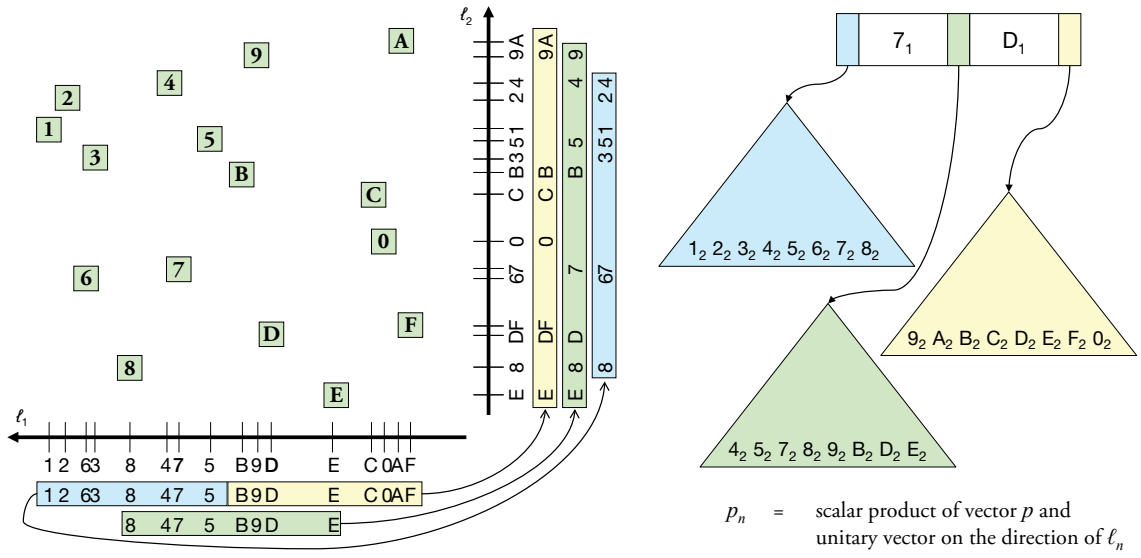


Figure 3.16: The construction of an individual voter in PvS. Points are projected on line 1, segments are created and then reprojected in line 2. The associated ternary tree is shown on the right. Note that in this case, the tree has only two levels: the root and the leaves, which are, themselves, B⁺-trees

The construction of a voter works as following:

1. A set of random projection lines is chosen, one line for each level of the voter's tree. The lines are chosen with the restriction that they must be nearly orthogonal to each other (the cosine of their angle must be less than 0,1);
2. Starting with line $\ell = 1$, we project all points on line ℓ, grouping them into a single, large segment;
3. We take each segment we have created so far and divide them into m equally populated segments, accordingly to the position of their projections on line ℓ;

4. We create m–1 overlapping segments, between those created in step 3. The overlapping segments will have as many elements as those create in step 3, half from each of the segments they overlap;
5. We then *reproject* each segment into line ℓ+1;
6. If line ℓ+1 is not the last line, we set $\ell = \ell + 1$ and repeat steps 3–6.

Each segment is associated to a node on the voter's tree. Since each segmentation operation (step 3) creates $2m$–1 subsegments, this will be the order of the tree.

The non-leaf segments store a set of $2m$–2 decision values, which help to decide which of the subtrees will be explored in the search operation: these are the values of the projections in the current line ℓ, of the points which are midway between overlapping segments (e.g., points "7" and "D" in Figure 3.16).

The leaf nodes are sorted lists with the values of the projections of the points on the last line, associated to the point identifiers. In the original algorithm, those sorted lists are stored in B^+-trees, on the disk.

The process is illustrated in Figure 3.16 for a tree of 2 levels (thus, 2 projection lines) and $m = 2$.

In the construction of a voter, there is a compromise between the height and the order of the tree: in order to obtain the desired segment size one can choose between shallow trees of higher order, or deep trees of lower order.

For example, if one has 64.000 points and wants segments of 1.000 points, using a uniform segmentation strategy, one can either have a tree of height 7 and order 3 ($m = 2$, thus breaking the point set in halves 6 times); have a tree of height 4 and order 7 ($m = 4$, breaking the points in quarters 3 times), or a tree of height 3 and order 15 ($m = 8$, breaking the points in eighths twice).

To perform the search, at least two voters are needed, and three are recommended. For each voter, the most appropriate leaf node will be sought, using the stored decision values. For each level of the tree, the query is projected in the corresponding projection line and the value of the projection compared to the decision values. The subtree chosen is that for which the query falls between the two consecutive decision values. E.g., in Figure 3.16, if the projection of the query on ℓ_1, falls to the right of "7", the right subtree is chosen; if it is between "7" and "D", the middle subtree is chosen, if it falls to the left of "D", the left subtree is chosen. The objective is always to choose the subtree where the query is the furthest from the boundaries.

Once a leaf segment is determined for each voter, the OMEDRANK algorithm (§ 3.8) will be run on them. The projection of the query on the last line of each voter will be used to look for the nearest points in each sorted list. As usual, the first point to appear in more than half the lists is considered the nearest neighbour.

The main advantage of PvS, when compared to OMEDRANK, is that each voter uses multiple straight lines in order to determine the proximity of the points. Since the points are projected into one line, segmented using the values of this projection, and then reprojected on a *different* line, the only way two points will fall in the same segment and have similar projection values, is if they are close to each other in *both lines*. This avoids the main problem of OMEDRANK: points that are far apart in the space appearing close on the projected lines.

The segmentation, however, introduces arbitrary discontinuities between the points. If we allowed the query to fall in the boundary of a segment, the neighbouring segment would have to be explored. The overlapping segments exist to avoid this situation.

The overlapping segments, however, are also a disadvantage. The management of insertions and deletions become delicate, as a single modification may span across several segments. Contrasting with the simplicity of the OMEDRANK, PvS has much more complex storage data structures.

Besides, the overlapping segments impose an overhead in terms of disk storage, which is exponential to the height of the tree. For a tree with height h and m non-overlapping segments per level (resulting in a total of $2m-1$ segments per level), we will have an overhead factor of:

$$\left(2-\frac{1}{m}\right)^{h-1} \qquad \textbf{Eq. 3.17}$$

The pre-processing time will also be proportional to this factor. This means that the trees have to be rather shallow in order for the method to be feasible.

Any dissimilarity which works for OMEDRANK should also work for PvS.

PvS has been recently improved and renamed NV-tree (Nearest Vector tree) [Lejsek 2008]. The new version introduces two alternative partitioning schemes: one which takes into account the distance between the data points (instead of the number of points per segment), and a hybrid approach (which avoids generating trees too unbalanced). It also provides algorithms for insertions and deletions through an amortised repartitioning of the leaf nodes, after the number of changes exceeds the capacity of a "change buffer".

Neither PvS nor NV-tree provides precision control for the approximation.

3.10 Study of case: Pyramid trees

The pyramid technique [Berchtold 1998] is based on a mapping between the d-dimensional space and 1-dimensional keys. This allows using a sorted list to handle the queries (the authors use the B^+-tree).

The idea is that trying to partition the space in a balanced way among the dimensions is unfeasible for high-dimensional data, because the geometry of the partitions will be such that most of the queries are likely to intersect most of the partitions. Instead, the authors advocate the partitioning "like the peels of an onion". The idea is to have $2d$ pyramids (for d dimensions) which share the top at the centre of the space, and whose bases are the $d-1$-hyperplanes which limit the space.

Considering the space normalised to the unit hypercube, every point will be associated to a key: the number of the pyramid in which they are located (the pyramids are simply numbered 0, 1, … $2d$) plus the "height" of the point, which is the difference between the coordinate of the

point and the top of the pyramid in the dimension orthogonal to the base of the pyramid (Figure 3.17).

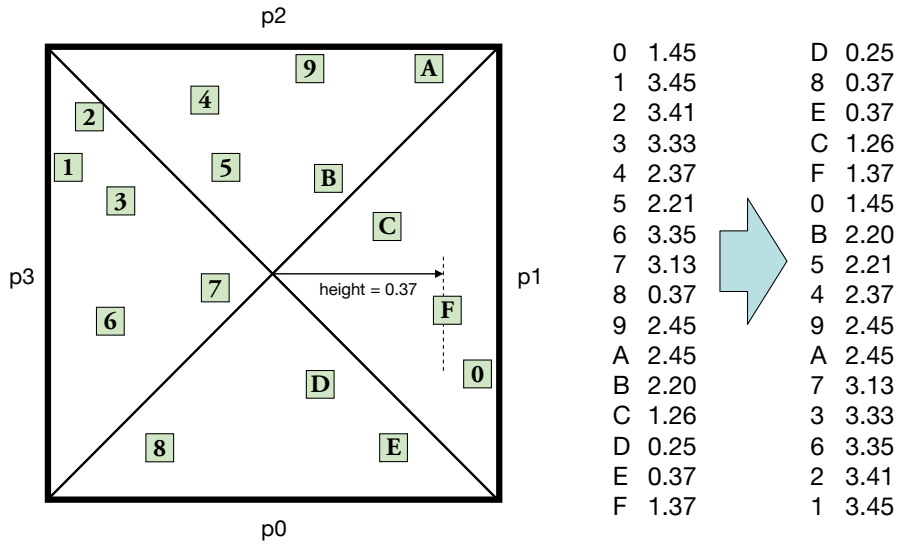


Figure 3.17: Building the pyramid tree. In this 2-dimensional example, there are four "pyramids": p0, p1, p2 and p3. The points are associated to a key which is the number of the pyramid plus the "height" of the point. The points are then stored in a B$^+$-tree, sorted by this key

The pyramid technique was developed having hypercubic range queries in mind. The argument for the "onion peel" partitioning can be convincingly illustrated in this case. Suppose we have a million 30-dimensional data points, uniformly distributed in the unit hypercube. If we want to use balanced partitioning, where the criterion for the choice of the splitting dimension is the spread of the data, it is easy to see that some dimensions will be partitioned once (no more than 20, since this will already create 2^{20} = 1.048.576 partitions) and the other dimensions will never be partitioned at all. Thus the partitions will be all hyperrectangles where the length in at most one of the dimensions will be 0.5 and the length in all other dimensions will be 1. Now, consider that we want a hypercubic range search to select 10 points. Because of the curse of dimensionality, and of the effects described in § 3.2.3, this turns out to be a hypercube of side 0.68. Clearly, no matter where we position this query, it will intersect all the partitions (Figure 3.18).

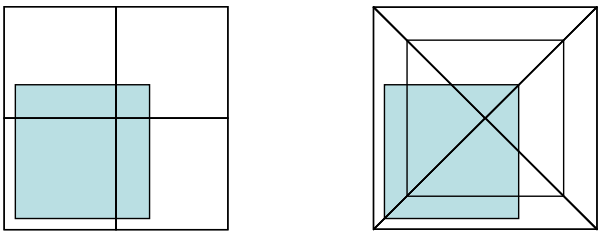


Figure 3.18: In balanced partitioning (left) most of the partitions will tend to intersect large range queries; using the Pyramid technique (right) this can be avoided. The effect is more dramatic for higher dimensionalities

The argument holds for the kNN search if we consider that there is an implicit range query whose size is unknown *a priori*. The queries will be hypercubic only if we use the Chebychev distance, but it is easy to see that the hyperspheric query induced by the Euclidean distance would also intersect most of the regions, for a large enough radius.

Range searching in pyramid trees is a complex operation. First, one has to determine which pyramids the query region intersects. Then, the interval of "heights" within each pyramid has to be computed. One then reads all the points within each interval and checks weather the points are effectively inside the query range. Zhang et al. [Zhang 2004] extends the technique to allow the kNN search by using a hypercubic range around the query point. The range is increased gradually until enough points are found. They conceived an algorithm which preserves information about which pyramids (and which nodes of the B⁺-tree in each pyramid) have already been visited, in order to avoid searching repeatedly the previous query ranges.

Some extensions to the basic pyramid tree have been proposed.

Berchtold et al. propose the *extended pyramid tree*, which has the vertices of the pyramids on the centre of the data (taken as the point whose coordinates are the median of the correspondent coordinate of the data). This allows distributing the points more uniformly among the pyramids [Berchtold 1998].

Zhang et al. have proposed the P⁺-tree, a mixed approach for clustered data, where the start by doing a conventional (balanced) segmentation of the space, in order to separate the clusters, and then build a pyramid tree for each segment. This approach, however, requires separating the data into clusters, which is expensive [Zhang 2004].

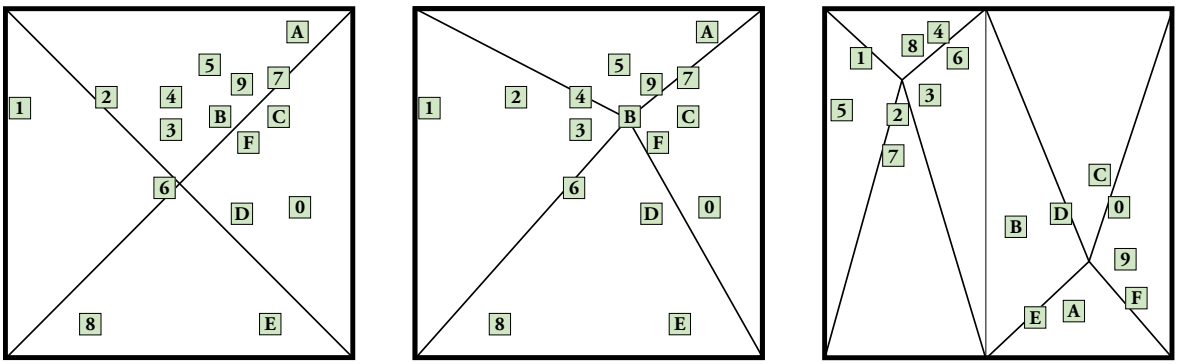


Figure 3.19: The pyramid tree (left) performs badly for skewed data; the extended pyramid tree (centre) puts the vertices of the pyramids on the centre of the cluster; The P⁺-tree creates one pyramidal partition for each cluster

The pyramid tree allows for easy manipulation of dynamic indexes, since it is based on one-dimensional B⁺-trees. The extended variant and the P⁺-tree will tolerate insertions and deletions, as long as the data distribution doesn't change significatively, otherwise the performance of the index will degrade.

The pyramid tree (and its variants) is an exact search, in the sense that the neighbours found can be confidently considered the best. The method could be transformed into an approximate $(1+\varepsilon)$-kNN search, by adapting the filtering rules to not discard any pyramid which is at more than a $1+\varepsilon$ factor distant than the best candidates so far. This would avoid making random accesses to read pyramids which have little chances of improving the results.

3.11 Study of case: the space-filling curves family

3.11.1 Space-filling curves

Fractal geometry shows many unexpected results: the concept of "fractional" dimensionality, the existence of figures of finite area but infinite perimeter, and the possibility of relatively simple rules generating very complex ("chaotic") sets. Not less surprising is the existence of continuous mappings from the unit interval $[0; 1]$ and any unit hypercube $[0; 1]^d$. Simply stated, this shows that there exist continuous curves which completely cover any d-dimensional hypercube.

When Georg Cantor (1845–1918) demonstrated that the unit interval has the same cardinality of any finite-dimensional manifold (e.g. a unit hypercube) [Cantor 1878], Giuseppe Peano (1858–1932), seeking to illustrate this counter-intuitive result, proved that there is a densely self-intersecting curve which passed through every point of the unit square [Peano 1890]. Later, many such curves were discovered and described (Figure 3.20).

Most space-filling curves are constructed by a recursive procedure, in which the space is progressively divided in smaller cells, which are then traversed by the curve. In those cases, it useful to define the concepts below:

- *Motif*: in a recursive space-filling curve, the motif is the building block of the curve, which is subsequently repeated in the progressive refinements of the curve;
- *Order* (m): is the number of iterations which have been carried out in the infinite series of a space-filling curve, i.e., the degree of refinement obtained so far. The expressions "m-order curve", "mth-order curve" and "curve of order m" are all equivalent. We will also say that the actual space-filling curve (the one to which the series converges) is of "infinite" order;
- *Grid*: as we iterate in the process of constructing a recursive space-filling curve, it's useful to consider the division of the space into a regular lattice, whose cells contain the motif. At each iteration, this lattice is further subdivided;
- *Approximation*: a space-filling curve of a given *finite* order is said to be an *approximation* of the named space-filling curve. Though it doesn't pass through every point of the space, it passes through each cell in the grid;
- *Embedding dimensionality* (d): is the dimensionality of the space that the curve is filling, when we talk about a d-dimensional curve, we mean that the curve fills a d-dimensional space.

The dimensionality deserves further clarification. When we take the actual space-filling curve, of infinite order, its (Hausdorff-Besicovitch or "fractal") dimension is equal to the dimension of the space it fills. Rigorously speaking, the dimension of all the finite approximations is strictly equal to one. Informally, however, we will talk about 2D, 3D or d-D curves, even when referring to the approximations.

For an in-depth introduction to the subject of the space-filling curves, their mathematical properties and the proofs of the most important theorems, see [Sagan 1994].

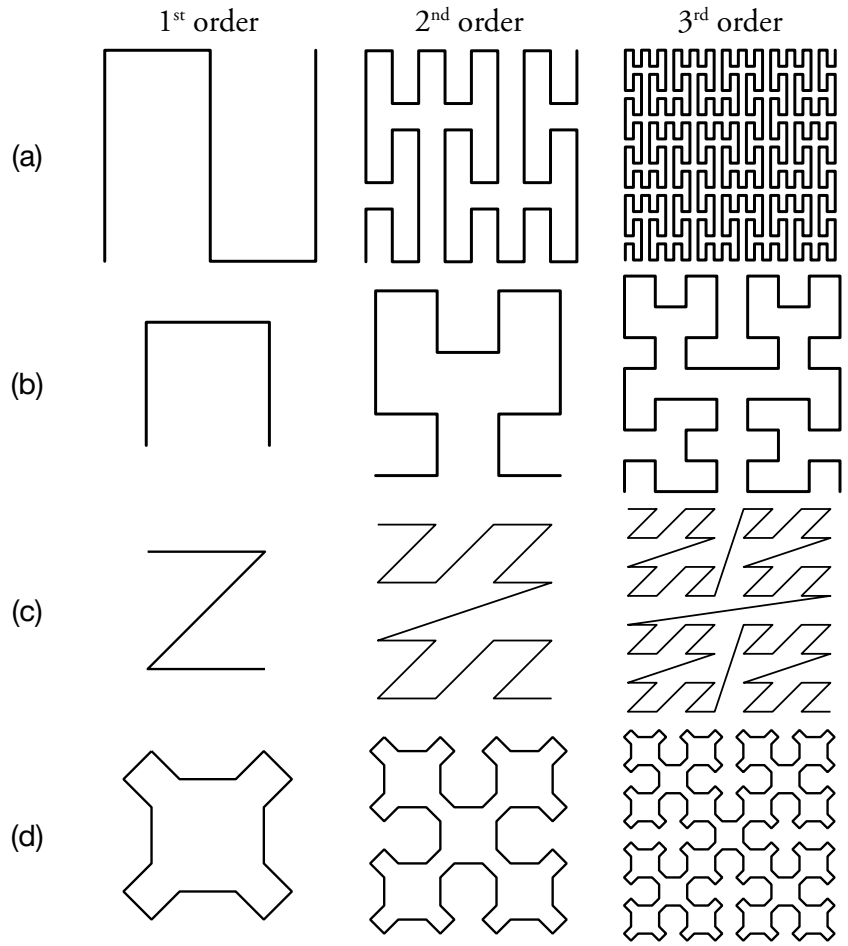


Figure 3.20: Some recursive 2D space-filling curves. a) Peano*; b) Hilbert; c) Lebesgue or "Z-order"; d) Sierpiński. Three iterations of each curve are shown: the actual space-filling curve is the infinite limit of those iterations

3.11.2 kNN search using space-filling curves

The use of space-filling curves to perform the kNN search in multidimensional spaces is not new. It is difficult to trace reliably who conceived the idea, but the earliest reference we could find is by Morton, who used the Z-order curve to index two-dimensional spatial data [Morton 1966]. The concept seemingly disappeared for a long time, until it resurfaced in two independent works by Tropf and Herzog†, who devised a way of using the Z-order curves and dynamically balanced trees to perform range queries, and Tamminen, who used it in conjunction with an extendible hashing function [Tamminen 1981; Tropf 1981].

---

* There is a great deal of ambiguity in the literature about the geometric interpretation of the original Peano curve, as the demonstrations in [Peano 1890] are entirely analytical. The fact that the term *peano curve* is sometimes used as a synonym for any space-filling curve exacerbates the incertitude. The interpretation shown here is the one of Moore [Moore 1900].

† Tropf and Herzog, however, cite an earlier publication of Bentley, who in his turn makes reference to an unpublished suggestion by McCreight "that the keys be 'shuffled' together bitwise" in order to make associative search [Bentley 1975].

It is interesting to note that those authors make no reference to space-filling curves, but instead call the idea *bit shuffling*, *bit interlacing* or *bit interleaving*. Due to the fact that the Z-order is obtainable by interleaving the bits of the individual spatial coordinates, the notion of a curve is implicit in those works.

Apparently Faloutsos was the first author to explicitly refer to the concept of space-filling curves. He was also the first to suggest that other curves could perform better than the Z-order, first proposing the Gray-code curve and then the Hilbert curve [Faloutsos 1986; Faloutsos 1988; Faloutsos 1989].

Skubalska-Rafajłowicz and Krzyżak make a seemingly independent development of the use of the curves to perform kNN, using the generalised form of the Sierpiński curve [Skubalska-Rafajłowicz 1996].

All the methods so far are conceptually very simple. They map the high-dimensional points in the curve and then perform a one-dimensional similarity search, which is straightforward, using the obtained extended-keys. The hypothesis is that points that are near to each other in the curve always correspond to points that are near to each other in the space, so it is a good heuristic to take the nearest points in the curve as the nearest points in the space.

Unfortunately, the converse of the hypothesis is not true, in the sense that near points in the space are not always near in the curve. This is because of the boundary effects, which tend to put very far apart points in certain regions of the curve. The matter is seriously aggravated as the dimensionality increases (Figure 3.21).

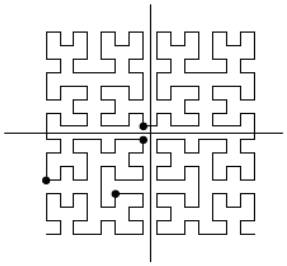


Figure 3.21: The problem with boundary effects on the space-filling curves. The points in the centre of the space are further apart in the curve than the points in the lower-left quadrant

In order to conquer the boundary effects, Megiddo and Shaft suggest the use of several curves at once, hoping that at least one of them will not pose boundary problems for any given query. They pose the idea in a very general way, without describing which curves should be used and how they should be made diverse [Megiddo 1997].

Shepherd et al. develop on this idea, by specifically recommending the use of several identical Hilbert curves where different copies of the data points are mapped, after random transformations (rotations and translations) [Shepherd 1999]. Whether or not the set of transformations could be optimised, is left unanswered by the authors.

Finally, Liao et al. solve the problem of choosing the transformations, by devising the necessary number of curves and the optimal set of translations to obtain a bounded approximation error in the kNN search. They also calculate the upper bound of this approximation [Liao 2001].

Those computations, while of theoretical interest, give little practical guidance for two reasons: first, the number of curves necessary to guarantee the bounded error is prohibitively high; second, only the upper bound of the error is computed, ignoring all constant factors.

Working independently from those three works, Pérez and Vidal also prescribe the use of multiple curves, using the transformation of the points before computing the extended-keys [Pérez-Cortés 1998].

The method we propose, multicurves is also based on the use of multiple Hilbert curves, but with the important difference that each curve is only responsible for a subset of the dimensions of the data (see § 4.3).

A depart from those methods was suggested recently by Mainar-Ruiz and Pérez-Cortés. Instead of using multiple curves, they propose using multiple instances of the same point in only one curve. Before inserting those instances in the curve, the algorithm disturbs their position, to give them the opportunity of falling into different regions of the curve. In that way even if the query falls in a problematic region, chances are it will be reasonably near to at least one of the instances [Mainar-Ruiz 2006]. We gain two advantages: first, we only have to manage one sorted list structure; second, when performing the search, we can do less random accesses. The main disadvantages are the difficulty in controlling the optimal number of representants for each point. Also, the method ignores that some regions in the curve are much more problematic than others, and simply associate each point to the same number of representants.

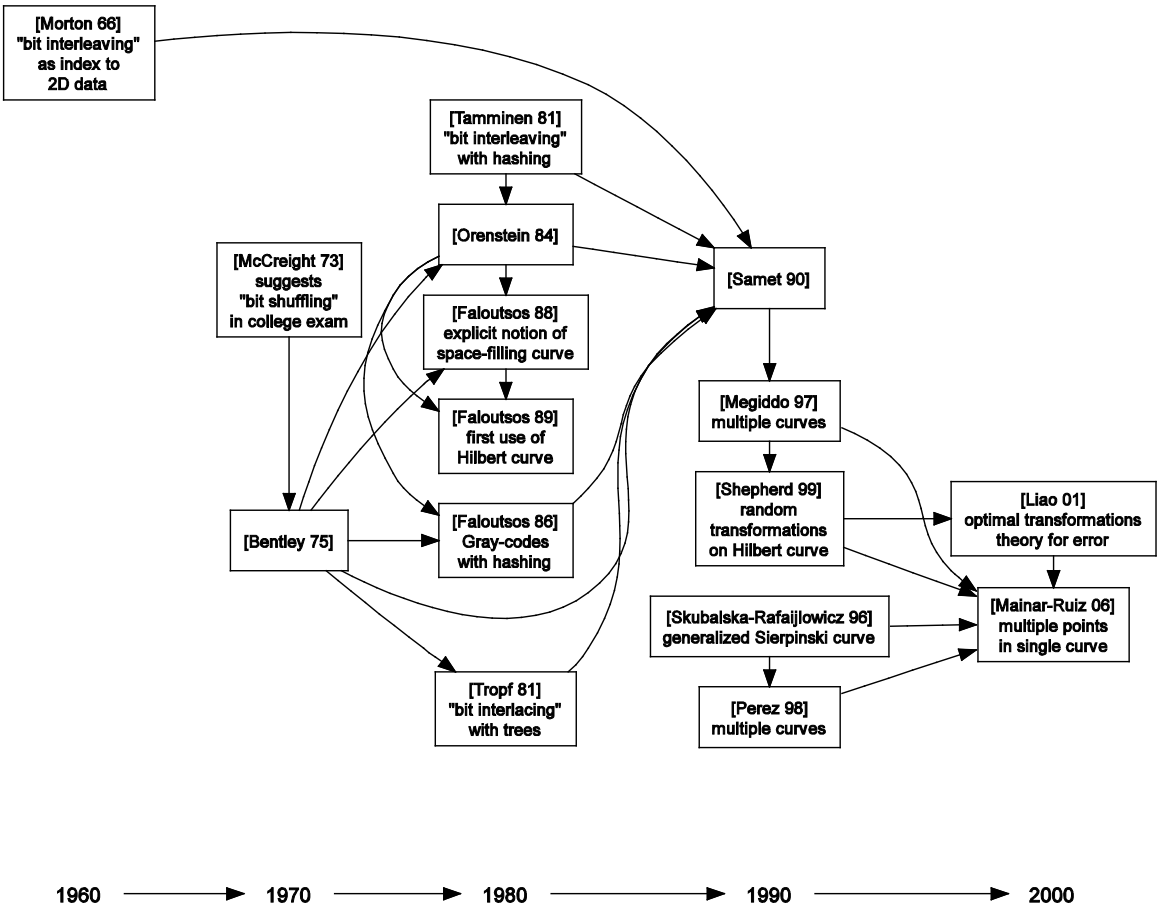


Figure 3.22: A genealogy of the ideas on spatial query using space-filling curves, based on citations. The original contributions are highlighted

A tentative genealogy of those ideas, using the citations as evidence, is shown in Figure 3.22. For an in-depth account of the bibliography on the subject up to the late 1980's, see [Samet 1990, pp. 105–109].

3.11.3 The choice of the curve

In comparison with other space-filling curves, like the Peano curve, the Z-order curve or the Gray-code curve, the Hilbert curve has better clustering properties, which is important for the kNN search and other applications (see [Moon 1996]). Other important properties of the curve, which resulted in its widespread application, are its "fairness" (symmetric behaviour on all dimensions) and the absence of distant jumps [Mokbel 2003].

Asano et al. have proposed a specially tailored space-filling curve for the task of range queries. Unfortunately, the absence of efficient, iterative algorithms to do the mapping between the space and this curve limits its applicability [Asano 1997].

Another interesting alternative is offered by the generalised Sierpiński curve $(p,1)$, which is highly symmetrical [Sierpiński 1912; Skubalska-Rafajłowicz 1994]. When using a closed curve, however, care should be taken to choose the point where the extremes of the interval "connect". Otherwise, points which are very near in the space will receive the further possible extended-keys in the curve.

3.11.4 The Hilbert curve

The Hilbert curve is the infinite limit of a family of recursive self-similar curves which are constructed hierarchically: in the plane, the second-order curve is composed of four first-order curves; the third-order curve is composed of four second-order curves, and so on [Hilbert 1981]. Ignoring rotations and symmetries, there is only one 2D curve (Figure 3.20b).

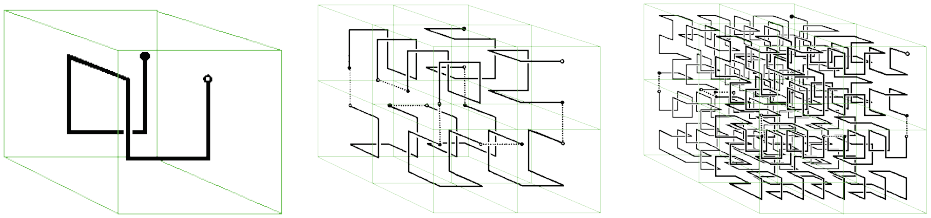


Figure 3.23: Three iterations of a 3D Hilbert Curve [Gilbert 2006, reproduced with permission]

Though the Hilbert curve can be generalised for hypercubes of any dimensionality (Figure 3.23), its formal definition has been somewhat elusive until the theoretical framework proposed by Alber and Niedermeier, for there are many different possible curves. Each curve can be uniquely described by the choice of the motif and by the orientation chart, which allows passing from the first-order to the second-order curve (Figure 3.24). Alber and Niedermeier describe which choices lead to well-formed, continuous, curves and compute how many possible curves exist for each dimensionality [Alber 1998].

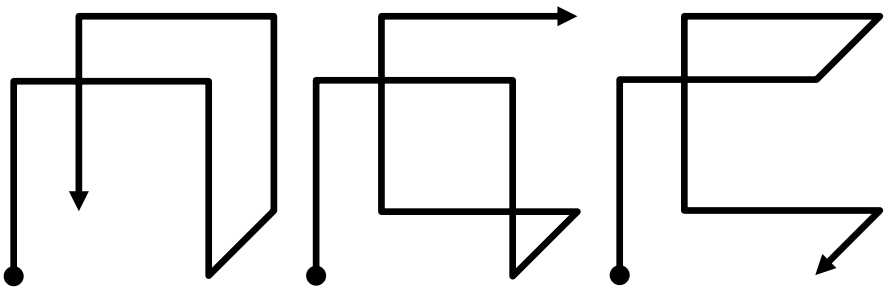


Figure 3.24: Three possible motifs for the 3D Hilbert Curve

3.11.5 Mapping between the space and the Hilbert curve

In order to use the Hilbert curve in any intended application, it is necessary to compute a mapping between the coordinates in the hyperdimensional space and the one-dimensional ordinal position in the curve (which we will call *extended-key*).

We normally use a finite order approximation instead of the actual Hilbert curve. The choice of the order depends on the quantisation of the dimensions of the hyperspace being considered. If each one of the d dimensions is represented by a number of m bits, then the order necessary will be exactly equal to m, resulting in a grid composed of m^d hypercubes. The resulting extended-key in the curve will need at most $m \times d$ bits to be represented.

Several mapping methods exist. The simplest one recognise that the Hilbert curve can be represented by a 2^d-ary tree of height m. Mapping between the space coordinates and the extended-key is done by traversing the tree from root to leaf. Unfortunately the size of the tree grows exponentially both with the order and with the dimensionality of the curve, which makes this method impractical.

Bially was the first to propose the use of finite-state machines to do the mapping [Bially 1967; Bially 1969]. Observing that the tree has many identical nodes, it is possible to represent it as a finite-state machine, in which each type of node corresponds to a state. Unfortunately, the size of the states in the machine is also exponential to the dimensionality of the curve (but not to the order), and even if it grows much slower than the tree, it still limits the application of the method to rather low-dimensional curves.

Bially describes how the states of the finite-state machine can be derived from a state generator table. Faloutsos then suggests this table to be used without storing the states themselves [Faloutsos 1989]. The problem again is that the generator table also grows exponentially, though more slowly than the states.

To overcome those difficulties, there are algorithms which use a constant amount of memory, and perform the mappings by direct calculation. Some of those algorithms are recursive and have an exponential worst case, which, again, limits their applicability.

Fortunately there is an algorithm which uses little memory, is iterative, and can map any curve using only $\mathcal{O}(md)$ bit operations [Butz 1971]. The original algorithm proposes only the mapping from the extended-key to the space coordinates, but the inverse mapping is not difficult to deduce, and was described in [Lawder 2000]. The algorithms are composed of complex bit operations, and provided without any justification of how those operations relate to the curve.

In our description of the algorithms, we try to use a notation consistent with the one used in [Butz 1971] and [Lawder 2000], using different conventions only where theirs clashes with the ones we have been using through this text.

First, some definitions:

- d: the number of dimensions in the space (n in the original articles);
- m: the order of the curve passing through the space;
- i: the iteration of the algorithm, in the extended-key $\{1,\ldots,m\}$;
- r: the extended-key in the Hilbert curve, expressed as a number of $d \times m$ bits;
- ρ_j^i: a bit of r (i.e., $\rho_j^i \in \{0,1\}$), such that $r = \rho_1^1\rho_2^1\ldots\rho_d^1\rho_1^2\rho_2^2\ldots\rho_d^2\cdots\rho_1^m\rho_2^m\ldots\rho_d^m$;
- ρ^i: the ith binary word of d bits of r, $\rho^i = \rho_1^i\rho_2^i\ldots\rho_d^i$;
- a_j: the jth coordinate of the point in space whose extended-key in the curve is r, expressed as a number of m bits;
- α_j^i: the ith binary digit of the coordinate a_j, such that $a_j = \alpha_j^1\alpha_j^2\ldots\alpha_j^m$;
- *principal position*: is the position of the last bit different from the rightmost bit, and if all bits are equal, it is the position of the rightmost bit. It is easier to grasp the concept by example: the principal position of 10$\underline{0}$11 is 3, the principal position of 1011$\underline{0}$ is 4, and the principal position of 1111$\underline{1}$ is 5.
- *parity*: the number of bits in a word which are set to 1.

Both Butz and Lawder describe the algorithms by listing the variables which shall be calculated in sequence, in order to perform the computation. Here we adopt pseudo-code, but we keep the same variable names, where possible. We also show diagrams illustrating the ith iteration of the algorithms.

Butz observes that the steps 5–7 of Algorithm 3.1 actually perform the Gray-code of ρ^i. Lawder suggests that they can be replaced by the single operation:

$$\sigma = \rho^i \oplus (\rho^i / 2), \qquad \textbf{Eq. 3.18}$$

which can be performed quickly in most architectures using a bitwise exclusive-or and a shift to the right.

Joly presents Algorithm 3.2 in a more formal way, providing a brief commentary on its mechanism and how it relates to the Hilbert curve. First he defines α^i as being a word of d bits composed of the ith bits of each coordinate a_j, i.e., $\alpha^i = \alpha_1^i\alpha_2^i\ldots\alpha_d^i$.

Then he considers that the algorithm corresponds to the equation:

$$\rho^i = \mathrm{P}(\psi_{i-1}(\alpha^i)), \quad i \in [1..m] \qquad \textbf{Eq. 3.19}$$

He defines P as a binary function defined by a loop of exclusive-ors*:

$$z = \mathrm{P}(x) \quad \Leftrightarrow \quad \zeta^i = \bigoplus_{k=1}^{i} \xi^k, \qquad \textbf{Eq. 3.20}$$

where ζ^i and ξ^i are the ith bit of, respectively, z and x. Note that this is the same function performed by the steps 10–12 in Algorithm 3.2.

The function ψ is defined recursively by a set of rules, which, in practice will compute the iterations of Algorithm 3.2:

1. $\psi_{i-1}(\alpha^i) = (\alpha^i \oplus \omega_{i-1}) \circlearrowleft S_{i-1}, \quad i \in [2..m];$ **Eq. 3.21**
2. $S_{i-1} = \sum_{k=1}^{i-1} J_{i-1}, \quad S_0 = 0;$
3. J_{i-1} is the principal position of ρ^{i-1};
4. $\omega_{i-1} = \bigoplus_{k=1}^{i-1} \tilde{\tau}^k, \quad \omega_0 = 00\ldots0;$
5. $\tilde{\tau}^{i-1} = \tau^{i-1} \circlearrowleft S_{i-2};$
6. τ^{i-1} is obtained by taking ψ_{i-2}, complementing the dth bit, and if the parity of the resulting number is odd, complementing also the J_{i-1} th bit.

Joly notes that when $m = 1$, i.e., the first-order curve, we have simply $r = \rho^1 = \mathrm{P}(\alpha^1)$, which corresponds to the conversion of the space coordinates of a d-dimensional space with one bit per dimension. We can visualise those coordinates as being the vertices of a d-dimensional hypercube, and see that the function P defines the motif of the curve.

For higher orders, this conversion is applied recursively, with the difference that the motif must be reoriented accordingly to its position. So, before applying the transformation P, the binary position vector α^i is transformed in a way that ultimately depends from the lower-order binary positions. Analysing the definition of the function ψ, we will note that the exclusive-or performs the reflections, while the bit rotation performs the axes permutations [Joly 2005].

As we mentioned before, for higher dimensions, many Hilbert curves do exist. The algorithms presented arbitrarily choose one of those curves, always the same for a given dimension and order.

* This big $\oplus$ symbol is an iterated exclusive-or, borrowing from the notation of summations.

```
{  The variables are explained in the definitions above (page 82), the
variables
   σ, τ, σ̃, τ̃, ω are all words of d bits, used in the original algorithms
without
   further clarifications

   a[] in an array with n words of m bits corresponding to the space
coordinates

   α is a word of d bits containing the ith bits of the a[]'s

   J is the principal position of ρⁱ and S is the sum of the values of J-1 so
far
}


HilbertKeyToSpace (d, m, r) → Returns array with the space coordinates


1. ω ← 00…0
2. S ← 0
3. For i ← 1 to m do
4. First bit of σ ← first bit of ρⁱ
5. For k ← 2 to d do
6.     kth bit of σ ← kth bit of ρⁱ xor (k-1)th bit of ρⁱ
7. Next
8. σ̃ ← rotate σ S bits to the right
9. α ← ω bitwise xor σ̃
10. For k ← 1 to d do
11.     ith bit of a[k] ← ith bit of α
12. Next
13. τ ← σ
14. nth bit of τ ← not nth bit of τ
15. J ← principal position of ρⁱ
16. If parity of τ is odd then
17.     Jth bit of τ ← not Jth bit of τ
18. τ̃ ← rotate τ S bits to the right
19. ω ← ω bitwise xor τ̃
20. S ← S + J-1
21. Next
22. Return a[]
```

Algorithm 3.1: Converting from extended-key to space coordinates [Butz 1971]

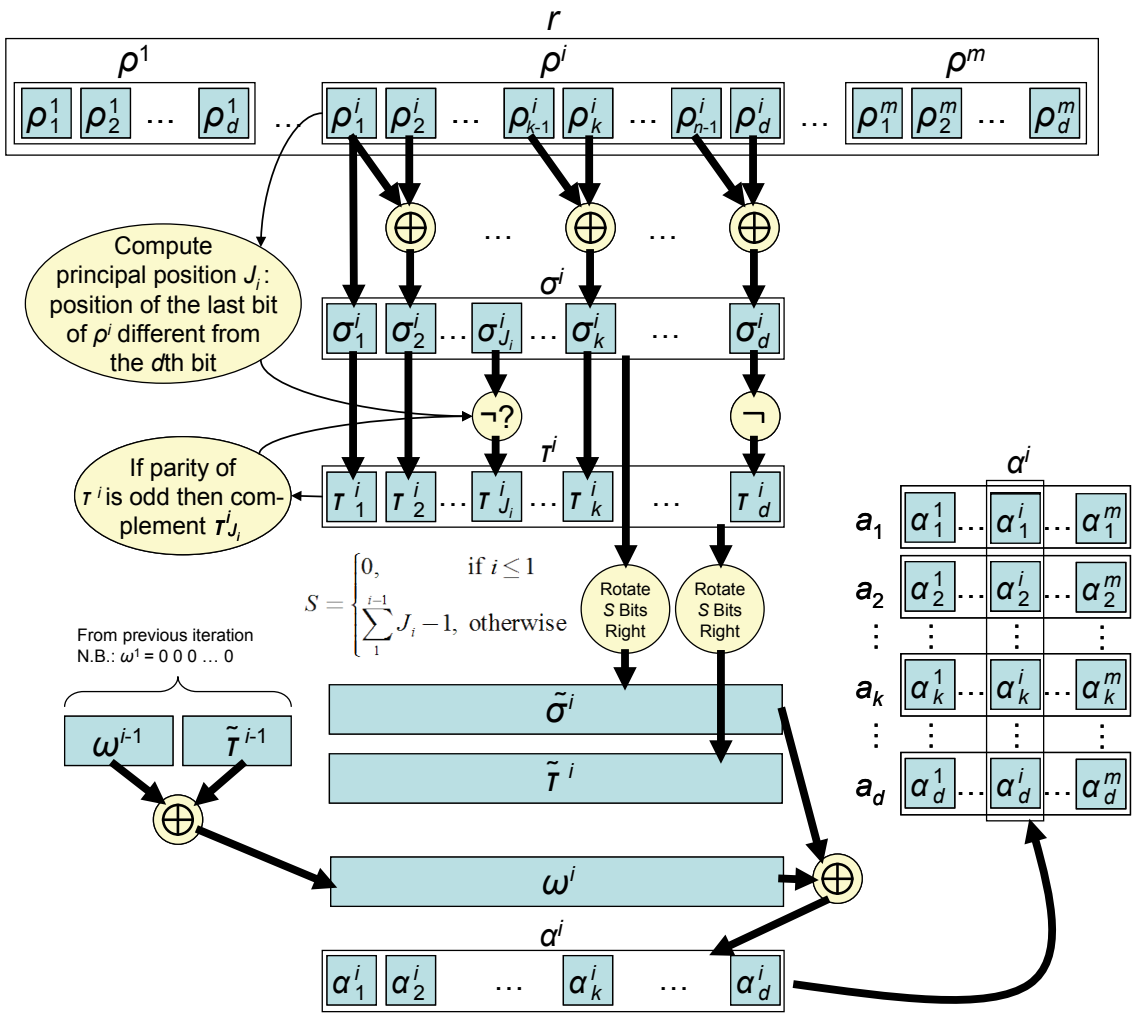


Figure 3.25: The *i*th iteration of Algorithm 3.1

```
{  The variables are explained in the definitions above (page 82), the
variables
   σ, τ, σ̃, τ̃, ω are all words of d bits, used in the original algorithms
without
   further clarifications

   a[] in an array with n words of m bits corresponding to the space
coordinates

   α is a words of d bits containing the ith bits of the a[]'s

   J is the principal position of ρ^i and S is the sum of the values of J-1 so
far
}


HilbertSpaceToKey (d, m, a[]) → Returns an integer of m x d bits


1. ω ← 00…0
2. S ← 0
3. For i ← 1 to m do
4. For k ← 1 to d do
5.     ith bit of α ← ith bit of a[k]
6. Next
7. σ̃ ← α bitwise xor ω
8. σ ← rotate σ̃ S bits to the left
9. First bit of ρ^i ← first bit of σ
10. For k ← 2 to d do
11.     kth bit of ρ^i ← kth bit of σ xor (k-1)th bit of ρ^i
12. Next
13. τ ← σ
14. nth bit of τ ← not nth bit of τ
15. J ← principal position of ρ^i
16. If parity of τ is odd then
17.     Jth bit of τ ← not Jth bit of τ
18. τ̃ ← rotate τ S bits to the left
19. ω ← ω bitwise xor τ̃
20. S ← S + J-1
21. Next
22. Return r
```

Algorithm 3.2: Converting from space coordinates to extended-key [Lawder 2000]

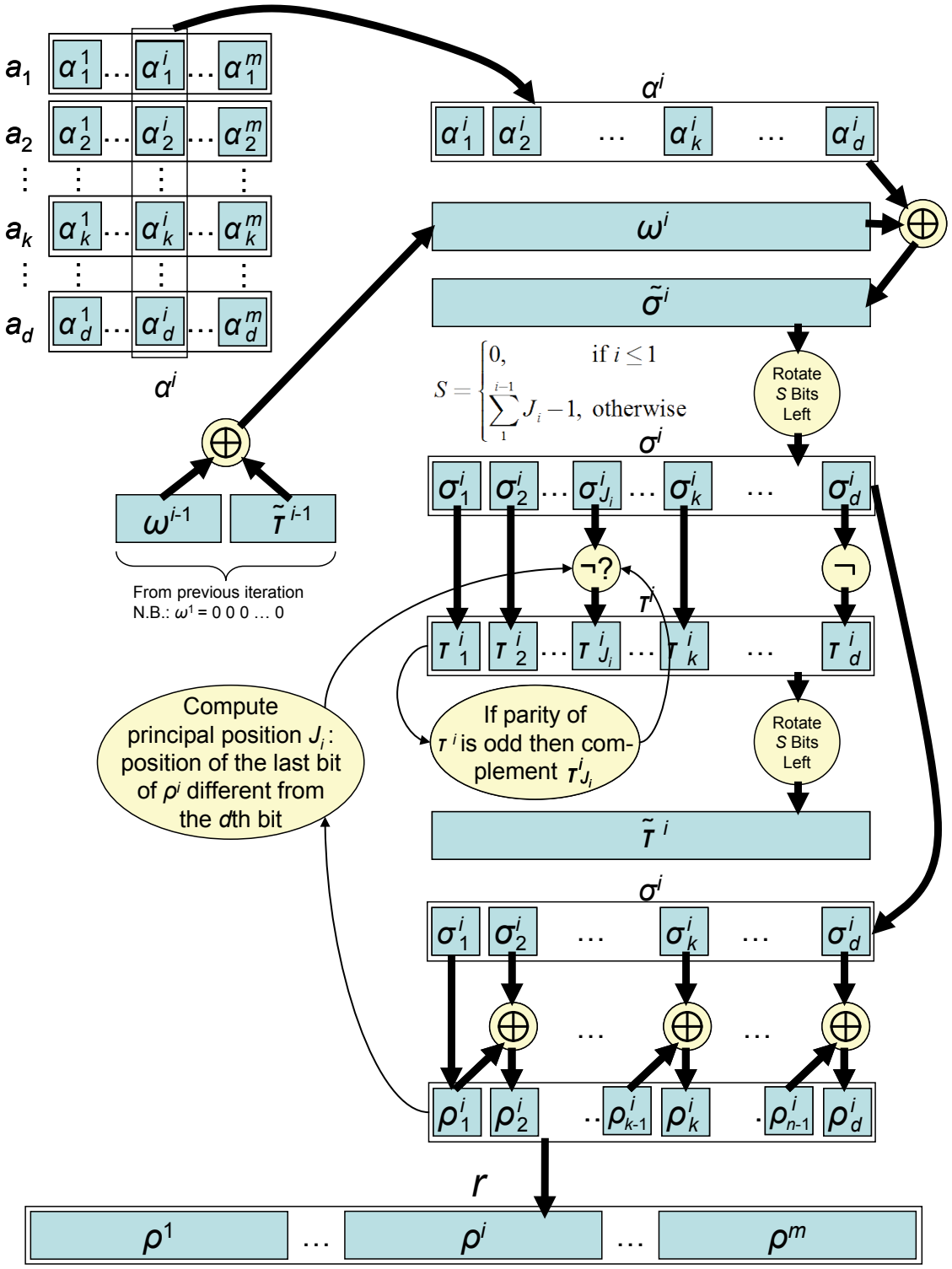


Figure 3.26: The *i*th iteration of Algorithm 3.2

3.12 Alternatives to kNN search

Given the technical and theoretical challenges of the kNN search it is not surprising that alternatives have been sought to bypass it altogether in the matching of the descriptors.

One has to consider that nearest neighbours of a descriptor are only valuable when they actually correspond to truly matched descriptors. This can affect our way of judging the quality of the searches. While in kNN search we measure the error by counting how often we find the exact nearest neighbours (probabilistic definition) or how near the points we found are from the exact nearest neighbours (nearby definition), a more pragmatic approach would be to count how many of the descriptors we have found correspond to real matches.

3.12.1 Range search

Range search is an alternative to kNN search where we are interested in finding all the points which are contained in a certain volume in the space. The kNN search can be considered a range search where one tries implicitly to optimise the range in order to obtain the k points nearest to the query.

The range search is simpler than the kNN search because the region of interest is known *a priori*, making easier to determine which portions of the index to visit. The main difficulty is the choice of the range: if it is too small, one risks obtaining too few points, if it is too big, one can get too many points.

Joly argues that in local descriptors databases, the range query is a more justifiable option than the kNN search [Joly 2005]. His argument is that since local descriptors express features which may be present in an unknown number of images, one cannot determine *a priori* a meaningful value for k. The apparent difficulty of range queries (obtaining too few or too many points) is then just a consequence of the nature of the data: when a descriptor expresses a popular feature, it will generate many matches, when it expresses a rare or unique feature, it will generate few matches. This approach, however, assumes the distribution of the distances between matching descriptors is well known, in order to be possible to select an appropriate value for the range. Acquiring this knowledge may be very challenging.

Some authors propose a compromise between kNN and range search, finding the nearest points of a query which fall within a given range. Those methods have the additional difficulty in making the geometry of the range compatible with the geometry of the distance used in the kNN search. In [Nene 1997], for example, the ranges are hyperrectangular, but the distance is the Euclidean distance, which induces a hypersphere, introducing undesirable boundary effects.

3.12.2 Statistical search

Joly exchanges the kNN search for a statistical approach to similarity search [Joly 2005]. His search is akin to a range search, but instead of using a geometric criterion to select the region of interest (e.g., all points with distance at most x from the query), he advocates the use of a statistical criterion.

In this approach, one starts by modelling the statistical distribution of the distortion observed between matching descriptors. The rationale is that if each descriptor is associated to a certain feature (e.g., the corner of an object), descriptors computed on transformations of that feature (i.e., rotations, scale changes, viewpoint changes…) can be considered as pertaining to the same motif (Figure 3.27).

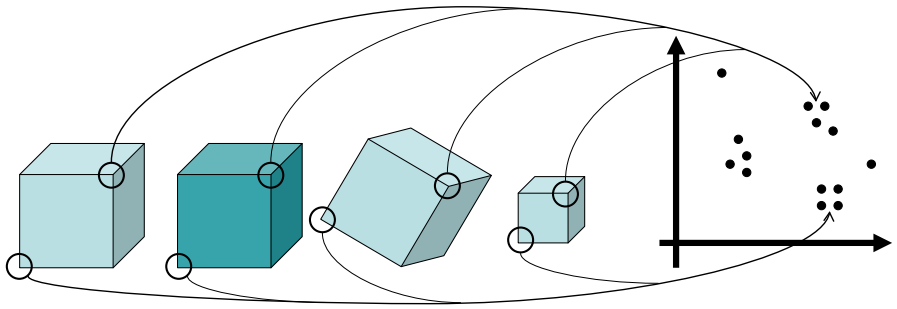


Figure 3.27: Local descriptors can be considered as variations of a motif, where each motif is associated to a given feature

Let us first make some definitions:

- T is the set of all possible transformations on the features (according to § 2.1);
- M is the set of feature motifs;
- $\mathrm{desc}(m)$ is the local descriptor associated to a given feature, assumed to be a vector in a d-dimensional space;
- $\Delta\mathrm{desc} = \mathrm{desc}(m) - \mathrm{desc}(\mathrm{t}(m))$ is the distortion on the descriptor of motif m introduced by transformation t;
- $\mathrm{p}_{\Delta\mathrm{desc}}$ is the probability density function for all possible $\Delta\mathrm{desc}$ ($\mathrm{t} \in T, m \in M$);
- $|\Delta\mathrm{desc}| = \Delta(\mathrm{desc}(m), \mathrm{desc}(\mathrm{t}(m)))$ is the distortion on the descriptor of motif m introduced by transformation t;
- $\mathrm{p}_{|\Delta\mathrm{desc}|}$ is the probability density function for all possible $|\Delta\mathrm{desc}|$ ($\mathrm{t} \in T, m \in M$);
- φ is the expected value of the fraction of relevant descriptors retrieved in the search.

The exact computation of $\mathrm{p}_{\Delta\mathrm{desc}}$ will be normally be impossible, since both sets T and M are huge. One can, however, make some suppositions about the nature of the distribution and estimate its parameters empirically. In his work, Joly supposes that $\mathrm{p}_{\Delta\mathrm{desc}}$ follows a Gaussian probability where each coordinate is independent and have the same standard deviation σ and zero mean.

One possible approach to ensure that the expected fraction of relevant descriptors is retrieved is to determine a radius r such that:

$$\int_0^r p_{|\Delta\mathrm{desc}|}(x)\,dx = \varphi, \qquad \textbf{Eq. 3.22}$$

We can then perform a range search centred on the query and with radius r, to retrieve the matches. It is easy to see that there is trade-off between minimising the volume of the range (which is exponential in r) and maximising φ. If we want to have $\varphi = 1$, we will have an enormous range, thus hindering the performance of the search and creating many false positives.

The proposed approach, however, is not based on the range search. Instead it relaxes eq. **3.22** by specifying that the search region V has to satisfy the statistical constraint without stipulating the geometry:

$$\int_V p_{\Delta\text{desc}}(x-q)\,dx = \varphi\,, \qquad \textbf{Eq. 3.23}$$

We have thus more liberty in defining the search region, and can do it in a way which maximises the system performance. The actual choice for V depends on the underlying index used. We can, for example, avoid accessing a partition of the data when there is small gain in the expected value of matches retrieved. Joly reports a reduction factor of 24 to 82 (depending on the value of φ) of the number of partitions intercepted by the query region, when using the more relaxed condition.

For a complete indexing scheme based on these ideas, the reader is referred to [Joly 2005].

3.12.3 Redesigning the search

As we have seen in § 3.2.5, in high-dimensional spaces distances tend to become indistinct. In those situations, the concept of kNN often becomes meaningless. One way to avoid this is assuring that the distance keep its power of discrimination, either by warranting that the data are highly clustered, or by using specially designed distant functions (§ 3.2.5).

Another way is to redesign the concept of kNN search. An interesting technique is the *projected kNN search*, where we use a local projection, adequate to each query, in order to compute the distance between the points. The idea is that though the distance between the points may be indistinct in full dimensionality, there may be a particular subspace for each cluster, in which the points are close to one another [Hinneburg 2000].

3.13 Conclusion

If it is possible to perform exact kNN search for low-dimensional data (less than 10), we are forced to accept approximate solutions for higher dimensionalities. The trade-off between efficiency and efficacy becomes progressively more severe as dimensionality grows, due to a phenomenon known as “curse of dimensionality”.

Approximation can be conceptualised in two aspects: are the points we find are “close enough” to the true points (nearby kNN search); have we identified a significant fraction of the true points (probabilistic kNN search). Which concept is more appropriate depends on the application — for descriptor matching, the probabilistic notion makes more sense.

In the plethora of existing methods, we had chosen to concentrate in the families of space partitioning and projection, for they do not rely on expensive algorithms for index construction (see § 3.3.6). In those families, two methods caught our attention: the KD-trees and

MEDRANK/OMEDRANK, the former because it has been used successfully in the indexing of SIFT descriptors [Beis 1997; Lowe 2004], and the latter for the clever use of multiple subindexes, and the possibility of employing efficient sorted list structures on those subindexes. The methods we propose in the next chapter were partially inspired on them.

While working with multiple subindexes we got also interested on the methods based on space-filling curves, for they allowed us to conquer the challenge of conciliating multidimensional subindexes with a total order of the points.

An advantage of those methods which should not be neglected is their simplicity. While this may sound of secondary importance, simplicity of implementation greatly enhance the chances a method will find success in real-world applications. It also makes easier the task of code optimisation, which can have a significant impact on the performance of a method.

Chapter 4

Proposed Methods for Nearest Neighbours Search

In this chapter, we propose three original approaches for approximate kNN search.

The first method, **3-way tree**, is based on the KD-tree. Since in high-dimensional spaces the KD-tree suffers from serious boundary effects, we propose to build a tree with overlapping regions, and use them to keep the query away from the edges. The 3-way tree allows fast and precise matching, but because it trades-off disk storage and pre-processing time for performance, it is adequate only for moderately large database (10 million points or less).

The second technique, named **projection KD-forests**, is based on multiple KD-trees, each projecting the data on a subset of the dimensions. Since each individual KD-tree is of moderate dimensionality, boundary effects are minimised. Inspired by MEDRANK, this was our first attempt to perform the search on several subindexes and aggregate the results. The motivation is to conquer the difficulties of MEDRANK (the lost of the spatial relationship between the points) by using multidimensional subindexes instead of straight lines. We obtain good precision, and scalability for any database size conceivable in our context.

The use of multidimensional subindexes, however, comes with a serious penalty: we lose the total order of the points and no longer can use simple sorted lists in the implementation. In comparison to MEDRANK, the index is more awkward to build and to update (insert or delete points). With **multicurves**, our idea is to keep the good results obtained with the partitioning of the dimensions, while recovering the advantages of the total ordering. The technique projects the data on multiple space-filling Hilbert curves of moderate dimensionality. The extended-keys on the curves provide a total order, and the data can be stored on very efficient data structures (like B^+-trees), allowing good performance even on huge databases.

In all cases, we keep in mind the need to create fast, disk-friendly structures, with reasonable building times. Therefore, pervasive in the designs are the choices to minimise random accesses and to avoid complex pre-processing.

4.1 The 3-way tree

The KD-tree* is a classic data structure for multidimensional indexing, which, in its various adaptations, has shown consistently good results, including in the indexing of SIFT descriptors [Beis 1997]. As we have explained in detail in § 3.5, it basically consists of a binary search tree where each node splits the search space along a given dimension. The leaf nodes, which effectively store the points, are called buckets. All splits are parallel to the existing dimensions — this simplicity allows for fast implementations, since no expensive arithmetic has to be performed.

However, on high-dimensional spaces, its performance suffers seriously, because the query will almost always fall near to several boundaries, forcing us to explore a large number of buckets in order to obtain acceptable results (see §§ 3.2, 3.5).

Every time the KD-tree accesses a bucket (which is stored in the disk, unless the whole structure fits in main memory), it must make a random access. Because the order in which the buckets are accessed is essentially unpredictable, there is no way to optimise the i/o. If many buckets are visited, severe i/o costs may result.

In order to avoid boundary effects, it would be desirable to choose a region where the query is the most centralised possible. However, because the regions are built in the offline phase, for some queries it will be impossible to find a region where the query will be centralised. In fact because of the boundary effects of high-dimensional spaces, it is almost certain that the query will fall near to the boundaries of at least one of the dimensions.

Customising the partitioning for each query would be the ideal solution, but of course, its cost prevents it from being of practical consideration. The next best solution is providing *some* redundancy in the partitioning scheme, in the form of overlapping regions. In that way, the most adequate region can be selected, accordingly to the current query. This is the idea behind the 3-way trees.

4.1.1 Basic structure

The 3-way trees are ternary trees where the left and right subtrees are equivalent to the ones of a balanced KD-tree, each containing half the points of the parent node. The middle subtree, however, is overlapping. It adds redundancy to the tree, containing half the same points of each of the other subtrees (Figure 4.1).

The essential feature of the 3-way tree is the fact that now we can choose a "path of the middle", if the query happens to fall too near the boundary between the left and right subtree. This could also be obtained by simply extending the existing branches so that they would overlap each other, but then, each individual node would be larger, and we would have a much deeper tree before reaching buckets of convenient size.

To build 3-way tree, it is necessary to establish, at each node, the dimension used to split the data (as it is the case for the KD-tree). But instead of a single *pivot*, now we need *three* bound-

* As we mentioned before, it is an unfortunate nomenclature tradition that the "k" in kNN and in KD-tree mean different things, respectively, the number of neighbours and the number of dimensions.

ary points: one to split the left and right regions, and the others to determine the boundaries of the middle region. We create then the three nodes and restart the process, recursively, unless the nodes have reached a small enough size, in which case, each node becomes a new leaf node, i.e., a bucket. All data points are stored in the buckets, and because of the redundancy, a given point can appear in many buckets.

It is at the moment of choosing the boundary points that the amount of overlapping (and therefore the redundancy overheads) will be determined. We found generally very hard to conceive any decision based on data which would not be either so time consuming as to become prohibitive, or so imprecise as to become misleading. We have preferred to use a generous, constant, margin of overlapping, independently of the data.

To perform the search, we start at the root and then choose, at each node, the one whose region keeps the query most centralised. Once we reach a bucket, we simply examine every point it contains and return the k nearest to the query.

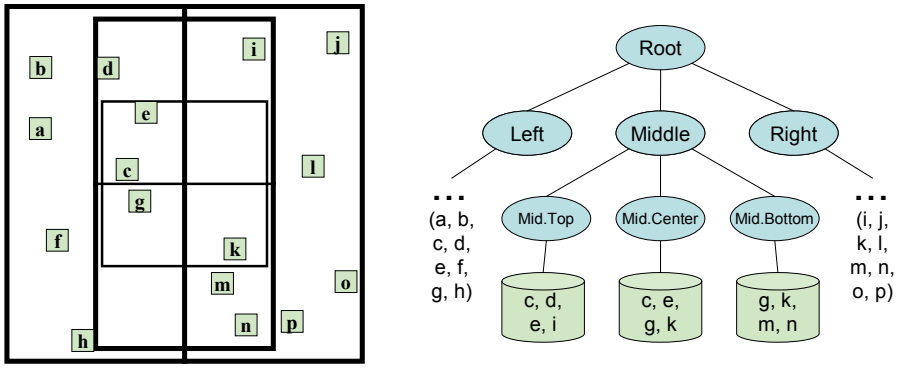


Figure 4.1: A set of data points covered by overlapping regions and the associated 3-way tree. For reasons of clarity, only the middle region and subtree are developed

4.1.2 Construction algorithm

In order to build a 3-way tree, a recursive algorithm is used, choosing the splitting dimension, sorting the data-points on this dimension, computing the boundaries of the regions and the decision limits, separating the points in three somewhat redundant datasets (the further we advance in the recursion, the more the redundancy increases), and calling itself recursively to build the three subtrees (Algorithm 4.1).

Like in the KD-tree, each node has a splitting dimension, which is chosen by the criterion of maximum data spread, using the *interquartile range*.

Definition 4.1 — Median, quartiles, interquartile-range:

- The *quartiles* are the three elements which divide a list of sorted elements into four equal parts;
- The *median* is the *second quartile*, i.e., the element which divides the sorted list into two halves;
- The difference between the values of the third and the first quartile is called *interquartile range*.

  Consider, for example, the set of values { 10, 13, 15, 20, 21, 27, 35, 42, 50, 51, 53 }. The first, second and third quartiles are, respectively, 15, 27 and 50, and the interquartile range is 35 (= 50–15).

■

Though other statistics could theoretically be used to determine the spread, some are either too sensitive to outliers (like the range, which simply takes the difference between the maximum and the minimum values) or too expensive to compute (like the variance, which demands many arithmetic operations).

After the splitting dimension is chosen, the data points are sorted in that dimension. The lesser half is put on the left subtree and the greater half on the right subtree. The half data points between the first and the third quartiles are put on the middle subtree. This scheme warrants a balanced tree.

The internal (non-leaf) nodes of the tree do not store any data points (Figure 4.2). They only have information about the regions, in order to allow a fast traversal of the tree during the search. This information consists of the splitting dimension and two "decision limits", which are simply the arithmetic mean of the overlapping boundaries of the two regions, i.e., the value equidistant from the two boundaries.

Comparing the decision limits with the value of the query on the splitting dimension, the algorithm can quickly verify which subtree to follow. If the value is lesser than the inferior limit, the traversal proceeds on the left subtree, if it is greater than the superior limit, the traversal proceeds on the right, and otherwise, it proceeds on the middle. This scheme guarantees that we always choose the subtree that keeps the query the furthest away from the boundaries.

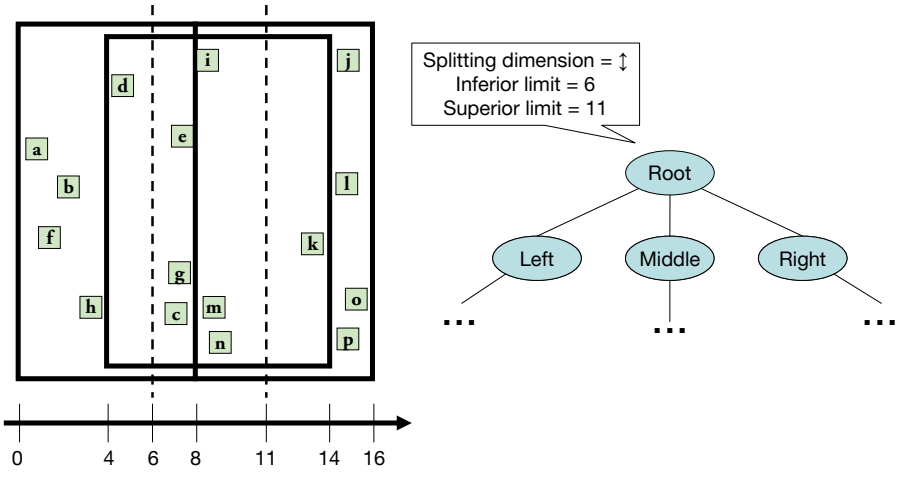


Figure 4.2: The decision limits and the internal nodes on a 3-way tree

There is only one parameter in the construction of the 3-way tree: the *desired size for the buckets* which will determine the height of the tree and the redundancy overhead. As we will see in § 4.1.4, this parameter is quite critical, because it determines not only the compromise between precision and efficiency, but also the storage overhead due to the redundancy on the tree.

In the unlikely event that the quartiles have the same value, the node will degenerate. If all three quartiles are equal, it means that the data points are much too clumped together, even in the most spread dimension. In that case, the algorithm opts for an early termination of the recursion, and a *large bucket* is created containing all remaining points. If only two of the quartiles are equal, there is always the possibility that another dimension will separate the data further down on the tree. In those cases, a *pseudo-binary node* is created with the middle subtree pointing to the same node as either the left or the right subtree (Figure 4.3).

Large buckets and pseudo-binary nodes are very rare occurrences. In our tests with both real data derived from image descriptors and data generated by pseudo-random distributions, we counted at most two occurrences of pseudo-binary nodes (in a tree with more than 260 000 nodes). As for the large buckets, we observed not a single incidence — in order to test the algorithm in their presence, we had to manually fabricate a degenerate dataset.

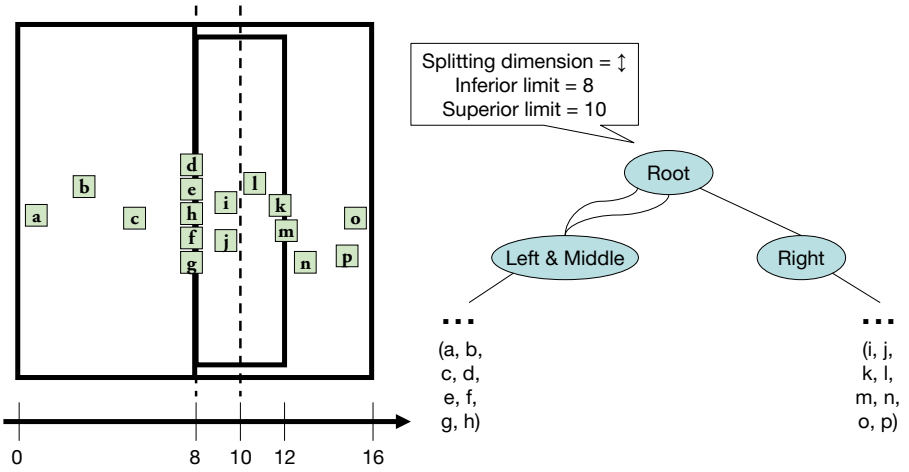


Figure 4.3: A degenerate *pseudo-binary node* may happen in the 3-way tree in very rare cases

```
{  points is the list of data points to build the node
   size_buck is the max. number of points in a bucket   }

Build3WayTree(points) → Returns a tree node

1.  If number of points in points < size_buck then
2.  Return a bucket with all the points in points
3.  splitDimension ← the dimension of points with the largest interquartile
    range
4.  Sort points on the values of splitDimension
5.  Find the three quartiles of points: firstQuartile, median and
    thirdQuartile; keep also their positions firstQuartile_index, median_index and
    thirdQuartile_index
6.  If firstQuartile = thirdQuartile then
7.  Return a large bucket with all the points in points
8.  infLimit ← (firstQuartile + median) / 2
    supLimit ← (thirdQuartile + median) / 2
9.  Build a new internal node with splitDimension, infLimit and supLimit
10. If firstQuartile = median then
        { Creates pseudo-binary node on the left }
11. leftPoints ← sublist of points from the first point to thirdQuartile_index
12. Left and middle subtrees ← Build3WayTree(leftPoints)
13. rightPoints ← sublist of points from median_index to the last point
14. Right subtree ← Build3WayTree(rightPoints)
15. Else if thirdQuartile = median then
        { Creates pseudo-binary node on the right }
16. leftPoints ← sublist of points from the first point to median_index
17. Left subtree ← Build3WayTree(leftPoints)
18. rightPoints ← sublist of points from firstQuartile_index to the last point
19. Right and middle subtrees ← Build3WayTree(rightPoints)
20. Otherwise
        { Common case – creates normal ternary node }
21. leftPoints ← sublist of points from the first point to median_index
22. Left subtree ← Build3WayTree(leftPoints)
23. midPoints ← sublist of points from firstQuartile_index to thirdQuartile_index
24. Middle subtree ← Build3WayTree(midPoints)
25. rightPoints ← sublist of points from median_index to the last point
26. Right subtree ← Build3WayTree(rightPoints)
27. Return the internal node just created
```

Algorithm 4.1: Construction of the 3-way tree*

* All algorithms in this chapter follow the conventions explained on page viii.

The analysis of the construction algorithm (Algorithm 4.1) is simple, if we ignore the large buckets and pseudo-binary nodes. Finding the interquartile range can be done naïvely by sorting the points on the dimension and taking the quartiles, which can be done in $\mathcal{O}(n \log n)$, but it is possible to adapt QuickSort (data on main memory) or MergeSort (data on disk) to compute it in $\mathcal{O}(n)$. Since the computation must be done for every dimension, the complexity of step 3 is of $\mathcal{O}(dn)$.

To simplify the exposition, in step 4, we sort all the points in the subtree, which takes $\mathcal{O}(n \log n)$ steps, but this is not strictly necessary, and the algorithm can simply separate the points between the three quartiles, which takes $\mathcal{O}(n)$. Actually the optimal solution is to combine steps 3–5 in a single pass.

Then, all other steps take constant time, except the steps 21, 23 and 25, which are $\mathcal{O}(n)$ (we ignore steps 6–7, which concern large buckets and 10–19 which concern pseudo binary nodes).

Three recursion calls occur, at steps 22, 24 and 26. Until we stop (when we reach the desired bucket size) every new call processes half the points as the current one. Since the complexity of a single call is $\mathcal{O}(dn + n \log n)$, the final complexity is given by:

$$\mathcal{C} = \sum_{i=0}^{h} \mathcal{O}\left(\frac{3^i}{2^i} \cdot dn\right), \text{ where } h \text{ is the height of the three.}$$

Simplifying this expression, we can finally arrive at a closed formula, highlighting the exponential dependency on h:

$$\mathcal{C} = \mathcal{O}\left(\left(3/2\right)^h \cdot nd\right) \qquad \textbf{Eq. 4.1}$$

4.1.3 Search algorithm

The search algorithm is, comparatively, very simple. The tree is traversed from top to bottom, always choosing the subtree where the query falls the most centralised, until a bucket is found. Then, all data points in the bucket are sequentially compared to the query (Algorithm 4.2).

This is the greatest advantage of the 3-way tree: it keeps the query roughly centralised all the way down the search tree, avoiding the problems brought by boundary effects. In that way, only one bucket has to be explored.

The analysis is quite simple. The loop on steps 2–10 will be executed as many times as the height of the tree (h). The step 11 computes as many distances as the size of the bucket ($size_{buck}$). So, for p-norm distances — which are $\mathcal{O}(d)$ — the final complexity is given by:

$$\mathcal{C} = \mathcal{O}\left(h + d \cdot size_{buck}\right) \qquad \textbf{Eq. 4.2}$$

```
NearestNeighboursSearch(query) → Returns a list of k nearest neighbours

1. currentNode ← root of the 3-way tree
2. While currentNode is not a bucket do
3. queryValue ← value of query on the dimension currentNode.splitDimension
4. If queryValue < currentNode.infLimit then
5.     currentNode ← currentNode.leftSubtree
6. Else if queryValue > currentNode.supLimit then
7.     currentNode ← currentNode.rightSubtree
8. Otherwise
9.     currentNode ← currentNode.middleSubtree
10. End While
11. Search sequentially for the k nearest neighbours on currentNode
12. Return the list of nearest neighbours found
```

Algorithm 4.2: kNN search on a 3-way tree

4.1.4 Parameterisation and storage needs

The 3-way tree requires the adjustment of just one parameter — the size of the bucket* ($size_{buck}$) — but it affects almost all aspects of performance: the time spent on search, the precision obtained, the overhead due to the redundant coverage of the data, and the height of the tree (h).

Consider first the height of the tree and how it can be deduced from the size of the bucket. During the construction algorithm, at each new level of the tree, the number of points is cut in half. Thus, the height at which the desired bucket size will be reached is given by:

$$h = \left\lceil \log \frac{n}{size_{buck}} \right\rceil + 1 , \qquad \textbf{Eq. 4.3}$$

remembering that n is the number of points in the database.

To compute the size of the index, we ignore the very rare abnormalities (pseudo-binary nodes and large buckets), and the small irregularities due to quantisation. Then, there is a simple relationship between the size of the buckets ($size_{buck}$), the height of the tree (h), and the size of the index (n_{store}).

Because of the redundancy, each additional level of the tree causes an increase in size, as we can see in Figure 4.4.

* Obviously, it is not possible to choose arbitrarily the bucket size, the options being restricted by the fact the tree is balanced.

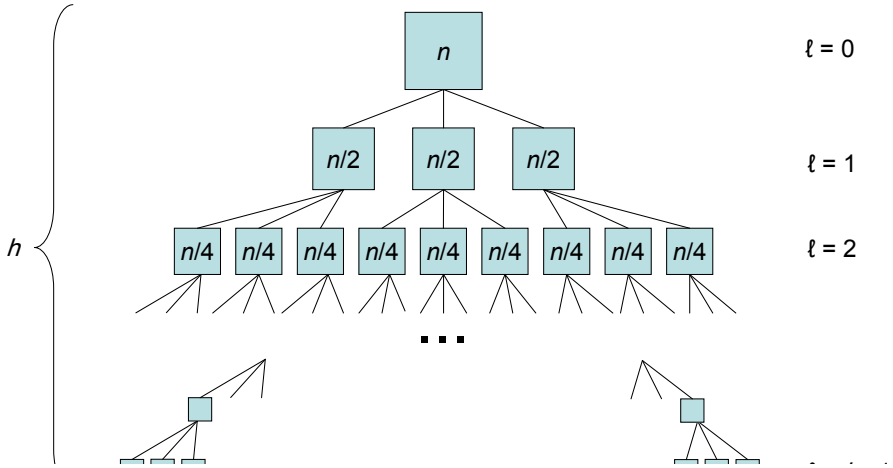


Figure 4.4: Computing the size of a 3-way tree

We have already seen that the internal nodes of the tree do not actually store any points. All they keep is some geometric information to allow the traversal of the tree. The total number of data points stored in the buckets will be given by:

$$n_{stored} = n\left(\frac{3}{2}\right)^{h-1} \qquad \textbf{Eq. 4.4}$$

In what concerns the internal nodes, since they occupy a few bytes at most, they represent a negligible fraction of the structure occupancy, and can be made to fit on primary memory even for huge databases (billions of data points),

What must be retained from Eq. 4.3 and Eq. 4.4 is that the size of the index grows exponentially with the height of the tree, and this, in turn, depends inversely on the bucket size. Therefore, for the index size to be feasible, we have to limit ourselves to a moderate database (less than 10 million points), or to accept a large bucket.

Space occupancy may not be a great concern in itself, as disks are cheap nowadays, but in some applications it can be critic. Besides, construction time is proportional to the size, and this, for large indexes can become a problem, especially if we have to rebuild them periodically.

The only way to reduce this overhead is to decrease the overlapping of the regions, but this is detrimental to the boundary avoiding mechanism, which is the core purpose of the 3-way tree. Conversely, the tree could have more aggressive overlapping, but this would incur in even more serious space overheads.

We could theoretically conceive an m-order tree, with a basic partition into m subregions, plus $m-1$ overlapping regions. The advantage is that we would have a shallower tree (since the logarithm of Eq. 4.3 would now be to the base m). Even if the compromise on the height would become more severe (since the base of the exponential of Eq. 4.4 would now be $m/m-1$) the final balance is still positive. There are, however, two serious drawbacks with this approach: first, the construction of the tree becomes much more difficult, since we are forced so compute several percentiles and make a complex decision on which dimension has the best values of

those percentiles; second, we slice the data in smaller regions, considering only one dimension, running the risk of hasty decisions in the upper levels of the tree.

We were generally satisfied with the quality of the results of the 3-way tree, but not so much with its space overhead and the fact it is a completely static index. Therefore, we pursed a method which would conquer those difficulties.

4.2 The projection KD-forests

Inspired by MEDRANK (§ 3.8), we wanted to devise a method that would perform similarity search in low to moderate-dimensional subspaces and then combine the partial results to obtain the final answer. We wanted, however, to avoid the drawbacks of MEDRANK, mainly the lack of correlation between proximity in a single dimension and the proximity in the high-dimensional space. The solution we envisaged was to consider several dimensions at the same time.

We decided to use several parallel KD-tree subindexes, each responsible for a projection of the dataset in a subspace and called the method *projection KD-forest*. The choice of the KD-tree came in part from our familiarity with the method, but mainly the simplicity of this technique, which allows, when necessary, aggressively optimised implementations.

The idea behind KD-forests is simple and presents several advantages:

- Each tree, being built on a projection of the complete space, has moderate dimensionality. The less dimensional the trees, the more precise the search;
- In comparison with MEDRANK, there is a greater correlation between proximity in the subspaces and in the original space;
- We avoid the collisions, which plague MEDRANK, where, for very high dimensionalities and quantised data, on each dimension many points coincide exactly with the query;
- And finally, since we have multiple trees, we can alleviate boundary effects, because we explore different regions in the different trees.

This last factor is very important, since it will allow us to make a radical choice when searching the trees: just the best bucket (i.e., just the bucket whose region contains the query) will be explored. This allows us to keep the number of random accesses to the bare minimum.

A note on terminology: the KD-forest **index** is composed of multiple KD-tree **subindexes** which are built from the data projected in **subspaces** of the whole database. Therefore, subindex is the structure, and subspace is the projection which concerns the structure. However, since there is a bijection between the two, we sometimes use the terms a little loosely.

4.2.1 Construction algorithm

The construction of the projection KD-forests is simple, once the following parameters have been established:

- d : the dimensionality of the domain of the data points;
- n_{trees} : the number of trees to build;
- d_i : the dimensionality of the i^{th} tree;
- A: an association between the dimensions of the data and the dimensions of the trees.

The association can be either very simple (distribute the dimensions of the data domain evenly among the curves), or very complex (distribute the dimensions considering the data distribution; use trees of different dimensionalities), as shown in Figure 4.5.

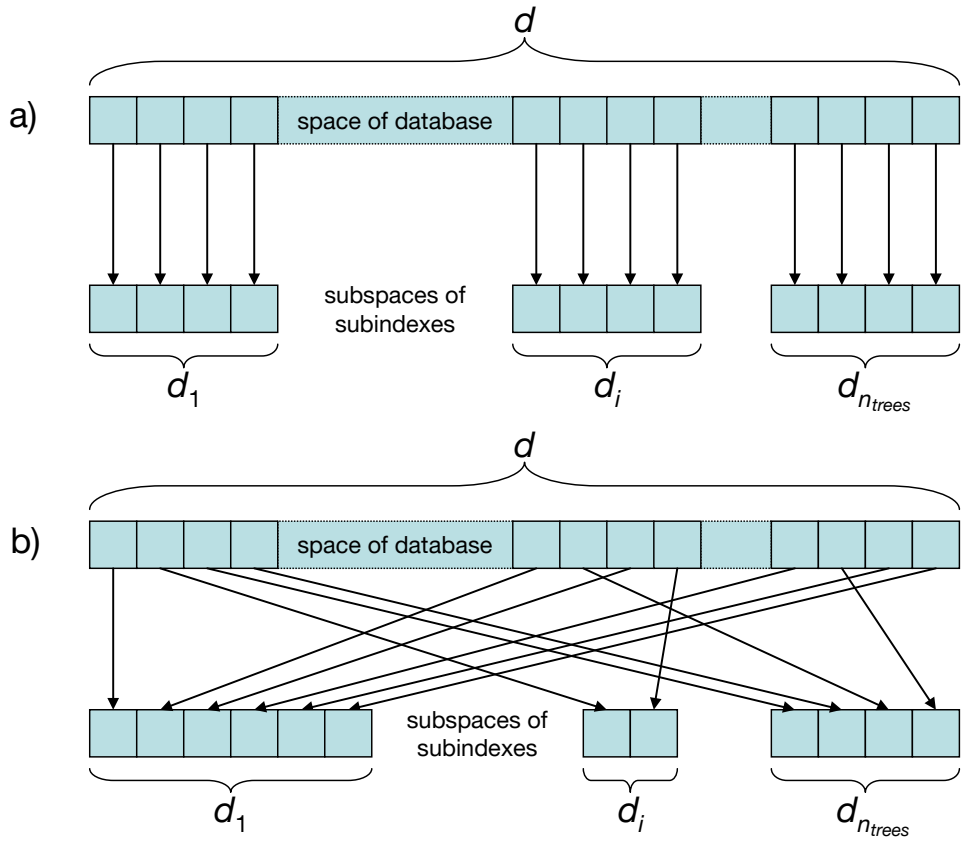


Figure 4.5: Two sample parameterisations of the KD-forest showing different choices for the parameters n_{trees}, d_i, and the association A, leading either to a very regular scheme (a), or not (b)

Naturally, the parameters for the component KD-trees have to be decided (e.g., size of the buckets, criteria for splitting, etc…). We use the interquartile range to choose the splitting dimension and the median as pivot. The size of the bucket must agree with the desired probe depth (number of points examined), since we examine just one bucket per subindex.

The complexity of the algorithm is basically the complexity of building n_{trees} KD-trees (which was deduced in [Friedman 1976]).

The computation of the projections (steps 3–9), takes at most $\mathcal{O}(dn)$ steps. The construction of each tree (step 10) can be analysed in the same fashion as the construction of the 3-way tree,

with the great difference that only two subtrees are build, and the number of points is cut in half at each subtree. This means that, at each recursion level, the number of points is constant, which simplifies the analysis. The complexity of a single call is, like before, $\mathcal{O}(dn)$, but now, the complexity to build the tree is:

$$\mathcal{C} = \sum_{i=0}^{h} \mathcal{O}(dn), \quad \text{where } h = \log\left\lceil \frac{n}{size_{buck}} \right\rceil$$

Simplifying, and ignoring the constant factors, we arrive at the closed formula for a single KD-tree:

$$\mathcal{C} = \mathcal{O}(dn \log n) \qquad \textbf{Eq. 4.5}$$

The final complexity is given by:

$$\mathcal{C} = \mathcal{O}(n_{trees} \cdot dn \log n) \qquad \textbf{Eq. 4.6}$$

```
{  The parameters are explained in the definitions above (page 103)
   points is the list of data points to build the index
   point[i] is the value of the ith dimension of point
   d[i] is the dimensionality of the ith tree (di)
   A[][] is an association between the dimensions of the data space and the
      subindexes space such that A[i][j] indicates the jth dimension (from
the
      data space) that should be attributed to the ith subindex, i.e.,
      if M[2][3] = 10, it means that the 3rd dimension of the 2nd subindex is
      in fact the 10th dimension of the data space
   projection[] is the point projected on the selected subspace
   trees[] is an array of ntrees KD-trees  }

BuildKDForest (ntrees, d[], A[][], points) → Returns array of ntrees KD-trees

1.  For tree ← 1 to ntrees do
2.  projectionList ← new empty list of projections
3.  For each point in points do
4.     projection[] ← new array with d[tree] elements
5.     For dimension ← 1 to d[tree] do
6.        projection[dimension] ← point[A[tree][dimension]]
7.     Next
8.     Put projection in projectionList
9.  Next
10. trees[tree] ← new KD-tree containing the projections in projectionList
11. Next
12. Return trees[]
```

Algorithm 4.3: Construction of the projection KD-forest

4.2.2 Search algorithm

Contrarily to MEDRANK, the search algorithm for the projection KD-forest is not based on rank aggregation, but on exploring all the points on the best bucket of each subindex, computing their distances to the query, and keeping the k best (Algorithm 4.4).

Implementing rank aggregation on the KD-forest would not be straightforward, because the data are retrieved by buckets, not individually, making the voting process very imprecise.

The loop on steps 2–17 is executed once for each tree. The internal loop on lines 4–6, which computes the projections, has complexity $\mathcal{O}(d)$. Since step 7 has a complexity equivalent to the height of the tree, the loop on steps 9–16 is executed once for each point in the bucket, and assuming that the distance function on step 11 has complexity $\mathcal{O}(d)$ the final complexity is:

$$\mathcal{O}\left(n_{trees} \cdot \left(\log \frac{n}{size_{buck}} + d \cdot size_{buck}\right)\right) \qquad \textbf{Eq. 4.7}$$

```
{  The parameters are explained in the definitions (page 103) and
   in Algorithm 4.3
   query[i] is the value of the ith dimension of query  }


SearchKDForest(n_trees, d[], A[][], probeDepth, k, trees[], query) →
   Returns a list of k nearest neighbours

1. best ← new empty sorted list of pairs <distance, point> sorted by distance
2. For tree ← 1 to n_trees do
3. projection[] ← new array with d[tree] elements
4. For dimension ← 1 to d[tree] do
5.    projection[dimension] ← query[A[tree][dimension]]
6. Next
7. Find the bucked in trees[tree] whose region contains projection
8. candidates ← list with all points in the bucket
9. For each candidate in candidates do
10.   point ← the point on the original space associated to candidate
11.   distance ← distance from point to query
12.   If distance < last distance in best then
13.      Put pair <distance, candidate> in best
14.   If best has more than k entries then
15.      Remove last entry in best
16. Next
17. Next
18. Return best
```

Algorithm 4.4: kNN search on the projection KD-forest

4.2.3 Discussion

Since the implementation of our approach, we discovered that Berchtold et al. had already considered the idea of splitting a multidimensional query into multiple queries in lesser-dimensional subspaces, a technique they call *tree-striping* [Berchtold 2000]. They are not interested on any particular implementation, but mention that the technique can be used for any kNN search method.

In their paper, which concerns range queries, they propose a theoretical model for the cost of querying the set of trees and use it to determine the optimal number of trees. Some consideration is also given on how to divide the dimensions among the trees. They perform experiments to validate the model, using with R*-trees and X-trees on the sub-indexes, but obtain an agreement only in low dimensionalities (< 10).

Based on this work, Dellis et al. propose a method for kNN search [Dellis 2005]. They propose a new cost model (concerning kNN search) and a more advanced way to assign the dimensions to the trees, taking in consideration the fractal dimensionality of the data. They also propose an algorithm to incrementally merge the results obtained from the subindexes, quite similar to rank aggregation. They perform experiments to validate their model, using R-trees on the subindexes, and small databases of image features and colour histograms. A reasonable agreement between the model and experiments is obtained, but only the dimensions 16 and 32 are evaluated.

The greatest difference to our work is that our focus is not at all on the cost model, (to which both previous works give foremost importance) or other theoretical considerations. We are interested in performing kNN search fast and accurately, and doing as few random accesses as possible.

Because of this, instead of simply adopting an unmodified pre-existing algorithm on the subindexes, we adapt the KD-trees to take advantage of the context and explore just the best bucket (which collaterally, allows a great simplification of the search algorithm). So, while the method of Dellis et al. reports typically 1 000 disk page access (4 KiB pages, no information on how many are sequential and how many are random) for a 10-NN search on a 16-dimensional database, projection KD-forests can perform a 20-NN search on a 128-dimensional database, needing 50 disk page access (4 KiB pages) of which all but four are sequential.

4.2.4 Parameterisation and storage needs

The most important parameter for the construction of the projection KD-forests is the number of subindexes. Once this is set, we may start to worry about how the dimensions are distributed among the subindexes.

Berchtold et al. propose a model to estimate the cost of performing *range* queries on multiple subindexes generated by dimensional partitioning, and use this model to compute the optimal number of subindexes. They also propose simply partitioning the dimensions roughly equally among the subindexes, unless some *a priori* information about the selectivity of the dimensions

is available. In that case, it is better to group together the dimensions with better selectivity (this may sound counterintuitive, but makes sense for their algorithm of range search, which has the option to bypass the subindexes of lower selectivity, if the estimated cost of querying them is higher than the cost of linear scan in the current result subset) [Berchtold 2000]. Their model, however, is of little use to us because the hypothesis it assumes (*range* querying, moderate dimensionality, uniformly distributed data, an expensive step of merging the results after the subindexes are searched) do not apply at all to our case.

Dellis et al. also propose a cost model, this time for kNN search and without the assumption of uniformly distributed data. Again, they use the model to determine the optimal number of indexes, and then distribute the dimensions using a heuristic which at the same time attempts to maximise the correlation fractal dimension (d_2) of each subindex and to keep those dimensions roughly equitable among the subindexes [Dellis 2005]. But, again, the model was only validated for moderate dimensional data (32 dimensions or less), and, above all, the dimension assignment step is very expensive to compute.

Our own approach is much more pragmatic. The cost of random access will anyway prevent any scheme with more than 8–10 subindexes. At such limited ranges, and for the large dimensionalities we are processing, the rule of thumb is using as many subindexes as efficiency restrictions will allow — i.e., we use the highest number of trees which allows us to stay below the desired search time. The actual value must be determined empirically (we do so for our databases in the next chapter).

As for the dimension assignment, we opt for the simple partitioning proposed by Berchtold et al. This means that the dimensions are simply distributed among the subindexes in a balanced way.

For much skewed data taking a representative sample of the database and performing the analysis proposed by Dellis et al. might improve the performances. Expecting to perform the analysis on the entire database, however, is unrealistic, as it calls $\mathcal{O}(d^2)$ times the "box counting" algorithm (§ 3.2.6) to compute the correlation fractal dimension, which already is expensive in itself.

The size of the bucket should also be considered, but, in contrast to the 3-way tree, the impact of the bucket size on the index size is minimal (it may slightly affect the overhead due to the internal node storage), so the choice is much less constricted. The main concern to be considered on the choice is the number of points which get to be examined during the search.

The storage needs are easy to compute. The forest is basically a set of KD-trees, which by themselves have no redundancy, just storing the data points. So, the total number of stored points will be given by:

$$n_{stored} = n \times n_{trees} \qquad \textbf{Eq. 4.8}$$

where n is the number of points in the database.

Analogously, the number of internal nodes will be replicated for each tree. Because the internal nodes have so little data, they can be stored in primary memory even for huge data sets.

If the 3-way tree had an important space overhead, the projection KD-forests are scalable for any database size we can possibly imagine in our context.

However, the use of several simultaneous dimensions in the subindexes, which allows conquering the problems of MEDRANK, brings also a great disadvantage: the loss of the total order relationships, preventing us from using the fast and convenient sorted list data structures. In order to regain those advantages, we devised the multicurves method, described in the next section.

4.3 The multicurves

We were very attracted by the idea of using a few moderate dimensional subindexes at once, instead of either a single high-dimensional index or a many one-dimensional straight lines. This technique, as we have seen for the projection KD-forests, allows a good compromise between minimising the curse of dimensionality (reducing it on each individual subindex) and loosing the spatial relationship between the points (which plagues the approaches based on straight lines).

However, passing from straight lines to more-dimensional indexes incurs in a serious penalty: we lose the total order of the points. We are no longer able to simply use sorted lists (which can be managed by very efficient and widely available data structures) and are forced to implement more complex indexing mechanisms.

Trying to overcome this loss, we turned our interest to the space-filling curves (§ 3.11). They allowed us to create multiple moderate-dimensional subindexes, retaining a total order in each one of them.

The idea of using multiple curves at once to avoid boundary effects is not new, and has been used by many methods in the literature (§ 3.11.2). The originality of multicurves is that each curve is only responsible for a subset of the dimensions. So not only we gain the intrinsic advantages of using multiple curves (i.e., points that are incorrectly separated in one curve will probably stay together in another), but also, we lower the boundary effects inside each one of the curves.

4.3.1 Basic structure

Multicurves has a simple structure: divide the dimensions of the data points among a certain number of subindexes. On each subindex, take the entries (which are composed of the chosen dimensions of each point in the database), compute their extended-keys in the space-filling curve, and put each pair <extended-key, point> in a list sorted by extended-key.

The search is done the same way: the query is decomposed into several projections (corresponding to the dimensions belonging to each subindex) and each projection has its extended-key

computed. Then, for each subindex, we explore the points whose extended-keys are the nearest to the corresponding query extended-key.

We decided to use the Hilbert curve for two reasons. First, it has been shown to have good clustering properties, which is an important feature for kNN search [Mokbel 2003; Moon 1996]. Second, there is an efficient algorithm to compute its extended-keys (§ 0).

Any adequate data structure can be used to store the sorted lists.

4.3.2 Construction algorithm

An efficient (linear-time) algorithm to compute the extended-keys is essential for the success of the method, lest the mapping onto the curves will become a bottleneck. In our experiments we use the algorithm by Butz (see § 0, [Butz 1971]).

The choice of the data structure for the sorted list should be based on practical considerations.

Very dynamic indexes, in which insertions and deletions are expected to occur often, require some forethought. If the index is being implemented in main memory, some kind of balanced tree, like the red-black tree, may be a good option. In disks, the B^+-tree should be the best solution.

If the index is completely static, a simple sorted array may suffice at least for main memory. For disks, a two-level indexing is better, to avoid wasteful random accesses to the disk. Those are the solutions we adopt in our experimental implementations.

The following parameters should be determined before the construction of the index:

- d : the dimensionality of the domain of the data points;
- m: the number of bits needed to represent each dimension of the data points, which will correspond to the order of the Hilbert curve to be generated;
- n_{curves} : the number of curves to build;
- d_i : the dimensionality of the i^{th} curve;
- A: an association between the dimensions of the data and dimensions of the curves (see Figure 4.5).

The multicurves consist in a set of sorted list, one for each Hilbert curve generated. Each data point is decomposed in n_{curves} projections, accordingly to the association of dimensions A. Those projections are used to compute the extended-key in each curve, using Algorithm 3.2. The pairs <extended-key, data point> are then inserted in the curves.

```
{  The parameters are explained in the definitions above (page 109)
   points is the list of data points to build the index
   point[i] is the value of the ith dimension of point
   d[i] is the dimensionality of the ith curve (di)
   A[][] is an association between the dimensions of the data space and the
      subindexes space such that A[i][j] indicates the jth dimension (from
the
      data space) that should be attributed to the ith subindex, i.e.,
      if M[2][3] = 10, it means that the 3rd dimension of the 2nd subindex is
      in fact the 10th dimension of the data space
   curves[] is an array of ncurves sorted lists  }

BuildMulticurves(ncurves, d[], A[], m, points) → Returns array of sorted lists

1.  For curve ← 1 to ncurves do
2.  curves[curve] ← new empty sorted list of pairs <extended_key, point>
          sorted by extended_key
3.  Next
4.  For each point in points do
5.  For curve ← 1 to ncurves do
6.     projection[] ← array with d[curve] elements
7.     For dimension ← 1 to d[curve] do
8.        projection[dimension] ← point[A[curve][dimension]]
9.     Next
10.    extended_key ← HilbertSpaceToKey(d[curve], m, projection)
                                                  { see Algorithm 3.2 }
11.    Put the pair <extended_key, point> into the list curves[curve]
12. Next
13. Next
14. Return curves[]
```

Algorithm 4.5: Construction of the multicurves

4.3.3 Search algorithms

Three modalities of search are possible on the multicurves:

- Search by direct **distance comparison**: in this modality, the number of points to explore in each subindex is decided *a priori*, and those points are explicitly compared to the query, using the distance function;
- Search by **rank aggregation**: inspired from MEDRANK, this modality uses the relative position of the points on the subindexes, without need to perform any actual distance computations;
- **Mixed search**: in this modality, we do a search by rank aggregation, for more neighbours than we actually need, and then, do a distance computation only for those neighbours. We return the k best answers.

In the simplest algorithm, search by distance comparison, basically we choose a set probe depth (number of points to examine) and then explore them in a conventional way: computing all the distances to the query. It is more precise, but it is CPU intensive, especially if the distance function requires lots of computations. It is shown in Algorithm 4.6.

The search by median rank aggregation does not compute the distances. Instead it uses a system of votes. It keeps a cursor on each curve, starting from the nearest point (the nearest to the corresponding extended-key in the curve). It keeps a table of all points seen so far, and the number of votes they received. The first point to receive votes from more than half the number of curves is declared the first nearest neighbour, and so on. The search works exactly like MEDRANK or OMEDRANK, but using the curves instead of the straight lines (Algorithm 4.7).

This scheme is less CPU intensive than the first one (since no distances whatsoever are computed), and has the additional advantage that only an identifier for the points must be stored in the sorted lists, not the actual points. It has, however some disadvantages:

- It is less precise than the first method, in particular, there will be some inversions in the order of the nearest neighbours list;
- There is a small probability it may degenerate for some queries, this probability can grow considerably for bad data distributions. The degenerate case has to be detected and stopped; otherwise one risks going through the entire database, which will effectively "freeze" the system for some queries.

The modality of mixed search, combining the two algorithms above, does a median rank aggregation search to find the k' nearest neighbours (where k' is the number of neighbours explored, $k' > k$), computes the distances of the found answers to the query and returns the k best results ().

The complexity analysis of the search by distance comparison is easy (Algorithm 4.6): the construction of the extended-key (line 7), using the Butz algorithm makes $\mathcal{O}(d_i m)$ bit operations. The whole operation, however is fast enough so that we can consider that this step takes constant time.

The time spent looking for the values nearest to the extended key (line 7) is dependent on the underlying data structure, but generally it can be assumed to take at most $\mathcal{O}(\log n)$ steps. This step is prone to be expensive, for here we are forced to make at least one random access to the data.

The complexity of steps 9–15 is known beforehand and grows linearly with the probe depth. The expensive operation here is the computation of the distances, which takes $\mathcal{O}(d)$ arithmetic operations, for the p-norm distances.

The dimensionality has also a "hidden" influence, in that it increases linearly the amount of data which must be transferred by the algorithm. This is especially important if we use secondary storage.

The whole operation (steps 2–16) is repeated once for every subindex, which means a linear growth with this parameter.

In summary, the time spent on the search by distance comparison grows linearly by the probe depth, the number of indexes and the number of dimensions in the data. It also grows logarithmically on the number of points in the database.

The analysis for rank aggregation is much more difficult, given that the *a priori* estimation of probe depth is very complex. Fagin et al. prove that, for MEDRANK, under several hypothesis and with high probability, the probe depth for $k = 1$ is of the order $\mathcal{O}\left(n^{1-2/(n_{lines}+2)}\right)$, where n_{lines} is the number of subindexes [Fagin 2003, § 2.1.3]. This estimative can be used as a pessimistic upper bound.

Once the probe depth is estimated, the analysis becomes straightforward: the step of finding the correct position in each curve is similar to the one in Algorithm 4.6 . Database lookup is proportional to $\log n$ and extended-key-computation takes $\mathcal{O}\left(d_i m\right)$ bit operations. Each hashtable lookup on the loop 15–22 can be performed in constant time, and the "hidden" influence of dimensionality in data transfer remains.

Summing up, the algorithm is linear with the probe depth and the number of dimensions in the data. It also grows sublinearly (but not logarithmically!) on the number of points in the database. The number of subindexes has an indirect influence, as the probe depth grows with it.

There is, however, a hidden constant, which depends on the value of k. As we will see in § 5.1.3, the probe depth grows quite fast on this parameter.

```
{  The parameters are explained in the definitions (page 109)
   and in Algorithm 4.5
   query[i] is the value of the ith dimension of query
   probeDepth is the number of data items to examine in each curve   }

SearchMulticurvesDist(n_curves, d[], A[],probeDepth, k, curves[], query)
   Returns a list of k nearest neighbours

1. best ← new empty sorted list of pairs <distance, point> sorted by distance
2. For curve ← 1 to n_curves do
3. projection[] ← array with d[curve] elements
4. For dimension ← 1 to d[curve] do
5.     projection[dimension] ← query[A[curve][dimension]]
6. Next
7. extended_key ← HilbertSpaceToKey(d[curve], m, projection)
                                              { see Algorithm 3.2 page 86 }
8. candidates ← list with the probeDepth points nearest to extended_key in
          the sorted list curves[Curve]
9. For each candidate in candidates do
10.    distance ← distance from candidate to query
11.    If distance < last distance in best then
12.        Put pair <distance, candidate> in best
13.    If best has more than k entries then
14.        Remove last entry in best
15. Next
16. Next
17. Return best
```

Algorithm 4.6: Searching the multicurves by comparison of distances

```
{  The parameters are explained in the definitions above (page 109)
   and in Algorithm 4.5  }

SearchMulticurvesRank(n_curves, d[], A[], m, k, curves[], query)
   Returns a list of k nearest neighbours

1. positionHi[] ← new array of n_curves integer values
2. positionLo[] ← new array of n_curves integer values
3. For curve ← 1 to n_curves do
4. projection[] ← array with d[curve] elements
5. For dimension ← 1 to d[curve] do
6.    projection[dimension] ← query[A[curve][dimension]]
7. Next
8. extended_key ← HilbertSpaceToKey(d[curve], m, projection)
                                          { see Algorithm 3.2 page 86 }
9. positionHi[curve] ← index of the nearest element to extended_key in
       the sorted list curves[curve]
10. positionLo[curve] ← positionHi[curve] - 1
11. Next
12. best ← new empty list of point
13. candidates ← new empty hashtable of pairs <point, votes> indexed by point
14. While number of points in best < k do
15. For curve ← 1 to n_curves do
16.    If positionHi[curve] < number of points in database then
17.       Check the point at positionHi[curve], if it is not in candidates,
             insert it with votes = 1. Otherwise, increment its votes. If
   votes
             reached n_curves/2, then puts the point on best.
18.       positionHi[curve] ← positionHi[curve] + 1
19.    If positionLo[curve] > 0 then
20.       Check the point at positionLo[curve], if it is not in candidates,
             insert it with votes = 1. Otherwise, increment its votes. If
   votes
             reached n_curves/2, then puts the point on best.
21.       positionLo[curve] ← positionLo[curve] - 1
22. Next
23. End while
24. Return best
```

Algorithm 4.7: Searching the multicurves by rank aggregation

```
{  The parameters are explained in the definitions above (page 109)
   and in Algorithm 4.5
   k' is the number of neighbours to explore  }

SearchMulticurvesMixed(n_curves, d[], A[], m, k, k', curves[], query)
   Returns a list of k nearest neighbours

1. best ← new empty sorted list of pairs <distance, point> sorted by distance
2. candidates ←
      SearchMulticurvesRank(n_curves, d[], A[], m, k, curves[], query)
3. For each candidate in candidate s do
4. distance ← distance from candidate to query
5. If distance < last distance in best then
6.    Put pair <distance, candidate> in best
7. If best has more than k entries then
8.    Remove last entry in best
9. Next
10. Return best
```

Algorithm 4.8: Searching the multicurves in the mixed mode

4.3.4 Parameterisation and storage needs

As for projection KD-forests, the most important parameter for the construction of the multicurves is the number of subindexes, which we propose to set as the highest value which efficiency will allow (the highest value for which we can still obtain a reasonable the search). Naturally, the actual value must be established empirically, and this will be done in the next chapter. As we have done for the KD-forests, the dimensions are simply partitioned equally among the curves.

The parameterisation for the search depends on the modality chosen. For rank aggregation, there are no parameters whatsoever to fix. In distance comparison, the probe depth must be decided *a priory*, and in mixed search, we must decide how many neighbours in excess we explore (k').

All three main parameters (number of subindexes, probe depth and factor of approximation) are a subject of empirical evaluation in the next chapter.

The storage needs for multicurves are simple to compute, and are strictly dependent on the number of subindexes, the size of the generated extended-keys and the amount of overhead needed by the data structure used in the management of the sorted lists. Let us consider, for simplicity sake, that this overhead is of a fixed factor f.

Each sorted list will store an entire copy of the data set, accompanied by the correspondent extended-key, which will have $d_i \times m$ bits. Considering that s_{point} is the size of a data point in bytes and that n is the number of data points in the data set, the total data space is given by:

$$s_{multicurves} = f \times n \times \left(\left(n_{curves} \times s_{point} \right) + \frac{\sum d_i \times m}{8} \right) \quad \text{bytes} \qquad \textbf{Eq. 4.9}$$

Looking at the big picture, Eq. 4.9 means that the storage overhead of the multicurves index is of order $\mathcal{O}\left(n_{curves}\right)$.

4.4 Conclusion

In this chapter we presented three new methods for approximate kNN search, considering the need to minimise the number of random access and avoiding complex pre-processing.

The first method, 3-way trees, is based on the use of overlapping regions which allow keeping the query away from the boundaries. In that way we can examine just one bucket of the tree (making just one random access) and still have good results. The method has good precision and fast searches, but presents an undesirable compromise between the bucket size (which we want to keep small, in order to examine few points) and the total index size. Therefore it is more adequate for moderately large database (10 million points or less).

To second technique, projection KD-forests, manages the boundary effects in a completely different way. It projects the data on different subspaces and creates a KD-tree for each one of those subspaces. Again, we visit just one bucket on each tree (making one random access per tree), and expect that boundary effects will be minimised for two reasons: each individual tree is less dimensional than the whole space, and we have several trees (and thus, a good chance that the query will fall in a favourable region in at least one of them).

The final method, multicurves, keeps the idea of partitioning the dimensions, but uses space-filling curves instead of KD-trees. The advantage is that the subindexes can now be stored on very efficient sorted list data structures, which means a more convenient implementation, and above all, the possibility of a dynamic index.

In the next chapter, we will evaluate the methods, and compare them to a few existing techniques from the literature.

Chapter 5

Experimental Results

In this chapter we present the results of our experiments, which comprise the evaluation of kNN search methods and the evaluation of the different image identification system architectures.

Our evaluation of the methods is empirical, using databases representative of the kind of data we are expecting to process. The experiments are presented in three different sections:

- In the first group, we are concerned purely with kNN search, with very little relation with the context of image identification. The only aspect of image identification on those experiments is the underlying database;
- In the second group, we keep focusing the kNN search but introduce the context of image identification by the bias of the ground truth;
- Finally, in the third group, we change our focus to the image identification system, instead of the kNN search component. We contrast global-descriptor with local-descriptor based systems, showing that the latter are much more effective.

5.1 kNN methods evaluation

5.1.1 Evaluation protocol

In this section we describe some important aspects of our experimentations in kNN methods evaluation, mainly the databases we have used and the measurements we have made.

5.1.1.1 Databases

A summary of the databases and query sets used in our experiments can be found at the end of this section, in Table 5.1.

In the literature, kNN methods are frequently evaluated with synthetic data, often uniformly distributed random points. However, uniform data are one of the hardest on which to perform kNN search — it may even be debated if the notion of kNN search is meaningful in high-dimensional uniform datasets (see § 3.2.5). Therefore, evaluating on uniform data leads to overpessimistic results, which may not correspond to how methods compare in the real world.

Our synthetic dataset, **Clusters**, is generated from Gaussian clusters distributed around uniformly spread centres. First we choose 50 000 centres from a uniform distribution in the space $\{0,\ldots,255\}^{128}$. Then, for each centre, we choose 20 points where the difference between the point and the centre, on each dimension, is randomly chosen from a normal distribution with zero mean and a standard deviation of 2. In that way, the final set has 1 000 000 points.

The queries are determined by choosing, at random, 1 000 of the same centres as the databases, and then drawing a random point using the same Gaussian distribution. This guarantees that the query points belong to the same clusters as the data points and that the search is meaningful.

Though this way of clusterising the data ensures that kNN search is feasible, it does not necessarily reflect the distribution of real-world databases. The hypothesis of Gaussian distortion between query and target points is challenged by Aggarwal, who argues that on high-dimensional data, a few components may present very large distortions [Aggarwal 2001].

For those reasons, the large majority of our experiments is conducted on databases composed of real local descriptors.

The **APM** database is composed of SIFT descriptors generated from image transformations of a selection of 100 original images. The images are old photographs (XIX and first half of XX centuries) from the collection of the Arquivo Público Mineiro, the State Archives of the Brazilian State of Minas Gerais [Cultura 2007]. Each image suffered three rotations, four scale changes, four non-linear photometric changes (gamma corrections), two smoothings and two shearings — a total of 15 transformations. Each transformed image had its SIFT descriptors calculated and those descriptors compose the database. The query points are the SIFT descriptors calculated from the original images. We have created a second, smaller, query set by selecting randomly 30 000 of those descriptors.

The **Yorck** database is composed of SIFT descriptors generated from the images of the Yorck Project, a collection of images of works of art put in public domain by the German company Directmedia Publishing [Directmedia 2008]. The images were downloaded from the Wikime-

dia Commons website [Wikimedia 2008], reduced to a smaller resolution, and had their SIFT points computed and put on the database. The queries are the SIFT descriptors calculated from images of the Yorck Project selected at random and then transformed. One hundred images were selected, of which, 20 were rotated, 20 were resized, 20 suffered a gamma correction, 20 were sheared and 20 suffered a dithering (halftoning).

The SIFT descriptors were computed using the binaries provided by Lowe [Lowe 2005], after reducing the images to ¼ of their original size. All image manipulations were done with the Netpbm package [Netpbm 2007].

The two aspects of the database which are most important to us are its size and its dimensionality. The latter, especially, affects critically the performance of the methods. The embedding dimensionality (d), i.e., the dimensionality of the space in which the points are immersed, is easy to understand and to verify, but does not correlate well with the observed performance of the methods. The so called "fractal" dimensionalities are more representative of the intrinsic dimensionality of the data, and indicate better the difficulty of the kNN search (see § 3.2.6).

APM had a d_0 of 17.7 ± 0.5 and a d_2 of 11 ± 2, while Yorck had a d_0 of 18.9 ± 0.3 and a d_2 of 11 ± 2. What this effectively means is that, despite the embedding dimensionality of those databases is of 128, the difficulty in performing kNN search in them should be roughly comparable to that of a database of uniformly distributed points of between 11 and 18 dimensions.

The analysis for Clusters is suspicious, since this synthetically generated base is all but self-similar. Its plot of $\log(S_p)$ versus $\log(r)$ is flat everywhere but on the smallest value of r, where the inclination is of 67.4 for d_2 and of 21.6 for d_0. This can be interpreted from two angles: on one hand, it hints that the fractal dimension cannot be reliably used on every database; on the other hand, it reinforces what we said before — that it is difficult to generate synthetic databases which are representative of real-world data.

Name	Data	# base points	# query points	Dimensionality d
Clusters	Synthetic random data	1 000 000	1 000	128
APM	SIFT descriptors	2 871 300	263 968 30 000	128
Yorck	SIFT descriptors	21 591 483	166 315	128

Table 5.1: Summary information on the databases and queries used in the evaluation of kNN search

5.1.1.2 Ground truth

The ground truth is the set of the correct nearest neighbours for all query points, according to the Euclidean distance. It was computed using the sequential search, a slow method, but which guarantees exact results.

The ground truth always has more neighbours than the value of k we are evaluating. This is to guarantee that if the k^{th} nearest neighbour is at the same distance to the query as the $k{+}1^{\text{th}}$, $k{+}2^{\text{th}}$ … $k{+}n^{\text{th}}$ neighbours (a not so rare occurrence, as we observed in our databases, probably due to the quantization of the space into 8 bits), any of them will be considered correct when computing the metrics.

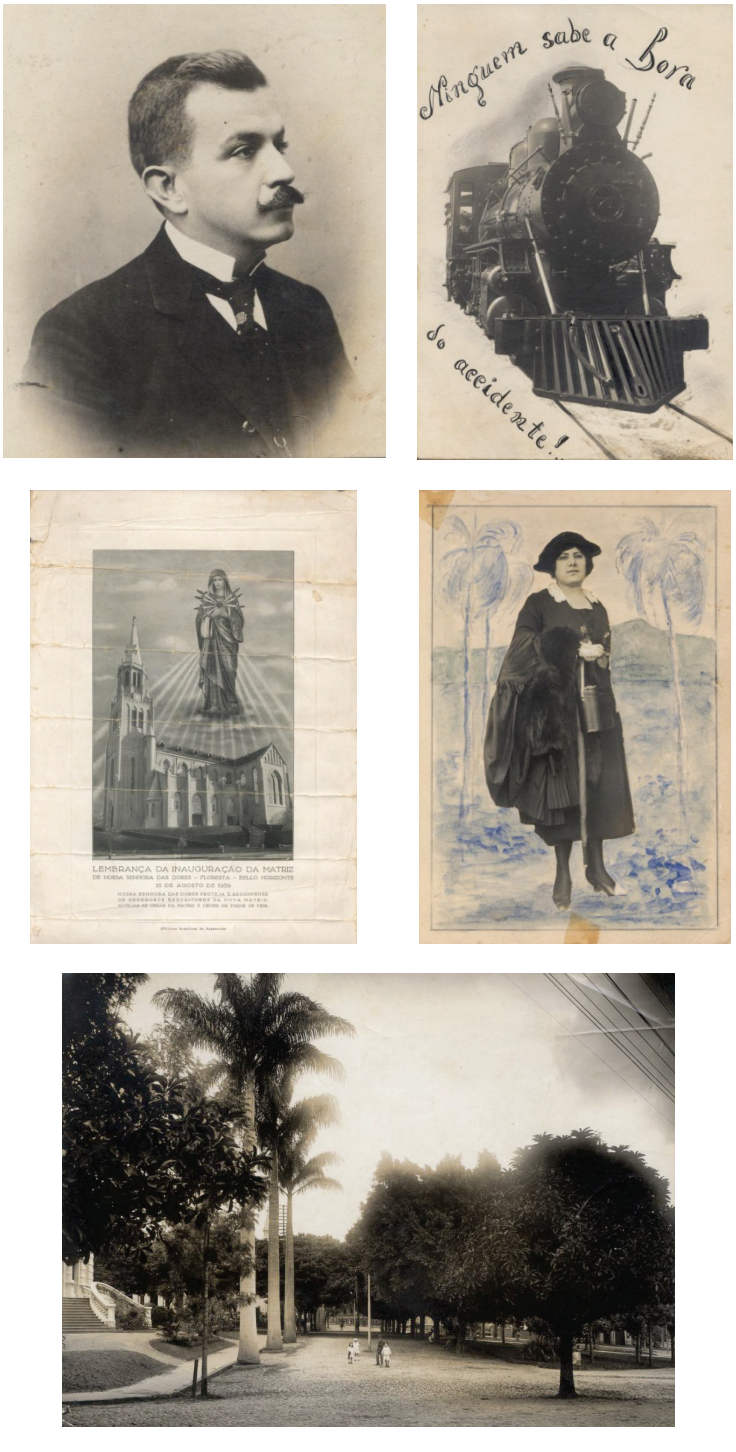


Figure 5.1: Some original images from the APM database of historical photographs

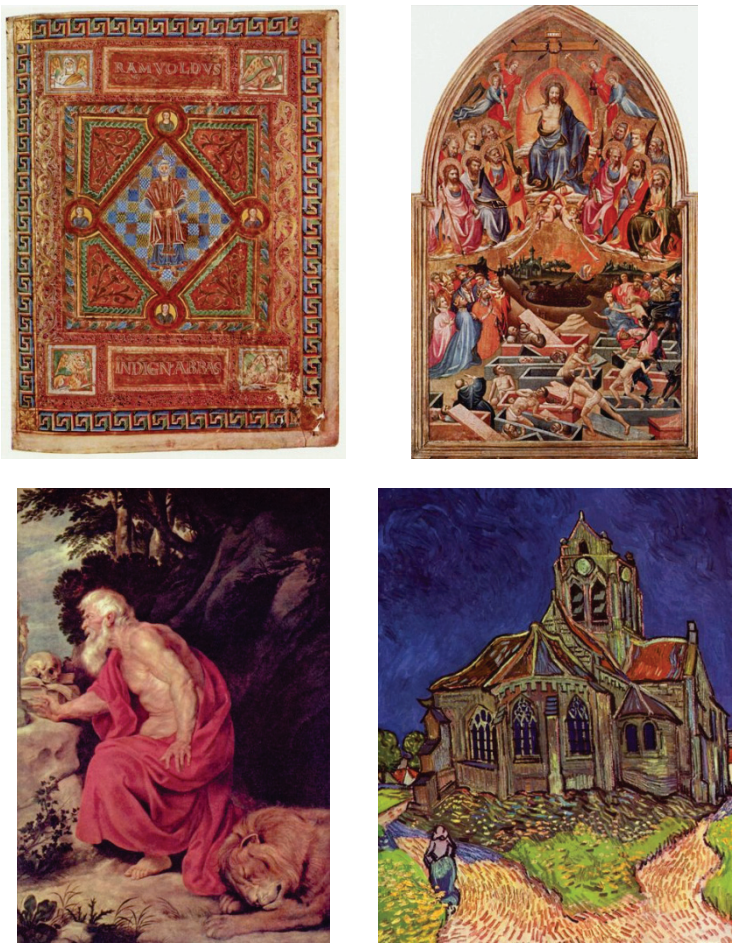


Figure 5.2: Some original images from the Yorck database of works of art

5.1.1.3 Efficacy metrics

Efficacy is the capability of the method to return the correct results, or a good approximation, in the case of the nearby kNN search. The efficacy metrics we use are defined on Table 5.2.

Symbol	Metric	Definition
RAE	Relative approximation error	$RAE = \mathrm{Avg}\left(\frac{1}{k}\sum_{i=1}^{k}\left\|\frac{\Delta(q,s_i)-\Delta(q,s'_i)}{\Delta(q,s_i)}\right\|\right)$
PFr	Probability of finding the r^{th} neighbour	$PF_r = \Pr\left[\exists s'_i \in S' : \Delta(q,s'_i) = \Delta(q,s_r)\right]$
P@k	Fraction of correct neighbours = Precision at rank k	$P@k = \mathrm{Avg}\left(\mathrm{F}(k)\right)$, where $\mathrm{F}(r) = \frac{\#\text{ of correct } r\text{-nearest neighbours found}}{r}$

Table 5.2: Metrics of efficacy for kNN search: q is the query point, $s'_i \in S'$ and $s_i \in S$ are the i^{th} element from, respectively, the ordered set of the found and the true nearest neighbours, the averages are taken over the queries performed on a given experiment

The approximation error (RAE) is related to the notion of nearby kNN search (since it measures the average value of the ε obtained), while the fraction of correct neighbours ($P@k$) is re-

lated to the notion of probabilistic kNN search (measuring the average value of the α obtained). Therefore, the usefulness of those measures will depend on which approximation notion (geometric or probabilistic) is more adequate for the intended application.

Neither of those metrics, however, takes into consideration *which* of the neighbours were found. The probability of finding the r^{th} neighbour (*PFr*) may be used when one of the neighbours (normally the first — *PF1*) is more important than the others. The definition we provide may seem unnecessarily complicated, but it is explained by the fact that, when multiple neighbours are at the same distance from the query, any of them is an acceptable r^{th} neighbour.

5.1.1.4 Efficiency metrics

Efficiency is the capability of the method to use as little resources as possible: processor time, main memory, disk space, etc. The most important efficiency metrics are explained on Table 5.3.

Symbol	Metric	Definition
	Wall time	Real time spent on operation (as measured by the "user's watch")
	Processing time	Machine time spent on operation (considers multiple processors, multitasking, etc.)
	Comparisons made	Number of comparisons between distances
	Distances computed	Number of distance computations
PD	Points examined Probe depth	Total number of points examined on each index
DT	Data transferred to/from disk	Amount of data read from or written to disk, sequentially or not
Seeks	Random seeks on disk	Number of non-sequential movements of to the disk i/o heads
	Main memory needs	Total space occupied by the method on primary memory
Size	Secondary memory needs	Total space occupied by the indexes on the disk

Table 5.3: Metrics of efficiency for kNN search

From the point of view of the user, the most critical efficiency metric is the wall time spent on the search, but using it to compare the methods may be misleading, since it depends heavily on the machine, the operating system, and even on the current load (concurrent tasks) at the time the experiment is performed. Processing time, which takes into consideration the concurrent tasks, may be more useful for comparisons, but is still much too dependent on the system. Since it is usually obtained through system calls, the measurement may, by itself, vary from one system to another. For example, weather i/o waiting time counts or not on this metric is very important for its meaningfulness, and especially, for its comparability.

We choose, therefore, to compare the methods by counting how many operations they perform among those more significant: random accesses on the disk, distance computations, comparisons, points examined, and so on.

Some operations are much more expensive than others and end up taking most of the time. Methods which use secondary memory, for example, end up by spending most of the time performing i/o, instead of computation. If both random and sequential accesses are performed, the first will quickly dominate the time spent. However, measuring only the most expensive operation can be misleading, especially because different machines have different costs for each operation.

Very often in the literature, only the time-related metrics are measured, while the space occupancy is ignored. It is also common to measure only the aspects related to the query operation, ignoring the construction of the indexes. However, for very large databases those factors cannot be ignored.

5.1.2 KD-trees *versus* 3-way trees

In this section we evaluate the 3-way tree (§ 4.1) to the method in which it is based, the KD-tree (§ 3.5), in order to verify how much improvement we obtain.

To compare the methods, we used the APM database (with the full query set) and the Clusters database. We stipulated that the methods would spend the same number of distance computations, distance comparisons, random seeks and data accesses (i.e., the same efficiency), and then compared the efficacy.

The experiment consisted in finding the 20 nearest neighbours of each query descriptor. This value of $k = 20$, somewhat arbitrary, corresponds to the 15 transformations in the base APM, and a small extra margin. The efficacy metrics were the probability of finding the first neighbour (*PF1*), the fraction of correct answers among the 20 (*P@20*) and the relative approximation error (*RAE*).

Since the performance of both methods should be dependent on the bucket size (which determine the number of points which get to be examined), we varied this parameter, checking the performance obtained at each value.

A summary version of these results was published in [Valle 2007a] and [Valle 2007b].

5.1.2.1 Implementation details

We compared the 3-way trees with our own implementation of the KD-tree, both using the interquartile range to determine the dimension to divide the space, and the median on this dimension as the pivot (see § 4.1).

In both methods, just the first bucket was explored, in order to allow only one random access, thus keeping query times the same.

All methods were implemented in Java, using the Java Platform, Standard Edition v. 1.6. We have, naturally, implemented our method (3-way trees) but also reproduced the algorithm for the KD-tree, with the necessary modifications to use the interquartile range as a criterion of splitting. We also implemented the necessary classes to analyse the results and obtain the efficacy metrics.

5.1.2.2 Results

In the experiments we have set the maximum bucket sizes to 1 024, 2 048, 4 096, 8 192 and 16 384 descriptors. The actual bucket size was indicated in the horizontal axis of the graphs. Remember that the bucket size also correspond to the probe depth (*PD*) and distances computed, per query. The efficacy metrics are shown on the vertical axes of the graphs.

The 3-way trees showed very good performance on both databases, being able to find the first neighbour almost always, and, in almost all instances, more than half the correct neighbours among the 20. The fraction of correct neighbours returned (*P@k*) varied from 49% to 64% in the APM database (Figure 5.3) and from 85% to 87% in the Clusters database (Figure 5.5). As for the probability of finding the first neighbour, it was at least 99% in all instances.

It is interesting to compare the results on the two bases. The fraction of correct points is much higher in Clusters than in APM. This suggests an association to the observation of Aggarwal that the norm-based distances (like the Euclidean distance, used here), work better when the difference between the query and target points is composed of several small fluctuations spread among many dimensions (see § 3.3.5.2, [Aggarwal 2001]). If we accept this hypothesis, then Gaussian clusters are not a good model for the SIFT descriptors.

It is somewhat surprising to see that the approximation errors are much higher in Clusters, but this can be explained by the fact that for each query point in this synthetic database, there are precisely 20 target points at a reasonable distance. If we miss any one of those points, the next candidate can be anywhere in the space, since the cluster centres are uniformly distributed.

The efficacy of both methods grows with the size of the bucket, which is not surprising, since more descriptors get to be examined.

Even the worst case of the 3-way tree showed more efficacy than the best case of the KD-tree. For the APM database, there was a relative increase of at least 30% in *P@k*, and of 8–15% for *PF1*. In the Clusters database, the most obvious improvement was the reduction of the approximation error (*RAE*), where the values for the KD-tree where at least 87% (and up to 320%) higher than the ones for 3-way tree.

All data show that, when keeping the same query times, the 3-way tree is much more precise.

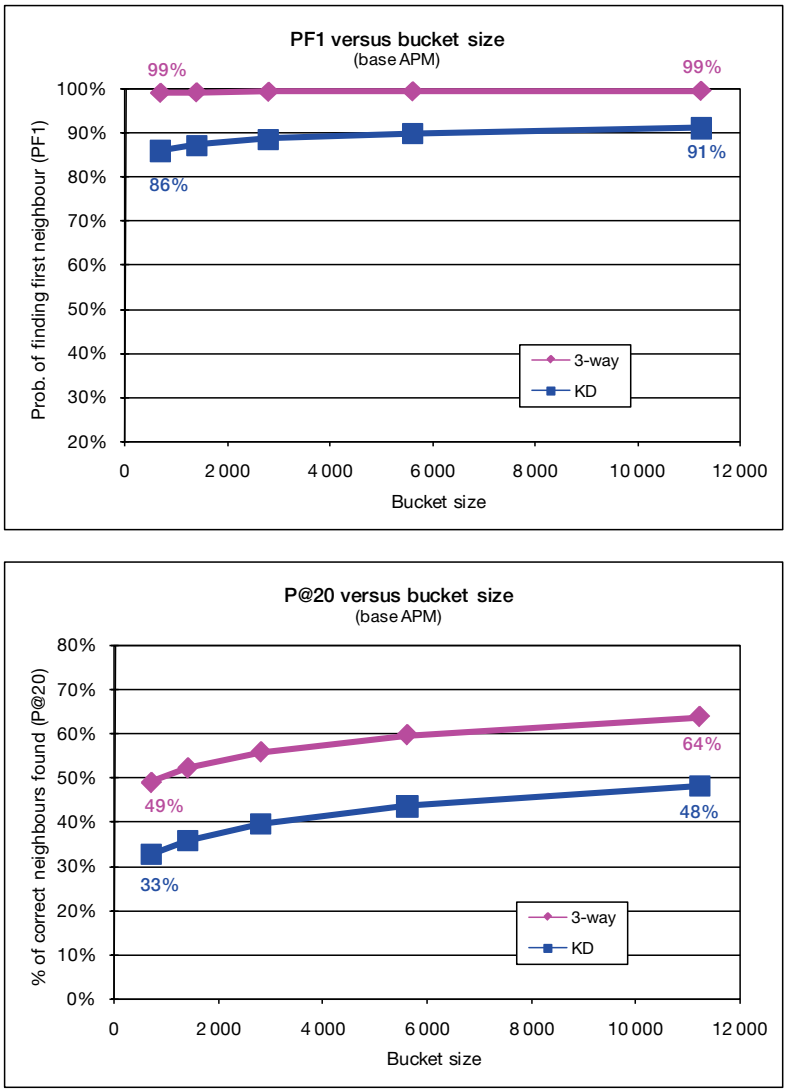


Figure 5.3: Comparison between the 3-way tree and the KD-tree — efficacy metrics related to the probabilistic kNN search (*PF1* and *P@20*) on the APM database, versus the size of the bucket

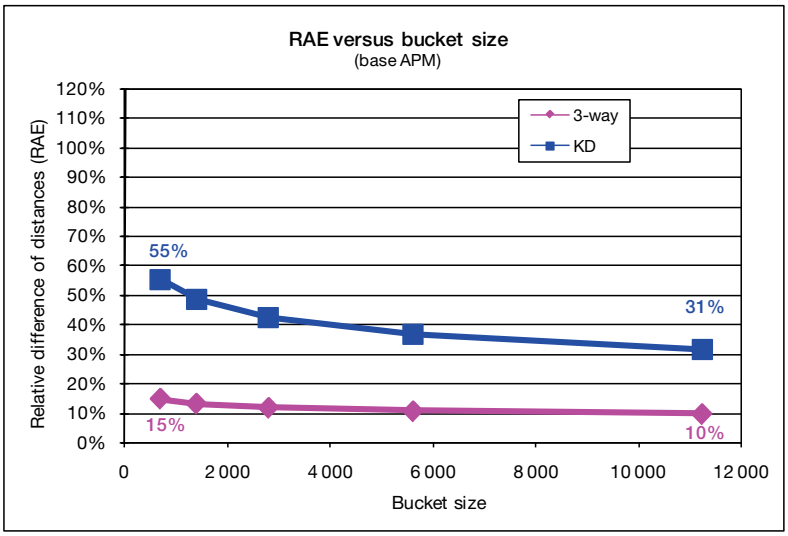
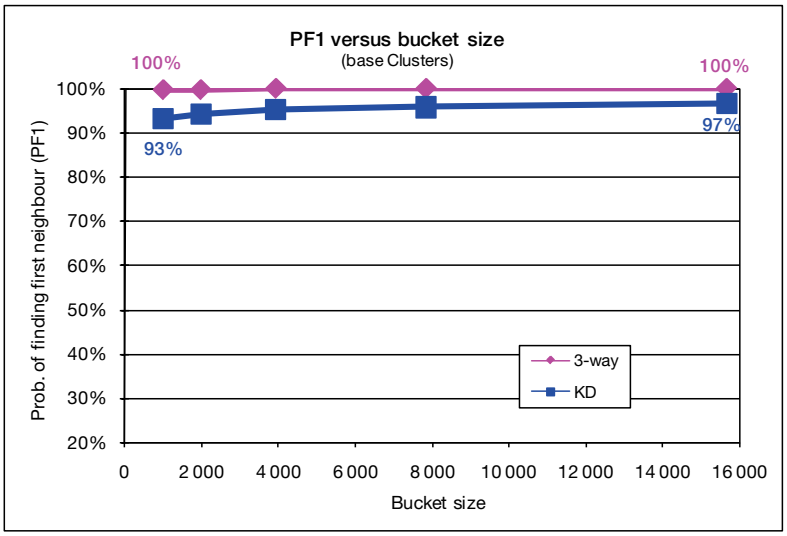


Figure 5.4: Comparison between the 3-way tree and the KD-tree — efficacy metric related to the nearby kNN search (*RAE*) on the APM database, versus the size of the bucket



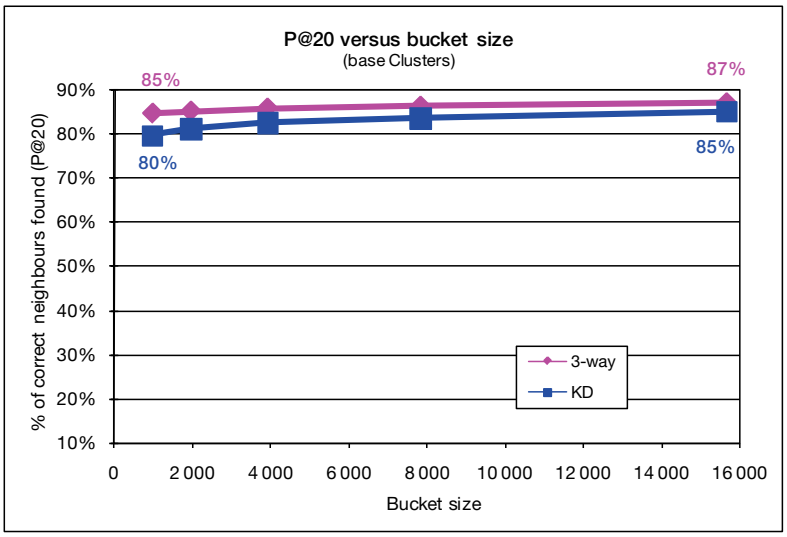


Figure 5.5: Comparison between the 3-way tree and the KD-tree — efficacy metrics related to the probabilistic search (*PF1* and *P@20*) on the Clusters database, versus the size of the bucket

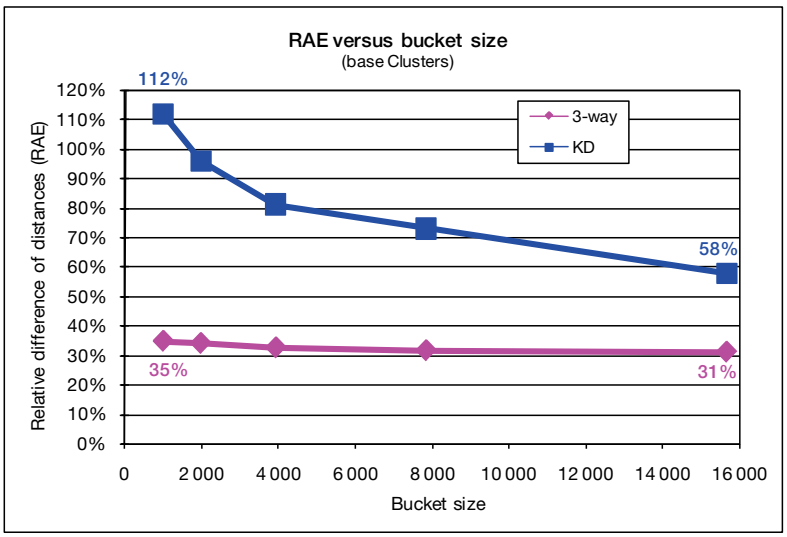
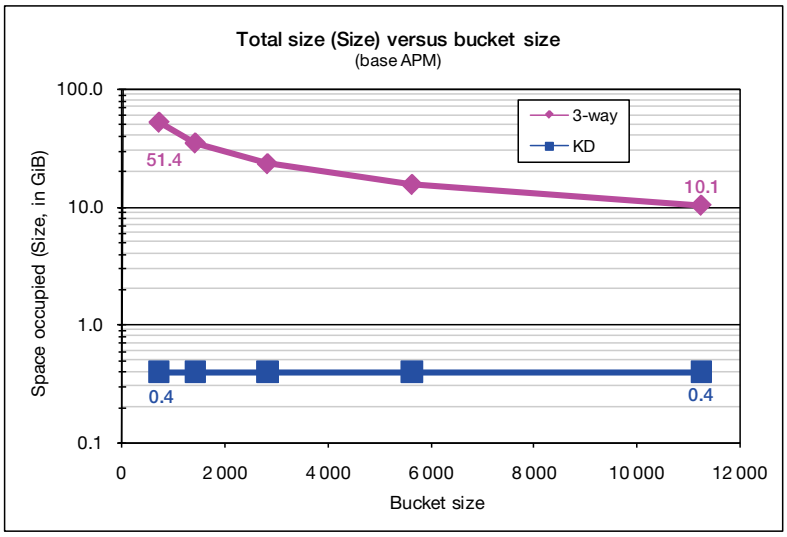


Figure 5.6: Comparison between the 3-way tree and the KD-tree — efficacy metric related to the nearby kNN search (*RAE*) on the Clusters database, versus the size of the bucket

5.1.2.3 Index size and construction

The 3-way tree presents much more efficacy than the KD-tree for the same querying times, but this is obtained by an increase in the index size, which is occupied by the redundant data. The relationship between the index size and the bucket size can be seen on Figure 5.7 and confirms our theoretical discussion of § 4.1.4 — a steep growth of the index as the bucket size shrinks. The time spent on the construction of the index is roughly proportional to its size.

As we have discussed before, the double compromise between the index size and the database size / bucket size makes the 3-way tree inadequate for databases much larger than 10 million descriptors (see § 4.1.4). Though enlarging the buckets contains the index growth (with the additional advantage of more precise results), that makes the search operation slower, since all points in the bucket have to be examined.



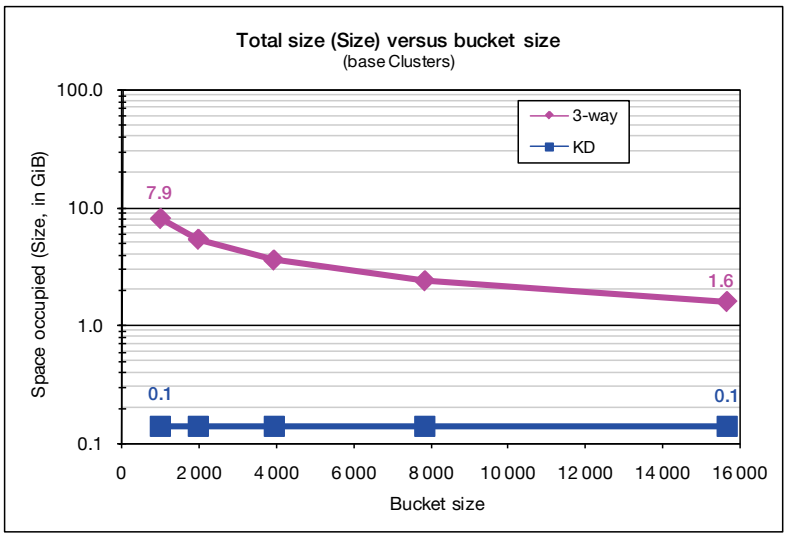


Figure 5.7: Comparison between the 3-way tree and the KD-tree — space usage

5.1.3 Projection KD-forest *versus* multicurves *versus* OMEDRANK

In this section we compare three strategies of kNN search based on the projection of the points on multiple subspaces: the algorithm OMEDRANK (§ 3.8) from the literature and our algorithms projection KD-forest (§ 4.2) and multicurves (§ 4.3).

OMEDRANK projections are one-dimensional subspaces (straight lines). Projection KD-forests and multicurves use n-dimensional subspaces. OMEDRANK and multicurves can use three modalities of search (distance comparison, rank aggregation and mixed strategy — see §§ 3.8.2 and 4.3.3), while KD-forests can only use distance comparison.

For those experiments we used the APM database (with the randomly selected query of 30 000 points). The experiment consisted in finding the 20 nearest neighbours of query descriptor. This value of $k = 20$, somewhat arbitrary, corresponds to the 15 transformations in the base APM, and a small extra margin. The efficacy metrics were the probability of finding the first neighbour (*PF1*), the fraction of correct answers among the 20 (*P@20*) and the relative approximation error (*RAE*). When using rank aggregation, no actual distances are computed, so we did not evaluate the approximation error in this modality.

The main efficiency metric is the probe depth per query (PD). Consider also that:

- Each one of the subspaces used by the method forms a *subindex* or a *voter* (this latter designation inspired on the theoretical background of rank aggregation);
- All methods make one random access per query per subindex (e.g., a KD-forest with four trees will make four random accesses per query);
- When using the search modality of distance comparison, the number of distances computed is roughly equal to the probe depth. There is a slight difference, because when a point is found in several subindexes, the distance is only measured once, but these are rare occurrences at the small probe depths used on this search modality;
- When using the mixed search modality, even if we are searching k nearest neighbours, the actual number of neighbours searched (which we call neighbours explored) will be given by another parameter (k') which also corresponds to the number of distances computed per query.

The most important parameter for all the methods is the number of subindexes, so we evaluated how the performance changed accordingly to this parameter. We do not explore the behaviour of the methods on a large number of subindexes (beyond 8), because we want to keep the search operations fast, and each subindex introduces a new random access and the augmentation of the probe depth.

In the modality of rank aggregation we also evaluated how the number of neighbours searched affected the performance, since the efficiency of the method is very dependent on this parameter. In the modality of mixed search, which is an extension of rank aggregation, we evaluated the influence of the number of neighbours explored.

5.1.3.1 Implementation details

All methods were implemented in Java, using the Java Platform, Standard Edition v. 1.6. Besides our own methods (multicurves, projection KD-forests) we also reimplemented

OMEDRANK (following the precise specifications in [Fagin 2003]), and the KD-tree (§ 3.5). The single Hilbert curve corresponds to a multicurve with one curve representing the entire space. We also implemented the necessary classes to analyse the results and obtain the efficacy metrics.

On the KD-tree, the buckets have 351 descriptors on average. When searching the KD-tree, we examine just the one bucket where the query falls, thus having only one random access per query.

On the Hilbert method, we create a single Hilbert curve with all the 128 dimensions of the descriptors. Then we examine the 512 descriptors which are nearest to the query in the curve.

On OMEDRANK and multicurves, three modalities of search were executed: by comparison of distances, by median rank aggregation and by combination of the two.

For the search modality of distance comparison, the probe depth per query is decided *a priori* as 512 descriptors per line (or curve, or tree). It is worthy to note that the actual number of descriptors examined on the projection KD-forest is slightly less: 351 descriptors on average. We examined the performance of the methods for a growing number of subindexes.

For multicurves and projection KD-forests, the dimensions are assigned by a simple partition of contiguous blocks. The process is better explained through a concrete example: for our 128-dimensional descriptors, if we have four subindexes, we will allocate the dimensions 0 to 31 to the first subindex, the dimensions 32 to 63 to the second one, the dimensions 64 to 95 to the third and the dimensions 96 to 127 to the last one. This will result in four 32 dimensional subindexes. When the exact division cannot be performed (which happens for 6 subindexes), we try to keep the division as balanced as possible.

All trees used the interquartile distance to choose the splitting dimension and the median as the pivot.

5.1.3.2 Results on the efficacy and efficiency of the search

We start by comparing two different search modalities (rang aggregation and distance comparison) for the two methods which are compatible with both modalities (i.e., multicurves and OMEDRANK).

Multicurves presented good performances in all search modalities: rank aggregation, distance comparison and mixed mode. The comparison between the two former modalities can be seen in the Figures 5.8–5.9. The results for mixed mode are on a separate plot (because we varied a different parameter) and can be seen in Figures 5.16–5.17. The best results were in the modality of distance comparison, were for small probe depths, it typically found half the correct neighbours and almost always found the first neighbour correctly

Thus, unless in a particular architecture the computation of the distances is much more expensive than the access to the descriptors, the modality of distance computation is to be preferred.

On the other hand, OMEDRANK, which was originally conceived to work with rank aggregation, does not adapt well to distance comparison. The comparison between those two modalities can be seen on Figure 5.10 and Figure 5.11. Even if rank aggregation has very large probe depths (at least 10 000 descriptors and, for 8 subindexes, up to 11% of the database), efficacy is so low in the modality of distance comparison, that it rules out its use.

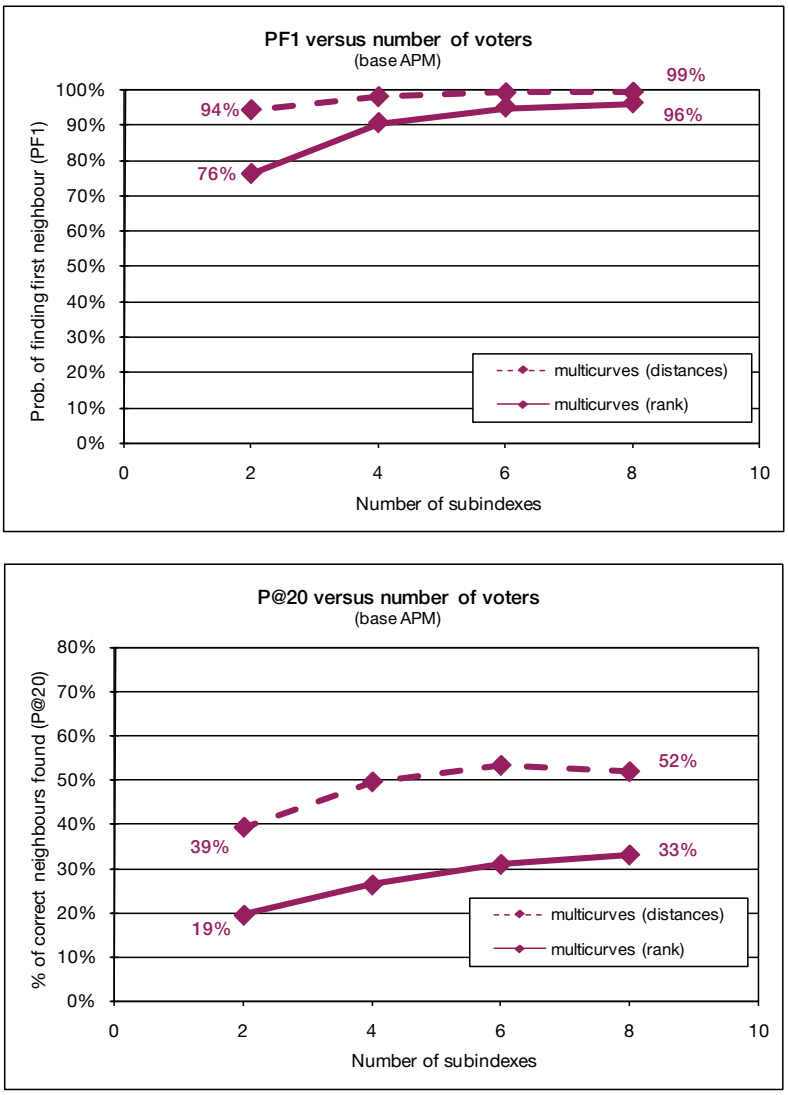


Figure 5.8: Comparison between two different modalities of multicurves, on the APM database, versus the number of subindexes — efficacy metrics

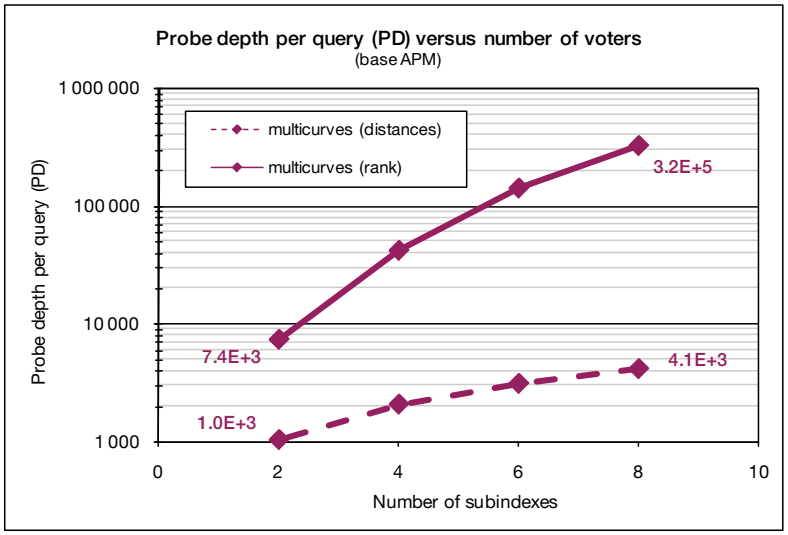


Figure 5.9: Comparison between two different modalities of multicurves, on the APM database, versus the number of subindexes — efficiency metrics

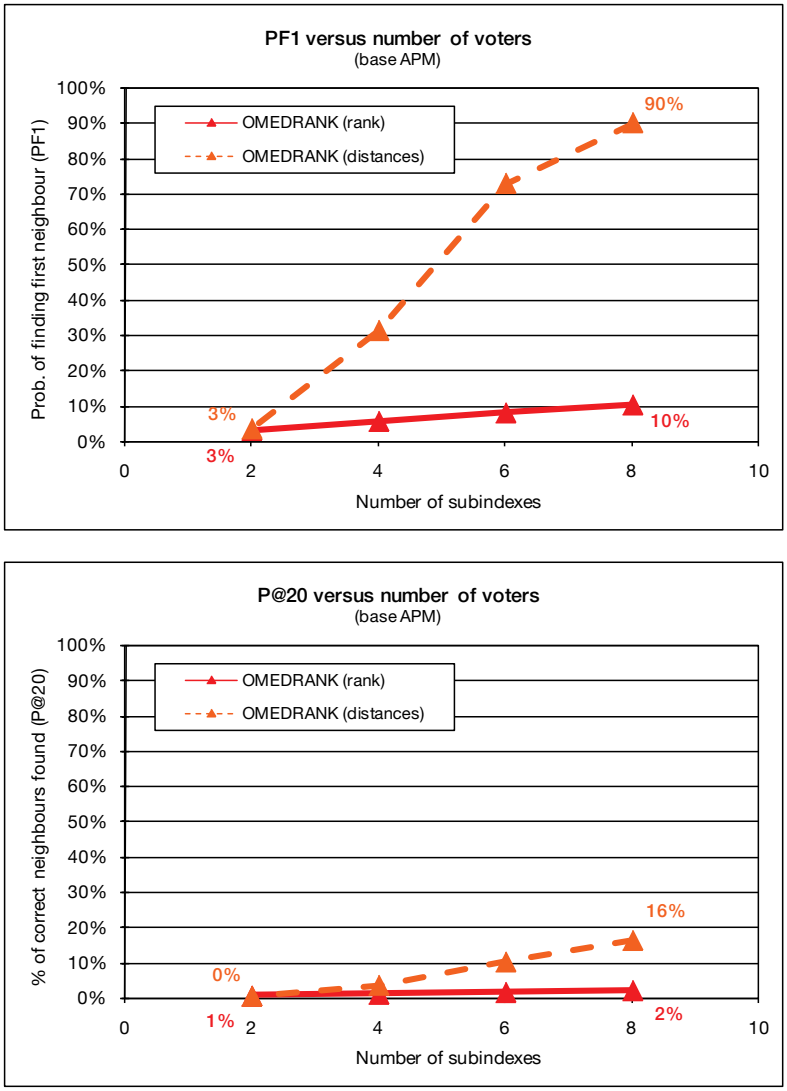


Figure 5.10: Comparison between the different modalities of OMEDRANK, on the APM database, versus the number of subindexes — efficacy metrics

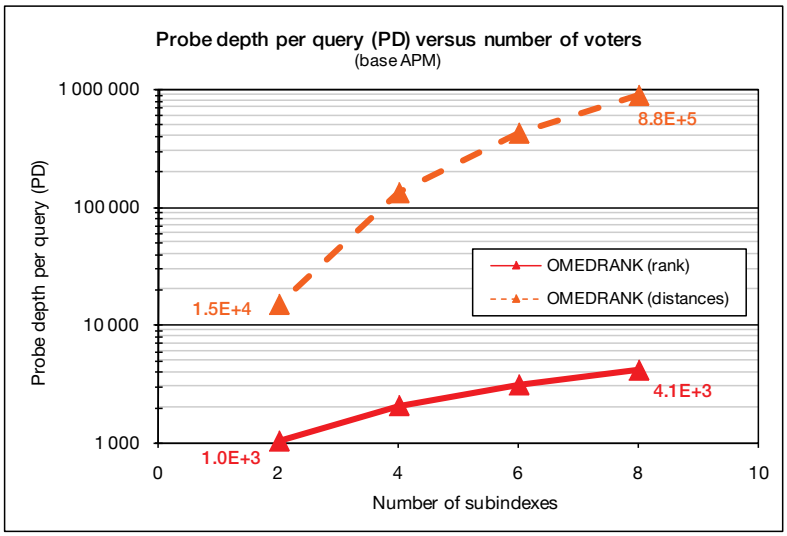


Figure 5.11: Comparison between two different modalities of OMEDRANK, on the APM database, versus the number of subindexes — efficiency metrics

In both methods, for the modality of rank aggregation, both efficacy and efficiency depended much on the number of neighbours searched (the parameters k), which is shown on Figure 5.14 and Figure 5.15.

The rapid growth of the probe depth versus k puts a heavy penalty on mixed search modality (a variation on rank aggregation were we search more neighbours than we actually need, and then sort them by distance). As Figures 5.16–5.17 show, for OMEDRANK there was an almost linear compromise between precision and probe depth — double the latter to double the former. But since the probe depth values were already high in rank aggregation, mixed mode does not seem an interesting alternative — if we are willing to increase the probe depth, augmenting the number of subindexes seems to have more impact in the quality of OMEDRANK. As for multicurves, the efficacy gains were too little in this modality, while the efficiency penalties where heavy.

In all experiments, whatever the modality of search chosen, the performance of multicurves was considerably better than that of OMEDRANK (Figures 5.12–5.17).

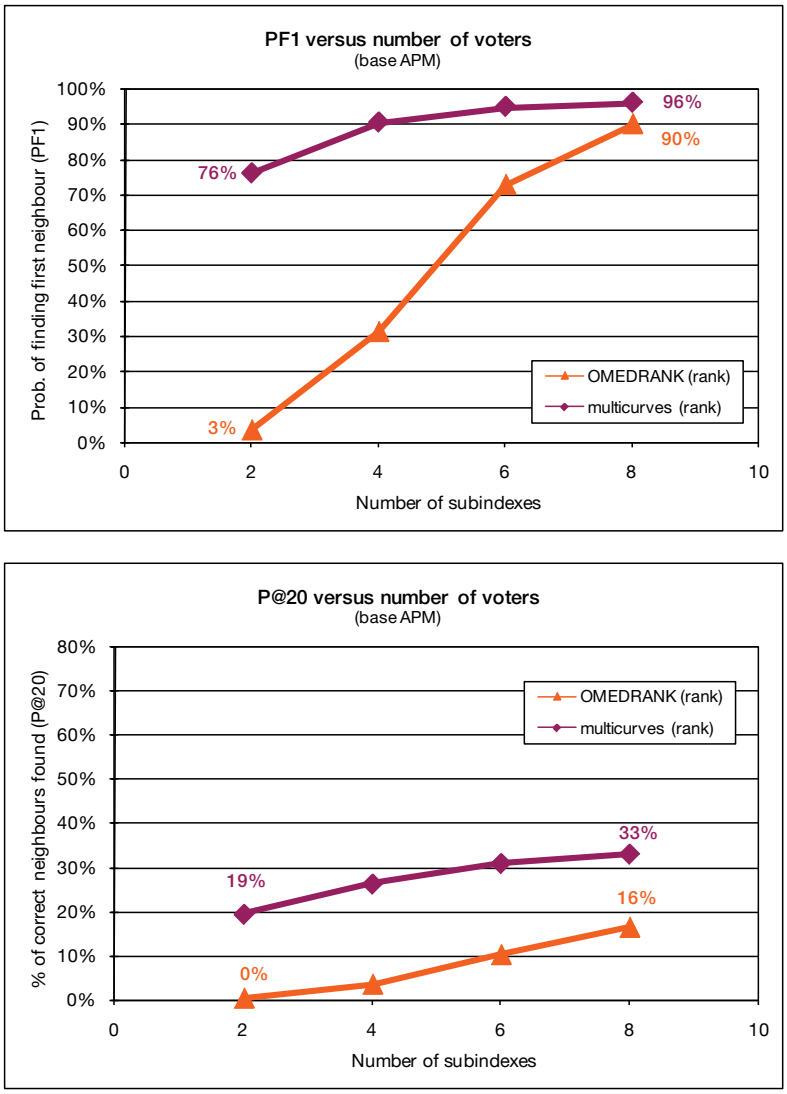


Figure 5.12: Comparison between OMEDRANK and multicurves on the APM database, in the rank aggregation modality, for k = 20 Performance versus the number of subindexes — efficacy metrics

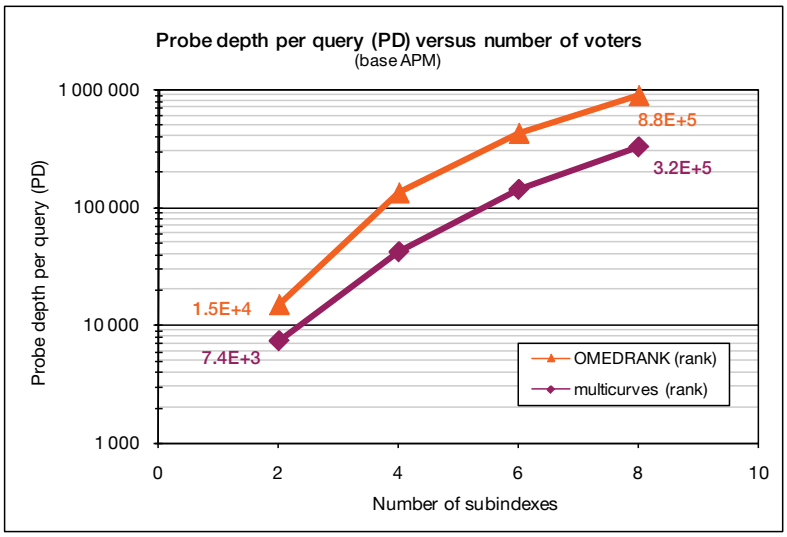


Figure 5.13: Comparison between OMEDRANK and multicurves on the APM database, in the rank aggregation modality, for k = 20 Performance versus the number of subindexes — efficiency metrics

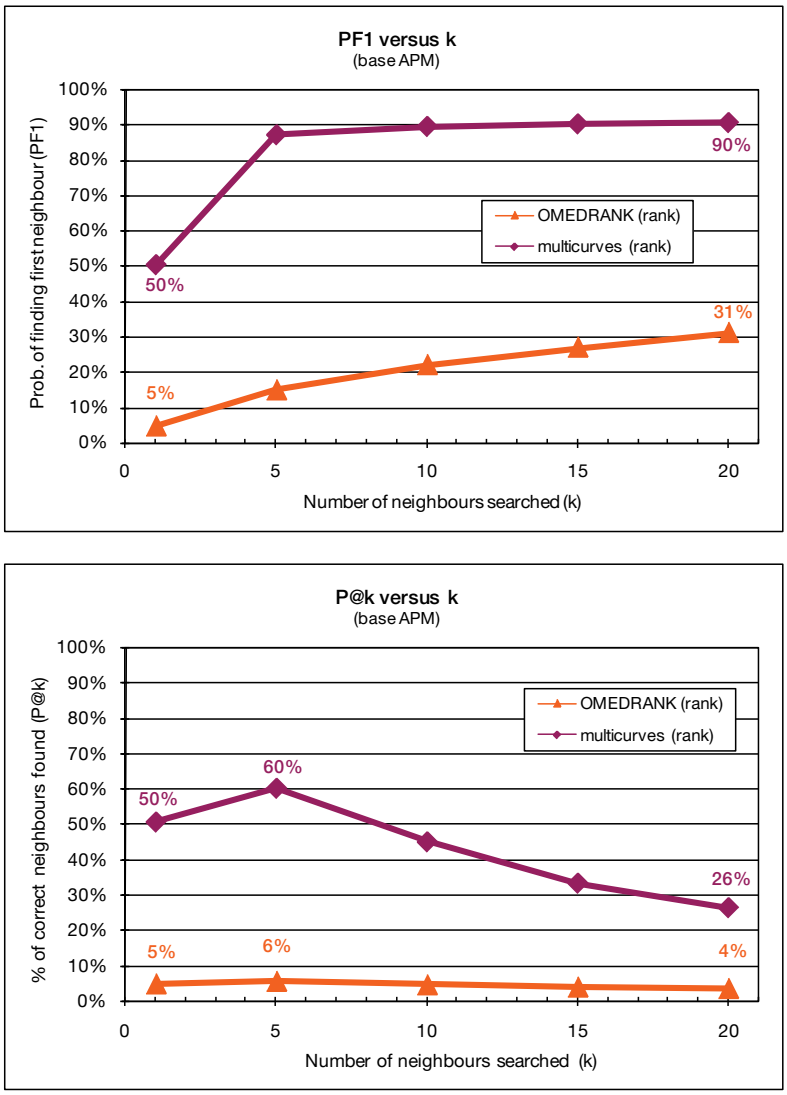


Figure 5.14: Comparison between OMEDRANK and multicurves on the APM database, in the rank aggregation modality, using 4 subindexes — efficacy metrics

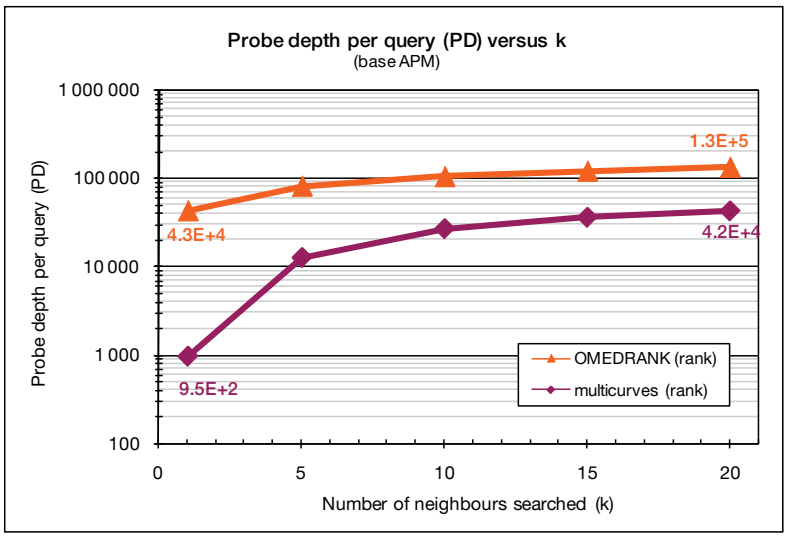


Figure 5.15: Comparison between OMEDRANK and multicurves on the APM database, in the rank aggregation modality, using 4 subindexes — efficiency metrics

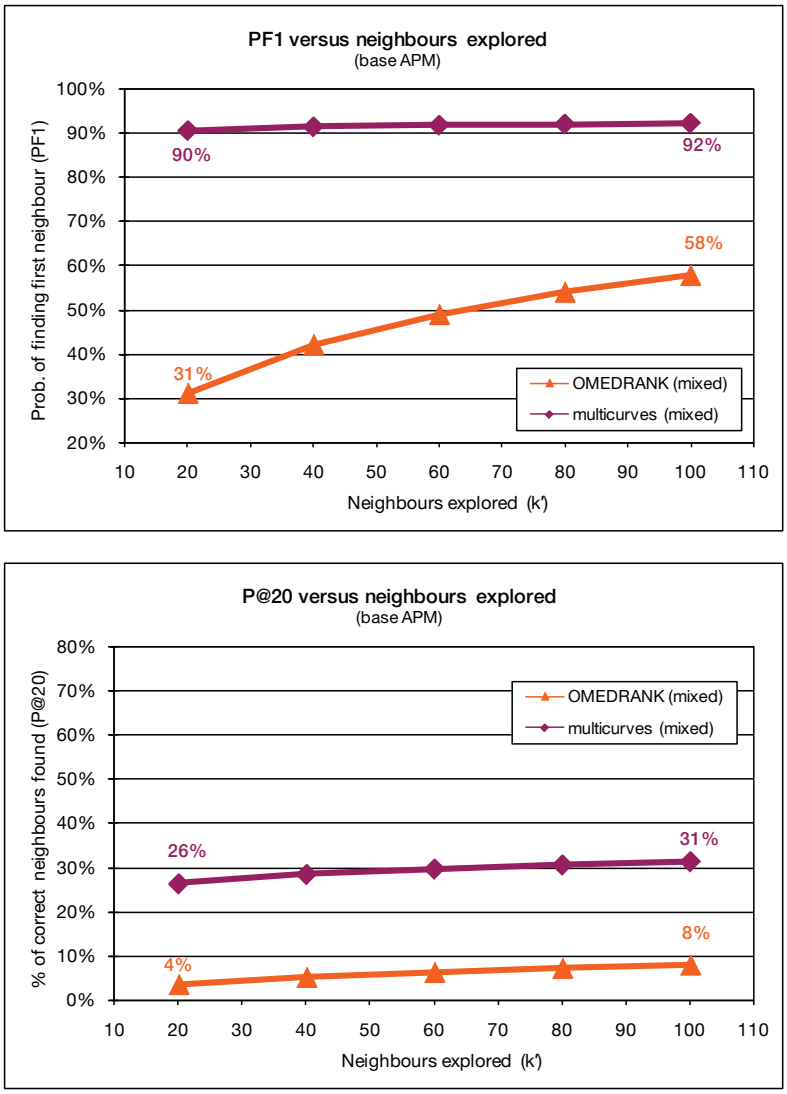


Figure 5.16: Comparison between OMEDRANK and multicurves on the APM database, in the mixed search modality, for k = 20 Performance versus the neighbours explored (k') — efficacy metrics

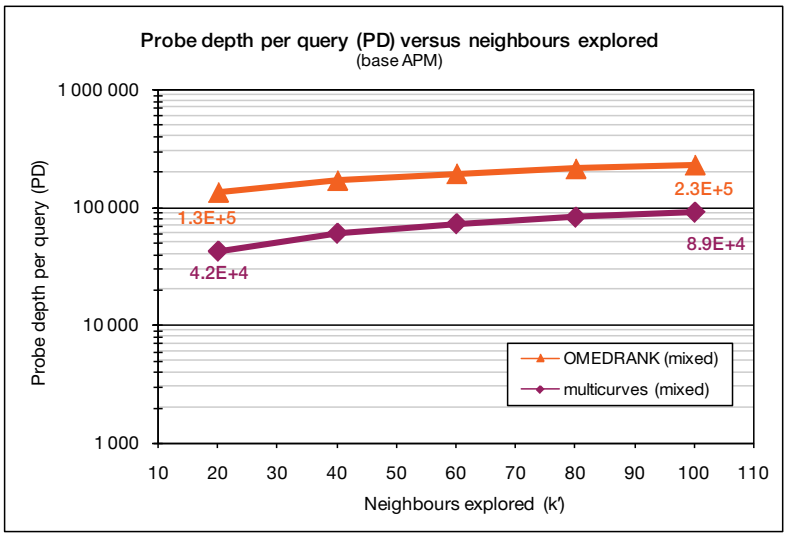


Figure 5.17: Comparison between OMEDRANK and multicurves on the APM database, in the mixed search modality, for k = 20 Performance versus the neighbours explored (k') — efficiency metrics

The probe depths found on the rank aggregation modality were generally too high. Distance comparison seems to offer a better efficiency, and, though OMEDRANK did not adapt well to it, multicurves benefited greatly from this modality. The bad adaptation of OMEDRANK is due to the fact that a very large number of descriptors coincide exactly with the query in a single straight line, due to the fact the space if quantised. A probe depth of 512 is not nearly enough to discriminate among the projections. The curve, even with the problem of boundary effects, is more able to tell the descriptors apart.

In our next study (Figure 5.18 and Figure 5.19), which comprised all methods in distance comparison, we included the projection KD-forests (which are incompatible with rank aggregation or mixed search). They performed surprisingly well, especially considering they have a smaller probe depth (and thus perform less distance computations) than the other methods.

The comparison between KD-tree versus KD-forest, and Hilbert curve versus multicurves shows that the use of multiple subindexes improves considerably the efficacy. The improvement is particularly dramatic on the approximation error.

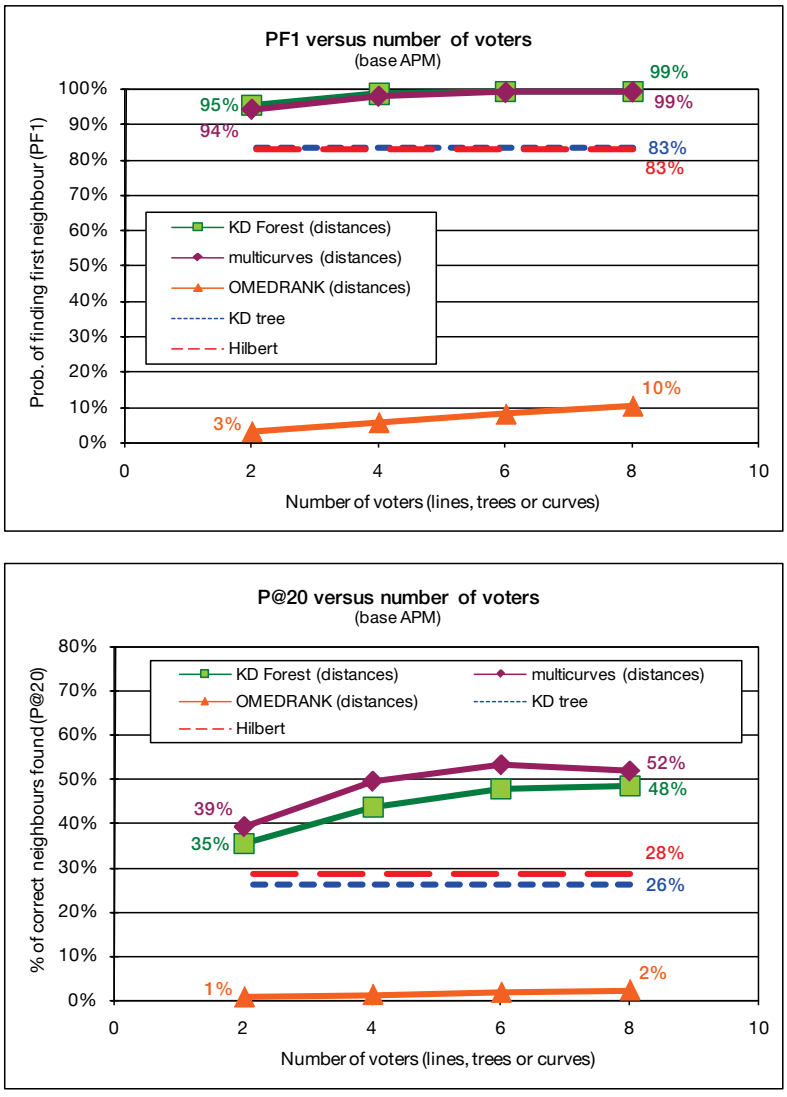


Figure 5.18: Comparison between OMEDRANK and multicurves on the APM database, in the mixed search modality, for k = 20 Performance versus the neighbours explored — efficacy metrics

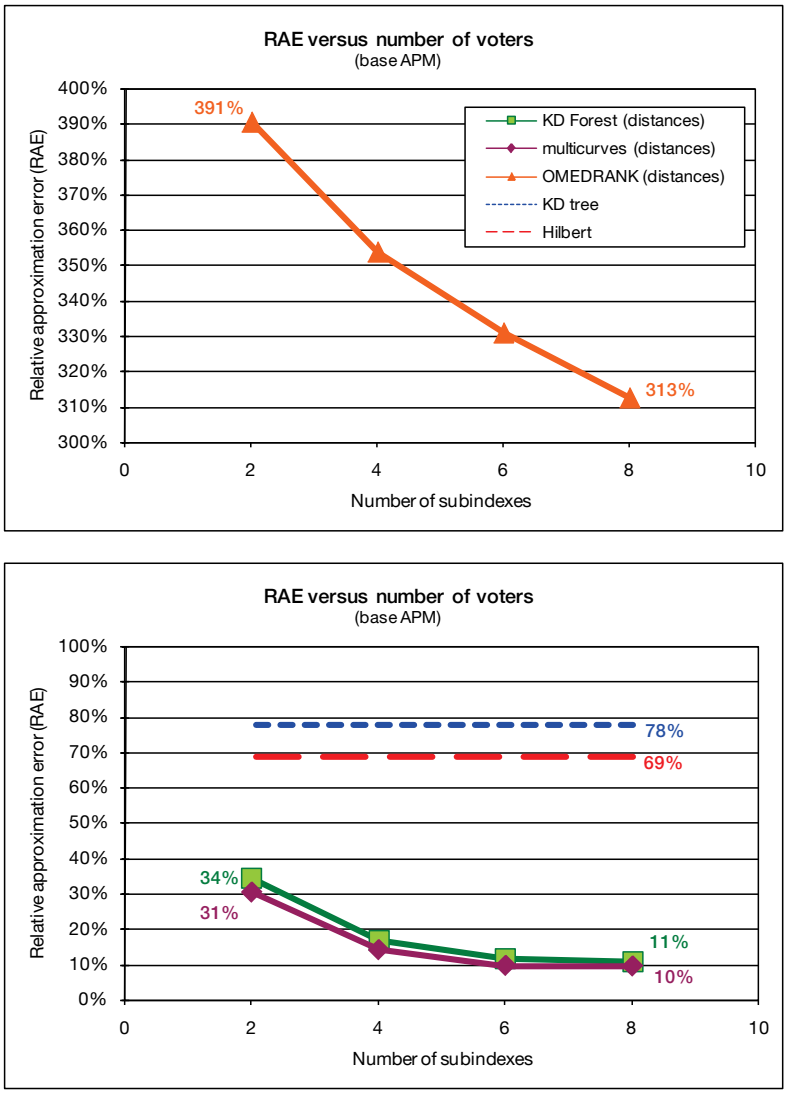


Figure 5.19: Comparison between OMEDRANK and multicurves on the APM database, in the mixed search modality, for $k = 20$ Performance versus the neighbours explored — efficacy metrics. (The results are presented in two separate graphs, to avoid compressing excessively the vertical axis)

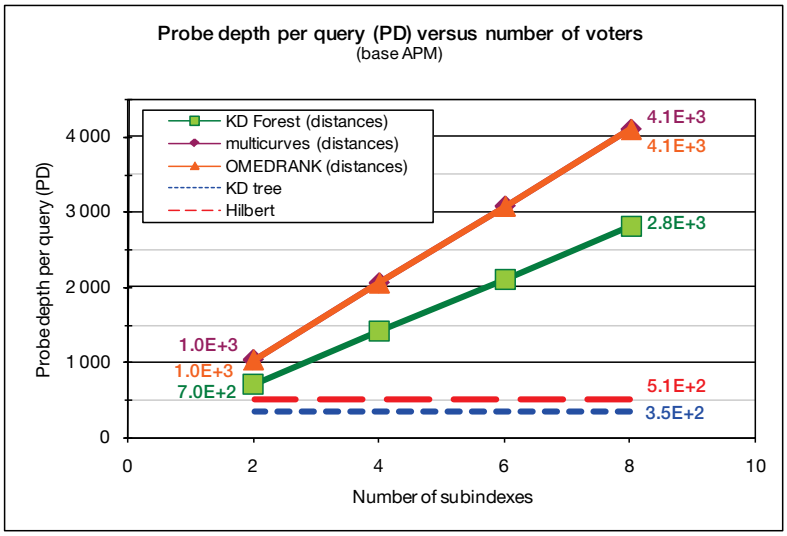


Figure 5.20: Comparison between the OMEDRANK, multicurves and projection KD-forest. Probe depth versus the number of subindexes. Using distance comparison and $k = 20$. The performance of the KD-tree and of the single Hilbert curve is indicated on dashed lines, for reference

In all experiments, the multicurves performed better than OMEDRANK, showing at least twice the efficacy and the efficiency, and in general much more. OMEDRANK only starts to have a somewhat acceptable performance at 8 subindexes, and by then its probe depth reaches 11% of the database.

Projection KD-forests performed surprisingly well, and may be the method of choice for databases which are not updated often (since they are harder to maintain than the multicurves — see §§ 4.2 and 4.3).

5.1.3.3 Results on index size

The use of multiple subindexes not only impacts the search time (adding more random accesses and increasing the probe depth), but also augments the size occupied by the index, and, accordingly, the time spent on the construction and maintenance of the index. The index sizes obtained for each method are shown in Figure 5.21.

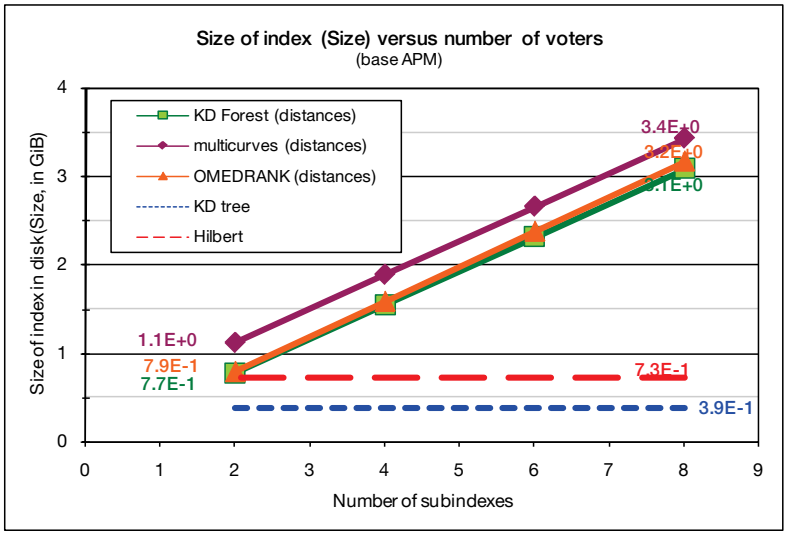


Figure 5.21: Comparison between the OMEDRANK, multicurves and projection KD-forest. Size of index versus the number of subindexes

OMEDRANK and projection KD-forests are slightly smaller than the corresponding multicurves, because the latter have to store the curve's extended key (which have the same size, in bits, as the corresponding address in the space).

In terms of construction, the KD-forests are more complex, since the process of finding the appropriate splitting dimensions and *pivots* for each tree is time consuming. Both multicurves and OMEDRANK subindexes are simply sorted lists, and can be constructed in $\mathcal{O}(n \log n)$ time. Multicurves has the extra burthen of computing the extended keys, but with the use of the Butz algorithm (see § 0, [Butz 1971]) this should not become a bottleneck.

5.1.4 Methods based on space-filling curves

In this section we compare three strategies of kNN search based on space-filling curves: our multicurves (§ 4.3) to state-of-art algorithms by Liao et al. [Liao 2001] and Mainar-Ruiz et al. [Mainar-Ruiz 2006].

The main advantage of those methods is that the curves impose a total order on the data, which can thus be stored in an efficient data structure (any structure capable of handling a sorted list). Sometimes, however, points which are near in the space are mapped very far apart in the curves, generating undesirable boundary effects. The methods we compare vary in the way they try to minimise those boundary effects.

The method by Liao et al. uses multiple Hilbert curves. When a point enters the index, a copy of it is placed on every curve, but first, a translation (which is different for every curve) is applied to the point.

Multicurves also uses multiple Hilbert curves, but uses projections on subspaces instead of translations. A point is projected into multiple subspaces and those projections are placed on the curves.

The method by Mainar-Ruiz et al. uses only one curve, but uses *multiple representants of the data* in the same curve. Each point is given a certain number of representants, of which all but one are randomly displaced around their original positions.

To compare the methods, we performed two sets of experiments.

The first set of experiments evaluated the performance of the methods according to the number of subindexes (considering that each one of the multiple representants in the single file of the Mainar-Ruiz method is conceptually a subindex). Here we used the APM database (with the randomly selected query subset of 30 000 points), and asked the methods to find the 20 nearest neighbours.

The second set of experiments evaluated the influence of the probe depth. We repeated the same queries on the APM database, but also included a query for the 10 nearest neighbours on the Yorck database.

Since the performance of both methods should be dependent on the bucket size (which determine the number of points which get to be examined), we varied this parameter, checking the performance obtained at each value.

The efficacy metrics were the probability of finding the first neighbour (*PF1*), the fraction of correct answers among the 20 (*P@20*) and the relative approximation error (*RAE*).

5.1.4.1 Implementation details

All methods were implemented in Java, using the Java Platform, Standard Edition v. 1.6. Besides our own method (multicurves) we reimplemented the methods of Liao et al. and Mainar-Ruiz et al. following, respectively the detailed descriptions in [Liao 2001] and [Mainar-Ruiz 2006]. The former was reimplemented "to the letter", but in the latter there was an adaptation: while the original algorithm uses a generalised Sierpiński curve, we use the Hilbert curve (both curves are very similar in terms of their boundary domains). We also implemented the necessary classes to analyse the results and obtain the efficacy metrics.

The allocation of dimensions among the subindexes of multicurves was implemented as explained in § 5.1.3.1.

Since the workings of the methods are very different, we did not feel it was useful or informative to cover all the parameter scale for all methods — instead, we covered the range adequate to make the general tendencies unequivocal, in order to allow a fair comparison.

5.1.4.2 Results

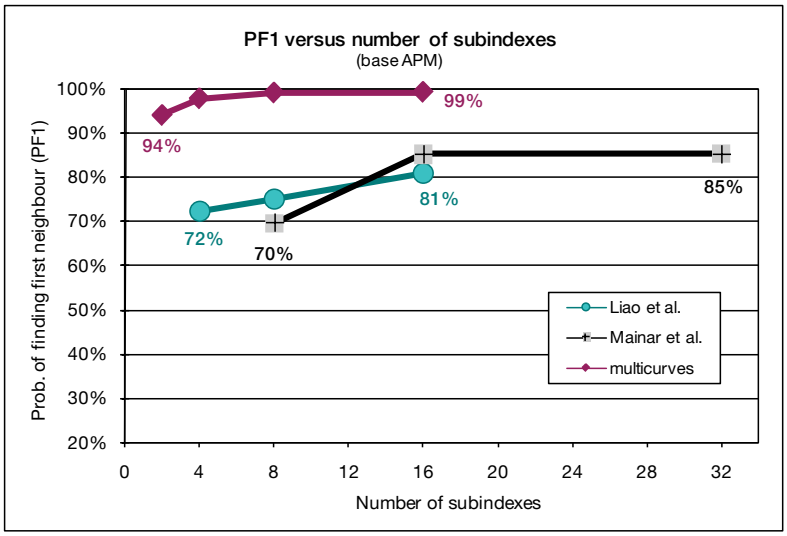
Multicurves performed very well in the APM database, typically finding half the correct neighbours and almost always finding the first neighbour (Figure 5.22–Figure 5.25).

The more modest results in the Yorck database are not so much related to number of points in the database, but to its nature (Figure 5.27). In APM, the queries are the points from the original images, and the points in the database are those of 15 transformations (while in Yorck it is the opposite). So, the number of points which are actually in the same "cluster" than the query is much smaller (and unpredictable) in Yorck than in APM.

This explains also the apparently contradictory results for the RAE, which are better in Yorck than in APM (compare Figure 5.23 and Figure 5.28). Since most points are actually spread away from the query, losing one of the correct neighbours will not make much difference in terms of relative error.

The comparison of random accesses per query is shown on Figure 5.30. The method by Mainar-Ruiz et al. performs just one random access per query, which may be an advantage depending on the penalty the i/o system puts on random access.

Independently of the base, multicurves showed almost always more efficacy than the other methods (the exception: Mainar-Ruiz et al. was slightly better in the $P@10$ metric on the base Yorck — see the lower graph on Figure 5.25) often by a considerable margin. And Figure 5.29 shows that though its probe depth matched with the one of Mainar-Ruiz et al., the actual amount of data transferred from the disk is lower, since it uses smaller extended keys, and thus, smaller index records.



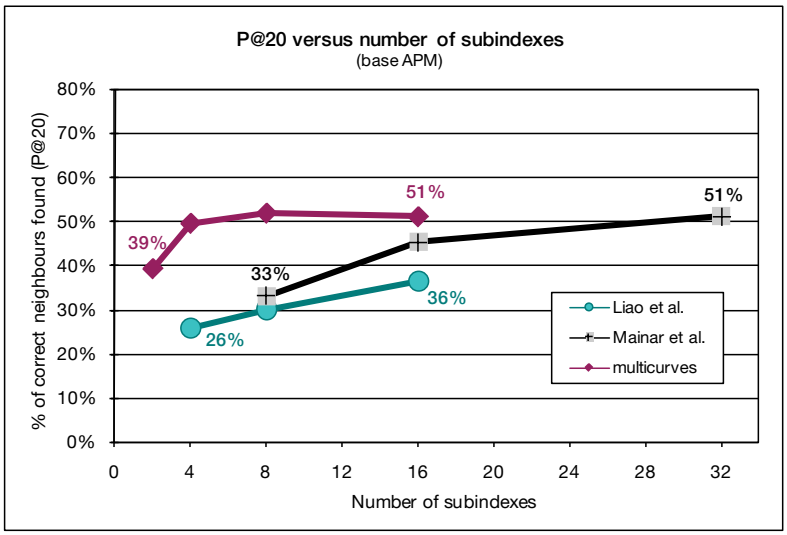


Figure 5.22: Comparison between Liao et al., Mainar-Ruiz et al. and multicurves, using a probe depth of 512 per subindex per query — efficacy metrics related to the probabilistic kNN search on the APM database, versus the number of subindexes

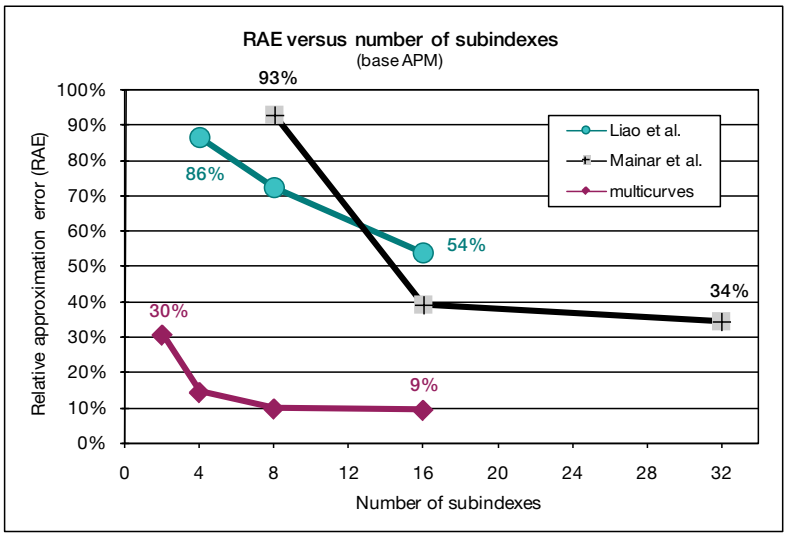


Figure 5.23: Comparison between Liao et al, Mainar-Ruiz et al. and multicurves, using a probe depth of 512 per subindex per query — efficacy metrics related to the nearby kNN search on the APM database, versus the number of subindexes

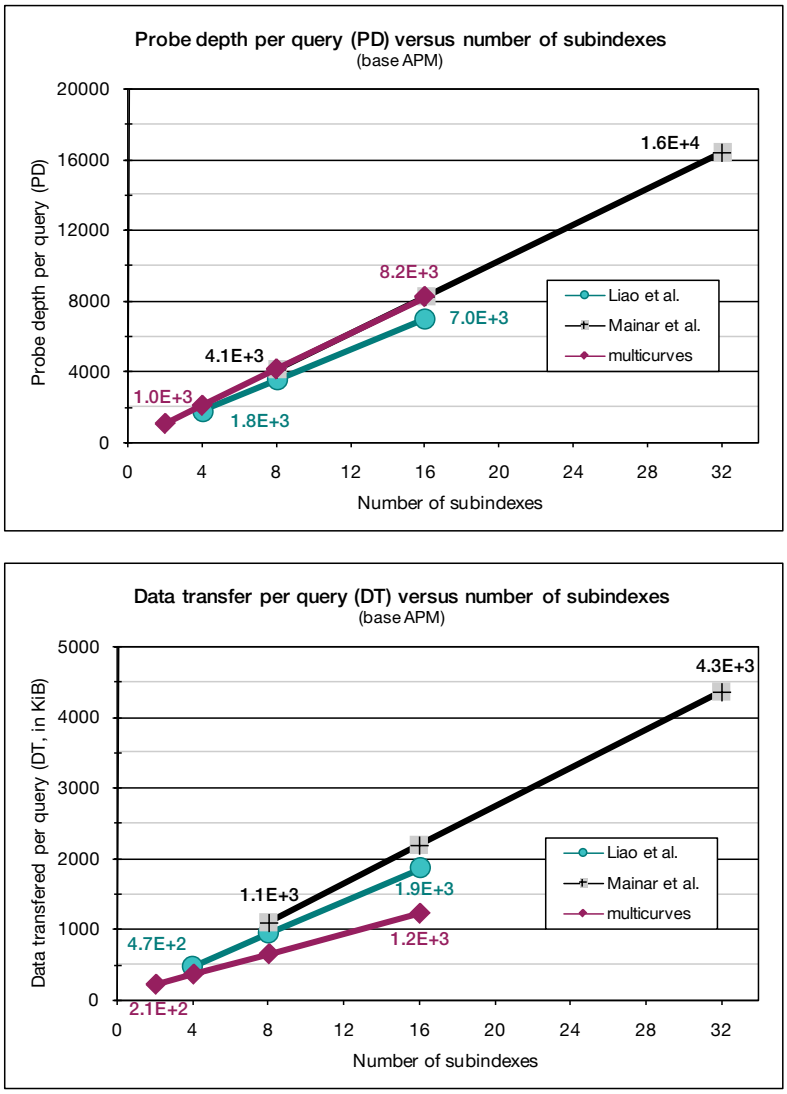
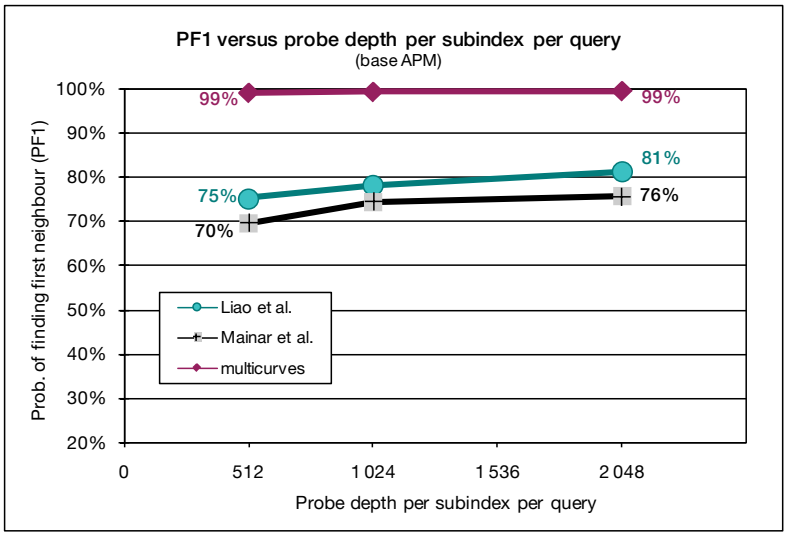


Figure 5.24: Comparison between Liao et al., Mainar-Ruiz et al. and multicurves, using a probe depth of 512 per subindex per query — efficiency metrics on the APM database, versus the number of subindexes



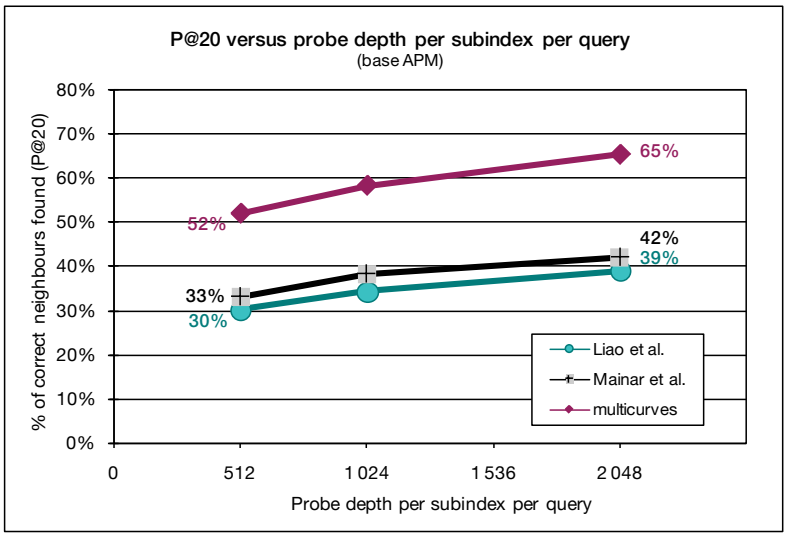


Figure 5.25: Comparison between Liao et al., Mainar-Ruiz et al. and multicurves, using 8 subindexes — efficacy metrics related to the probabilistic kNN search on the APM database, versus the probe depth on each subindex

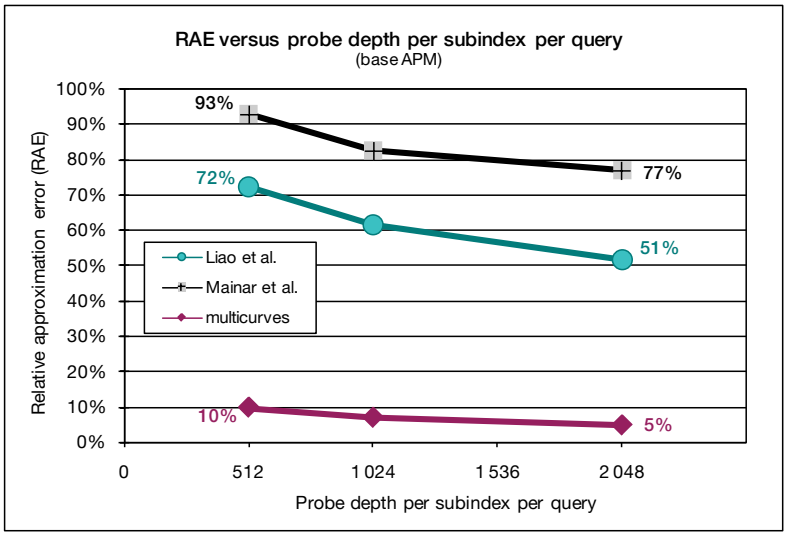


Figure 5.26: Comparison between Liao et al., Mainar-Ruiz et al. and multicurves, using 8 subindexes —efficacy metrics related to the nearby kNN search on the APM database, versus the probe depth on each subindex

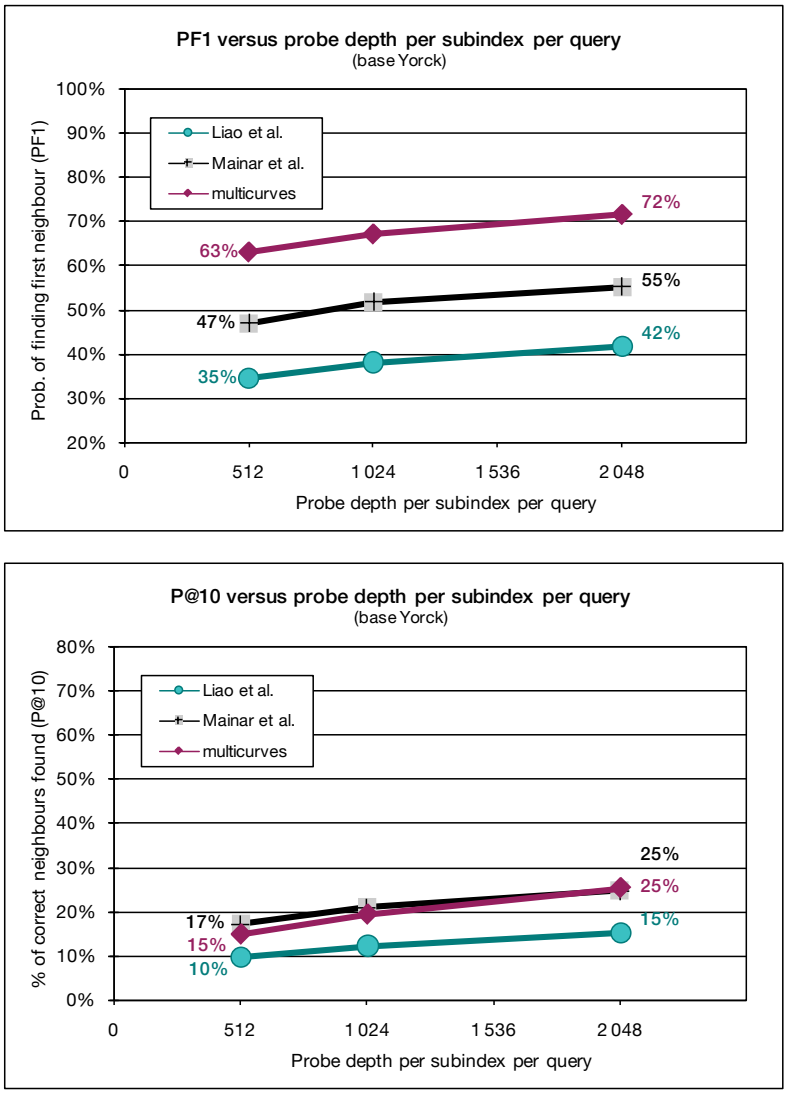


Figure 5.27: Comparison between Liao et al., Mainar-Ruiz et al. and multicurves, using 8 subindexes —efficacy metrics related to the probabilistic kNN search on the Yorck database, versus the probe depth on each subindex

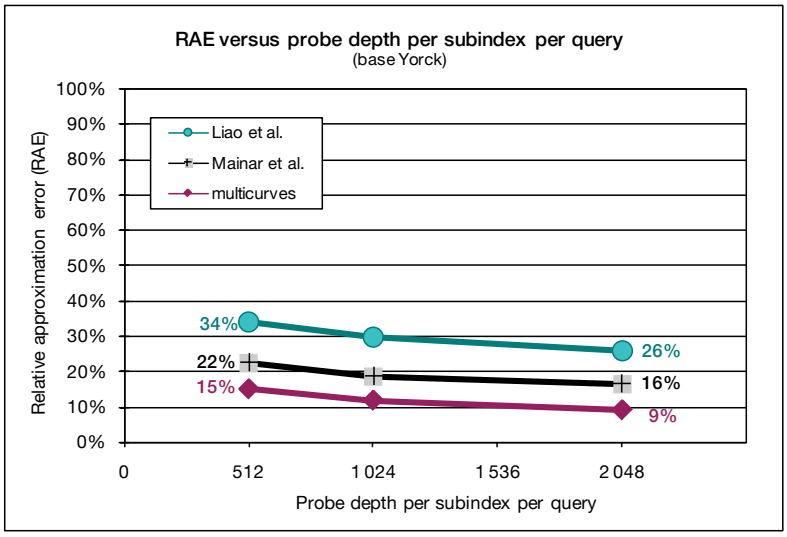


Figure 5.28: Comparison between Liao et al., Mainar-Ruiz et al. and multicurves, using 8 subindexes —efficacy metrics related to the nearby kNN search on the Yorck database, versus the probe depth on each subindex

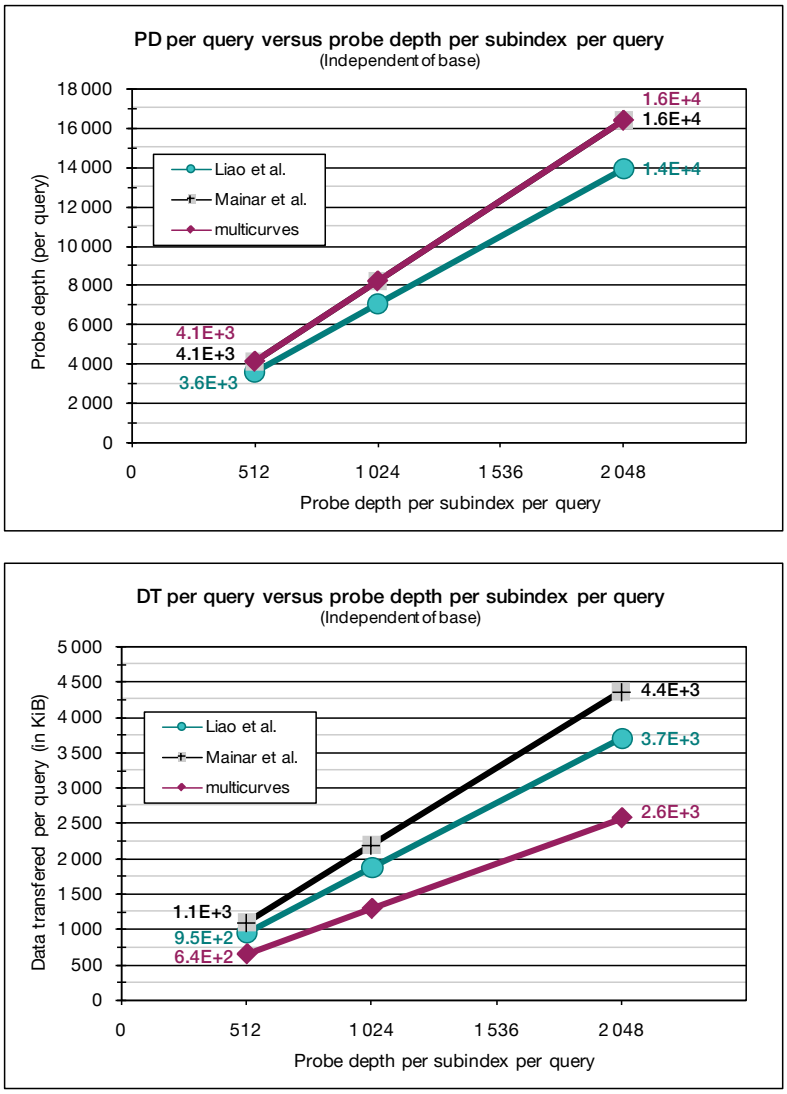


Figure 5.29: Comparison between Liao et al., Mainar-Ruiz et al. and multicurves, using 8 subindexes —efficiency metrics on both databases, versus the probe depth on each subindex

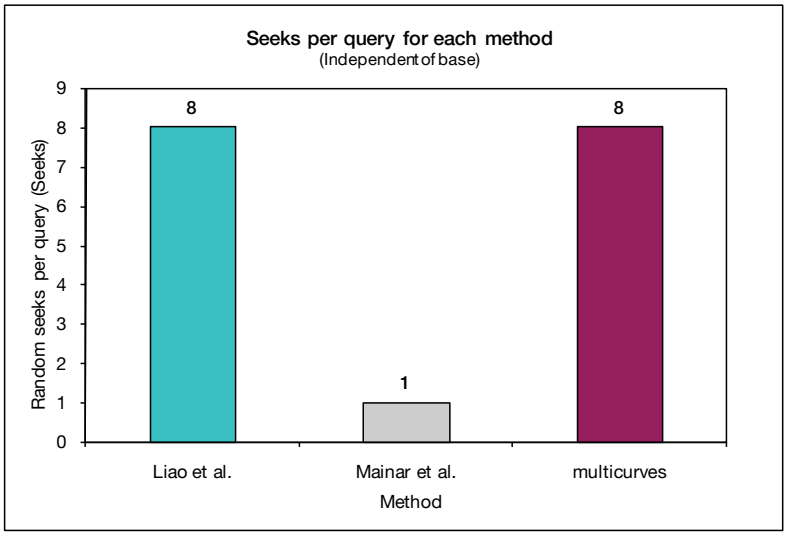


Figure 5.30: Comparison between Liao et al., Mainar-Ruiz et al. and multicurves — number of random seeks for each method, using 8 subindexes

5.1.4.3 Results on index size

The number of subindexes (considering the representants on Mainar-Ruiz as subindexes) impacts linearly the size of the index. Multicurves has slightly smaller indexes, because it stores extended-keys for the subspaces, while the other methods do it for the entire space.

Mainar-Ruiz has a "cleaning" step where it removes all duplicate representants in the same window (the size of the window being determined by the intended probe depth). The impact of this on the size of the index depends on the database. Figure 5.31 shows that it had relatively large impact in APM, but a lesser impact in Yorck (perhaps due to the fact that in Yorck, the probe depth is much smaller in comparison to the size of the database).

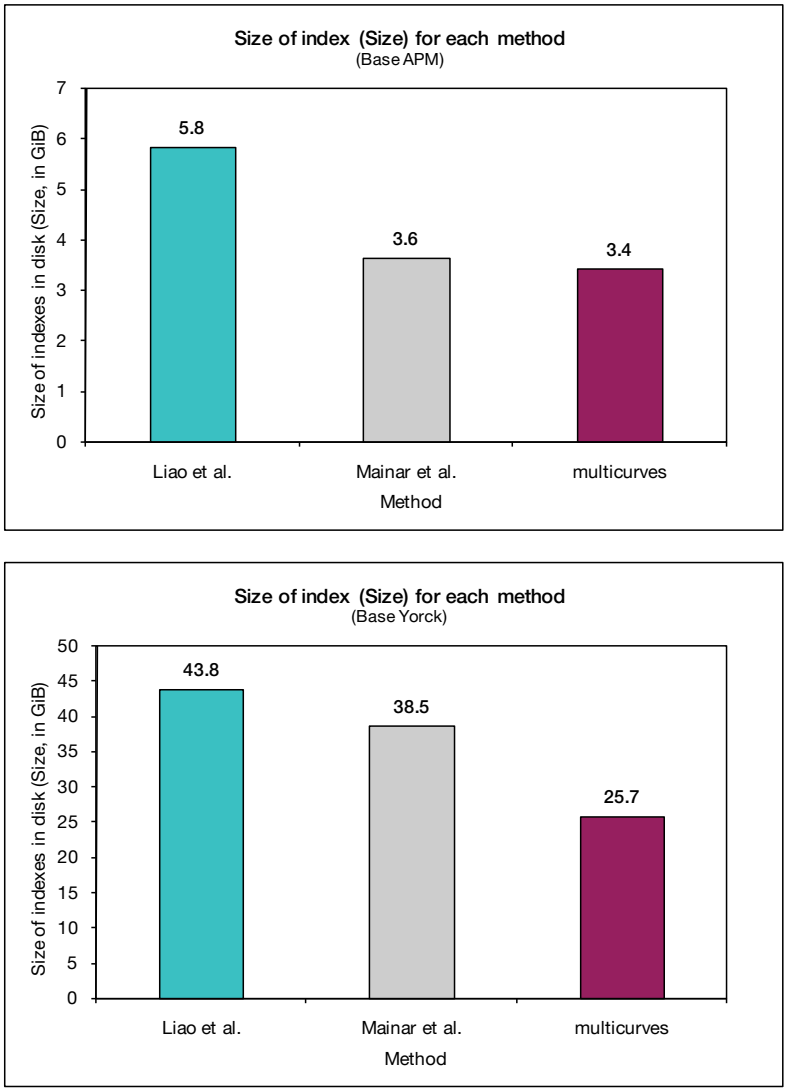


Figure 5.31: Size of index for each method — Liao et al., Mainar-Ruiz et al. and multicurves — using 8 subindexes

All indexes can be constructed incrementally using a sorted-list structure, like a B^+-tree, but, if all data are available since the start, bulk loading is much cheaper. In Mainar, we have to build a single index of the same size of the multiple indexes necessary for multicurves and Liao. Though the management of a single index is simplified, the sorting of such a large index is actually a penalty (since sorting is a slightly more-than-linear operation).

The smaller extended-keys of multicurves mean also less computational effort, though, with the use of fast algorithms (see § 0, [Butz 1971]) this step should not become a bottleneck for any of the algorithms.

5.2 The kNN methods on the context of image identification

So far, the only image identification aspect of our evaluation of kNN methods has been the fact we use databases of descriptors. When measuring the efficacy, we consider that the method is more successful if it finds the nearest points, in terms of the Euclidean distance.

However, not all nearest points are equally useful for image identification. In fact, the match can only be considered correct, when it corresponds to a real match in the image. In this section we will repeat some of the previous evaluations taking this fact into account, by the bias of a refined ground truth.

5.2.1 Evaluation protocol

The fact a descriptor is the nearest to the query in the database does not guarantee it is a good match in the sense of the visual features. Those which actually correspond to good matches are, of course, the only ones which count in the quality of the results of the image identification system.

To take this into account we created a refined ground truth, which takes the visual features into account (from this point on we will call the previous ground truth the *brute* one). Instead of simply matching each query descriptor to their nearest descriptors in the database, we make a more precise kNN search directly from the query descriptors to the set of descriptors in the correct target images (which are known beforehand). Each target image is processed in turn.

For every pair of query-target images, after matching all descriptors, we keep only the matches which are geometrically consistent (which is done via a RANSAC model fitting [Fischler 1981]). The images which are supposed to match to a given query image are known beforehand. After we process them all, we will have a list of the target descriptors which are at the same time discriminating (from the point of view of the descriptor vector) and correct (from the point of view of the visual features).

The resulting ground truth is usually more “forgiving” than the brute one, since it has less matches per query, many queries with zero matches, corresponding to the query descriptors with no useful matches (and which will be ignored in the evaluation), and usually has the easier descriptors to find (i.e., those which are the nearest to the query).

In the refined ground truth, every query point may have a different number of matching points. More than that, this number will be typically smaller than value of k used in the methods we are evaluating (while in the brute ground truth we made sure to have more matches

than the value of k). The metric of the correct answers among the k given ($P@k$) is not very indicative, since it would unfairly penalise those queries which match to fewer points. So we will use instead the mean average precision (MAP), a classic efficacy metric of image retrieval, whose definition is given below:

$$MAP = \mathrm{Avg}\left(\frac{1}{\substack{\#\text{ of answers} \\ \text{in ground truth}}} \sum_{j=1}^{k} \begin{cases} \mathrm{F}(j), & \text{if } j^{\text{th}} \text{ answer} \in \text{ground truth} \\ 0, & \text{otherwise} \end{cases} \right), \qquad \textbf{Eq. 5.1}$$

$$\text{where } \mathrm{F}(j) = {}^{\#\text{ of answers } 1^{\text{st}} \ldots j^{\text{th}} \in \text{ ground truth}} \Big/ {}_{\text{j}}$$

When the entries in the ground truth have at most one match (which happens for the Yorck database), the MAP reduces to the mean reciprocal rank (MRR):

$$MRR = \mathrm{Avg}(RR),$$

$$\text{where } RR = \begin{cases} 1/rank, & \text{if the answer is on the ground truth} \\ 0, & \text{otherwise} \end{cases} \qquad \textbf{Eq. 5.2}$$

The efficiency metrics we will use are the probe depth (PD) and the size of the indexes ($Size$).

A difficulty in creating this evaluation is the fact that two images which we would not initially consider as matches in the ground truth, may share visual features. For example, two otherwise unrelated pictures of interiors, but which have the same reproduction of Monet's "Impression, Sunrise", may end up generating an unexpected match, because of the common feature. Because deciding when those matches are legitimate and when they are not is conceptually very problematic, we tried to avoid this situation in the APM database, which we assembled ourselves, and which is relatively small, but it happens sometimes in Yorck. Because of this, though we have chosen the queries in a mostly random manner, we have eliminated those which suffered from this problem.

5.2.2 Evaluated methods and parameterisation

We are not as exhaustive in the exploration of the parameter space as in the last section. Instead we concentrate in comparing the methods where they showed an acceptable compromise of efficacy / efficiency.

For the APM database, we have selected:

- The 3-way tree with maximum bucket size of 8 192 descriptor (actual bucket size of 5 608);
- The projection KD-forests with 4 subindexes and maximum bucket size of 512 descriptor (actual bucket size of 351);
- The multicurves with 4 subindexes, search modality of distance comparison and a probe-depth of 512 descriptors per subindex per query;
- The method by Liao et al. with 8 subindexes and a probe depth of 512 descriptors per subindex per query;

- The method by Mainar-Ruiz et al. with 8 subindexes and a probe depth of 512 descriptors per subindex per query;
- The simple KD-tree with maximum bucket size of 8 192 descriptor (actual bucket size of 5 608);
- The simple Hilbert curve with a probe depth of 512 descriptors per subindex per query.

For the APM database, all methods were allowed to return 20 nearest neighbours (corresponding to the 15 image transformations in the database and a small margin).

For the Yorck database, the chosen methods were:

- The multicurves;
- The method by Liao et al.;
- The method by Mainar-Ruiz et al.

For the Yorck database, all three methods were executed with 8 subindexes and a probe depth of 512 descriptors per subindex per query. They were allowed to return 10 nearest neighbours.

Again, all trees are built using the interquartile range and the median to determine respectively the splitting dimension and the pivot. Also, we only visit one bucket per tree (the one where the query falls).

5.2.3 Results

Unsurprisingly, all methods have better scores for the *MAP* in the refined ground truth than in the brute ground truth (Figure 5.32 and Figure 5.34). What is noteworthy is that the relative difference between the two values of the MAP is not the same for all methods, which indicates that some have a particular bias towards the "correct" descriptors. This is particular noticeable in the projection KD forests.

This difference, is, of course, very pronounced in all methods for the Yorck database (Figure 5.34–Figure 5.35), because of the lower proportion of useful matches in the refined ground truth for this database.

We make two plots of efficiency *versus* efficacy for the studied methods, one considering the probe depth per query (*PD*) and the other considering the size of the index (*Size*).

As we can see in Figure 5.33 the methods with the best compromise of efficacy and efficiency are the projection KD-Forests and the multicurves. The 3-way trees have the better performance of all methods, but the size of their indexes is more than the double of the second worst (Liao et al.). The other methods lagged behind (though it is curious to note that the simple KD-tree has acceptable efficacy, for a very small index size).

This trend is confirmed in Figure 5.35, where we compare again multicurves, Liao et al. and Mainar-Ruiz et al., this time on the Yorck database. It is interesting to note, comparing the upper plots of Figure 5.32 and Figure 5.34, that while the *MAPs* obtained in Yorck with the brute ground truth are much lower than the ones obtained in APM, the *MAPs* obtained with the refined ground truth remain roughly comparable.

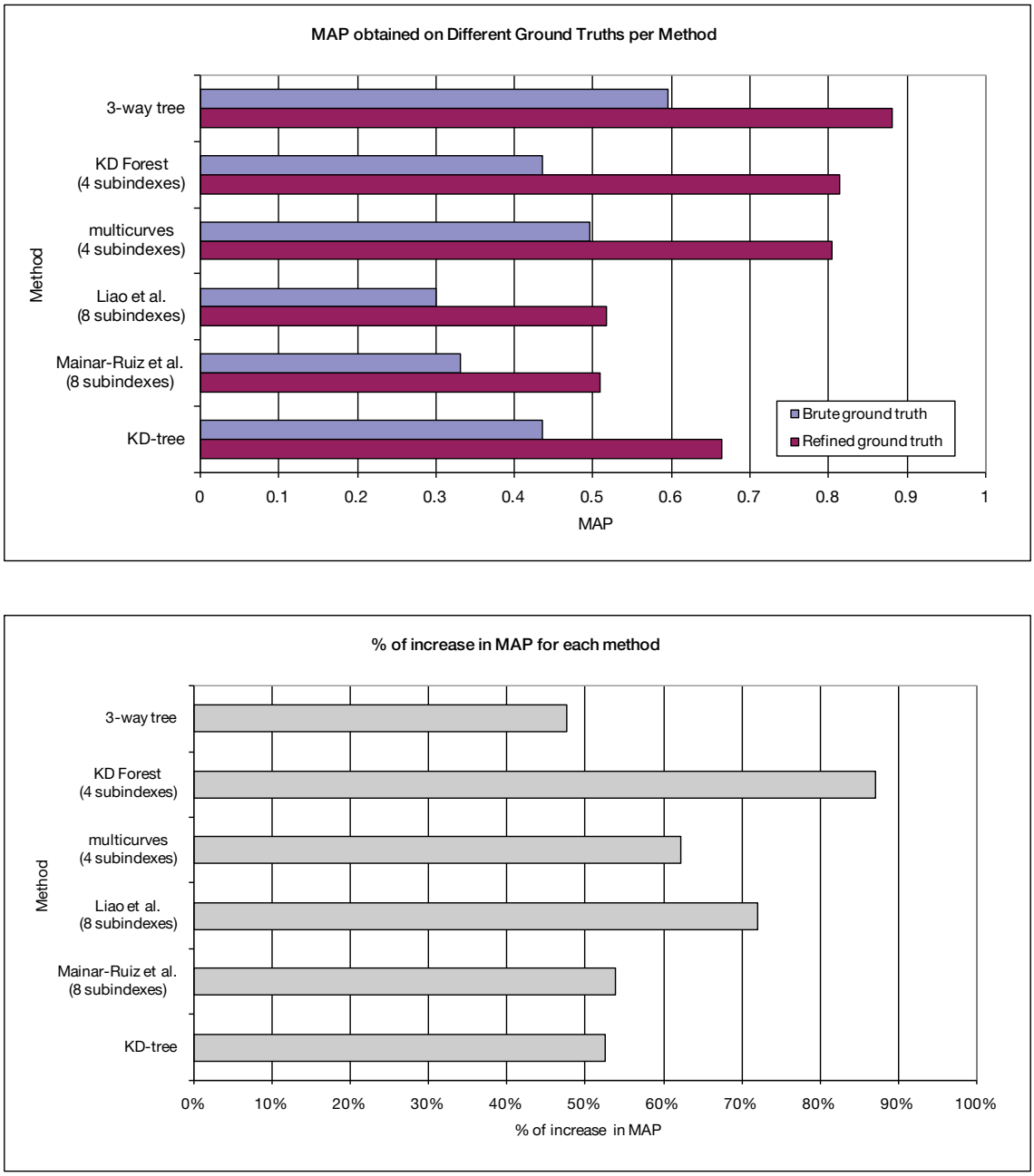
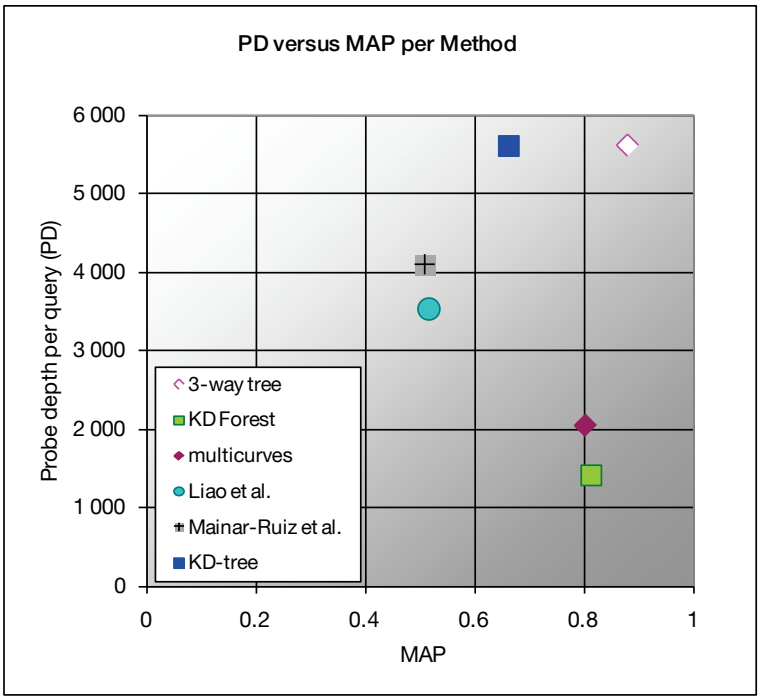


Figure 5.32: The *MAP* obtained for the two ground truths on the APM database, for each method



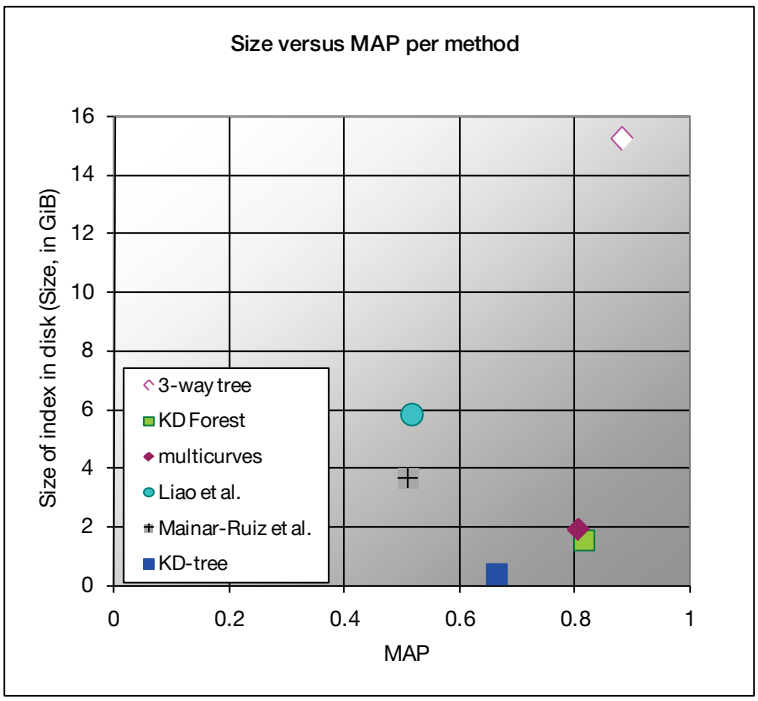


Figure 5.33: Plot of the efficacy versus efficiency for the APM database, for each method. The best compromise is on the lower right corner (darker areas)

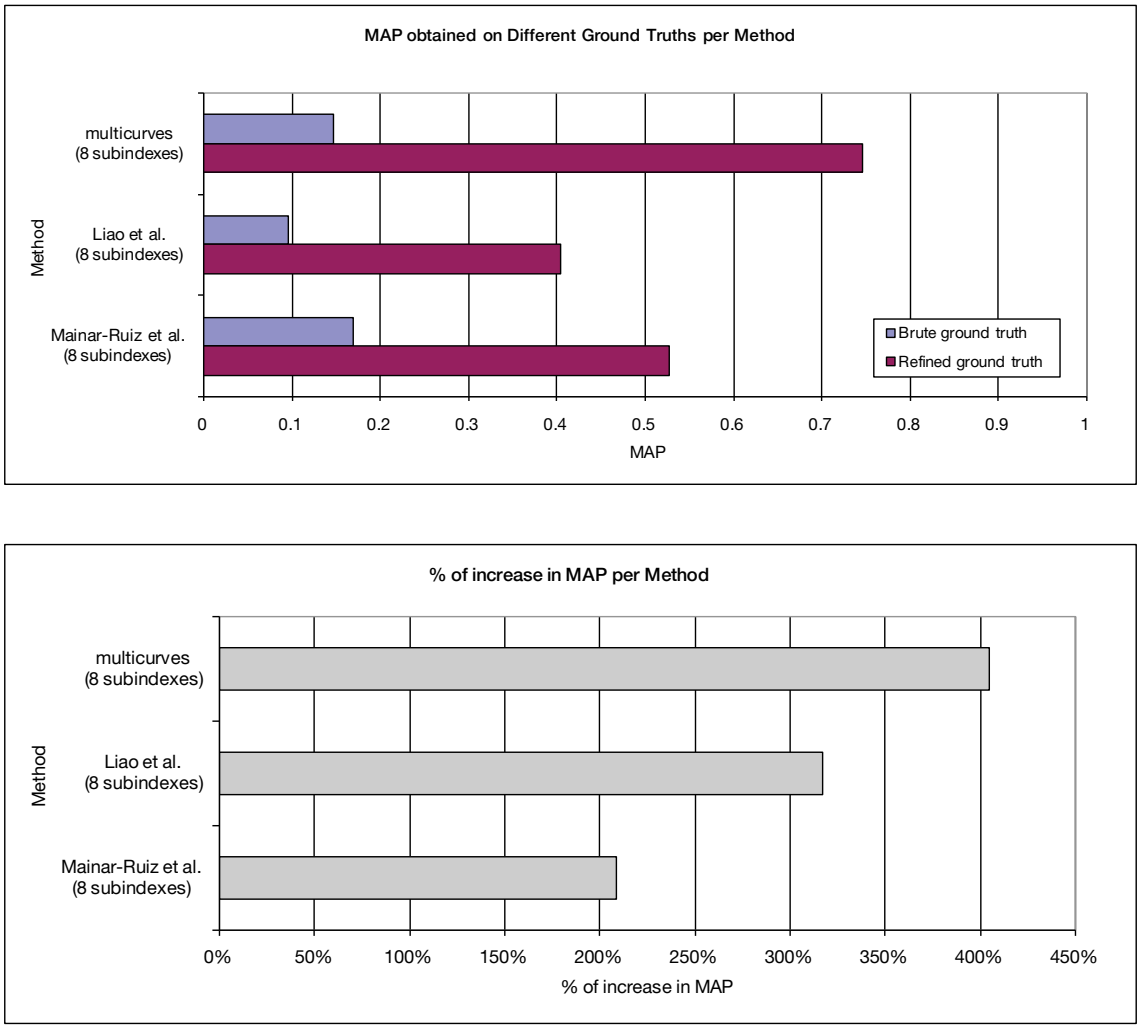


Figure 5.34: The *MAP* obtained for the two ground truths on the Yorck database, for each method

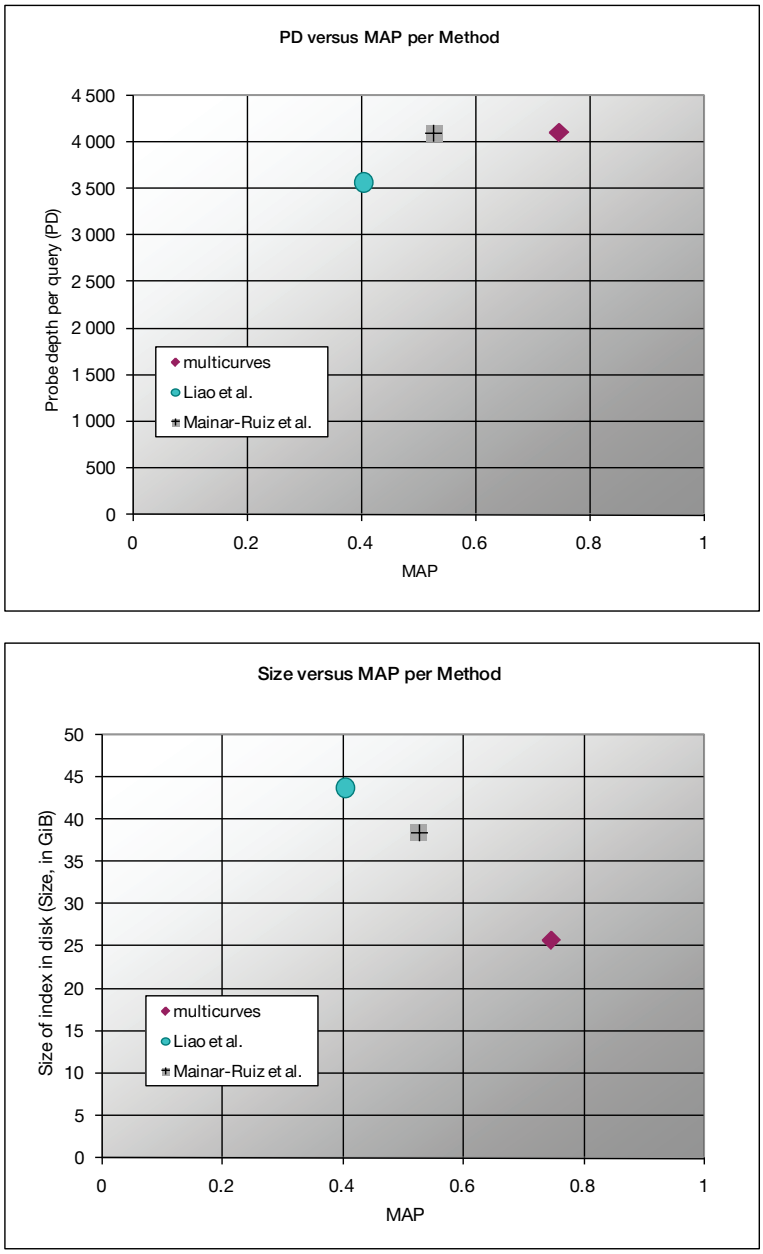


Figure 5.35: Plot of the efficacy versus efficiency for the Yorck database, for each method. The best compromise is on the lower right corner (darker areas)

5.3 Image identification systems evaluation

If so far we have evaluated the subcomponent of descriptor matching, in this section we broaden our perspective to encompass the whole architecture. We present an empirical comparison between our local-descriptor based image identification scheme, and the one used in the ARTISTE system [Lewis 2004]. We also report on our participation in the ImagEVAL campaign [ImagEVAL 2006].

ImagEVAL was a campaign funded by the French program "Techno-Vision", having as objective the evaluation of retrieval, classification, pattern matching and machine learning in image databases. One of the tasks concerned image identification. Seven contestants, including both academy and industry, participated in this specific task.

5.3.1 Evaluation protocol

Image identification can be performed in two ways:

- Using the transformed image as query, find the original image from which it derives. This is the most common task, and the most frequent way in which the problem is presented;
- Using the original image as query, find all the derivative images in the database.

The performance of the system, of course, depends on several factors, and not only on the quality of descriptor matching. One of the most critic issues is the invariance of the local descriptors under the different image transformations. SIFT descriptors are invariant under similarity geometric transformation (rotations, and scale changes — up to a factor of 4 to 6) and linear photometric changes. Non-linear photometric changes are also somewhat contemplated. It is not invariant, though, to all affine transformations (like shearings) [Lowe 2004].

In our evaluation we used the three databases from the ImagEVAL campaign of 2006. Summary information on these databases can be found in Table 5.4.

The first ImagEVAL base which we used was **IE.Test**, which corresponded to the database of the first test of ImagEVAL, but without the transformation of negative images (SIFT is not invariant to the negative of an image).

The **IE.T11** base is a large database with the final test of ImagEVAL (including the negative images, for which we created a special processing). This database is composed of 42 500 transformed images from a set of 2 500 originals. A subset of 50 of those originals was used as query.

The **IE.T12** base is a database with the final test of ImagEVAL (also, including the negative images). This database is composed of 2 500 originals. A set of 60 transformed images from those was used as query.

The transformations comprise rotations, shearings, scale changes, cropping, dithering, addition of noise, lossy compression, gamma correction, etc. A sample of those can be found in Figure 5.37.

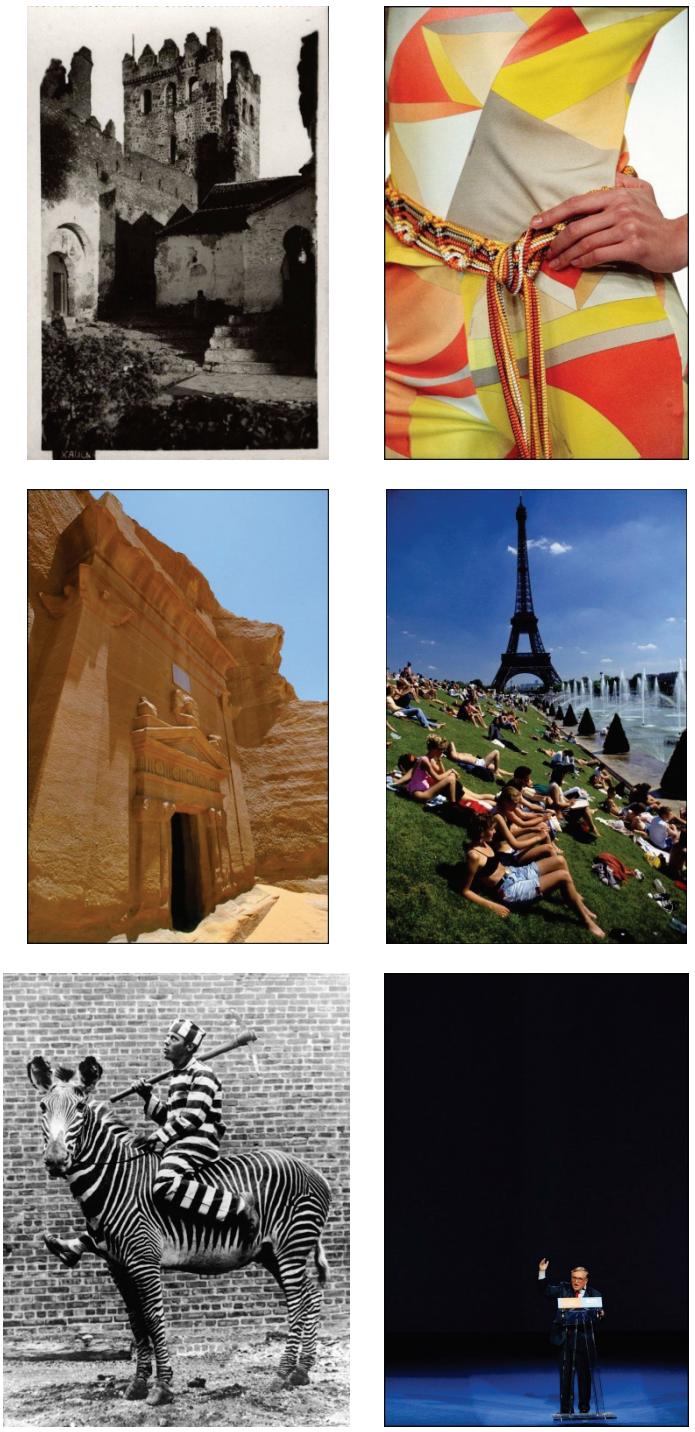


Figure 5.36: Some original images from ImagEVAL, showing a wide range of visual characteristics

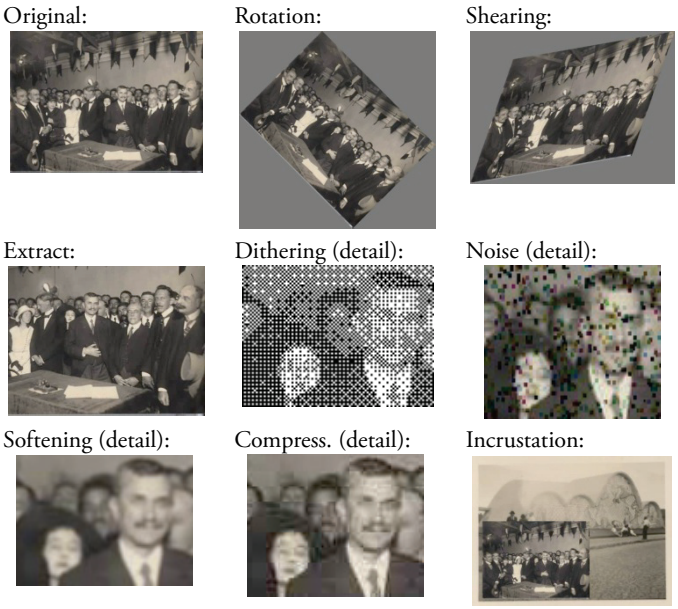


Figure 5.37: Sample transformations

The systems themselves are also evaluated both in efficacy and efficiency. The efficiency criteria do not differ from those used for the evaluation of kNN search: query time, number of operations performed, size of indexes, etc. (§ 5.1.1.4).

The efficacy metrics, however, differ in one significant aspect: the relevant (correct) answers have no inherent internal ordering (contrarily to the nearest neighbours, which have a first, second, third, and so on, ranked answer given by the distance to the query). If the answers appear to have a ranking, it is due to the internal mechanisms of the identification system (see § 2.6).

One possible efficacy metric is the **success rate**, i.e., frequency in which the correct answer is among the *top x* answers. The rationale for this criterion is that the images will be presented to the user in the form of thumbnails in a screen of results, and there are only so many simultaneous miniatures the user can view comfortably, without having to resort to scrolling or navigation (between 10 and 50 images, depending on the size and resolution of the screen). A more precise metric is the **mean reciprocal rank**, which measures the average value of the inverse of the value of the rank of the good image (Eq. 5.3).

Those metrics are useful when at most one correct answer is known to exist in the database for every query. When several correct answers potentially exist, the **mean average precision** may be more indicative of efficacy (Eq. 5.3).

$$MAP = \mathrm{Avg}\left(\frac{1}{\#\text{ of relevant answers}} \sum_{j=1}^{\ell} \begin{cases} \mathrm{F}(j), & \text{if } j^{\text{th}} \text{ answer is relevant} \\ 0, & \text{otherwise} \end{cases} \right), \qquad \textbf{Eq. 5.3}$$

$$\text{where } \mathrm{F}(j) = {}^{\#\text{ of relevant answers in top } j \text{ answers}} \Big/ _{\mathrm{j}}$$

and ℓ is the number of answer images returned by the system.

Name	# base images	# query images	Type of task	Transformations
IE.Test	960	60	From original find transformed	Rotations, colour, dithering, noise, softening, shearing, cropping, occlusion, frame, combination of images (with scaling), compression.
IE.T11	42 500	50	From original find transformed	Same as IE.Test, plus negative image and combined transformations.
IE.T12	2 500	100	From transformed find original	Same as IE.Test, plus negative image and combined transformations.

Table 5.4: Summary information on the databases and queries for the evaluation of image identification

5.3.2 Points of interest *versus* hierarchic regions

In this section we present one of our early experiments, a comparison between a system of image identification based on the SIFT points of interest [Lowe 2004] to a good archetype of the time-honoured indexing methods, the CCV (colour coherence vector), which was successfully implemented in a large scale European software project [Lewis 2004]. We demonstrate experimentally the efficacy of the local descriptor-based approach, which achieved a 99.2% success rate, against 61.0% for the Multiscale-CCV, in a database of photos, drawings and paintings.

To compare the two methods, we used the IE.Test database.

The experiment consisted in submitting each one of the transformed images as query, and collecting the 25 most similar answers, accordingly to each method. The efficacy metric was the success rate of finding the original image among the returned set. The choice of 25 answers, if somewhat arbitrary, was based on the number of thumbnails a user can view simultaneously with some comfort. The idea is that the user will receive a screen of results and decide, in a few moments, if the correct answer is among them.

The results of this section were published in [Valle 2006].

5.3.2.1 Implementation details

For the system based on Multiscale-CCV, we created the descriptors for each image on the database [Lewis 2004, § III, pp. 303–306; Pass 1997]. For each query, a unique CCV descriptor was computed, using the entire image. This descriptor was then compared sequentially with all descriptors in the database, and the 25 best candidates from different images were kept.

For the system based on SIFT, the images in the database were first pre-processed with a Gaussian smoothing, and had its PoI detected and described, following the architecture described in § 2.6. Each query was also smoothed and had its descriptors computed. Each query descriptor was matched against the descriptors in the database using a KD-tree with the best-bin-first strategy (§ 3.5). The matched descriptors voted for the images to be considered. After removing geometric inconsistent votes using a RANSAC algorithm with a 2D affine transformation model [Fischler 1981], we counted the votes and kept the 25 images which had more votes.

The SIFT descriptors were computed using the binaries provided by Lowe [Lowe 2005], after reducing the images to ¼ of their original size. All image manipulations were done with the Netpbm package [Netpbm 2007].

We reimplemented the Multiscale-CCV, following the instructions on [Lewis 2004]. We also implemented the final steps for the local-descriptor system: vote-counting and geometric consistency, based on the RANSAC algorithm. All our code was in Java.

5.3.2.2 Results

The results of the experiments are presented in Table 5.5. We showed the percentage of successes, classified by type of transformation of the query.

In our approach, an important parameter is the number of neighbours to match to each query point (parameter k). If the value of k is too small, some images in the correct category will not be found, because the corresponding points will be outside the neighbourhood set. Conversely, if the value of k is too large, false positives will proliferate, decreasing the quality of the classification. Ideally, k should be exactly the number of correct matching points for each query point (which corresponds, roughly, to the number of images in each category). In most practical cases this number may be impossible to determine *a priori*. We evaluated the results for 5 and 10 nearest neighbours.

The performance is better for $k = 10$ than for $k = 5$ and would probably be optimal at $k = 16$ (the actual number of transformations of each image). Unfortunately, for the KD-tree with best-bin-first, as k increases, so does the compromise between running time and imprecision.

Query	Hist.	CCV	Local Desc.	
			k = 5	k = 10
Original	100%	100%	100%	100%
Colour	18%	20%	100%	100%
Dithering	0%	0%	100%	100%
Noise	60%	45%	100%	100%
Softening	95%	90%	100%	100%
Rotation	42%	47%	96%	98%
Extract	93%	96%	100%	100%
Shearing	65%	82%	85%	95%
Occlusion	97%	100%	100%	100%
Frame	63%	75%	100%	100%
Incrustation	22%	20%	93%	98%
Compression	90%	97%	100%	100%
Average	**58.6%**	**61.0%**	**97.7%**	**99.2%**

Table 5.5: Success rate for each image identification scheme, by query type

For both $k = 10$ and $k = 5$, our approach performed better than or as equally well as the other methods, for all the transformations.

CCV performance was particularly poor in the colour transformations and was catastrophic under the dithering, because this transformation completely modifies both the value and the coherence of the colours.

For the sake of reference, we included the performance of a simple colour histogram, using the same colour quantization of the CCV (RGB colourspace, 64 equal "cubes"). The results are inferior to the CCV for most of the transformations. The softening and impulsive noise, are remarkable exceptions, explained by the fact they disturb the coherence of the colours.

Though the local descriptor approach had a much better efficacy, its computing times were also much higher: a single query took several minutes of wall time (against a few seconds for CCV).

5.3.3 The ImagEVAL 2006 campaign — Tasks 1.1 and 1.2

The results of ImagEVAL 2006 confirm these observations about the efficacy of methods based on local descriptors. In both tasks of image identification (Tasks 1.1 and 1.2) of ImagEVAL the teams which obtained the best results used local descriptors [Moëllic 2007].

Task 1.1 consisted in taking the original as query and finding all the transformations in a database of 42 500 images (database IE.T11 in Table 5.4). The corresponding descriptor database had more than 16 million SIFT descriptors.

Task 1.2 consisted in taking the transformed image as query and finding the original in a database of 2 500 images (database IE.T12 in Table 5.4). The corresponding descriptor database had more than 3.3 million SIFT descriptors.

In this campaign of ImagEVAL, time was less a concern than obtaining the best possible results (though the organization tried to incorporate efficiency measures, as an afterthought). To match the descriptors, we used a simple KD-tree with the best-bin first strategy (§ 3.5) and a large probe depth (20 000 descriptors).

Our method obtained perfect results for Task 1.2, sharing ex-æquo the first position in the rank with the two other techniques. In Task 1.1, it has obtained a third place, with a *MAP* of 0,9833 (the first in the rank obtained a *MAP* of 0,9950).

The only pre-processing we used was to reduce the size of the images, in order to reduce the number of descriptors obtained. In Task 1.1, we reduced the images to ¼ of their original size, while in Task 1.2, which had a smaller database, we made experiments reducing the images both to ¼ and to ½ their original sizes (there was a slight difference in efficacy, the version with the factor of ½ was the one which gave the perfect results, the one with the factor of ¼ had a *MRR* of 0,9875).

The local descriptor we used, SIFT, is reasonably invariant to all the transformations proposed, except the conversion to the negative colours. This is because, contrarily to the other photometric/colorimetric transforms, it radically changes the direction of the gradient (actually inverting it). Since this was the *only* failure case, instead of seeking to make SIFT invariant to the inversion of the gradient, it seemed to us more reasonable to solve the problem by making two researches: one using the unmodified query, and one using the negative of the query. The results are later recombined, by intercalating the two answers set in the order of the greater number of descriptors matched.

The major difficulty faced by our method was how to deal with images completely void of features, or those in which they are too few (less than 10). If this occurs either on the target or in the query image, the method will either not work at all (no points) either work unreliably (too few points to allow a robust matching). Some images we found particularly hard to process can be seen in Figure 5.38: what mainly characterises them is a strong lack of high-frequency spectral components.

A report on our participation in the campaign is available in [Valle 2007c].

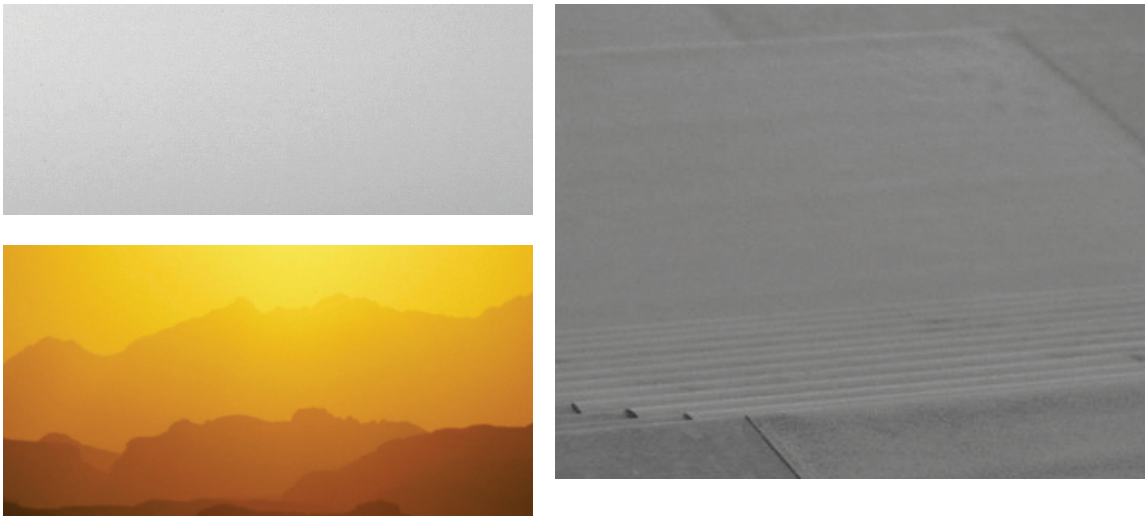


Figure 5.38: Some hard-to-treat images from the Task 1 of ImagEVAL. The general smoothness of those images is responsible for a lack of points of interest

5.4 Conclusion

The 3-way trees showed very good performance: even in their worst case, they showed more efficacy than the best case of the KD-trees. Projection KD-forests and multicurves, compared respectively with the KD-tree and the single Hilbert curve, showed that is very advantageous to use multiple subindexes. They also showed that it is better to stay away from one-dimensional subindexes, beating OMEDRANK in all experiments.

In the comparison between the different ways to use multiple space-filling curves, multicurves showed almost always more efficacy than the other methods, often by a considerable margin.

Considering our context and the methods we compared, multicurves and projection KD-forest are the methods of choice, consistently presenting good efficacy and efficiency. The choice between the two is a matter of the relative advantages in particular implementation scenarios. Multicurves is entirely based on sorted lists, which makes it very easy to implement over existing data management systems. This also makes it easier to implement a dynamic index, which is important in contexts where the database changes constantly.

If an acceptable compromise can be reached between the bucket size and the database size, the 3-way tree can be a method of choice, because of its precision, provided that the database does not change very often.

Our experiment on the system architecture confirms the efficacy of the local-descriptor based systems, though some “back-up” strategy may be useful for the images which present very few or no features at all. The same efficacy was observed on the competition ImagEVAL.

Chapter 6

Conclusion

Image identification is a task which finds several important applications, including the retrieval of documents lacking adequate source references, the recovery from accidental separation of a document and its metadata, and the detection of copyright infringements. Of all proposed solutions to tackle this mission, the systems specific for image identification present several advantages. Contrarily to textual search, they do not depend on annotations (which can be unavailable or incorrect). They do not depend on watermarks (which can be destroyed accidentally or on purpose). And compared to general-purpose CBIR systems, their notion of similarity is better suited to match the distorted queries to the originals.

Among the image identification systems, the superiority of those based on local-descriptors has been established empirically (§§ 5.3.2–5.3.3, [Ke 2004b]), which is not surprising, since local-descriptors are intrinsically robust to cropping and occlusions, and normally very robust to geometric and photometric distortions. Those systems, however, suffer an efficiency penalty, due to the need of matching hundreds, even thousands of descriptors to perform a single query.

For our system, we have chosen SIFT [Lowe 2004] and concentrated on the goal of accelerating the techniques of descriptor matching, in order to obtain an effective system without incurrent in serious efficiency drawbacks.

After a detailed analysis of the state of the art on approximate kNN search, concentrating on the techniques adequate to our context, we proposed three new methods:

- **The 3-way tree** (§ 4.1), which is an adaptation of the KD-tree, with the addition of redundant, overlapping, nodes. Those nodes allow to choose a "path of the middle", if the query happens to fall too near the boundaries (which in high dimensionalities, occurs quite often).
- **The projection KD-forest** (§ 4.2), which was the first answer to the problem of the excessive growth of the 3-way tree. Inspired on MEDRANK, this technique uses multiple subindexes, but with the important difference that each subindex has multiple dimensions (avoiding the main drawbacks of MEDRANK, which is the lack of correlation between proximity in a single dimension and in the high-dimensional space). If the 3-way tree grows

exponentially with the number of descriptors, the projection KD-forest grows only linearly, which means they are scalable for any database size we want to process.

- The **multicurves** (§ 4.3), based on space-filling curves, which regained the advantage of MEDRANK, lost by projection KD-forest, of having a total order on the subindexes, allowing thus the use of sorted list data structures. It keeps the idea of using multiple moderately dimensional subindexes, but each subindex is a simple sorted list where the entries are sorted by extended-keys taken from a space-filling curve.

In the context of descriptor matching, among the methods we compared, the 3-way trees obtained the best efficacy. Multicurves and projection KD-forest, however, obtained the best compromise between efficacy and efficiency (§ 5.2).

Multicurves should be the method of choice for most cases, as it is very easy to implement using any available, efficient sorted list data structure. Besides, if the data structure chosen to implement the sorted list allows, it is easy to keep the index dynamic. If the index is known to be static or almost-static, the projection KD-forests may be an alternative choice, since they have a slightly better performance. If the index is static and disk space is not a problem, the 3-way trees might be a feasible solution, in situations where precision is so crucial that its slightly better efficacy compensates for its inconveniences.

We tested our system architecture, based on local descriptor, comparing it with a real-world scheme based on global descriptors (§ 5.3.2). This empirical evaluation not only validated the robustness of our architecture, but reaffirmed the prediction that local-descriptor based schemes are much more effective in tackling the problem. Our architecture was also put through the test in the campaign ImagEVAL, with very good results (§ 5.3.3).

Perspectives

The current architecture of local-descriptor based systems is well established and empirically attested for the recognition of a specific target (but much less so for the recognition of general categories). However, little theoretical work has been developed to sustain it. For us, it is clear that the vote-counting mechanism is a pragmatic response to the need to perform the search in sublinear time, and that it approximates an implicit function of dissimilarity. It would be interesting to formalise this concept and explore how, without paying an excessive performance penalty, we could go beyond the simple vote count — especially in order to gain the generalisation necessary in the search for categories.

In what concerns kNN search, so many interesting questions remain to be answered, that even for such a specific subject, it does not feel absurd to cite the adage *ars longa, vita brevis*.

The distribution of the dimensions among the subindexes in projection KD-forests and multicurves is an interesting question. It would be desirable to provide a less costly way to make this decision than the method of Dellis et al. and see how it affects the performance.

We were initially very impressed with the idea of Mainar-Ruiz and Perez-Cortés of using multiple representants instead of multiple subindexes —for it allows reducing the number of random access to the bare minimum — and we were somewhat surprised when its efficacy came so much lower than multicurves. But the explanation is quite obvious: their algorithm uses a fixed number of representants, ignoring that most of the regions of the curve are quite well behaved, and that serious boundary effects happen in relatively few sites. The results of this homogeneous treatment are the pollution of the curve with an enormous quantity of unnecessary representants and an insufficient quantity for the really problematic points. It would be interesting to observe the performance of a smarter version of the algorithm, which takes those facts into account.

There is an important lesson in this latter paragraph: the conception of kNN search algorithms presents special challenges. On one hand, one should be very wary of the everyday geometric intuitions about "distance", "adjacency", "boundary" and "neighbourhood" — at the dimensionalities we are processing, they are treacherous. On the other hand, one has to keep constant attention on the harsh realities of implementations. Many a beautiful solution (and we, too, had our quota of false starts) fails miserably in practice because it forgets to address those limitations. The key to success is to have in mind both the theoretical foundations and the technological constraints.

Technology, of course, is a changing hypothesis. It is interesting to speculate what would happen in the presence of a disruptive advancement, like very fast and cheap DRAM or fast random access in secondary storage (or maybe the disappearance of the notion of memory hierarchy altogether). This might become true if magnetoresistive RAM (or MRAM) accomplishes its promises, but right now, it is one of those technologies which has been labelled imminent for the last 15 years.

We feel that the evaluation of kNN search on synthetic databases does not allow determining reliably how the method will behave in real databases. We initially hoped that our synthetic "Clusters" database would at least be comparable to the real descriptors "APM" database, but this was far from the truth.

As consequence, on one hand, it would be interesting to test our methods on real databases from applications other than descriptor matching. This would allow checking how dependent the optimisations we have chosen are on the application we targeted. On the other hand, it would also be interesting to study the possibility of creating synthetic data which behaves more like real data — perhaps using the comprehension we gained from fractal analysis.

The absence of a serious longitudinal study of performance is the Achilles' heel of kNN search (and, for that matter, of all high-dimensional point techniques — range search, closest-point search and inverse-kNN search). So far, every new proposed method is compared to a few proceeding methods, each time using different databases and different metrics. The general unavailability of the source codes, the inexistence of standardised databases and the complexity of

most methods (making a Herculean effort for a single author to reimplement, parameterise and test a significant amount of them) has prevented a reliable comparative study.

Perhaps the absence of a comprehensive theoretical framework has also prevented the development of comparative studies, for it is difficult to establish, for very different methods, equivalent parameterisations to produce fair comparisons. We believe that such a framework is possible, at least as an abstract goal. Seemingly unrelated methods like the NV-tree [Lejsek 2008] and the p-stable flavoured LSH [Datar 2004], in final analysis, are based on surprisingly similar geometrical insights (slicing the space regularly among several concurrent axes).

Incidentally, the theoretical advancements on the subject have so far been meagre. Despite some interesting work by Arya et al. and Kleinberg, among others, many important questions remain unanswered. A better theoretical understanding of the problem would be welcome, either to prevent further exploration of fruitless research directions, or to take away the uncanny sensation that an efficient solution for the kNN search is another unattainable goal.

Nevertheless, we are convinced that the subject of kNN search will continue to thrive, not only for the importance of its applications (which appeal to both the Ministry of Culture and the Ministry of Defence), but also for the lure of its theoretic, algorithmic and implementational challenges.

Publications

Our research, comprised on this thesis, has also been the subject of the following publications:

Eduardo Valle, Matthieu Cord and Sylvie Philipp-Foliguet. "Content-Based Retrieval of Images for Cultural Institutions Using Local Descriptors" in *Geometric Modeling and Imaging — New Trends*, pp. 177–182. London, UK, July 5–7, 2006. IEEE Computer Society, Washington, DC, USA. DOI: 10.1109/GMAI.2006.16

Eduardo Valle, Matthieu Cord and Sylvie Philipp-Foliguet. "3-Way-Trees: A Similarity Search Method for High-Dimensional Descriptor Matching" in *Proceedings of the 2007 IEEE International Conference on Image Processing — ICIP*, vol. 1, pp. I.173–176. San Antonio, TX, USA, September 16–19, 2007. IEEE Computer Society, Washington, DC, USA. DOI: 10.1109/ICIP.2007.4378919.

Eduardo Valle, Matthieu Cord and Sylvie Philipp-Foliguet. "Matching Local Descriptors for Image Identification on Cultural Databases" in *Proceedings of the 9th International Conference on Document Analysis and Recognition — ICDAR*, vol. 2, pp. 679–683. Curitiba, PR, Brazil, September 23–26, 2007. IEEE Computer Society, Washington, DC, USA. DOI: 10.1109/ICDAR.2007.4377001.

Eduardo Valle, Sylvie Philipp-Foliguet and Matthieu Cord. "Image Identification using Local Descriptors (Task 1)" in *ImagEVAL Workshop*. University of Amsterdam, Amsterdam, Netherlands, July 12, 2007.

Eduardo Valle, Sylvie Philipp-Foliguet and Matthieu Cord. "Content-based Image Identification on Cultural Databases" in *International Cultural Heritage Informatics Meeting — ICHIM*, 5 p. Toronto, ON, Canada, October 24–26, 2007. Archives & Museum Informatics, Toronto, ON, Canada.

Eduardo Valle, Matthieu Cord and Sylvie Philipp-Foliguet. "Fast identification of visual documents using local descriptors" in *Proceedings of the 8th ACM Symposium on Document Engineering — DocEng*, 4 p. São Paulo, SP, Brazil, September 16–19, 2008. ACM, New York, NY, USA. DOI: 10.1145/1410140.1410175.

Eduardo Valle, Matthieu Cord and Sylvie Philipp-Foliguet. "Fast identification of visual documents using local descriptors" in *Proceedings of the 17th ACM Conference on Information and Knowledge Management — CIKM*, 9 p. Napa Valley, CA, USA, October 26–30, 2008. ACM, New York, NY, USA.

References

[Aggarwal 2001] Charu C. Aggarwal. *Re-designing distance functions and distance-based applications for high dimensional data*. ACM SIGMOD Record, vol. 30, n. 1, pp. 13–18. ACM, New York, NY, USA. ISSN: 0163-5808. DOI: 10.1145/373626.373638. March 2001.

[Alber 1998] Jochen Alber and Rolf Niedermeier. "On Multi-dimensional Hilbert Indexings" in *Proceedings of the Computing and Combinatorics: 4th Annual International Conference, COCOON'98*, vol. 1449, pp. 199–231. Taiwan, Republic of China, August 1998. Springer, Berlin / Heidelberg, Germany.

[Amsaleg 2001] Laurent Amsaleg and Patrick Gros. *Content-based Retrieval Using Local Descriptors: Problems and Issues from a Database Perspective*. Pattern Analysis & Applications, vol. 4, n. 2–3, 108–124. Springer, London, UK. ISSN: 1433-7541. DOI: 10.1007/s100440170011. June 2001.

[Armitage 1997] Linda H. Armitage and Peter G. B. Enser. *Analysis of user need in image archives*. Journal of Information Science, vol. 23, n. 4, 287–299. SAGE / Chartered Institute of Library and Information Professionals. ISSN: 0165-5515. DOI: 10.1177/016555159702300403. 1997.

[Arya 1996] Sunil Arya. *Nearest neighbor searching and applications*. Ph.D. Thesis. University of Maryland at College Park, College Park, MD, USA. 1996.

[Asano 1997] Tetsuo Asano, Desh Ranjan, Thomas Roos, Emo Welzl and Peter Widmayer. *Space-filling curves and their use in the design of geometric data structures*. Theoretical Computer Science, vol. 181, n. 1, pp. 3–15. Elsevier Science Publishers Ltd., Essex, UK. ISSN: 0304-3975. DOI: 10.1016/S0304-3975(96)00259-9. July 1997.

[Ballard 1981] Dana H. Ballard. *Generalizing the Hough transform to detect arbitrary shapes*. Pattern Recognition, vol. 13, n. 2, pp. 111–122. Elsevier Inc. ISSN: 0031-3203. DOI: 10.1016/0031-3203(81)90009-1. 1981.

[Bayer 1972] Rudolf Bayer and E. M. McCreight. *Organization and maintenance of large ordered indexes*. Acta Informatica, vol. 1, n. 3, p. 173–189. Springer, Berlin / Heidelberg. ISSN: 0001-5903. DOI: 10.1007/BF00288683. September 1972.

[Beckmann 1990] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider and Bernhard Seeger. *The R*-tree: an efficient and robust access method for points and rectangles*. ACM SIGMOD Record, vol. 19, n. 2, pp. 322–332. ACM, New York, NY, USA. ISSN: 0163-5808. DOI: 10.1145/93605.98741. June 1990.

[Beis 1997] Jeffrey S. Beis and David G. Lowe. "Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces" in *Proceedings of the 1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, p. 1000–1006.

San Juan, Puerto Rico, June 06–19, 1997. IEEE Computer Society, Washington, DC, USA. DOI: 10.1109/CVPR.1997.609451.

[Bellman 1961] Richard Bellman. *Adaptive Control Processes: a guided tour.* Princeton University Press. 1961.

[Belussi 1995] Alberto Belussi and Christos Faloutsos. *Estimating the Selectivity of Spatial Queries Using the 'Correlation' Fractal Dimension*. (Computer Science Technical Report Series). Technical report CS-TR-3423. University of Maryland at College Park, Institute for Advanced Computer Studies, College Park, MD, USA. 1995.

[Bennett 1999] Kristin P. Bennett, Usama Fayyad and Dan Geiger. "Density-based indexing for approximate nearest-neighbor queries" in *Proceedings of the 5th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 233–243. San Diego, California, USA, August 15–18, 1999. ACM, New York, NY, USA. DOI: 10.1145/312129.312236.

[Bentley 1975] Jon Louis Bentley. *Multidimensional binary search trees used for associative searching*. Communications of the ACM, vol. 18, n. 9, pp. 509–517. ACM Press, New York, New York, USA. ISSN: 0001-0782. DOI: 10.1145/361002.361007. September 1975.

[Berchtold 1996] Stefan Berchtold, Daniel A. Keim and Hans-Peter Kriegel. "The X-tree: An Index Structure for High-Dimensional Data" in *Proceedings of the 22th International Conference on Very Large Data Bases*, pp. 28 - 39 September 03–06, 1996. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Berchtold 1998] Stefan Berchtold, Christian Böhm and Hans-Peter Kriegal. *The pyramid-technique: towards breaking the curse of dimensionality*. ACM SIGMOD Record, vol. 27, n. 2, pp. 142--153. ACM, New York, NY, USA. ISSN: 0163-5808. DOI: 10.1145/276305.276318. June 1998.

[Berchtold 2000] Stefan Berchtold, Christian Böhm, Daniel A. Keim, Hans-Peter Kriegel and Xiaowei Xu. "Optimal Multidimensional Query Processing Using Tree Striping" in *Proceedings of the 2nd International Conference on Data Warehousing and Knowledge Discovery*, pp. 244--257. London, UK, September 04--06, 2000. Springer-Verlag, London, UK. DOI: 10.1007/3-540-44466-1.

[Berrani 2002] Sid-Ahmed Berrani, Laurent Amsaleg and Patrick Gros. *Approximate k-nearest-neighbor searches: a new algorithm with probabilistic control of the precision*. Technical Report IRISA-PI - 02-1495 4675, 32 p. INRIA, Rennes, France. December 2002.

[Berrani 2004] Sid-Ahmed Berrani, Laurent Amsaleg and Patrick Gros. "Recherche d'images par le contenu pour la détection des copies" in *14ème Congrès Francophone AFRIF-AFIA de Reconnaissance des Formes et Intelligence Artificielle*. Toulouse, France, January 28–30, 2004. Association Française pour la Reconnaissance et l'Interprétation des Formes.

[Bially 1967] Theodore Bially. *A Class of Dimension Changing Mappings and Its Application to Bandwidth Compression*. Ph.D. dissertation. Polytechnic Institute of Brooklyn, Department of Electrical Engineering, New York, NY, USA. June, 1967.

[Bially 1969] Theodore Bially. *Space-filling curves: Their generation and their application to bandwidth reduction*. IEEE Transactions on Information Theory, vol. 15, n. 6, pp. 658–664. IEEE Transactions on Information Theory Society. ISSN: 0018-9448. November 1969.

[Binkley 1939] Robert C. Binkley. *Strategic Objectives in Archival Policy*. American Archivist, vol. 2, 162–168. July 1939.

[Böhm 2000] Christian Böhm, Bernhard Braunmüller, Hans-Peter Kriegel and Matthias Schubert. " Efficient similarity search in digital libraries" in *Proceedings of the IEEE Advances in*

Digital Libraries 2000, p. 193–199. Washington, DC, USA, May 22–24, 2000. IEEE Computer Society, Washington, DC, USA. DOI: 10.1109/ADL.2000.848382.

[Böhm 2001] Christian Böhm, Stefan Berchtold and Daniel A. Keim. *Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases*. ACM Computing Surveys (CSUR), vol. 33, n. 3, pp. 322–373. ACM, New York, NY, USA. ISSN: 0360-0300. DOI: 10.1145/502807.502809. September 2001.

[Butz 1971] Arthur R. Butz. *Alternative Algorithm for Hilbert's Space-Filling Curve*. IEEE Transactions on Computers, vol. C-20, n. 4, pp. 424–426. IEEE Computer Society. ISSN: 0018-9340. April 1971.

[Cantor 1878] Georg Cantor. *Ein Beitrag zur Mannigfaltigkeitslehre*. Crelles Journal für die reine und angewandte Mathematik, vol. 84, pp. 242–258. Germany. 1878.

[Cha 2002] Guang-Ho Cha, Xiaoming Zhu, Dragutin Petkovic and Chin-Wan Chung. *An efficient indexing method for nearest neighbor searches in high-dimensional image databases*. IEEE Transactions on Multimedia, vol. 4, n. 1, pp. 76-87. IEEE, New York, NY, USA. ISSN: 1520-9210. DOI: 10.1109/6046.985556. 2002.

[Chang 2003] Edward Chang, Beitao Li, Gang Wu and Kingshy Goh. "Statistical Learning for Effective Visual Information Retrieval" in *Proceedings of the 2003 International Conference on Image Processing (ICIP)*, vol. 3, pp. III.609–612. Barcelona, Spain, September 14–17, 2003. IEEE, Washington, DC, USA. DOI: 10.1109/ICIP.2003.1247318.

[Chávez 2001] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates and José Luis Marroquín. *Searching in metric spaces*. ACM Computing Surveys (CSUR), vol. 33, n. 3, pp. 273–321. ACM, New York, NY, USA. ISSN: 0360-0300. DOI: 10.1145/502807.502808. September 2001.

[Ciaccia 1997] Paolo Ciaccia, Marco Patella and Pavel Zezula. "M-tree: An Efficient Access Method for Similarity Search in Metric Spaces" in *Proceedings of the 23rd International Conference on Very Large Data Bases*, pp. 426–435. August 25–29, 1997. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Conway 1996] Paul Conway. *Preservation in the Digital World*. (CLIR Reports). Report pub62. Concil on Library and Information Resources, Washington, DC, USA. March 1996.

[Cord 2006] Matthieu Cord, Philippe H. Gosselin and Sylvie Philipp-Foliguet. *Stochastic exploration and active learning for image retrieval*. Image and Vision Computing, vol. 25, n. 1, pp. 14–23. ELSEVIER. DOI: 10.1016/j.imavis.2006.01.004. January 2007.

[Cultura 2007] Secretaria de Estado de Cultura, Minas Gerais. *Site da Secretaria de Estado de Cultura de Minas Gerais: Arquivo Público Mineiro*. April 23, 2008. Web page. Retrieved on April 24, 2008 from http://www.cultura.mg.gov.br/?task=home&sec=5.

[Datar 2004] Mayur Datar, Nicole Immorlica, Piotr Indyk and Vahab S. Mirrokni. "Locality-sensitive hashing scheme based on p-stable distributions" in *Proceedings of the 12th annual symposium on Computational geometry*, pp. 253–262. Brooklyn, New York, USA, June 08–11, 2004. ACM, New York, NY, USA. DOI: 10.1145/997817.997857.

[Dellis 2005] Evangelos Dellis, Bernhard Seeger and Akrivi Vlachou. "Nearest neighbor search on vertically partitioned high-dimensional data" in *7th International conference on data warehousing and knowledge discovery*. Copenhagen, Denmark, August 22--26, 2005. Springer, Berlin, Germany.

[Directmedia 2008] Directmedia, Publishing GmbH. *Digitale bibliothek homepage*. Web page. Retrieved on April 24, 2008 from http://www.digitale-bibliothek.de/.

[Dwork 2001] Cynthia Dwork, Ravi Kumar, Moni Naor and D. Sivakumar. "Rank aggregation methods for the Web" in *10th International World Wide Web Conference*. Hong Kong, May 1--5, 2001. Proceedings unpublished.

[Eakins 1999] John Eakins and Margaret Graham. *Content-based image retrieval*. Report n. 39, 65 p. Higher Education Funding Councils, Joint Information Systems Comittee, JISC Technology Applications Programme, Bristol, UK. October 1999.

[Fagin 2003] Ronald Fagin, Ravi Kumar and D. Sivakumar. "Efficient similarity search and classification via rank aggregation " in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pp. 301–312. San Diego, California, USA, June 09–12, 2003. ACM, New York, NY, USA. DOI: 10.1145/872757.872795.

[Faloutsos 1986]Christos Faloutsos. "Multiattribute hashing using Gray codes" in *1986 ACM SIGMOD International Conference on Management of data*, pp. 227–238. Washington, D.C., USA, May 28–30, 1986. ACM Press, New York, NY, USA. DOI: 10.1145/16894.16877.

[Faloutsos 1988]Christos Faloutsos. *Gray Codes for Partial Match and Range Queries*. IEEE Transactions on Software Engineering, vol. 14, n. 10, pp. 1381–1393. DOI: 10.1109/32.6184. October 1988.

[Faloutsos 1989]Christos Faloutsos and Shari Roseman. "Fractals for secondary key retrieval" in *Proceedings of the 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 247–252. Philadelphia, Pennsylvania, USA, March 29–31. ACM Press, New York, NY, USA. DOI: 10.1145/73721.73746.

[Faloutsos 1994]Christos Faloutsos and Ibrahim Kamel. "Beyond uniformity and independence: analysis of R-trees using the concept of fractal dimension" in *Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems* p. 4–13. Minneapolis, Minnesota, USA, May 24–27, 1994. ACM, New York, NY, USA. DOI: 10.1145/182591.182593.

[Fauzi 2002] Mohammad F. A. Fauzi and Paul H. Lewis. "Query by Fax for Content-Based Image Retrieval" in *Image and Video Retrieval*. pp. 107–130. (Lecture Notes in Computer Science, vol. 2383/2002). Springer, Berlin / Heidelberg, Germany. ISBN: 978-3-540-43899-1. DOI: 10.1007/3-540-45479-9_10. 2002.

[Ferhatosmanoglu 2000] Hakan Ferhatosmanoglu, Ertem Tuncel, Divyakant Agrawal and Amr El Abbadi. "Vector approximation based indexing for non-uniform high dimensional data sets" in *Proceedings of the 9th international conference on Information and knowledge management*, pp. 202–209. McLean, Virginia, USA, 2000. ACM, New York, NY, USA. DOI: 10.1145/354756.354820.

[Finkel 1974] R. A. Finkel and J. L. Bentley. *Quad trees: a data structure for retrieval on composite keys*. Acta Informatica, vol. 4, n. 1, pp. 1–9. Springer, Berlin / Heidelberg, Germany. ISSN: 0001-5903. DOI: 10.1007/BF00288933. March 1974.

[Fischler 1981] Martin A. Fischler and Robert C. Bolles. *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography*. Communications of the ACM, vol. 24, n. 6, pp. 381–395. ACM, New York, NY, USA. ISSN: 0001-0782. DOI: 10.1145/358669.358692. June 1981.

[Fredman 1984] Michael L. Fredman, János Komlós and Endre Szemerédi. *Storing a sparse table with O(1) worst case access time*. Journal of the ACM, vol. 31, n. 3, p. 538–544. ACM, New York, NY, USA. ISSN: 0004-5411. DOI: 10.1145/828.1884. July 1984.

[Friedman 1975] J. H. Friedman, F. Baskett and L. J. Shustek. *An Algorithm for Finding Nearest Neighbors*. IEEE Transactions on Computers, vol. 24, n. 10, pp. 1000–1006. IEEE Com-

puter Society, Washington, DC, USA. ISSN: 0018-9340. DOI: 10.1109/T-C.1975.224110. October 1975.

[Friedman 1976] Jerome Friedman, Jon L. Bentley and Raphael A. Finkel. *An Algorithm for Finding Best Matches in Logarithmic Expected Time*. Technical Report CS-TR-75-482. Stanford University, Stanford, CA, USA. 1976.

[Fuchs 1980] Henry Fuchs, Zvi M. Kedem and Bruce F. Naylor. *On visible surface generation by a priori tree structures*. ACM SIGGRAPH Computer Graphics, vol. 14, n. 3, pp. 124–133. ACM, New York, NY, USA. ISSN: 0097-8930. DOI: 10.1145/965105.807481. July 1980.

[Gaede 1998] Volker Gaede and Oliver Günther. *Multidimensional access methods*. ACM Computing Surveys (CSUR), vol. 30, n. 2, pp. 170–231. ACM, New York, NY, USA. ISSN: 0360-0300. DOI: 10.1145/280277.280279. June 1998.

[Geurts 2006] Pierre Geurts, Damien Ernst and Louis Wehenkel. *Extremely randomized trees*. Machine Learning, vol. 63, n. 1, 3–42. Springer Netherlands. ISSN: 0885-6125. DOI: 10.1007/s10994-006-6226-1. April 2006.

[Gilbert 2006] William Gilbert. *A Cube-filling Hilbert Curve*. September 19, 2006. Web page. Retrieved on April 24, 2008 from http://www.math.uwaterloo.ca/~wgilbert/Research/HilbertCurve/HilbertCurve.html.

[Gionis 1999] Aristides Gionis, Piotr Indyk and Rajeev Motwani. "Similarity Search in High Dimensions via Hashing" in *Proceedings of the 25th International Conference on Very Large Data Bases*, pp. 518–529. September 07–10, 1999. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Goh 2002] King-Shy Goh, Beitao Li and Edward Chang. "DynDex: a dynamic and non-metric space indexer" in *Proceedings of ACM international conference on Multimedia*, pp. 466–475. Juan-les-Pins, France, December 01–06, 2002. ACM Press, New York, New York, USA. DOI: 10.1145/641007.641107.

[Grétarsdóttir 2005] Ragnheiður Grétarsdóttir, Sigurðir Einarsson, Björn Jónsson and Laurent Amsaleg. "The Eff2 Image Retrieval System Prototype" in *IASTED International Conference on Databases and Applications*. Innsbruck, Austria, February 14–16, 2005.

[Guttman 1984]Antonin Guttman. "R-trees: a dynamic index structure for spatial searching" in *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pp. 47–57. Boston, MA, USA, 1984. ACM, New York, NY, USA. DOI: 10.1145/602259.602266.

[Heisterkamp 2003] Douglas R. Heisterkamp and Jing Peng. "Kernel VA-Files for Nearest Neighbor Search in Large Image Databases" in *Proceedings of 13th International Conference on Artificial Neural Networks/10th International Conference on Neural Information Processing*. Istanbul, Turkey, June 26–29, 2003.

[Henrich 1989] Andreas Henrich, Hans-Werner Six and Peter Widmayer. "The LSD tree: spatial access to multidimensional and non-point objects" in *Proceedings of the 15th international conference on Very large data bases*, pp. 45–53. Amsterdam, The Netherlands, 1989. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Henrich 1998] Andreas Henrich. "The LSD^h-Tree: An Access Structure for Feature Vectors" in *Proceedings of the 14th International Conference on Data Engineering* 362-369. Orlando, FL, USA, February 23–27, 1998. IEEE Computer Society, Washington, DC, USA. DOI: 10.1109/ICDE.1998.655799.

[Hilbert 1981] David Hilbert. *Über die stetige Abbildung einer Line auf ein Flächenstück* Mathematische Annalen, vol. 38, n. 3, 459-460. Springer, Berlin / Heidelberg. ISSN: 0025-5831 (Print)

1432-1807 (Online). DOI: 10.1007/BF01199431. September, 1891.

[Hinneburg 2000] Alexander Hinneburg, Charu Aggarwal and Daniel A. Keim. "What is the Nearest Neighbor in High Dimensional Spaces?" in *Proceedings of the 26th International Conference on Very Large Data Bases*, pp. 506--515. September 10--14, 2000. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Hough 1962] Paul Hough. *Method and Means for Recognizing Complex Patterns*. US Patent n. 3,069,654. December 18, 1962.

[IEC 2005] IEC. *Letter symbols to be used in electrical technology — Part 2: Telecommunications and electronics*. 60027-2 Ed. 3.0, 145 p. International Electrotechnical Commission, Technical Committee 25, Geneva, Switzerland. August 11, 2005.

[ImagEVAL 2006] Comité d'Organisation ImagEVAL. *imagEVAL homepage*. Web page. Retrieved on April 24, 2008 from http://www.imageval.org/.

[Indyk 1998] Piotr Indyk and Rajeev Motwani. "Approximate nearest neighbors: towards removing the curse of dimensionality" in *Proceedings of the 13th annual ACM symposium on Theory of computing*, pp. 604–613. Dallas, Texas, USA, May 24–26, 1998. ACM, New York, NY, USA. DOI: 10.1145/276698.276876.

[Indyk 2004] Piotr Indyk. "Nearest neighbors in high-dimensional spaces" in *Handbook of Discrete and Computational Geometry*. 2nd ed. p. 877–892. Jacob E. Goodman and Joseph O'Rourke (eds.). (Discrete Mathematics And Its Applications Series). CRC Press, Inc., Boca Raton, FL, USA. ISBN: 1-58488-301-4. 2004.

[Joly 2005] Alexis Joly. *Recherche par similarité statistique dans une grande base de signatures locales pour l'identification rapide d'extraits vidéo*. Thèse de doctorat, 270 p. Université de La Rochelle, Département de formation doctorale en informatique, La Rochelle, France.

[Katayama 1997] Norio Katayama and Shin'ichi Satoh. "The SR-tree: an index structure for high-dimensional nearest neighbor queries" in *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, pp. 369–380. Tucson, Arizona, United States, May 11–15, 1997. ACM, New York, NY, USA. DOI: 10.1145/253260.253347.

[Ke 2004a] Yan Ke and Rahul Sukthankar. "PCA-SIFT: A More Distinctive Representation for Local Image Descriptors" in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 506–513. Washington, DC, USA, June 27–July 2, 2004. IEEE Computer Society, Wasington, DC, USA. DOI: 10.1109/CVPR.2004.183.

[Ke 2004b] Yan Ke, Rahul Sukthankar and Larry Huston. "An efficient parts-based near-duplicate and sub-image retrieval system" in *Proceedings of the 12th ACM international conference on Multimedia*, pp. 869–876. New York, NY, USA, October 10–16, 2004. ACM, New York, NY, USA. DOI: 10.1145/1027527.1027729.

[Kemeny 1959] John Kemeny. *Mathematics without numbers*. Daedalus, vol. 88, pp. 577–591. 1959.

[Kleinberg 1997] Jon M. Kleinberg. "Two algorithms for nearest-neighbor search in high dimensions" in *Proceedings of the 29th Annual ACM symposium on Theory of computing*, pp. 599--608. El Paso, Texas, USA, New York, NY, USA. DOI: 10.1145/258533.258653.

[Lawder 2000] Jonathan Lawder. *Calculation of Mappings Between One and n-Dimensional Values Using the Hilbert Space-filling Curve*. Technical Report JL1/00, 13 p. University of London, Birkbeck College, London, United Kingdom. August 15, 2000.

[Lejsek 2005] Herwig Lejsek, Heiðar F. Ásmundsson, Jónsson Björnþór and Laurent Amsaleg. *Efficient and effective image copyright enforcement*. (Publication interne — IRISA, ISSN 1166-8687). Technical report IRISA-PI-05-1699, 24 p. Institut national de recherche en

informatique et en automatique, Institut de recherche en informatique et systèmes aléatoires, Rennes, France.

[Lejsek 2006] Herwig Lejsek, Fridrik H. Ásmundsson, Björn Thór Jónsson and Laurent Amsaleg. "Scalability of local image descriptors: a comparative study" in *Proceedings of the 14th Annual ACM International Conference on Multimedia* 589–598. Santa Barbara, CA, USA, October 23–27, 2006. ACM, New York, NY, USA. DOI: 10.1145/1180639.1180760.

[Lejsek 2008] Herwig Lejsek, Friðrik Heiðar Ásmundsson, Björn Þór Jónsson and Laurent Amsaleg. *NV-tree: An Efficient Disk-Based Index for Approximate Search in Very Large High-Dimensional Collections*. IEEE Transactions on Pattern Analysis and Machine Intelligence, n. preprint. IEEE Computer Society, Washington, DC, USA. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2008.130. May 20, 2008.

[Lepetit 2005] Vincent Lepetit, Pascal Lagger and Pascal Fua. "Randomized Trees for Real-Time Keypoint Recognition" in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, pp. 775–781. June 20–26, 2005. IEEE Computer Society, Washington, DC, USA. DOI: 10.1109/CVPR.2005.288.

[Lewis 2004] Paul H. Lewis, Kirk Martinez, Fazly Salleh Abas, Mohammed Faizal Ahmad Fauzi, Stephen C. Y. Chan, Matthew J. Addis, Mike J. Boniface, Paul Grimwood, Alison Stevenson, Christian Lahanier and James Stevenson. *An integrated content and metadata based retrieval system for art*. IEEE Transactions on Image Processing, vol. 13, n. 3, pp. 302–313. IEEE Signal Processing Society. ISSN: 1057-7149. DOI: 10.1109/TIP.2003.821346. March 2004.

[Li 2002] Chen Li, Edward Chang, Hector Garcia-Molina and Gio Wiederhold. *Clustering for approximate similarity search in high-dimensional spaces*. IEEE Transactions on Knowledge and Data Engineering, vol. 14, n. 4, pp. 792–808. IEEE Educational Activities Department, Piscataway, NJ, USA. ISSN: 1041-4347. DOI: 10.1109/TKDE.2002.1019214. July 2002.

[Liao 2001] Swanwa Liao, Mario A. Lopez and Scott T. Leutenegger. "High Dimensional Similarity Search With Space Filling Curves" in *Proceedings of the International Conference on Data Engineering*, pp. 615–622. Heidelberg, Germany, April 02–06, 2001. IEEE Computer Society Washington, DC, USA. DOI: 10.1109/ICDE.2001.914876.

[Lin 1994] King-Ip Lin, H. V. Jagadish and Christos Faloutsos. *The TV-tree: An index structure for high-dimensional data*. The VLDB Journal — The International Journal on Very Large Data Bases, vol. 3, n. 4, pp. 517–542. Springer, Berlin / Heidelberg, Germany. ISSN: 1066-8888. DOI: 10.1007/BF01231606. October 1994.

[Lowe 2005] David Lowe. *Demo Software: SIFT Keypoint Detector v. 4*. July 2005. Computer program. Retrieved on April 24, 2008 from http://www.cs.ubc.ca/~lowe/keypoints/.

[Lowe 1999] David G. Lowe. "Object Recognition from Local Scale-Invariant Features" in *Proceedings of the 7th International Conference on Computer Vision*, vol. vol. 2, p. 1150. September 20–25, 1999. IEEE Computer Society. DOI: 10.1109/ICCV.1999.790410.

[Lowe 2004] David G. Lowe. *Distinctive Image Features from Scale-Invariant Keypoints* International Journal of Computer Vision, vol. 60, n. 2, pp. 91–110. Springer, Netherlands. ISSN: 0920-5691 (Print) 1573-1405 (Online). DOI: 10.1023/B:VISI.0000029664.99615.94. November 2004.

[Mainar-Ruiz 2006] Gloria Mainar-Ruiz and Juan-Carlos Pérez-Cortés. "Approximate Nearest Neighbor Search using a Single Space-filling Curve and Multiple Representations of the Data Points" in *Proceedings of the 18th International Conference on Pattern Recognition*, vol. 2, pp. 502–505. Wan Chai, Hong Kong, August 20–24, 2006. IEEE Computer Society, Washington, DC, USA. DOI: 10.1109/ICPR.2006.275

[McKee 2004] Sally A. McKee. "Reflections on the memory wall" in *Proceedings of Conference on Computing frontiers*, pp. 162–167. Ischia, Italy, April 14–16, 2004. ACM Press, New York, New York, USA. DOI: 10.1145/977091.977115.

[Megiddo 1997] Nimrod Megiddo and Uri Shaft. *Efficient nearest neighbor indexing based on a collection of space filling curves*. Research Report RJ 10093. IBM Almaden Research Center, San Jose, California, USA. 1997.

[Mikolajczyk 2005] Krystian Mikolajczyk and Cordelia Schmid. *A Performance Evaluation of Local Descriptors*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, n. 10, pp. 1615–1630. IEEE Computer Society, Washington, DC, USA. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2005.188. October 2005.

[Mindru 2004] Florica Mindru, Tinne Tuytelaars, Luc Van Gool and Theo Moons. *Moment invariants for recognition under changing viewpoint and illumination*. Computer Vision and Image Understanding, vol. 94, n. 1–3, pp. 3–27. Elsevier Science Inc., New York, NY, USA. ISSN: 1077-3142. DOI: 10.1016/j.cviu.2003.10.011. April–June 2004.

[Moëllic 2007] Pierre-Alain Moëllic and Christian Fluhr. "ImagEVAL Official Results" in *ImagEVAL Workshop*. University of Amsterdam, Amsterdam, Netherlands, July 12, 2007.

[Moënne-Loccoz 2005] Nicolas Moënne-Loccoz. *High-Dimensional Access Methods for Efficient Similarity Queries*. Technical Report 05.05. University of Geneva, Computer Science Center Computer Vision Group, Geneva, Switzerland. May 2005.

[Mokbel 2003] Mohamed F. Mokbel, Walid G. Aref and Ibrahim Kamel. *Analysis of Multi-Dimensional Space-Filling Curves*. Geoinformatica, vol. 7, n. 3, pp. 179–209. Kluwer Academic Publishers Hingham, Massachussetts, USA. ISSN: 1384-6175. DOI: 10.1023/A:1025196714293 September 2003.

[Moon 1996] Bongki Moon, H. V. Jagadish, Christos Faloutsos and Joel H. Saltz. *Analysis of the clustering properties of Hilbert space-filling curve*. (Computer Science Technical Report Series). Technical Report CS-TR-3611, 24 p. University of Maryland at College Park, College Park, Maryland, USA. 1996.

[Moore 1991] Andrew W. Moore. *An Intoductory Tutorial on Kd-Trees*. Technical Report No. 209. University of Cambridge, Computer Laboratory, Cambridge, UK.

[Moore 1900] Eliakim Hastings Moore. *On Certain Crinkly Curves*. Transactions of the American Mathematical Society, vol. 1, n. 1, pp. 72–90. American Mathematical Society. ISSN: 00029947. DOI: 10.2307/1986405. January 1900.

[Morton 1966] G. M. Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. Technical Report. IBM Ltd., Ottawa, Ontario, Canada. 1966.

[Nene 1997] Sameer A. Nene and Shree K. Nayar. *A Simple Algorithm for Nearest Neighbor Search in High Dimensions*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, n. 9, pp. 989--1003. IEEE Computer Society, Washington, DC, USA. ISSN: 0162-8828. DOI: 10.1109/34.615448. September 1997.

[Netpbm 2007] Netpbm. *Netpbm home page*. July 16, 2007. Web page. Retrieved on April 24, 2008 from http://netpbm.sourceforge.net/.

[Overmars 1983] Mark H. Overmars. *The Design of Dynamic Data Structures*. 1st ed. (Lecture Notes in Computer Sciences, vol. 156). Springer-Verlag, Berlin / Heildelberg / New York / Tokyo. ISBN: 038712330X. 1983.

[Pass 1997] Greg Pass, Ramin Zabih and Justin Miller. "Comparing images using color coherence vectors" in *Proceedings of the 4th ACM international conference on Multimedia*, 65–73. Boston, Massachusetts, USA, November 18–22, 1996. ACM, New York, NY, USA. DOI: 10.1145/244130.244148.

[Peano 1890] Giuseppe Peano. *Sur une courbe, qui remplit toute une aire plane*. Mathematische Annalen, vol. 26, n. 1. Springer, Berlin / Heidelberg, Germany. ISSN: 0025-5831 (Print) / 1432-1807 (Online). DOI: 10.1007/BF01199438. March 1890.

[Pérez-Cortés 1998] Juan-Carlos Pérez-Cortés and Enrique Vidal. "An Approximate Nearest Neighbours Search Algorithm based on the Extended General Spacefilling Curves Heuristic" in *Workshop on Statistical Pattern Recognition*. Sydney, Australia, August 11–13, 1998.

[Philipp-Foliguet 2006] Sylvie Philipp-Foliguet, G. Logerot, P. Constant, Philippe H. Gosselin and Christian Lahanier. "Multimedia indexing and fast retrieval based on a vote system" in *IEEE International Conference on Multimedia and Expo*, pp. 1781–1784. 2006. DOI: 10.1109/ICME.2006.262897

[Procopiuc 2003] Octavian Procopiuc, Pankaj K. Agarwal, Lars Arge and Jeffrey Scott Vitter. "Bkd-tree: a dynamic scalable kd-tree" in *Advances in Spatial and Temporal Databases*. p. 46–65. (Lecture Notes in Computer Science, vol. 2750/2003). Springer, Berlin / Heidelberg, Germany. ISBN: 978-3-540-40535-1. DOI: 10.1007/b11839. 2003.

[Robinson 1981] John T. Robinson. "The KDB-tree: a search structure for large multidimensional dynamic indexes" in *Proceedings of the 1981 ACM SIGMOD international conference on Management of data* pp. 10–18. Ann Arbor, Michigan, April 29–May 01, 1981. ACM, New York, NY, USA. DOI: 10.1145/582318.582321.

[Sagan 1994] Hans Sagan. *Space-filling curves*. 193 p. Springer-Verlag, Berlin / New York. ISBN: 0387942653. September 2, 1994.

[Samet 1990] Hanan J. Samet. *The Design and Analysis of Spatial Data Structures*. 510 p. (Addison-Wesley series in computer science). Addison-Wesley, Reading, Massachusetts, USA. ISBN: 0-201-50255-0. 1990.

[Schmid 1997] Cordelia Schmid and Roger Mohr. *Local Greyvalue Invariants for Image Retrieval*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, n. 5, pp. 530–535. IEEE Computer Society, Washington, DC, USA. ISSN: 0162-8828. DOI: 10.1109/34.589215. May 1997.

[Shepherd 1999] John A. Shepherd, Xiaoming Zhu and Nimrod Megiddo. "A fast indexing method for multidimensional nearest neighbor search" in *SPIE Conference on Storage and Retrieval for Image and Video Databases VII*, vol. 3656, pp. 350–355. San Jose, California, USA, January 26-29, 1999. DOI: 10.1117/12.333854.

[Sierpiński 1912] Wacław Franciszek Sierpiński. *Sur une nouvelle courbe continue qui remplit toute une aire plane*. Bulletin de l'Académie des Sciences de Cracovie — Série A, pp. 463–478. 1912.

[Silpa-Anan 2005] Chanop Silpa-Anan and Richard Hartley. "Visual localization and loop-back detection with a high resolution omnidirectional camera" in *Workshop on Omnidirectional Vision*. 2005.

[Skubalska-Rafajłowicz 1996] Ewa Skubalska-Rafajłowicz and Adam Krzyżak. "Fast k-NN Classification Rule Using Metrics on Space-Filling Curves " in *Proceedings of the International Conference on Pattern Recognition*, vol. 2, pp. 121–126. Vienna, Austria, August 25–29, 1996. IEEE Computer Society, Washington, DC, USA. DOI: 10.1109/ICPR.1996.546736.

[Skubalska-Rafajłowicz 1994] Ewa Skubalska-Rafajłowicz. *The Closed Curve Filling Multidimensional Cube*. Technical Report, 16 p. Technical University of Wroc ław, Intitute of Engineering Cybernetics, Wroc ław, Poland. 1994.

[Smeulders 2000] Arnold W. M. Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta and Ramesh Jain. *Content-Based Image Retrieval at the End of the Early Years*. IEEE Transac-

tions on Pattern Analysis and Machine Intelligence, vol. 22, n. 12, pp. 1349–1380. IEEE Computer Society, Washington, DC, USA. ISSN: 0162-8828. DOI: 10.1109/34.895972. December 2000.

[Szumilas 2008] Lech Szumilas. *Scale and Rotation Invariant Shape Matching*. Ph.D. thesis, 150 p. Technischen Universität Wien, Fakultät für Informatik, Vienna, Austria. February 22, 2008.

[Tamminen 1981] Markku Tamminen. *Order preserving extendible hashing and bucket tries*. BIT Numerical Mathematics, vol. 21, n. 4, pp. 419–435. Springer Netherlands. ISSN: 0006-3835 (Print) / 1572-9125 (Online). DOI: 10.1007/BF01932839. December 1981.

[Traina Jr. 2000]Caetano Traina Jr., Agma Traina, Bernhard Seeger and Christos Faloutsos. "Slim-Trees: High Performance Metric Trees Minimizing Overlap between Nodes" in *Advances in Database Technology — EDBT 2000*. pp. 51–65. (Lecture Notes in Computer Science, vol. 1777/2000). Springer, Berlin / Heidelberg, Germany. ISBN: 978-3-540-67227-2. DOI: 10.1007/3-540-46439-5_4. 2000.

[Tropf 1981] H. Tropf and H. Herzog. *Multimensional Range Search in Dynamically Balanced Trees*. Angewandte Informatik, vol. 23, n. 2, pp. 71–77. Vieweg Verlag, Wiesbaden, Hessen, Germany. 1981.

[Valle 2003] Eduardo Valle. *Sistemas de Informação Multimídia na Preservação de Acervos Permanentes*. M.Sc. dissertation, 128 p. Universidade Federal de Minas Gerais, Departamento de Ciência da Computação, Belo Horizonte, MG, Brazil. February 21, 2003.

[Valle 2006] Eduardo Valle, Matthieu Cord and Sylvie Philipp-Foliguet. "Content-Based Retrieval of Images for Cultural Institutions Using Local Descriptors" in *Geometric Modeling and Imaging — New Trends*, pp. 177–182. London, UK, July 5–7, 2006. IEEE Computer Society, Washington, DC, USA. DOI: 10.1109/GMAI.2006.16

[Valle 2007a] Eduardo Valle, Matthieu Cord and Sylvie Philipp-Foliguet. "3-Way-Trees: A Similarity Search Method for High-Dimensional Descriptor Matching" in *Proceedings of the 2007 IEEE International Conference on Image Processing*, vol. 1, pp. I.173–176. San Antonio, TX, USA, September 16–19, 2007. IEEE Computer Society, Washington, DC, USA. DOI: 10.1109/ICIP.2007.4378919.

[Valle 2007b] Eduardo Valle, Matthieu Cord and Sylvie Philipp-Foliguet. "Matching Local Descriptors for Image Identification on Cultural Databases" in *Proceedings of the 9th International Conference on Document Analysis and Recognition*, vol. 2, pp. 679–683. Curitiba, PR, Brazil, September 23–26, 2007. IEEE Computer Society, Washington, DC, USA. DOI: 10.1109/ICDAR.2007.4377001.

[Valle 2007c] Eduardo Valle, Sylvie Philipp-Foliguet and Matthieu Cord. "Image Identification using Local Descriptors (Task 1)" in *ImagEVAL Workshop*. University of Amsterdam, Amsterdam, Netherlands, July 12, 2007.

[Ward 2001] Annette Ward, Neil Eliot, Margaret Graham, Jonathan Riley, Nic Sheen, John Eakins and Alice Barnaby. "Collage and Content-based Image Retrieval: Collaboration for enhanced services for the London Guildhall Library" in *Museums and the Web 2001*. March 15-17, 2001. Archives & Museum Informatics, Toronto, ON, Canada.

[Weber 1997] Roger Weber and Stephen Blott. *An Approximation-Based Data Structure for Similarity Search*. Report TR1997b. ETH Zentrum, Zurich, Switzerland. 1997.

[Weber 1998] Roger Weber, Hans-Jörg Schek and Stephen Blott. "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces" in *Proceedings of the 24rd International Conference on Very Large Data Bases*, 194–205. New York, NY, USA, August 24–27, 1998. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[White 1996] David A. White and Ramesh Jain. “Similarity indexing with the SS-tree” in *Proceedings of the 12th International Conference on Data Engineering*, pp. 516–523. New Orleans, LA, USA, February 26–March 01, 1996. IEEE. DOI: 10.1109/ICDE.1996.492202.

[Wikimedia 2008] Wikimedia, Foundation Inc. *Wikimedia Commons homepage*. Web page. Retrieved on April 24, 2008 from http://commons.wikimedia.org/.

[Young 1978] H. Peyton Young and Arthur Levenglick. *A Consistent Extension of Condorcet's Election Principle*. SIAM Journal on Applied Mathematics, vol. 35, n. 2, pp. 285–300. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA. ISSN: 00361399. September 1978.

[Zhang 2004] Rui Zhang, Beng Chin Ooi and Kian-Lee Tan. “Making the Pyramid Technique Robust to Query Types and Workloads” in *Proceedings of the 20th International Conference on Data Engineering*, pp. 313--324. Boston, MA, USA, March 30--April 2, 2004. IEEE Computer Society, Washington, DC, USA. DOI: 10.1109/ICDE.2004.1320007.

[Zhang 1996] Tian Zhang, Raghu Ramakrishnan and Miron Livny. *BIRCH: an efficient data clustering method for very large databases*. ACM SIGMOD Record, vol. 25, n. 2, pp. 103–114. ACM, New York, NY, USA. ISSN: 0163-5808. DOI: 10.1145/235968.233324. June 1996.

Image credits

Figure 2.1 Yorck Project, put in public domain by Directmedia Publishing GmbH.

Figure 2.2 Bibliothèque Nationale de France, Département des Estampes et Photographies.

Figure 5.1 Arquivo Público Mineiro.

Figure 5.2 Yorck Project.

Figure 5.36 © EYEDEA, all rights reserved.

Figure 5.37 Arquivo Público Mineiro.

Figure 5.38 © EYEDEA, all rights reserved.

Index

Composed with Garamond and Helvetica typefaces in Microsoft Word 2007.
Mathematical formulæ composed in Times New Roman and Euclid typefaces using MathType 6.
The layout on this file was optimised for reading on paper and for duplex-printing. For other uses, check on the author website www.eduardovalle.com for more convenient formats.