Proceedings

Third International Conference on SImilarity Search and APplications

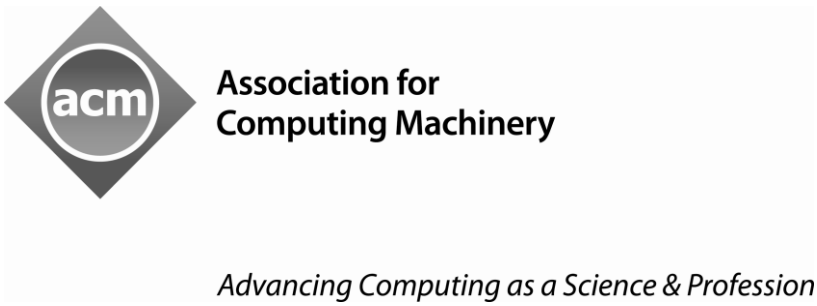
2010

Proceedings

Third International Conference on SImilarity Search and APplications

18-19 September 2010
Istanbul, Turkey

Editors:
Paolo Ciaccia and Marco Patella



The Association for Computing Machinery
2 Penn Plaza, Suite 701
New York New York 10121-0701

Notice to Past Authors of ACM-Published Articles

ACM intends to create a complete electronic archive of all articles and/or other material previously published by ACM. If you have written a work that was previously published by ACM in any journal or conference proceedings prior to 1978, or any SIG Newsletter at any time, and you do NOT want this work to appear in the ACM Digital Library, please inform permissions@acm.org, stating the title of the work, the author(s), and where and when published.

ACM ISBN: 978-1-4503-0420-7

Table of Contents

Applications

Demonstrations

Posters

Foreword

The *International Conference on Similarity Search and Applications (SISAP)* is a conference devoted to similarity searching, with emphasis on metric space searching. It aims to fill in the gap left by the various scientific venues devoted to similarity searching in spaces with coordinates, by providing a common forum for theoreticians and practitioners around the problem of similarity searching in general spaces (metric and non-metric) or using distance-based (as opposed to coordinate-based) techniques in general.

SISAP aims to become an ideal forum to exchange real-world, challenging and exciting examples of applications, new indexing techniques, common testbeds and benchmarks, source code, and up-to-date literature through a web page serving the similarity searching community. Authors are expected to use the testbeds and code from the SISAP web site (`www.sisap.org`) for comparing new applications, databases, indexes and algorithms.

After the very successful first events in Cancun, Mexico (2008) and Prague, Czech Republic (2009), this year SISAP conference has been held in Istanbul, Turkey, on September 18-19, 2010.

Preface

This volume contains 2 invited papers, 11 regular papers, 2 posters and 6 demos, presented at SISAP 2010 conference, held in Istanbul, Turkey, on September 18-19, 2010. The papers were selected from 29 submissions, by a renowned program committee. We received submissions from all over the world (Argentina, Chile, China, Cuba, Czech Republic, Germany, Greece, Italy, Japan, Mexico, Norway, Spain, and USA). In this third volume of the SISAP series, the acceptance ratio of 66% was achieved as a result of the increasing quality of submissions established last year. The papers were selected based on their originality, relevance, and technical strength.

The submissions cover the spectrum of similarity searching topics from novel indexing techniques, to approximate methods for speeding up similarity search in metric/non-metric spaces, to applications in computerized biology, image classifications, and content-based image retrieval. In the demonstrations section a large number of interesting prototypes of similarity search engines, related to content- and semantics-based retrieval of multimedia data (audio, images, and video), was presented. Finally, the posters present an approach for balancing splits in indexing techniques based on generalized hyperplane partitioning and a system for automatic video annotation based on similarity among detected shots.

Two invited speakers gave excellent and thought-provoking talks. Prof. Vladimir Pestov provided interesting connections between similarity searching in metric spaces, the phenomenon of concentration of measure on high-dimensional structures, and the Vapnik-Chervonenkis theory of statistical learning, deriving bounds on performance of well-know indexing structures and showing how statistical learning methods can be used to build indexing schemes in datasets drawn from distributions of lower intrinsic dimension. Prof. Tomáš Skopal discussed the impact of the metric indexing paradigm on real-world applications, analyzing the world of similarity searching in metric spaces from the point of view of practitioners, transforming his conclusions into research challenges and perspectives of the similarity search methods. These contributions given by leading researchers definitely contributed to an in-depth and up-to-date understanding of fascinating fields and will surely stimulate further research.

Extended versions of the best SISAP 2010 papers were collected and will be published in a special issue of Journal of Discrete Algorithms (Elsevier).

We are grateful to the program committee members, the local organizers, and the external reviewers for their committed and high-quality work. We highly appreciate the work done by Karina Figueroa for maintaining the Metric Library, and those who contributed to it. Reviewing of papers and preparation of the proceedings were done using the EasyChair conference system (`www.easychair.org`), for which we specially thank Andrei Voronkov. Last but not least, we thank Adrienne Griscti (ACM program manager for SIG publications) for her hard work in helping us publishing these proceedings.

Paolo Ciaccia and **Marco Patella**
Istanbul, Turkey
September 19th, 2010

Conference Organization

Program Chairs

Paolo Ciaccia, *Università di Bologna*
Marco Patella, *Università di Bologna*

Program Committee

Edgar Chávez, *Universidad Michoacana - CICESE*
Alfredo Ferro, *Università degli Studi di Catania*
Daniel Keim, *Universität Konstanz*
Daniel Miranker, *University of Texas at Austin*
Gonzalo Navarro, *Universidad de Chile*
Hanan Samet, *University of Maryland*
Tomáš Skopal, *Univerzita Karlova v Praze*
Agma Juci Machado Traina, *Universidade de São Paulo*
Pavel Zezula, *Masarykova Univerzita*

Local Organization

Cengiz Celik, *Bilkent University*

Referees

Maria Camila Nardini Barioni
Benjamin Bustos
Geoff Ellis
Karina Figueroa
David Hoksza
Curran Kelleher
Milos Krstajic
Jakub Lokoc
Mauricio Marin
Jiri Novak
Daniela Oelke
Oscar Pedreira
Vladimir Pestov
Giuseppe Pigola
Ives Pola
Alfredo Pulvirenti
Smriti Ramakrishnan
Humberto Razente
Nora Reyes
Matthias Schaefer
Andreas Stoffel
Andrada Tatu
Marcos Vieira
Leishi Zhang

Invited papers

SISAP 2010

Indexability, concentration, and VC theory

Vladimir Pestov
Department of Mathematics and Statistics, University of Ottawa
585 King Edward Avenue, Ottawa, Ontario, Canada K1N6N5
vpest283@uottawa.ca

ABSTRACT

Degrading performance of indexing schemes for exact similarity search in high dimensions has long since been linked to histograms of distributions of distances and other 1-Lipschitz functions getting concentrated. We discuss this observation in the framework of the phenomenon of concentration of measure on the structures of high dimension and the Vapnik-Chervonenkis theory of statistical learning.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information search and retrieval—*Search process*; F.2.2 [**Theory of Computation**]: Analysis of algorithms and problem complexity—*Geometrical problems and computations*

1. INTRODUCTION

At an intuitive level, at least for a limited class of indexing schemes the geometric and probabilistic origin of the curse of dimensionality is quite transparent. Let $W = (\Omega, \rho, X)$ denote a similarity workload, where ρ is a metric on a domain Ω and X is a finite subset of Ω (dataset). Let us say we are interested in indexing into W for deterministic, exact range queries. A traditional "distance-based" indexing scheme, stripped down to the bone, consists of a family of real-valued 1-Lipschitz functions f_i, $i \in I$ on Ω, either fully or partially defined:

$$|f_i(x) - f_i(y)| \leq \rho(x, y). \tag{1}$$

(For example, a pivot-based indexing scheme will be using distance functions $\rho(p, -)$ to the pivots p.) Given a query (q, ε), where $q \in \Omega$ and $\varepsilon > 0$, the algorithm chooses recursively a sequence of indices i_n, where i_{n+1} is determined by the previous values $f_{i_k}(q)$, $k \leq n$. The functions f_i serve to discard those datapoints which cannot possibly answer the query. Namely, if $|f_i(q) - f_i(x)| \geq \varepsilon$, then, by the 1-Lipschitz property of f_i, one has $\rho(q, x) \geq \varepsilon$, and so the point x is irrelevant and need not be considered (Figure 1).

SISAP 2010, September 18-19, 2010, Istanbul, Turkey.

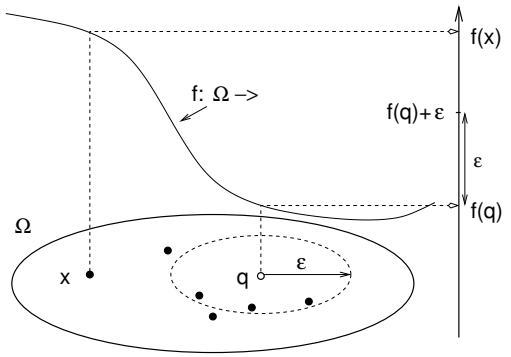


Figure 1: The datapoint x can be discarded.

After the calculation terminates, the algorithm returns all points which cannot be discarded, and checks each one of them against the condition $\rho(x, q) < \varepsilon$.

Next come two standard observations about high-dimensional data. The first one, known as the "empty space paradox," asserts that the average distance $\mathbb{E}(\varepsilon_{NN})$ to the nearest neighbour approaches the average distance $\mathbb{E}(\rho)$ between two datapoints as the dimension d goes to infinity, provided the number of datapoints, n, grows subexponentially in d. Cf. Figure 2, where we illustrate the point with a constant number of points ($n = 10^3$ and $n = 10^5$), and the distances are normalized so that the *characteristic size* of the gaussian space $(\mathbb{R}^n, \gamma^n)$,

$$\text{CharSize}\,(X) = \mathbb{E}_{\mu\otimes\mu}(\rho(x, y)), \tag{2}$$

is one.

The second observation is that the histograms of values of common 1-Lipschitz functions on high-dimensional data are concentrated near their mean (or median) values. This effect is already pronounced in moderate dimensions such as $d = 14$ in Figure 3. Here the function is a distance to a randomly chosen pivot p, and assuming the query point q is at a distance ≈ 1 from p, only the points outside of the region marked by vertical bars can be discarded.

The two properties combined imply that as $d \to \infty$, fewer and fewer datapoints can be discarded for an average range query, and the performance of an indexing scheme degrades rapidly. This mechanism has been discussed repeatedly, e.g. in [9], pp. 35–37, [32], [42], p. 487, to mention just a few.

To make this idea yield rigorous lower performance bounds, one needs to guarantee first that *every* histogram of dis-

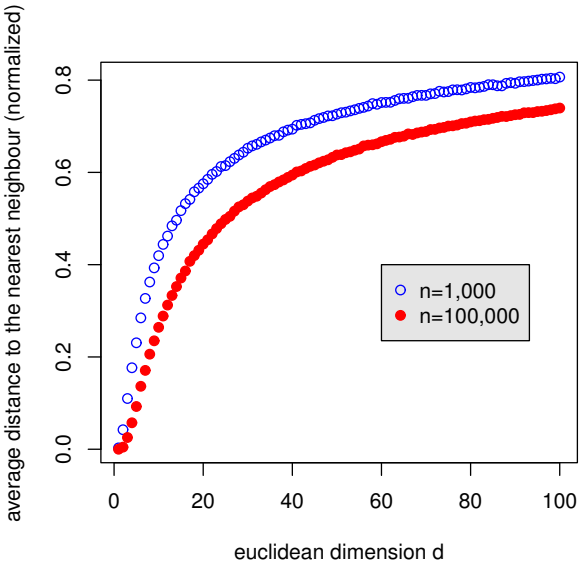


Figure 2: The normalized average distance to the nearest neighbour in a dataset of n points randomly drawn from a gaussian distribution in $\mathbb{R}^d$.

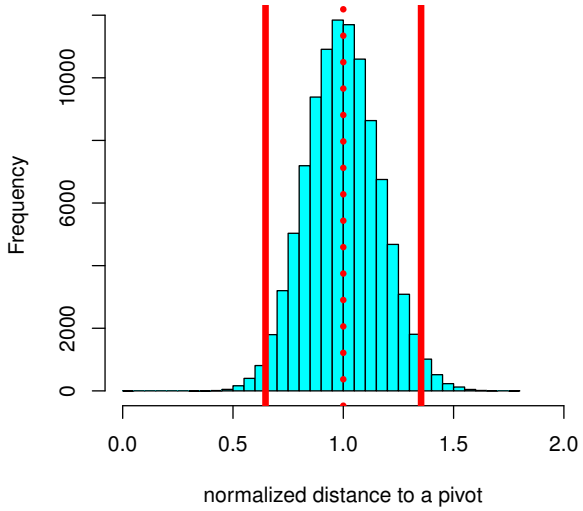


Figure 3: Histogram of distances to a randomly chosen pivot in a dataset X of $n = 10^5$ points drawn from a gaussian distribution in $\mathbb{R}^{14}$. The vertical lines mark the mean normalized distance $1 \pm \varepsilon_{NN}$.

tances of 1-Lipschitz functions used to build an indexing scheme is highly concentrated. In other words, if $\mathscr{F}$ denotes a class of 1-Lipschitz functions from which we choose the f_i, we want a small uniform upper bound on the variances of $f \in \mathscr{F}$. Results of this type are indeed well-known for a variety of geometric objects and are referred jointly as the *phenomenon of concentration of measure*.

Next problem is, how to link the concentration of functions f with regard to the underlying distribution μ on the domain Ω to concentration with regard to the *empirical measure* on the dataset X (this was essentially a criticism of [37] made in [43])? Here one needs the machinery of *statistical learning theory* of Vapnik and Chervonenkis, which can guarantee such results provided the class $\mathscr{F}$ has low capacity (e.g., a finite VC dimension). This way, one obtains $\Omega(n/d \lg n)$ lower bounds for the pivot table expected average performance, as well as superpolynomial in d lower bounds for metric trees.

Approximate NN queries seem to be in some sense free from the curse of dimensionality. In fact, the concentration of measure becomes a positive force here, and we will try to explain why, using the example of random projections method (the Johnson–Lindenstrauss lemma), as well as the approach of Kushilevitz, Ostrovsky and Rabani based on VC theory.

Getting back to exact search, the *Curse of Dimensionality Conjecture* calls for a much more general statement than the lower bounds discussed above, which would apply across the entire range of all possible indexing schemes. The conjecture is still open even for the Hamming cube $\{0,1\}^n$, and we discuss it briefly.

We conclude the article with remarks on the notion of intrinsic dimensionality of data, as well as on a spatial approximation algorithm based on Delaunay graphs.

2. CONCENTRATION

2.1 The concentration of measure phenomenon

Informally, the phenomenon can be stated as follows:

> On a typical "high-dimensional" structure, the variance of every 1-Lipschitz function is small.

This formulation is not quite exact (it is a slightly stronger statement), and in fact a usual way to deal with concentration is through a different dispersion parameter. Let a metric space (Ω, ρ) carry a probability measure μ. One defines the *concentration function* α_Ω of the metric space with measure Ω:

$$\alpha_\Omega(\varepsilon) = \begin{cases} \frac{1}{2}, & \text{if } \varepsilon = 0, \\ 1 - \inf\left\{\mu\left(A_\varepsilon\right) : A \subseteq \Omega, \quad \mu_\sharp(A) \geq \frac{1}{2}\right\}, & \text{if } \varepsilon > 0. \end{cases}$$

The value $\alpha_\Omega(\varepsilon)$ gives a uniform upper bound on the measure of the complement to the ε-neighbourhood A_ε of every subset A of measure $\geq 1/2$, cf. Fig. 4. On a typical high-dimensional geometric object Ω the function $\alpha(\varepsilon)$ drops off steeply near zero. For regular geometric objects such as Hamming cubes, Euclidean unit spheres and so on, there are usually gaussian upper bounds of the form $\alpha(\varepsilon) \leq \exp(-O(\varepsilon^2 d))$, where d is the dimension parameter. For example, the Hamming cube $\{0,1\}^d$ with the normalized Hamming metric and uniform measure satisfies a Chernoff bound $\alpha(\varepsilon) \leq \exp(-2\varepsilon^2 d)$ (obtained by combining Harper's isoperimetric inequality, see e.g. [16], with the classical Chernoff bound, cf. [49], 2.2.1). See Figure 5.

It follows easily that for every real-valued 1-Lipschitz function f on Ω and for each $\varepsilon > 0$ one has

$$\mu\{x \in \Omega\colon\ |f(x) - M_f| > \varepsilon\} \leq 2\alpha_\Omega(\varepsilon), \qquad (3)$$

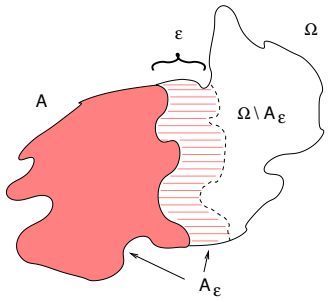


Figure 4: To the concept of the concentration function $\alpha_\Omega(\varepsilon)$.

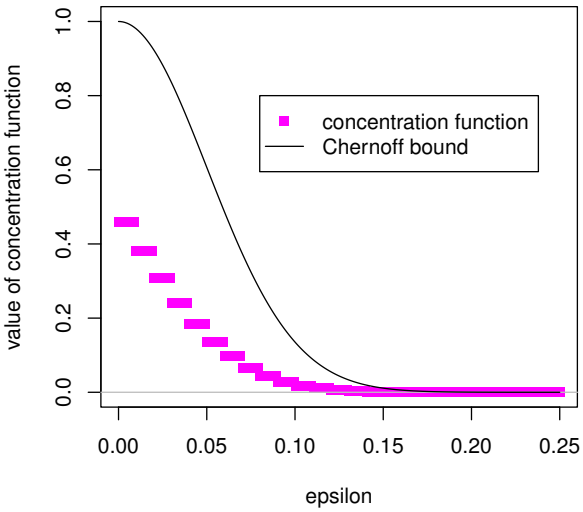


Figure 5: Concentration function of the Hamming cube of dimension $d = 100$ *vs* Chernoff bound.

where M_f is the median value of f. Under mild conditions (e.g. if Ω has finite diameter), this implies a uniform bound on variances. But usually this is Eq. (3) which gives the most useful form of concentration.

The phenomenon admits the following illustration. Draw $1,000$ points randomly from a high-dimensional sphere $\mathbb{S}^d$, choose a random orthogonal projection onto a two-dimensional subspace, and project both the sphere and the chosen points on this subspace. The points will concentrate near the centre, the more so the higher the dimension d, as seen in Figure 6 for $d = 20$ and $d = 1000$... except that the figure shows the results of the same experiment performed on the unit cubes of the same dimension d rather than spheres. However, for $d = 1000$ the only difference one can spot from a unit sphere, is the scale of the projection: the diameter of a unit d-cube is $O(\sqrt{d})$.

This illustrates an interesting feature of geometry of high dimensions: many high-dimensional objects look essentially the same to a low-dimensional observer. For instance, this is true of *all* convex bodies, as recently proved by Klartag [23]. For this reason, for an asymptotic study of lower bounds of

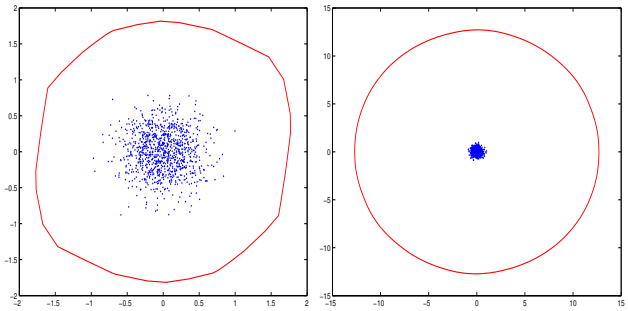


Figure 6: Orthogonal projection of a unit Euclidean d-cube and of $1,000$ random points inside the cube on a random 2-subspace, $d = 20$ (left), $d = 1000$ (right).

indexing schemes when $d \to \infty$ the choice of a particular family of domains (Euclidean spheres, balls, cubes, Hamming cubes) does not matter that much.

Among the books treating the concentration phenomenon, [30] is the most reader-friendly, [26] most comprehensive, and [18] contains a wealth of ideas. See also a survey [29].

2.2 Empty space paradox

Let us agree on the following assumptions on the similarity workload:

- the metric domain (Ω, ρ) is equipped with a probability measure μ, and datapoints are drawn from Ω in an i.i.d. fashion following the distribution μ (this is the model of [11]);
- the distance ρ on the domain is normalized so that the characteristic size of Ω is constant:
$$\mathrm{CharSize}(\Omega) = \mathbb{E}_{\mu\otimes\mu}(\rho) = O(1);$$
- Ω has concentration dimension d in the sense that the concentration function of Ω admits a gaussian upper bound
$$\alpha_\Omega(\varepsilon) = \exp(-\Omega(\varepsilon^2 d));$$
- the number n of datapoints grows faster than any polynomial function in d, but slower than any exponential function in d:
$$n = d^{\omega(1)}, \quad d = \omega(\log n). \tag{4}$$
(This is a standard assumption in the asymptotic analysis of algorithms, cf. [20]. An example of such a rate of growth is $n = 2^{\sqrt{d}}$.)

Denote ε_{NN} the nearest neighbour distance function on Ω, given by $\varepsilon_{NN}(\omega) = \rho(\omega, X)$.

THEOREM 2.1. *Under the above assumptions, with high confidence one has for every ε*

$$\mu\{\omega\colon\ |\varepsilon_{NN}(\omega) - \mathrm{CharSize}(\Omega)| > \varepsilon\} < \exp(-O(\varepsilon^2 d)).$$

The result applies to the Hamming cube, the Euclidean cube, the Euclidean space with gaussian measure, the Euclidean ball, etc. For a proof, see [40], Lemma 1.

COROLLARY 2.2. *Under the same assumptions, for every $\varepsilon > 0$ and sufficiently large d with high confidence the pairwise distances between datapoints of X are all in the range* $\mathrm{CharSize}\,(\Omega) \pm \varepsilon$.

For constant $n = |X|$ and a Euclidean domain the result was proved in [19].

3. VC THEORY

3.1 VC dimension

Let $\mathscr{C}$ denote a collection of subsets of the domain Ω. The VC dimension is a measure of combinatorial complexity of $\mathscr{C}$. A finite set $A \subseteq \Omega$ is *shattered* by $\mathscr{C}$ if every subset $B \subseteq A$ can be "carved out" of A with the help of a suitable element C of $\mathscr{C}$:

$$B = A \cap C.$$

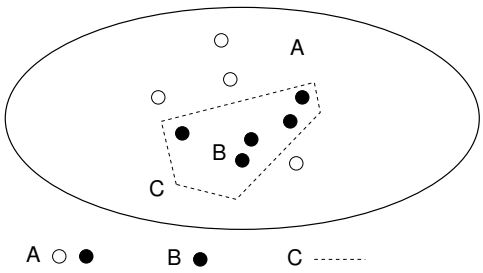


Figure 7: To the concept of a set A shattered by a class $\mathscr{C}$.

The *VC dimension* of $\mathscr{C}$, denoted $\mathrm{VC}(\mathscr{C})$, is the supremum of cardinalities of finite subsets of the domain shattered by $\mathscr{C}$. Here are some classical examples.

Family of sets	VC dimension
Intervals in $\mathbb{R}$	2
Half-spaces in $\mathbb{R}^d$	$d+1$
Euclidean balls of all radii in $\mathbb{R}^d$	$d+1$
Parallelepipeds in $\mathbb{R}^d$	$2d+2$
Family of n sets	$\leq \lg_2 n$
Balls in the Hamming d-cube	$\leq d + \lg_2 d$
Finite unions of intervals in $\mathbb{R}$	∞
Convex polygons in $\mathbb{R}^d$	∞
$\{\Omega \setminus C \colon C \in \mathscr{C}\}$	$\mathrm{VC}(\mathscr{C})$
$\mathscr{C} \cup \mathscr{D}$	$\leq \mathrm{VC}(\mathscr{C}) + \mathrm{VC}(\mathscr{D}) + 1$
k-fold intersections of els of $\mathscr{C}$	$\leq 2k \lg(ek)\mathrm{VC}(\mathscr{C})$

Proofs can be found in [49], Ch. 4. Estimating the VC dimension is often a non-trivial task, and for instance even for the collection of all cubes in $\mathbb{R}^d$ the value of this parameter seems to be unknown.

3.2 Uniform convergence of empirical measures

Now suppose $\mathscr{C}$ consists of *Borel subsets* of Ω, that is, members of the smallest family closed under countable intersections and complements and containing all open balls. (This assumption guarantees that the value $\mu(C)$ is well-defined for every probability measure μ on Ω.) The *empirical measure* of $C \in \mathscr{C}$ with regard to a finite sample $X = \{x_1, \ldots, x_n\}$ is just the normalized counting measure $\mu_n(C) = |\{i \colon x_i \in C\}|/n$. The VC dimension of $\mathscr{C}$ is finite if and only if, with high confidence, the empirical measures of every $C \in \mathscr{C}$ converge uniformly to the true value $\mu(C)$ as the sample size $n \to \infty$, no matter what μ is.

Here is a more exact formulation. The class $\mathscr{C}$ has the property of *uniform convergence of empirical measures*, or is a *uniform Glivenko–Cantelli class,* if there is a function $s(\delta, \varepsilon)$ (*sample complexity* of the class) so that, given a desired precision value $\varepsilon > 0$ and a risk level $\delta > 0$, whenever $n \geq s(\delta, \varepsilon)$, one has

$$\sup_{\mu \in P(\Omega)} P\left\{ \sup_{C \in \mathscr{C}} |\mu(C) - \mu_n(C)| \geq \varepsilon \right\} < \delta.$$

Here $P(\Omega)$ denotes the family of all probability measures on Ω. We quote the following as stated in [49], Theorem 7.8.

THEOREM 3.1. *A concept class $\mathscr{C}$ is uniform Glivenko–Cantelli if and only if $d = \mathrm{VC}(\mathscr{C}) < \infty$, in which case*

$$s(\delta, \varepsilon) \leq \max\left\{ \frac{8d}{\varepsilon} \lg \frac{8e}{\varepsilon}, \frac{4}{\varepsilon} \lg \frac{2}{\delta} \right\}.$$

The proof uses in an essential way the concentration of measure in the Hamming cube $\{0,1\}^n$.

Among a great selection of books treating VC theory, let us mention encyclopaedic sources [49] and [14], a classical monograph [47], and a lighter, but very well-written [2].

4. THE CURSE OF DIMENSIONALITY

4.1 Pivot tables

4.1.1 Reduction and access overhead

Every 1-Lipschitz mapping f from a domain Ω to a domain Υ can be viewed as a *projective reduction* of the exact similarity search problem to the new workload $(\Upsilon, f(X))$. This viewpoint is developed in some detail in [41]. If queries in Υ are easier to process than in Ω, it makes sense to retrieve all datapoints x in the ε-range query of $f(q)$ in Υ and then check them against the condition $\rho_\Omega(q, x) < \varepsilon$. The *access overhead* of the reduction f is defined as

$$\mathrm{acc}_f(q) = |X \cap f^{-1}(B_\varepsilon(f(q)))| - |X \cap B_\varepsilon(q)|.$$

This simple idea on its own can be surprisingly efficient, cf. [45].

4.1.2 Pivot-based reduction to $\ell^\infty(k)$

Every finite collection $f_1, f_2, \ldots, f_k$ of 1-Lipschitz functions on (Ω, ρ) defines a 1-Lipschitz mapping $f = \Delta_{i=1}^k f_i$ from Ω to $\ell^\infty(k)$ via

$$f(x) = (f_1(x), f_2(x), \ldots, f_k(x)).$$

Here $\ell^\infty(k)$ is the vector space $\mathbb{R}^k$ with the norm $\|x\|_\infty = \max_{i=1}^k |x_i|$. If the f_i are distance functions from pivot points $p_i \in \Omega$, the resulting mapping f is of the form

$$f(x) = (d(x, p_1), d(x, p_2), \ldots, d(x, p_k)) \in \ell^\infty(k). \quad (5)$$

In [48], it was proposed to use this reduction in case where the distance computations in Ω are so expensive that even a simple sequential scan of the image $f(X)$ in $\ell^\infty(m)$ is computationally cheaper. This idea was analyzed for more general similarity measures than metrics in [15]. By combining it

with other access methods on the space $\ell^\infty(m)$, further new indexing methods have been developed, see e.g. [8].

A m-NN similarity query is processed in (Ω, d, X) in time

$$k + \ell + (\mathrm{acc}_f(q) + m).$$

Here the first term stands for the calculation of k distances from a query point q to the pivots and ℓ is the processing time of a rectangular query in $\ell^\infty(k)$, while the latter expression lists the number of distance computations in Ω needed to separate false hits from k true positives. There is only one term above that can be minimized through the choice of the pivots. A classical paper on pivot selection is [6].

4.1.3 Lower query time bounds for pivot tables

Our next result (a slightly corrected version of the main theorem in [50]) is valid for the Hamming cube which is a testbed for asymptotic analysis of performance of indexing schemes, but also for the Euclidean space $\mathbb{R}^d$ with the gaussian measure, the cube $[0,1]^d$, and so forth.

THEOREM 4.1. *In addition to the assumptions of Subs. 2.2, suppose also that the VC dimension of the family of all balls in Ω is $O(d)$. Any pivot table with $k = o(n/d\log n)$ pivots will return an expected average number of $\Omega(n)$ datapoints. Consequently, the average total complexity of the performance of any pivot table for the resulting workload is $\Omega\left(n/d\log n\right)$.*

PROOF. Let ε_M denote the median value of the function ε_{NN}, so that for at least half query points q the distance to the NN in X is $\geq \varepsilon_M$. For each pivot p_i, $i = 1, 2, \ldots, k$, denote ρ_i^M the median value of the distance function $\rho(p_i, -)$. Because of concentration, the mesure of the spherical shell

$$S_i = \{q \in \Omega \colon \rho_i^M - \varepsilon_M/2 < \rho(p_i, q) < \rho_i^M + \varepsilon_M/2\},$$

is $1 - \exp(-\Omega(\varepsilon_M^2 d))$, and the measure of the intersection, $S = \cap_i S_i$, of all k shells is

$$1 - o(n/d)\exp(-\Omega(\varepsilon_M^2 d)) = 1 - \exp(-\Omega(\varepsilon_M^2 d)),$$

since n is subexponential in d. Thus, among all k-fold intersections of spherical shells (Figure 8), we have found a giant one, whose μ-measure is nearly one. To assure that it contains an accordingly high proportion of datapoints, consult the table in Subs. 3.1 to notice the family of all k-fold intersections of spherical shells in Ω has VC dimension not exceeding $2k\lg(ek)O(d) = o(n)$. By Theorem 3.1, the empirical measure $\mu_n(S)$ approaches $\mu(S)$ and therefore 1 with high confidence as $d \to \infty$. One concludes: about half query points will belong to S, and for them all datapoints belonging to S, that is, most datapoints, have to be returned. This gives expected average total complexity $\Omega(n)$. □

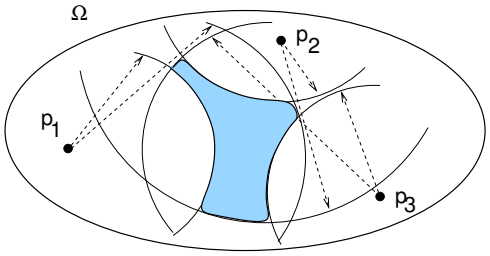


Figure 8: An intersection of spherical shells.

Notice that we allow the pivots p_i to be any points of the domain Ω. If we require that pivots be chosen from the dataset X, then the set S in the above proof is guaranteed to contain $n - k$ datapoints by Theorem 2.1 and Corollary 2.2, and we obtain (without using VC theory):

COROLLARY 4.2. *Under the assumptions of Subs. 2.2, if all pivots p_i belong to the dataset X, then the expected total complexity of the performance of the resulting pivot table is $\Omega(n)$.*

4.1.4 A remark on results of [15]

The above lower bounds agree with an exponential in d upper bound of $k + c^d$ derived in the influential paper [15], Theorem 3 within a similar model, with no restriction on a number n of datapoints, and with d a dimension parameter defined by a certain measure distribution density condition verified, e.g. by the Hamming cube $\{0,1\}^d$ or the Euclidean sphere $\mathbb{S}^d$. Here c is a constant depending on Ω, the smallest distortion parameter of a 1-Lipschitz embedding $f\colon \Omega \to \ell^\infty(k)$:

$$\forall x, y \in \Omega, \quad \|f(x) - f(y)\|_\infty \leq \rho(x,y) \leq c\,\|f(x) - f(y)\|_\infty .$$

However, the usefulness of the result is limited because of an imprecise claim (*loc. cit.*, Example 1) that for a bounded subset X of $\ell^2(d)$ there always exists a 1-Lipschitz function $f\colon X \to \ell^\infty(d+2)$ having distortion $c \leq 2$. In fact, an optimal constant here is on the order $O(\sqrt{d})$, and so the query performance estimate for the Euclidean domains made in Remark after the main Theorem 3, *loc.cit.*, becomes superexponential in d and thus meaningless.

This has created a misconception which led the authors of [6] to believe that using the prior knowledge of Euclideanness of the domain would make the construction of an optimal pivot-based scheme easy (cf. their remarks on p. 2358, end of first paragrph on the r.h.s., and at the beginning of Section 5).

4.2 Hierarchical metric tree schemes

4.2.1 Metric trees

For a finite rooted tree T we denote $L(T)$ the set of leaves of T and $I(T)$ the set of inner nodes. The symbol $*$ will denote the root node of T.

Let $\mathscr{F}$ be a class of 1-Lipschitz functions on Ω (possibly partially defined). A *metric tree* (*of type* $\mathscr{F}$) for a workload (Ω, ρ, X) is a hierarchical indexing structure consisting of

- a finite binary rooted tree $\mathcal{T}$,
- an assignment of a function $f_t \in \mathscr{F}$ (a *pruning,* or *decision function*) to every inner node $t \in I(T)$, and
- a collection of subsets $B_t \subseteq \Omega$, $t \in L(T)$ (*bins*), covering the dataset: $X \subseteq \cup_{t \in L(T)} B_t$.

For simplicity, we will assume that the tree T is binary and thus can be identified with a sub-tree of the prefix tree, that is, a subset of binary strings $\varepsilon_1\varepsilon_2\ldots\varepsilon_k$, $0 \leq k \leq n$, where $\varepsilon_i = \pm 1$ for all i. The indexing scheme is constructed so as to assure the following condition. Let $s = \varepsilon_1\varepsilon_2\ldots\varepsilon_m$ be a leaf node, and let x belong to the bin B_s labelled with s. Then for each $0 \leq k \leq m$, the value of $f_{\varepsilon_1\varepsilon_2\ldots\varepsilon_i}$ at x equals ε_{i+1}.

At each inner node $t = \varepsilon_1\varepsilon_2\ldots\varepsilon_l$ the value of the pruning function f_t at the query center q is evaluated. The condition $f_t(q) > \varepsilon$ gurantees that the child node $t(-1) =$

$\varepsilon_1\varepsilon_2\ldots\varepsilon_l(-1)$ need not be visited, because all elements x of the bins indexed with the descendants of $t(-1)$ are at a distance $> \varepsilon$ from q. Indeed, assuming $x \in B_\varepsilon(q)$, one has

$$|f_t(x) - f_1(q)| \leq d(x,q) \leq \varepsilon.$$

Similarly, if $f_t(q) < -\varepsilon$, then the node $t1 = \varepsilon_1\varepsilon_2\ldots\varepsilon_k 1$ can be pruned, because no bin labelled with descendants of $t1$ can possibly contain a point within the range ε from q.

However, if $f_t(q) \in [-\varepsilon, \varepsilon]$, then no pruning is possible and both children nodes of t have to be visited. The search branches out. In the presence of concentration, the amount of branching is considerable, and results in dimensionality curse.

The M-tree [10] is by now a classical example of a metric tree. However, metric tree-type indexing schemes are very numerous, cf. Sections 2.1-2.4 in [51] or Section 4.5 in [42].

4.2.2 Lower bounds for metric trees

THEOREM 4.3. *In addition to the assumptions of Subs. 2.2, suppose that the VC dimension of the class $\mathscr{F}$ of* 1*-Lipschitz functions used to construct a particular type of metric tree is* $\mathrm{poly}(d)$*. Then the expected average performance of a metric tree indexing structure is superpolynomial in d.*

That the assumption $\mathrm{VC}(\mathscr{F}) = \mathrm{poly}(d)$ is sensible, follows from a theorem of Goldberg and Jerrum [17]. Consider the parametrized class

$$\mathscr{F} = \{x \mapsto f(\theta, x) \colon \theta \in \mathbb{R}^s\}$$

for some $\{0,1\}$-valued function f. Suppose that, for each input $x \in \mathbb{R}^n$, there is an algorithm that computes $f(\theta, x)$, and this computation takes no more than t operations of the following types:

- the arithmetic operations $+, -, \times$ and / on real numbers,
- jumps conditioned on $>, \geq, <, \leq, =$, and $\neq$ comparisons of real numbers, and
- output 0 or 1.

Then $\mathrm{VC}(\mathscr{F}) \leq 4s(t+2)$.

ON THE PROOF OF THEOREM 4.3. (For details, see [40].) As the total content of bins B_t indexed with strings t of length superpolynomial in d has to be asymptotically negligible, we can assume the total number of bins to be $2^{\mathrm{poly}(d)}$. Statistical learning estimates imply that measures of bins cannot be too skewed, for a bin of large measure has to contain many points, leading to the desired estimate. Now concentration is used to prove that at least $\mathrm{poly}(d)$ bins B_t have size so large that the ε_M-neighbourhood of B_t has almost full measure. One deduces further that query centres q whose ε_{NN}-neighbourhood meets at least $d^{\omega(1)}$ bins have measure $1/2 - o(1)$. Processing the nearest neighbour query with such a centre q requires accessing all of these bins, let even to verify that it is empty. □

4.3 The curse of dimensionality conjecture

4.3.1 The problem

Of course the above are just particular results only applicable to specific indexing schemes. If one wants to validate the curse of dimensionality once and for all, here is a fascinating open problem.

CONJECTURE 4.4 (CF. [20]). *Let X be a dataset with n points in the Hamming cube $\{0,1\}^n$. Suppose $d = n^{o(1)}$ and $d = \omega(\log n)$. Then any data structure for exact nearest neighbour search in X, with $d^{O(1)}$ query time, must use $n^{\omega(1)}$ space.*

The data structure and algorithm are understood in the sense of the *cell probe model* of computation (cf. [31, 5]). It is a natural choice when one is interested in lower bounds on the performance of general indexing schemes.

4.3.2 Cell probe model

In the context of similarity search, the model can be described as follows. An abstract indexing structure for a domain Ω consists of

- a collection of functions f_t indexed with inner nodes of a rooted tree T,
- a collection of cells C_i, indexed with a set I, and
- a mapping $t \mapsto i(t)$ from T to I (not necessarily one-to-one).

Every function f_t is defined on a subset of Ω and besides takes a b-bit string σ as a parameter, except if $t = 0$ is the root. A value $f_t(\sigma; q)$ is a pair (τ, s), consisting of a b-bit string τ and a child s of the node t. If $i = i(t)$ corresponds to an inner node, the cell C_i can hold a b-bit string. If $i = i(t)$ where t is a leaf, then C_i can hold a datapoint $x \in X$. Often the problem is replaced with a weaker *decision version,* whereby a range parameter $\varepsilon_0 > 0$ is fixed and the algorithm is expected to tell whether there is an $x \in X$ at a distance $< \varepsilon_0$ from the query point. In such a case, C_i will hold a single bit (a "yes" or "no" answer).

Building the data structure at the preprocessing stage, given a dataset X, consists in storing in every inner node cell a b-bit string, and in every leaf node cell (a bin) a pointer to a datapoint $x \in X$ (or, in the decision version, a bit).

A memory image of the indexing structure $C_i, i \in I$ is created when the algorithm is initialized. Given a query point $q \in \Omega$, the tree is traversed down to the leaf level beginning with the root. At the inner node t, the content σ of the cell $C_{i(t)}$ is read and passed on to the function f_t as a parameter. The computed value $f_t(\sigma; q)$ consists of a string, to be written in the cell $C_i(t)$ instead of the current value, and of a child s of t to follow at the next step. When a leaf s is reached, the algorithm halts. The query time is the length of the branch traversed, or equivalently the number of cells probed during the execution of the algorithm.

The cell probe model is very liberal. Indeed, the cost of storing functions f_t is not even taken into account, the space is just the number of cells C_i. Similarly, the cost of computing the values of f_t is disregarded. For this reason, any lower bound obtained under the cell probe model will likely hold under any other model of computation. Since the nearest neighbour problem is stronger than the decision version ("near neighbour problem"), any lower bound for the decision version will serve as a lower bound for the conjecture.

As an example, the pivot table will be encoded within the cell probe model in a rather cumbersome way, because coordinates of each pivot p in the Hamming cube will occupy $\lceil d/b \rceil$ cells, and the distance function $\rho(p, -)$ will need to be computed recursively in $O(d/b)$ steps, with values of partial distances on substrings stored and then passed on further as parameters. For the same reason, our lower bound on the

query time of the pivot algorithm becomes $\Omega(n/\lg n)$.

4.3.3 *Current state of the problem*

The best lower bound currently known is $O(d/\log \frac{sd}{n})$, where s is the number of cells used by the data structure [36]. In particular, this implies the earlier bound $\Omega(d/\log n)$ for polynomial space data structures [3], as well as the bound $\Omega(d/\log d)$ for near linear space (namely $n \log^{O(1)} n$).

5. APPROXIMATE NN SEARCH AND DIMENSIONALITY REDUCTION

Approximate nearest neighbour search [35] is often said to be free from the curse of dimensionality, and the reason is that the (dimensionality) reduction maps f used in indexing are no longer 1-Lipschitz, but what may be called "probably approximately 1-Lipschitz". They no longer exhibit a strong concentration around their means. The price to pay is that we may lose some relevant datapoints, as some distances may get distorted. Curiously, the construction of reduction maps is often based on the concentration phenomemon and/or the VC theory.

5.1 Random projections

Let $\mathbb{S}^{d-1}$ denote the Euclidean sphere of unit radius sitting in the space $\mathbb{R}^d$. The projection π_1 on the first coordinate is a 1-Lipschitz function. For all pairs of points $x, y \in \mathbb{S}^{d-1}$, one has $|\pi_1(x) - \pi_1(y)| \le \|x - y\|$, and for antipodal pairs of points the equality is achieved. Now let $x, y \in \mathbb{S}^{d-1}$ be drawn at random. What is the expected value of the distortion $|\pi_1(x) - \pi_1(y)| / \|x - y\|$?

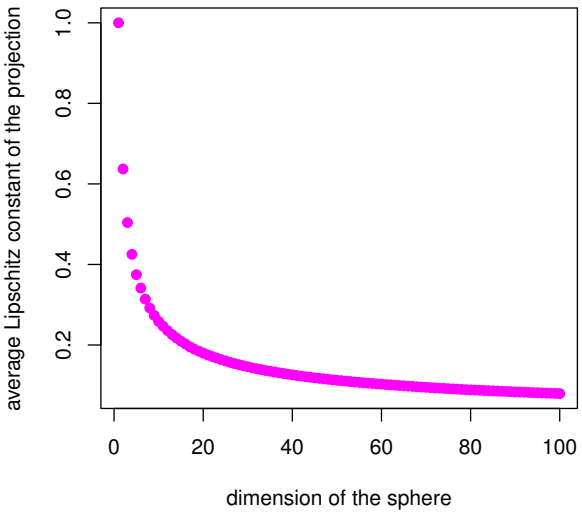


Figure 9: The expected distortion of one-dimensional projection of the d-dimensional sphere $\mathbb{S}^{d-1}$ over all pairs of points.

Figure 9 shows that for a vast majority of pairs of points, the projection distorts distances by the factor $O(1/\sqrt{d})$. A geometric explanation is very simple. A randomly chosen pair of points on the high-dimensional sphere, because to concentration, are at a distance $\approx \sqrt{2}$. At the same time, half of the points of the sphere project on the interval of length $O(1/\sqrt{d})$, and so are contained in the equatorial region (Figure 10).

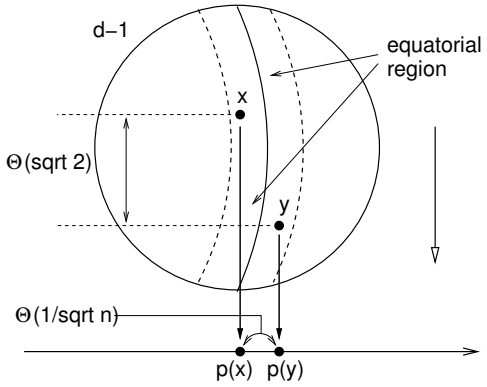


Figure 10: To the geometry of random projections.

If we define a new mapping

$$f(x) = C\sqrt{n}\pi(x),$$

for a suitably chosen constant C, it will be approximately 1-Lipschitz for a finite fraction of pairs of points. Now, in order to achieve a distortion in the range $1 \pm \varepsilon$ with high confidence, it is enough to combine $k = O(\log n/\varepsilon^2)$ mutually orthogonal projections as above, that is, to project on a randomly chosen k-dimensional subspace. This is the famous *Johnson–Lindenstrauss lemma* [22]. (See [27] for an up-to-date state of the lemma.) The projection is not quite as good as a genuine 1-Lipschitz map, because the distortion of a distance can exceed one, and occasionally very considerably. Yet, as a reduction mapping for *approximate NN search,* the projection map is quite OK. And its histogram is concentrated *no more.* This explains the efficiency of the random projection method for *approximate* NN search. Combined with an indexing scheme in a low-dimensional space $\mathbb{R}^k$, this dimensionality reduction leads to an indexing scheme for an $(1+\varepsilon)$-approximate NN search (Indyk and Motwani [21]).

5.2 Algorithm of Kushilevitz, Ostrovsky and Rabani

The indexing scheme constructed in [25] at the same time and independently from [21] is an application of VC theory.

Think of the Hamming cube $\{0,1\}^d$ as the set of all binary functions in the space $\ell^1(d) = L^1([d])$, where $[d] = \{1, 2, \ldots, d\}$ supports a uniform measure. If the dataset $X \subseteq \{0,1\}^d$ contains n points, then the VC dimension of X, viewed as a concept class on $\{1, 2, \ldots, d\}$, does not exceed $\lg_2 n$. According to the Glivenko–Cantelli theorem, if $O(\varepsilon^{-2} \lg_2 n)$ coordinates of the Hamming cube are chosen at random, then with high confidence the restriction mapping from X to the Hamming cube $\{0,1\}^{O(\varepsilon^{-2} \lg_2 n)}$ preserves the pairwise distances to within a factor of $1 \pm \varepsilon$. Cf. Figure 11.

A further analysis allows to conclude that the same will hold for the distances within $X \cup \{q\}$, for a *vast majority* of query points q in $\{0,1\}^d$. Since the new cube only contains $2^{O(\varepsilon^{-2} \lg_2 n)}$ points, a hash table storing nearest neighbourhs,

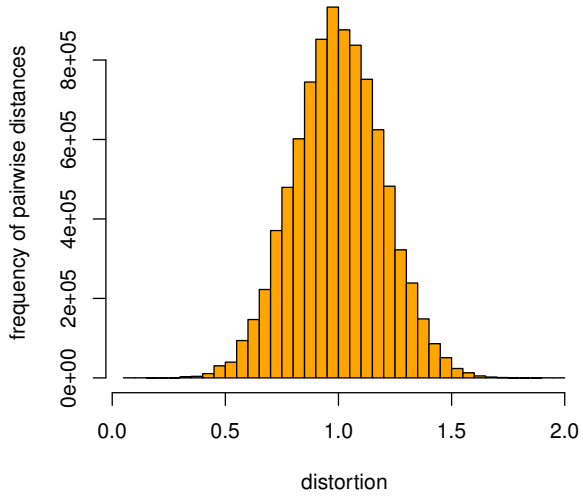


Figure 11: Histogram of distortions of all pairwise distances in a random dataset of $n = 3,000$ points in the $d = 500$ Hamming cube under a projection to a Hamming cube on randomly chosen $k = 25$ bits.

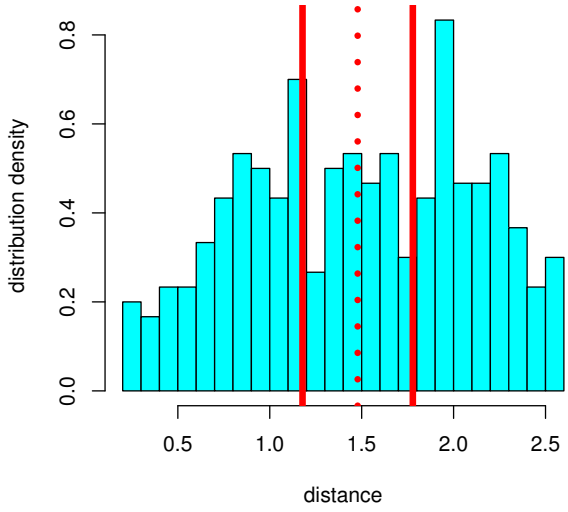


Figure 12: Empirical density histogram of distances from a pivot having the highest found value of dissipation for the NASA dataset. Vertical lines mark the mean $\pm$ tolerance range $\varepsilon = 0.275$. The ϵ-dissipation (0.747) is the area outside of extreme lines.

together with the reduction map f, produces an indexing scheme taking space polynomial in n and answering $(1+\varepsilon)$-approximate NN queries in time $O(\varepsilon^{-2} \lg_2 n)$.

This argument naturally extends to a framework where the metric ρ on the domain is an L^p-distance for some p in the range $1 \leq p < \infty$. In other words, there is a measure space (S, ν) with $\Omega \subseteq L^p(S, \nu)$.

The dimensionality reduction methods have been shown to be near optimal in the cell probe model [1].

6. CONCLUDING REMARKS

6.1 Intrinsic dimensionality

Merits of asymptotic analysis of indexing algorithms using artificial datasets sampled from theoretical high-dimensional distributions should be clear from [33]. At the same time, it is an often held belief that the real data does not have very high intrinsic dimension. This corresponds to the existence of 1-Lipschitz functions that are highly dissipating. Figure 12 shows the distance distribution to the points of the SISAP benchmark dataset of NASA images $X \subseteq \ell^2(20)$ of $40,149$ vectors in a 20-dimensional Euclidean space [6, 44] from a highly dissipating pivot, selected from a gaussian cloud around X with standard deviation on the order of the tolerance range $\varepsilon = 0.275$ retrieving on average 0.1% of data. This has to be compared to Figure 3.

Of a great variety of approaches to intrinsic dimension [13], at least two specifically measure the amount of concentration in data. The first one is the intrinsic dimension by Chávez *et al.* [9]

$$\dim_{dist}(X) = \frac{1}{2\mathrm{var}\,(d)}. \tag{6}$$

The second is the concentration dimension, studied within an axiomatic approach of [38, 39]:

$$\dim_{\alpha}(X) = \frac{1}{\left[2\int_0^1 \alpha_X(\varepsilon)\; d\varepsilon\right]^2}. \tag{7}$$

(In both cases we assume that $\mathrm{CharSize}(X) = 1$.) The value (7) is convenient for asymptotic analysis in the spirit of this paper, but is nearly impossible to estimate for a given dataset. On the other hand, (6) is readily calculated by sampling (e.g. $\dim_{dist}(X) = 5.18$ for NASA images) and forms a good statistical estimator for the dimension of the hypothetical underlying measure μ in the most (only?) interesting case where metric balls have low VC dimension. The shortcoming of (6) is that the parameter estimates the concentration/dissipation behaviour of a *typical* pivot distance function, while it is a few most dissipating pivots that really matter for indexing. One may envisage the emergence of further concepts of intrinsic dimension in the same spirit. For instance, the *local dimension* of Ollivier [34], Definition 3, deserves a close look.

6.2 Black box search model and Urysohn space

The *black box model* of similarity search was studied by Krauthgamer and Lee [24]. Given a finite metric space (X, d), a query is a one-point metric space extension $X \cup \{q\}$, where every distance $d(q, x)$, $x \in X$ is accessible via the distance oracle and can be evaluated in constant (unit) time. A preprocessing phase is allowed, and the indexing scheme occupies poly (n) space. The query time is a number of calls to the distance oracle.

Recall that the *Assouad* (or *doubling*) *dimension* of a met-

ric space (X, d) is the minimum value $\rho \geq 0$ such that every set A in X can be covered by 2^ρ balls of half the diameter of A. Denote this parameter by $\dim_{dbl}(X)$. The above authors have shown that a metric space (X, d) admits an algorithm requirying poly (n) space and taking polylog (n) time to answer a $(1 + \varepsilon)$-approximate nearest neighbour query, where $\varepsilon < 2/5$, if and only if

$$\dim_{dbl}(X, d) = O(\log \log n).$$

The *Urysohn metric space,* $\mathbb{U}$ [28], is a complete separable metric space uniquely defined by the following *one-point extension property:* suppose X is a finite subset of $\mathbb{U}$ and q a one-point metric space extension of X. Then $\mathbb{U}$ contains a point q' so that the distances from q and from q' to any point $x \in X$ are the same. The black-box model can be restated as a classical NN search problem with $\Omega = \mathbb{U}$ as the domain. A simple information-theoretic argument shows that any deterministic algorithm for exact similarity search in a finite metric space X within this model will take the worst case time $n = |X|$. However, if one requires the queries to follow the same underlying distribution as datapoints, the problem becomes more subtle, and we do not know the answer.

6.3 Indexing via Delaunay graph

By far not every indexing scheme for exact similarity search is "distance-based." Here is an example. The *Voronoi cell* $V(x)$ of a datapoint $x \in X$ in a domain Ω consists of all points $q \in \Omega$ having x as the nearest neighbour. The *Delaunay graph* has X as the set of vertices, with x, y being adjacent if their Voronoi cells intersect. Suppose the domain has the property that every two points x, y can be joined by a shortest geodesic path, not necessarily unique. (All the domains previously considered are such.) Then for any $q \in \Omega$ and $x \in X$, either x is the nearest neighbour to q, or else one of the datapoints y adjacent to x is closer to q than x is. (Proof: start moving along a shortest geodesic from x towards q.) This makes the Delaunay graph of X into an indexing scheme for exact nearest neighbour search. Denote S_x the list of points adjacent to each $x \in X$. Given a query q, start with an arbitrary $x_0 \in X$, and find

$$x_1 = \arg\min_{y \in S_x} d(q, y).$$

If $x_1 \neq x_0$, move to x_1 and repeat the procedure. Once $x_{i+1} = x_i$, the algorithm halts and returns x_i. This algorithm, already mentioned in [12], was studied for general metric spaces by Navarro [32]. See also [42], 4.1.6.

In order to be efficient, the average degree of the Delaunay graph has to be small. Navarro had observed (*loc. cit.,* Theorem 1) that this is not the case in general metric spaces. In fact, one can deduce from Corollary 2.2 that if Ω be either $\mathbb{R}^n$, or the sphere $\mathbb{S}^n$, or the Hamming cube, then under the assumptions of Subs. 2.2 the Delaunay graph of X is, with high probability, the complete graph on n vertices.

Thus, the indexing scheme in question suffers from the curse of dimensionality because of concentration of measure considerations, but the argument seems to be of a different nature from that either for pivots or for trees. What would a common proof for all three types of schemes look like? This highlights the difficulty of proving in a uniform way lower bounds in a general setting of the cell probe model for all possible indexing schemes at once.

To finish on an optimistic note, for real data the complexity of the Delaunay graph is lower than in an artificial asymptotic setting, and Voronoi diagrams are being successfully used for data mining algorithms in high dimensions, cf. [46]. It further seems the above algorithm could be especially efficient in hyperbolic metric spaces, and Alain Connes had suggested [7], pp. 138–141, that a long-term human memory is organized as a hyperbolic simplicial complex, where a search is performed in a manner similar to the above.

7. REFERENCES

[1] A. Andoni, P. Indyk, M. Pătrascu, On the optimality of the dimensionality reduction method, in: Proc. 47th IEEE Symp. on Foundations of Computer Science, pp. 449–458, 2006.

[2] M. Anthony and P. Bartlett, *Neural Network Learning: Theoretical Foundations.* Cambridge University Press, Cambridge, 1999.

[3] O. Barkol and Y. Rabani. Tighter lower bounds for nearest neighbor search and related problems in the cell probe model. In: *Proc. 32nd ACM Symp. on the Theory of Computing,* 2000, pp. 388–396.

[4] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, When is "nearest neighbor" meaningful?, in: *Proc. 7-th Intern. Conf. on Database Theory (ICDT-99),* Jerusalem, pp. 217–235, 1999.

[5] A. Borodin, R. Ostrovsky, and Y. Rabani, Lower bounds for high-dimensional nearest neighbor search and related problems, in: *Proc. 31st Annual ACS Sympos. Theory Comput.*, pp. 312–321, 1999.

[6] B. Bustos, G. Navarro, and E. Chávez, Pivot selection techniques for proximity searching in metric spaces. *Pattern Recogn. Lett.*, 24:2357–2366, 2003.

[7] J.-P. Changeux and A. Connes, *Conversations on Mind, Matter, and Mathematics,* Princeton University Press, Dec. 1998.

[8] E. Chávez, J.L. Marroquín, and R.A. Baeza-Yates, Spaghettis: An Array Based Algorithm for Similarity Queries in Metric Spaces, in: *Proceedings SPIRE/CRIWG,* pp. 38–46, 1999.

[9] E. Chávez, G. Navarro, R. Baeza-Yates, and J.L. Marroquín, Searching in metric spaces, *ACM Computing Surveys,* 33:273–321, 2001.

[10] P. Ciaccia, M. Patella and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB'97), (Athens, Greece)*, 426–435, 1997.

[11] P. Ciaccia, M. Patella and P. Zezula. A cost model for similarity queries in metric spaces, in: *Proc. 17-th ACM Symposium on Principles of Database Systems* (PODS'98), Seattle, WA, 59–68, 1998.

[12] K.L. Clarkson. An algorithm for approximate closest-point queries. In: *Proc. 10th symp. Comp. Geom.* Stony Brook, NY, 160–164, 1994.

[13] K.L. Clarkson. Nearest-neighbor searching and metric space dimensions. In: Nearest-Neighbor Methods for Learning and Vision: Theory and Practice, MIT Press, 2006, pp. 15–59.

[14] L. Devroye, L. Györfi, and G. Lugosi. A Probabilistic Theory of Pattern Recognition. Springer-Verlag, New York, 1996.

[15] A. Faragó, T. Linder, and G. Lugosi, Fast nearest neighbor search in dissimilarity spaces, *IEEE*

Transactions on Pattern Analysis and Machine Intelligence, 18:957–962, 1993.

[16] P. Frankls and Z. Füredi. A short proof for a theorem of Harper about Hamming-spheres, *Discrete Math.* 34:311–313, 1981.

[17] P.W. Goldberg and M.R. Jerrum. Bounding the Vapnik–Chervonenkis dimension of concept classes parametrized by real numbers. *Machine Learning* 18:131-148, 1995.

[18] M. Gromov. *Metric Structures for Riemannian and Non-Riemannian Spaces.* Progress in Mathematics **152**. Birkhauser Verlag, 1999.

[19] P. Hall, J.S. Marron, and A. Neeman. Geometric representation of high dimension, low sample size data. *J. R. Stat. Soc. Ser. B Stat. Methodol.* 67:427–444, 2005.

[20] P. Indyk. Nearest neighbours in high-dimensional spaces, In: J.E. Goodman, J. O'Rourke, Eds., *Handbook of Discrete and Computational Geometry*, Chapman and Hall/CRC, Boca Raton–London–New York–Washington, D.C. 877–892, 2004.

[21] P. Indyk and R. Motwani. Approximate nearest neighbours: towards removing the curse of dimensionality. Proc. 30th ACM Symp. Theory of Computing, 604–613, 1998.

[22] W.B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemp. Math.* 26:189–206, 1984.

[23] B. Klartag. A central limit theorem for convex sets. *Invent. Math.* 168:91–131, 2007.

[24] R. Krauthgamer and J.R. Lee, *The black-box complexity of nearest-neighbor search,* Theoretical Computer Science 348(2):262 - 276, December 2005.

[25] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces. *SIAM Journal on Computing* 30:457–474, 2000.

[26] M. Ledoux. *The concentration of measure phenomenon*. Math. Surveys and Monographs **89**, Amer. Math. Soc., 2001.

[27] J. Matoušek. On variants of the Johnson–Lindensrauss lemma. *Random Structures and Algorithms* 33:142–156, 2008.

[28] J. Melleray. On the geometry of Urysohn's universal metric space. *Topology and its Applications* 154:384-403, 2007.

[29] V. Milman. Topics in asymptotic geometric analysis. In: *Geometric and Functional Analysis,* special volume GAFA2000, pp. 792–815, 2000.

[30] V.D. Milman and G. Schechtman. *Asymptotic theory of finite-dimensional normed spaces (with an Appendix by M. Gromov)*. Lecture Notes in Math., **1200**, Springer, 1986.

[31] P.B. Miltersen. Cell probe complexity - a survey. In: *19th Conf. on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS),* 1999. Advances in Data Structures Workshop.

[32] G. Navarro. Searching in metric spaces by spatial approximation. *The VLDB Journal* 11:28–46, 2002.

[33] G. Navarro. Analysing metric space indexes: what for? Invited paper, in: *Proc. 2nd Int. Workshop on Similarity Search and Applications (SISAP 2009)*, Prague, Czech Republic, 2009, 3–10.

[34] Yann Ollivier. Ricci curvature of metric spaces. *C.R. Math. Acad. Sci. Paris* 345:643–646, 2007.

[35] M. Patella and P. Ciaccia. The many facets of approximate similarity search. Invited paper, in: *Proc. First Int. Workshop on Similarity Search and Applications (SISAP 2008),* Cancun, México, pp. 10–21, 2008.

[36] M. Pătrascu and M. Thorup. Higher lower bounds for near-neighbor and further rich problems. in: *Proc. 47th IEEE Symp. on Foundations of Computer Science*, pp. 646–654, 2006.

[37] V. Pestov, On the geometry of similarity search: dimensionality curse and concentration of measure. *Inform. Process. Lett.*, 73:47–51, 2000.

[38] V. Pestov. Intrinsic dimension of a dataset: what properties does one expect? in: *Proc. of the 22-nd Int. Joint Conf. on Neural Networks (IJCNN'07), Orlando, FL.*, pp. 1775–1780, 2007.

[39] V. Pestov. An axiomatic approach to intrinsic dimension of a dataset. *Neural Networks* 21:204-213, 2008.

[40] V. Pestov. *Lower bounds on performance of metric tree indexing schemes for exact similarity search in high dimensions.* preprint, arXiv:0812.0146 [cs.DS].

[41] V. Pestov and A. Stojmirović. Indexing schemes for similarity search: an illustrated paradigm, *Fund. Inform.*, 70:367–385, 2006.

[42] H. Samet. *Foundations of Multidimensional and Metric Data Structures.* Morgan Kaufmann Publishers Inc., San Francisco, CA, 2005.

[43] U. Shaft and R. Ramakrishnan, Theory of nearest neighbors indexability. *ACM Trans. Database Syst. (TODS),* vol. 31, pp. 814–838, 2006.

[44] SISAP metric space library, `http://sisap.org/Metric_Space_Library.html`

[45] A. Stojmirović and V. Pestov, Indexing schemes for similarity search in datasets of short protein fragments, *Information Systems* 32:1145-1165, 2007.

[46] K. Taşdemir and E. Merényi. Exploiting the data topology in visualizing and clustering of self-organizing maps. *IEEE Trans. Neural Networks* 20(4):549–562, 2009.

[47] V.N. Vapnik, *Statistical Learning Theory.* John Wiley & Sons, Inc., New York, 1998.

[48] E. Vidal, An algorithm for finding nearest neighbors in (approximately) constant average time, *Pattern Recognition Letters* 4:145–157, 1986.

[49] M. Vidyasagar, *Learning and Generalization, With Applications to Neural Networks.* Second Ed., Springer-Verlag, London, 2003.

[50] I. Volnyansky and V. Pestov. Curse of dimensionality in pivot-based indexes. In: *Proc. 2nd Int. Workshop on Similarity Search and Applications (SISAP 2009),* Prague, Czech Republic, 2009, pp. 39-46.

[51] P. Zezula, G. Amato, V. Dohnal and M. Batko, *Similarity Search. The Metric Space Approach.* Springer Science + Business Media, New York, 2006.

Where are you heading, metric access methods? A provocative survey.

Tomáš Skopal
SIRET research group
Charles University in Prague, FMP, Department of Software Engineering,
Malostranské nám. 25, 118 00 Prague, Czech Republic
http://siret.ms.mff.cuni.cz

ABSTRACT

In this paper the impact of the metric indexing paradigm on the real-world applications is discussed. We pose questions whether the priorities in research of metric access methods (MAMs) established in the past decades reflect the actual needs of practitioners. In particular, we formulate the following pragmatic questions: Are the established MAM cost measures relevant? Isn't the metric space model too general when the majority of real-world applications use L_p spaces? On the other hand, isn't the metric model too restrictive with respect to the growing community of practitioners using non-metric distances? Are the simple similarity queries competitive enough? Have the real-world similarity search engines ever used a general metric access method, or do they use specific indexing? Is there a real demand for content-based similarity search or will the annotations and keyword search win the game? We present justification of these questions, investigating relevant literature and search engines. Finally, we try to transform the questions into answers and suggestions to the future research on MAMs.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Retrieval models*

General Terms

algorithms, performance, design

Keywords

metric access methods, MAM, similarity, content-based search

1. INTRODUCTION

The content-based search in multimedia and other unstructured data becomes steadily more important nowadays, while the similarity search concept provides a general and intuitive model. Given a database of descriptors (i.e., features extracted from the original multimedia objects), a query descriptor is submitted such that objects similar to the query are returned as an answer. Hence, such content-based retrieval model separates the actual retrieval algorithm from the semantic model (the similarity of content). In a narrow sense, the pair-wise similarity function is often required to fulfill the metric postulates which are the basic properties that allow to index the database for efficient (fast) query processing. Here we talk about a class of database methods called *metric access methods* (MAMs), or *metric indexes*.

In this paper, we pragmatically analyze both, the experimental practices in the MAM research, and also the potential "market" for metric indexing – applications in content-based multimedia retrieval. Unlike other survey papers, we do not give a systematic overview of the achievements in the research area, but we try to critically discuss the weak points of MAM research. The aim of the constructive critique is to prevent the research of MAM from isolating into a bubble of theoretical and artificial achievements, and to motivate the research to a constant and tight connection with the real-world applications.

1.1 Questions

In particular, we ask the following six provocative questions, more or less rhetorical, but all aimed at impact of the MAM research on practical applications:

Q1: Isn't the metric space model too general?
Q2: Are the established MAM cost measures relevant?
Q3: Is there a real demand for general metric indexing?
Q4: Are the simple similarity queries competitive enough?
Q5: Have the real-world search engines ever used a MAM?
Q6: Isn't the metric model too restrictive?

The set of questions was not assembled by an arbitrary thought process of the author, but it came out as an impression of the analysis we will present in the following text. In addition to identifying the questions in Sections 2 and 3, in Section 4 we discuss possible answers and future solutions.

1.2 Metric access methods

Before we start the survey, we need to remember the very basic mission of the metric access methods.

$$MAM =$$
Set of algorithms and data structure(s) providing efficient (fast) similarity search under the metric space model.

SISAP 2010, September 18-19, 2010, Istanbul, Turkey.

The metric space model itself is determined by its mathematical foundations. The database $\mathbb{S} \subset \mathbb{U}$ to be searched is considered as a set of unstructured (black-box) descriptors, so that only a distance function $\delta(x, y)$ is defined between any two descriptors x, y from the descriptors universe $\mathbb{U}$. The distance is required to be a metric distance, i.e., δ is non-negative, identical ($\delta(x, y) = 0 \Leftrightarrow x = y$), symmetric ($\delta(x, y) = \delta(y, x)$) and triangular ($\delta(x, y)+\delta(y, z) \geq \delta(x, z)$).

There were many metric access methods developed so far (see the literature referenced in the following section), addressing various data management aspects, namely: main vs. secondary memory index, static vs. dynamic database, exact vs. approximate search, continuous vs. discrete metric distances, centralized/serial vs. distributed/parallel implementation, etc.

2. EXPERIMENTAL PRACTICES IN MAM RESEARCH

In this section we present the results of analysis within the 40 years old database-oriented research on similarity search. In particular, we have analyzed papers cited in the major "bibles" for the MAM community – the classic survey [6], the classic monographs [26, 20] and a recent book chapter [13]. Among the hundreds of papers referenced in the mentioned sources, we selected 77 for our research, that propose *general* metric access methods and prove their contribution in an *experimental evaluation*. Hence, we did not consider distance-specific indexing methods, and also theoretical papers. In addition to the 77 papers, we analyzed also 18 relevant papers from the SISAP 2008 and 2009 conferences, giving us the total number of 95 analyzed papers.

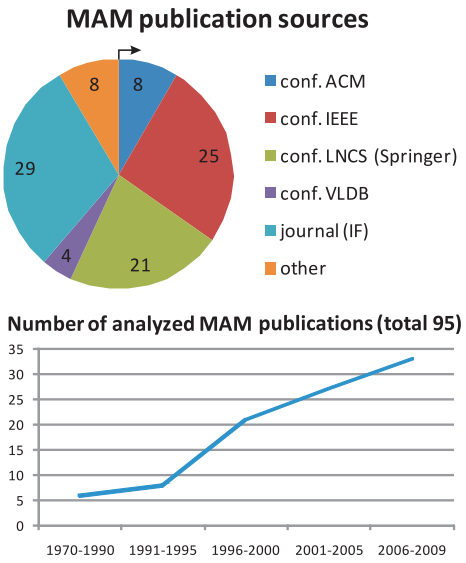


Figure 1: Analyzed experimental papers on MAMs.

The structure of the papers and their distribution in time is shown in Figure 1. We can observe that the majority of papers was published in renowned journals and in proceedings of international conferences, while the majority of the contributions was published in the past ten years. It should be also mentioned that 49.5% of the papers were co-authored by somebody from the SISAP program committee (12 people through the years 2008-2010). However, this fact does not support a possible biased paper selection, it simply points out that SISAP conference gathers a significant proportion of people interested in MAM research.

2.1 Distances & Databases

In the first part of the MAM papers analysis, we focused on the metric space instances and testbeds used in the papers' experiments. In particular, we aggregated the information about the type of space, the size of databases used, and especially the distance metric used.

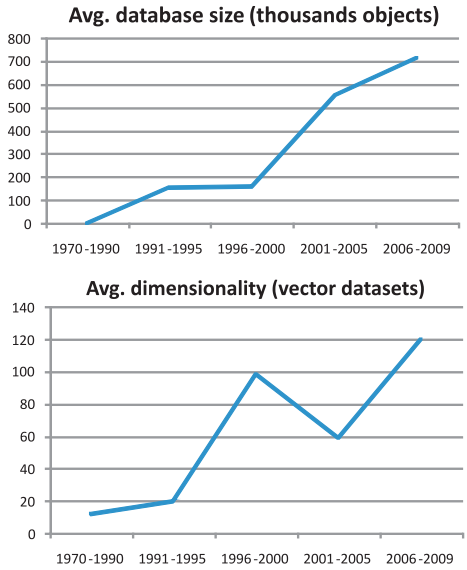


Figure 2: Databases used through the years.

In Figure 2, see the evolution of database size and dimension (when vector space used) in time. In 1970-1990 we can hardly speak about "databases", as the numbers of objects were around 1000. In the past ten years, however, the database sizes got to volumes of almost a million objects per database (on average). In such volumes the sequential search already becomes a bottleneck in the retrieval process, so the indexing efforts pay off. In case of vector spaces (equipped by a metric distance), the dimension grows from less than ten to more than one hundred (on average). This growth not only increases the volumes of databases, but also indicates growing complexity of descriptors (high-dimensional vectors).

The most interesting result of this subsection is shown in Figure 3, where the usage of different types of metric spaces is summarized. As expected, the majority of papers evaluate their contribution on vector spaces, which mostly means the Euclidean space (+ several L_∞, L_1 spaces), followed by a few others, like Hamming or angle space. Since low-level descriptors represented as vectors of independent dimensions are very popular over many domains, the employment of Euclidean distance seems natural. The second most frequent is the string space under the edit distance. The instances of string databases used are, however, far less multifarious than the Euclidean ones, as they mostly include English and Spanish vocabularies (+ several others, like biological sequences). Other types of metric spaces are quite rare, including also several non-vectorial databases (sets of elements, time series, geometries). The expensive metric

distances (i.e., of complexity $\geq O(n^2)$) mostly reduce to the mentioned edit distance, followed by several others, like Hausdorff distance, quadratic form distance, or variations on the edit distance (sequence/string alignments solved by dynamic programming)[1].

An alarming fact, that denies the common assumption that metric distances are *expensive*, is shown on the last line of the figure, saying that almost 50% papers use only cheap distances ($O(n)$) in their experiments!

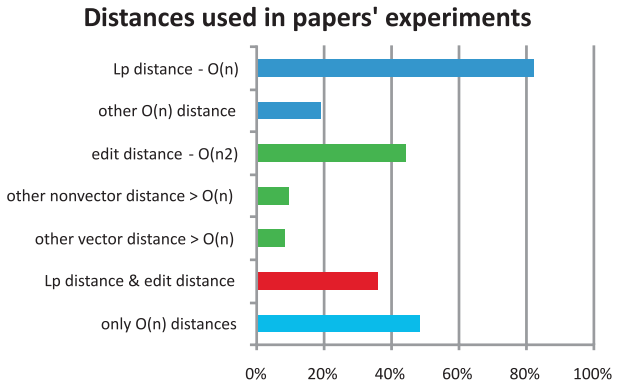


Figure 3: Distance spaces used in experiments.

Had we interpret the observations resulting from Figures 2 and 3, we could speculate about two alternative possibilities:

- The experimental part of the papers (on *general* MAM) is considered as a toy problem, while the main focus is given to the description of the method itself, rather than proving its properties on *general* metric spaces.
- The experiments in the papers are correct and reflect the actual application needs. However, that would mean there is actually only little demand for general MAM, while the attention should be given to more specific access methods, or simply indexes for L_2 or generally (combinations of) L_p distances.

Either of the two speculations indicates a problem. In the first case, the MAM research does not stick to real-world experiments in general spaces, focusing on the narrow L_2 space or edit distance stereotype. The second case is worse, as it leads to the first question of this paper, *Q1: Isn't the metric space model too general?* In other words, isn't the universality of the metric model just a technical simplification for indexing much more specific spaces, e.g., L_p?

2.2 Cost measures

In the second part of the MAM papers' analysis, we discuss the methodologies of measuring a MAM's performance (efficiency) in terms of cost types used in experiments. The overall picture is shown in Figure 5, showing the proportion of each cost type utilized in the papers' experiments. Before we analyze the consequences of Figure 5 in Section 2.2.5, we discuss all of the relevant cost types in the following sections.

2.2.1 Distance computations

From the very dawn of MAM research, the number of distance computations spent during indexing/querying (*DC cost*) has established as the most respected and frequently used performance cost. To justify the employment of DC cost, it is generally assumed that evaluation of single distance $\delta(\cdot,\cdot)$ is *computationally expensive*, so that other types of cost become marginal. The advantage of DC cost is its independence on code optimization, programming language, software and hardware platform, thus allowing to separate the essence of the proposed MAM's efficiency from the irrelevant runtime factors.

However, the assumption on expensive distance is critical, while experiments using the DC cost alone (without showing also other types of cost) are only appropriate when:

1. an expensive distance is used (i.e., having time complexity $\geq O(n^2)$ and/or large n),
2. rather small database is used (e.g., fits main memory),
3. contribution of other cost type to realtime is negligible, e.g., internal time/space cost, I/Os, networking, synchronization of parallel/distributed processing, etc.

Hence, the DC cost is a performance measure useful to analyze the qualitative behavior of the MAM (including tuning of internal parameters or comparing MAMs), rather than to be presented as the objective cost to the "end-users".

2.2.2 I/O cost

The *I/O cost*, mostly interpreted as the number of random accesses to disk pages (reads/writes), has been established in the pioneer times of database research when the hard disk drives (HDDs) were the biggest bottleneck of the data management. In particular, the research field of *spatial access methods* [4] – the closest relative to MAM research – still widely uses I/Os as the major cost type. Although the HDD technology has improved tremendously over the years, the seek time component in an I/O operation has not changed much, so the I/O cost is still relevant for methods requiring random access to disk.

However, in the context of MAM research, the I/O cost has to be used carefully. In particular, experiments using the I/O cost alone are only appropriate when:

1. I/O time dominates the other types of cost,
2. the competing MAMs share the same I/O model

The latter condition is especially important, because improper usage of I/O cost could be totally misleading. In particular, consider sequential search (as trivial MAM) and a hierarchical MAM, like the M-tree [8]. The sequential search can be easily optimized such that only single seek operation is needed to process the whole file. On the other hand, such an optimization cannot be implemented for the M-tree (without heavy disk prefetching leading to sequential search), since metric data in hierarchical indexes cannot be (natively) linearly ordered, and so random access I/Os are necessary for them.

In the following (silly) example we illustrate the danger of incorrect I/O cost usage. Let us have two 100 MB indexes (sequential file and M-tree), 4 kB disk page file system (i.e., 25,600 pages per index), seek time 8 ms and read time 50 MB/s (i.e., today low-cost HDD). Let us also suppose that an M-tree query needs to access just 1% of pages, while the sequential file needs to access, of course, 100% pages.

[1]The variable n is the size of a descriptor.

The realtime needed for M-tree is 256 I/Os, that is $0.008 \times 256 + 0.1 = 2.148$ seconds. An improper (random access) implementation of sequential search would take $204.8 + 2 = 206.8$ seconds, however, optimized (one seek) variant would take only 2.008 seconds![2]

Anyways, as the HDD technology will be soon (hopefully) defeated by the SSD technology that removes the seek time overhead, we can expect a renaissance of random access methods (and better performance of hierarchical MAMs).

2.2.3 *Internal cost*

Apart from distance computations (and I/Os), i.e., cost measures universally applicable to every MAM (managing secondary-memory index), the time/space requirements of a particular MAM could be also measured by a specific *internal* cost measure. The internal cost is not very frequently presented in the experiments, however, it could be crucial to the overall MAM's efficiency, especially when the DC cost is not dominant (i.e., cheap metric is used). To illustrate the impact of internal cost, let us discuss two examples:

- The methods based on Pivot tables, e.g., the classic LAESA [18], use a matrix consisting of distances from the database objects to a set of pivots. When a query is evaluated, the distance matrix is sequentially processed (either entire using single pass, or just a part but using multiple passes), which constitutes a significant internal overhead. For instance, consider 128 dimensional vector space under L_2 distance and 128 pivots. Then the one-pass LAESA query processing of distance matrix under L_∞ distance is equivalent to the sequential search in the original L_2 space. Here the DC cost, dramatically higher for the sequential search, is not appropriate due to the internal cost of LAESA.

- The incremental kNN algorithm [15] by Hjaltason and Samet can be implemented in any MAM. Although the algorithm was proved as optimal in terms of DC cost (i.e., equivalent to range query with radius equal to the distance to the kth neighbor), it suffers from high internal cost. Specifically, the algorithm utilizes a heap that contains the set of not-yet-processed regions of a MAM's index. Depending on the intrinsic dimensionality of the space [5] and the MAM used, the heap is usually largely inflated until the first neighbor is found, followed by a rapid heap reduction.

2.2.4 *Realtime cost*

Finally, we get to the very objective type of computation cost – the "dirty" *realtime cost* (or wall-clock time), measured in seconds or processor cycles spent in the MAM's process. The realtime cost is not very popular in analyzing the MAMs' efficiency because it is hardware-, platform-, language- and compiler-dependent, it requires proper optimizations of the code, etc. It mixes many different costs into a single aggregate, making hard to recognize the underlying causes of a MAM's (in)efficiency.

On the other hand, only the realtime cost is the moment of truth for an end user that wants to be oriented in the jungle of various MAMs. The realtime cost means "no cheating is allowed" – a MAM is either fast or slow (given a particular database context). To demonstrate a possible discrepancy between the DC cost and realtime, see an experiment in Figure 4, where a database of peptides (pieces of proteins) was indexed as 32-dimensional vectors under a linear variant of the Hausdorff distance (intrinsic dim. ≈ 3). The left-hand figure shows an expected superior performance of Pivot tables (LAESA) in terms of DC cost, being on 5% of sequential search. However, in terms of realtime (right-hand figure) the M-tree and even the sequential search run faster for databases over 1.5 million objects (2.5, resp.), due to the internal cost of Pivot tables and the cheap distance.

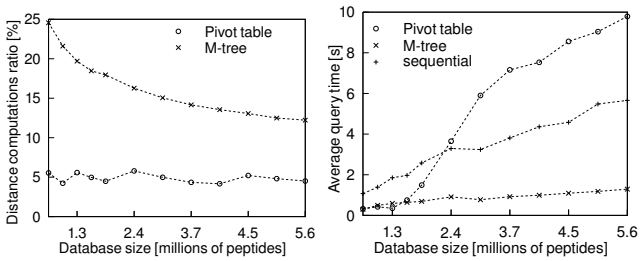


Figure 4: Example: DC cost vs. realtime

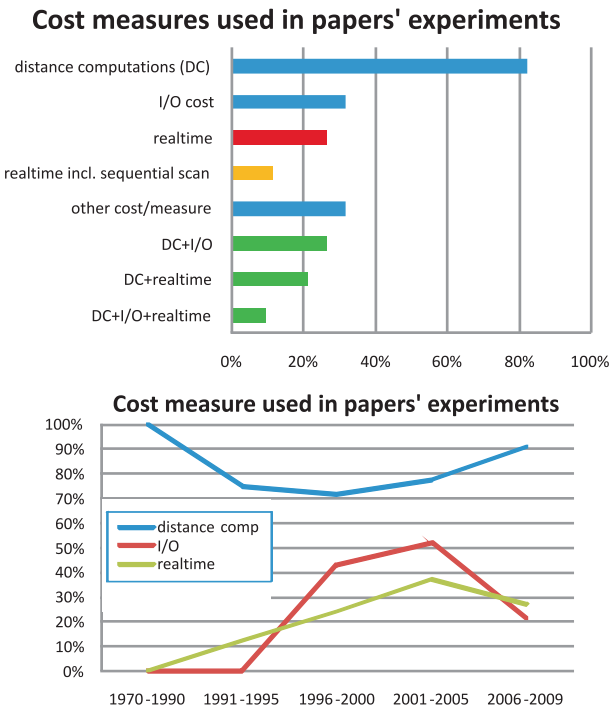


Figure 5: Cost measures used in experiments.

2.2.5 *Discussion*

In Figure 5 see the structure of cost measures used in the examined papers' experiments. Among other, it shows the DC cost was the most popular (in more than 80% papers), the realtime cost was used in 25% of papers and all costs (DC+I/O+realtime) were used in 10% papers. Some 12% papers provided a realtime comparison with the sequential search, which is seemingly an unimportant detail, but it prevents from commending an expensive MAM over even more

[2]Note that here we consider just the I/O cost contribution to realtime, while the overall realtime could be significantly different (due to possibly expensive distance computations).

expensive competitors as "really fast".

In summary, the structure of cost measures used in the papers is quite rich, however, it is not very convincing for the practitioners that need a MAM for their application. This is documented not only by the relatively small number of papers using realtime, but also by the alarming observation that 21% papers used only DC cost and, at the same time, only cheap ($O(n)$) distances in their experiments! Hence, we end up the discussion with the second question of this paper, *Q2: Are the established MAM cost measures relevant?*

3. APPLICATIONS

In this section, we leave the area of "intra-MAM" research, and move to applications that are expected to profit from metric indexing (in the future). In particular, we investigate trends in content-based image retrieval, the existing multimedia search engines, and the recent attempts to employ similarity measures more general than metric distances.

3.1 Content-based image retrieval

As a representative summary of the advances in content-based image retrieval (CBIR), we chose the respected survey by Datta et al. [10], referencing almost 300 papers related to CBIR. Apart from presenting the recent feature extraction techniques, belonging more to Computer vision than to the area of Image retrieval, we summarize several observations interesting for applications of MAMs in CBIR.

3.1.1 Modeling vs. indexing in CBIR

The situation in CBIR could be uncovered slightly by two citations from the survey: *"... we do not have yet a universally acceptable visual model for content-based search..."*, and *"... the indexing techniques were largely overshadowed by research on similarity modeling ..."*

The good news for MAM applications in CBIR is the uncompromising interpretation of the content-based retrieval as a similarity search based on image features. Hence, there is a clear distinction between the model (similarity function) and retrieval algorithm (an access method). The bad news for MAMs is the indexing for efficient search was either not solved at all (implicit sequential search), or the semantic model itself was prepared to reuse a well-known indexing technology. Following the latter way, a popular concept is an automatic annotation of the image content (including even more popular segmentation) by tags/keywords. The images described by tags are subsequently indexed using the well-known boolean or vector model of information retrieval [2], leading to inverted files at the implementation level. At this moment we come out with the third question of this paper, *Q3: Is there a real demand for general metric indexing?*

Nowadays, MAMs suffer from many limitations that prevent their straightforward (naïve) utilization in CBIR. Although the metric space model is quite general, from the complex CBIR point of view the metric indexing brings many obstacles. In particular, MAMs rarely allow modifications to the metric space during their indexes' lifetime, making hard to learn/tune the similarity measure, to rearrange the structure of descriptors, or to include user preferences. Furthermore, as metric distances must fulfill the triangle inequality, they are limited in measuring *local* similarity that usually leads to nonmetric behavior (discussed in Section 3.3). Nevertheless, even if MAM will not become the core of image retrieval techniques, its role in CBIR could be substantial (as discussed later in Sections 3.1.3 and 4.2).

3.1.2 Similarity measures

When modeling the distance space in CBIR, the complexity is more propagated into the descriptor semantics (see also the later discussion in Section 4.1), rather than into the distance measure. In turn, the most popular distances measuring similarity in CBIR are the usual Euclidean or L_1 distance, statistical (non)metric distances (e.g., Kullback-Leibler divergence), while some approaches use more expensive distances measuring histogram similarity, like quadratic form distance or earth mover's distance. As in the previous section, also these observations do not indicate inevitable benefits of general metric indexing.

Fortunately, when taking another citation from the survey, *"...the richness in the mathematical formulation of signatures (descriptors) grows alongside the invention of new methods for measuring similarity ..."*, the need for more sophisticated similarity measuring could lead to more expensive distance measures that would require general (metric) indexing.

3.1.3 Retrieval models

Had we classify the retrieval models used in CBIR, we could distinguish three design levels:

- **Pseudo-CBIR** – proprietary add-ons of text-based image search engines (surveyed in Section 3.2). The usual search of images based on keywords extracted from the surrounding web page is augmented by a limited CBIR functionality. For example, the images are additionally labeled by tags representing certain extracted features, like "contains face", "is illustration", "mostly red color", etc. At query time, the user can select some of the tags as an additional filter to the keyword query. There is no room for MAMs at all.

- **Single-model similarity search** – a true content-based search, where the retrieval procedure is based on similarity search using single-descriptor representation and single distance. Although this model would position a "simple" MAM into the prominent role of the core technology inside a CBIR system, it is not very likely to happen due to the limitations mentioned in Section 3.1.1. On the other hand, MAMs could be utilized for particular tasks, as suggested in Section 4.2.

- **Hybrid-model similarity search** – a true content-based search, where the complex retrieval procedure is split into a hierarchy of simple similarity searches. In particular, an image is represented by multiple local subdescriptors (e.g., image segments), where each subdescriptor could be modeled in its own distance space. A query image is modeled the same, so that multiple similarity searches are performed for a single query. The obtained intermediate results (ordered lists of subdescriptors) are finally ranked by an aggregating function, e.g., the top-k operator [12] or reranking [16] based on user preferences/feedback, etc. The MAMs could be utilized in the separate local searches, provided the local distances are metric and static. Here the above mentioned incremental kNN algorithm by Hjaltason and Samet is suitable due to unknown k required by the aggregating function.

Based on the observations discussed in this section (i.e., tags-based search, multiple local searches + aggregation), we come with the fourth question of the paper,
Q4: Are the simple similarity queries competitive enough? In other words, should the MAM research focus also on a native support of more complex similarity queries than the simple range/kNN?

3.2 Search engines

After the more or less academic discussion on MAMs in CBIR systems, the following analysis investigates the impact of MAMs on real-world engines. However, because most of the engines were not much documented and/or patented, this part of the paper should be considered with caution.

3.2.1 Mainstream multimedia search engines

At first, we have focused on 32 mainstream web sites providing multimedia retrieval, including search engines, hosting servers, and stock servers.

The *multimedia search engines* do not constitute standalone solutions, they are rather add-ons extending the classic web search engines. In particular, we considered web sites for image search (Google Image Search, Bing Image Search, AllTheWeb, PicSearch), video search (Bing Video Search, Lycos, AOL Video Search, SearchForVideo, BlinkX) and audio search (KaZaA, FindSounds, Skreemr, Yahoo Music Search). In addition to search engines, we included also hosting servers for images (Flickr, PhotoBucket, ImageShack, Google Picasa, DeviantArt) and videos (YouTube, DailyMotion, Yahoo Video, MySpace, MetaCafe, Google Video, MSN Video). Finally, we included major (micro)stock servers (Corbis, Getty, iStockPhoto, ShutterStock, Fotolia, DreamsTime, Alamy, Veer) that offer a paid multimedia content (image, video, audio, vector, flash) to professional designers.

Among all the listed web sites, just 7 (Google, Bing, PicSearch, FindSounds, Flickr, Picasa, ShutterStock) support a kind of content-based retrieval. However, only FindSounds supports true similarity search (though information on the similarity and index is not available), while the rest of the sites provide a kind of Pseudo-CBIR (see Section 3.1.3).

3.2.2 Content-based image retrieval engines

In order to increase the number engines providing similarity search, we have analyzed CBIR systems listed at Wikipedia [24], namely, Elastic Vision, Gazopa, Imense, Imprezzeo, Incogna, Like.com, MiPai, idee Visual Search Lab, Empora, Shopachu, TinEye, Tiltomo, eBay More Like This, ALIPR, Anaktisi, BRISC, Caliph & Emir, CIRES, FIRE, GNU Image Finding Tool, ISSBP, img(Rummager), imgSeek, IKONA, MUVIS, PIRIA, RETIN, Retrievr, SIMBA, TagProp, MUFIN. Among the 29 engines, 25 use similarity search concept, while 7 of them certainly use a metric distance (for the rest of engines the information was not available.) Only MiPai and MUFIN were identified as MAM-based. Anyways, as there is not much evidence (or even promotion) that current content-based search engines use MAMs, we ask the fifth question of this paper,
Q5: Have the real-world search engines ever used a MAM?

3.3 Beyond the metric space model

As pointed out in a recent survey [22], the playground for similarity search is much larger than the usual area of multimedia retrieval. In particular, similarity search tasks become even more common in areas like biometric databases, various scientific databases (bioinformatics, chemoinformatics, medical data), social networks, etc. Moreover, it was shown that domain experts develop constantly more complex similarities that have to reflect higher demands on retrieval effectiveness, leaving the simple distances like L_p metrics or edit distance. The new complex distances are often being generalized in order to become better parameterizable for a given domain. Due to such extensive similarity modeling, the new distances often lose their closed form (i.e., concise mathematical formula) and become heuristic algorithms. In consequence, the more complex distance, the more likely it will violate the metric postulates, so it becomes a *nonmetric*. As an example we name the (nonmetric) Smith-Waterman alignment [23], generalizing the edit distance to better model functional similarity of proteins (including scoring matrices, local alignment and gap penalizations).

The domain experts often do not care whether their distance is a metric or is not, because their similarity search tasks are usually not (yet) large-scale and the sequential search is sufficient for them at the moment. On the other hand, in case a model gets matured in the particular domain ("surviving" a certain period), the demand for better scalability could reach a higher priority, so that a kind of nonmetric indexing will be needed. Apparently, the MAMs cannot be directly utilized here, as they require metric distances. This observation leads us to the last question of the paper, *Q6: Isn't the metric space model too restrictive?*

3.3.1 "Metric nonmetric" indexing

Nevertheless, there appear transformational approaches that put the MAMs back into the game also for the purpose of nonmetric similarity search. This could bring fascinating opportunities for indexing by similarity, not yet discovered by the database community. In particular, the proposed TriGen algorithm [21] constitutes a mapping of a semi-metric space into an (approximation of) metric space, so that MAMs can be used without limitations. The mapping is achieved by finding suitable concave function, so that the triangle inequality becomes satisfied while the intrinsic dimensionality of the new space is kept as low as possible.

3.3.2 Alternative indexing

There appear also alternative approaches that completely abandon the metric space model and propose a qualitatively different mathematic formalism for general similarity indexing. For example, the recently introduced concepts of *ptolemaic indexing* [14] (replacing triangle inequality by ptolemaic inequality) or *indexing fuzzy similarity* [11] (replacing metric properties by fuzzy logic operators) represent the first attempts to natively nonmetric indexing.

4. DISCUSSION AND SUGGESTIONS

Based on the observations summarized in the previous sections, in the following text we give some suggestions to the future MAM research from the application point of view. Our aim is to strengthen the competitiveness of metric access methods in the context of content-based retrieval, by addressing the questions formulated in this paper.

As a leitmotif, an active attitude of the MAM community to domain problems is necessary, in order to achieve a larger success of MAM research in "enterprise" software ap-

plications. Hence, in addition to the "formal" experiments, (some of) the MAM proposals should dive into the real-world problems, showing that MAMs can really contribute to the performance of complex data management.

4.1 Balancing the model complexity

To address the questions Q1, Q2, Q6, the structure of the similarity model complexity deserves an increased attention. The formulation of questions Q1 and Q2 was motivated by the almost exclusive use of cheap $O(n)$ metric distances (mostly Euclidean vector spaces) and by the closely related problem of cost measures relevancy. In the following text, we discuss the two conceptual possibilities when modeling a similarity – either a low-level descriptor space and a complex distance, or a high-level descriptor and a simple distance.

4.1.1 Complex distance + low-level descriptor

The most promising opportunity for MAMs would be seeking for applications that require *complex* and *expensive* metric distances. The advantages for MAMs are two-fold, first, for complex metric distances (often non-vectorial) the alternative indexing methods (e.g., spatial access methods) cannot be efficiently employed, and second, for expensive distances the DC cost (being the MAMs' optimization priority) becomes relevant due to the negligible contribution of the other cost types.

From the semantic point of view, this "complex distance" concept assumes most of the retrieval logic lies inside a complex distance function, while the descriptor is large and contains rather low-level (raw) features produced by some elementary feature extraction procedure. In fact, the complex distance algorithm is supposed to finish the feature extraction at the moment of distance evaluation, however, during the evaluation also the second descriptor is available. Hence, such "online" feature extraction is able to integrate both descriptors into the process, allowing thus their better comparison.

As an example, we could consider the time series matching using the dynamic time warping (DTW) distance [17]. Instead of applying just the Euclidean distance on the time series, the DTW distance at first finishes the feature extraction by aligning the closest value pairs between the two series, while the resulting Euclidean distance is computed on this optimal alignment.

Unfortunately, as the complex and expensive distances are often not metrics, a preprocessing step that maps the nonmetric space into metric space, e.g., the TriGen algorithm, is needed (just the case of the DTW distance). It is also questionable, whether the "cleverness" of the complex distance could pay off the computational expensiveness (when compared with the opposite approach, that follows).

4.1.2 Simple distance + high-level descriptor

Nowadays, it seems the "complex distance" concept is less popular than the inverse concept that supposes a *simple* and *cheap* distance. From the semantic point of view, the "simple distance" concept aims to put the essence of the retrieval logic right into the descriptors. Hence, the descriptor (often vector) contains rather high-level features produced by a sophisticated feature extraction procedure[3]. The distance is "degraded" to simple aggregation of internal distances within the particular descriptor features. In most cases, this leads to the popular model of vector space with non-correlated dimensions + an L_p distance. Apparently, in the "simple distance" concept the position of MAMs is not as advantageous as in the "complex distance" concept.

To demonstrate the properties of both concepts in the same domain, we consider, again, the example of time series matching, but now using the "simple distance" concept [25]. Instead of low-level features (e.g., the time series itself), the time series could be modeled as a linear combination (or concatenation) of some representative subseries. Hence, the time series becomes a high-level vector modeled in space of subseries, while the Euclidean distance is used as similarity.

4.1.3 Complex vs. simple distance

When reasoning pragmatically, the "simple distance" concept promises more benefits for the practitioners (which is rather bad news for MAMs). In particular, the increased cost needed for the high-level feature extraction procedure is amortized within the frequently repeated search by cheap distance. Here we can see a motivation similar to the very purpose of indexing, where an expensive one-shot indexing phase is paid by multiple efficient searches.

Nevertheless, to give MAMs a better prospect, we can formulate the following question. Can always be the model complexity put into "canonized" descriptors within the "simple distance" concept, or do there exist (important) problems requiring inherently a complex distance? A possible positive answer to this question is suggested in Section 4.3.

4.2 MAMs in search engine architectures

In order to address the questions Q3, Q4, Q5, in the following section we discuss the role of MAMs in various architectures of similarity search engines (as categorized in Section 3.1.3).

4.2.1 MAM as single-model engine

As the single-model engine assumes single-descriptor space under a complex similarity, a MAM could be used as the core technology, provided the similarity is (mapped to) a metric distance. Although the single-model engines have clear semantics of the search (described by a rigorous model), they are quite limited in flexibility as discussed in Section 3.1.1. Hence, because there is not much room for adjusting the distance function after the indexing phase, the expressive power of the retrieval could be increased by offering a larger portfolio of similarity queries.

In addition to the simple kNN/range queries that allow just two parameters (a single example + a radius or k), there have been more complex query types proposed, providing more detailed query specification, yet remaining fully consistent with the single model (e.g., multi-example queries [9], metric skylines [7]).

4.2.2 MAM as a part of hybrid-model engine

In the hybrid-model engine, a MAM is nested deeper in the architecture (say, at "middleware" level), but still representing the most important part. It combines several similarity searches into an aggregated output, providing thus more flexible retrieval. As an example, we mention a shape retrieval method [3], where multiple queries are performed on M-tree indexes, while the results are finally aggregated by a nonmetric ranking.

[3]To be complete, there are many approaches (even the majority?) using simple distance *and* low-level descriptors.

Note the final aggregation could not only combine output of several metric indexes, but it can incorporate also results of searches that are not content-based, e.g., the popular keyword search.

4.2.3 MAM as a tool

In some implementations of similarity-search engines, the MAMs cannot compete with specific indexing models. For example, consider a CBIR based on the model of visual words [19], where the descriptor of an image is modeled as a sparse 10^6-dimensional vector (i.e., million of visual words). The dimensions of the vector correspond to tf-idf weights of visual words, which is a concept adopted from the vector model of information retrieval [2]. The distance between two vectors is evaluated as the well-known cosine similarity. It is also well-known that for searching a collection of sparse vectors under cosine similarity, the inverted file is extremely efficient (due to only traversing the lists corresponding to nonzero weights in the query vector). Any MAM in such an extremely high-dimensional space is condemned to fail.

When creating the image descriptor, for each image segment[4] (for its 128-dimensional SIFT vector, respectively) the most similar visual word (also SIFT vector) is found, while all the visual words are organized in a vocabulary. Hence, the database and query images are transformed into the space of visual words, using the nearest neighbor search (under L_2 or L_1 distance) in the vocabulary. Since the vocabulary (million dense 128D vectors) needs to be efficiently searched, an index is necessary, while this is an opportunity for MAMs. Hence, the MAM within the CBIR engine could not only provide the retrieval of images, it could serve as a particular tool speeding the descriptor preparation (the feature extraction, respectively). Moreover, the role of MAM as a tool is not limited to retrieval engines, as it could be applicable in other areas, such as multimedia data mining.

4.3 Bidirectional motivation

We end up the discussion with a highly speculative meditation on how to bring closer the interests of MAM research and domain-specific research. Apparently, the worlds of databases and various applied sciences requiring management of data are separated. The gap caused by different concerns of each world is even magnified by different terminology, where for a database researcher it is often difficult to identify a possible similarity function within a proprietary retrieval algorithm. The popular BLAST method used in proteomics research could be an example [1], where measuring the similarity of protein sequences is mixed with the access method (a search tree).

Had we establish a picture of usual thinking stereotype of a domain expert when modeling a content-based retrieval technique, we could consider two variants (see Figure 6 top).

4.3.1 All-in-one stereotype

In the first one, the domain expert does not distinguish between the content-based semantics and the access method (e.g., the BLAST example), which turns out in a monolithic retrieval solution (usually a heuristics without a rigorous formal model)[5].

[4]An image is segmented into more than 3000 segments, so we get 3000 nonzero weights per each million-dimensional representation of an image (i.e., 99.7% vector sparsity).

[5]Actually, BLAST aims to approximate the similarity model of Smith-Waterman alignment, but it is still a heuristics.

4.3.2 Separated similarity + sequential search

Second, the expert views the retrieval task as sequential search, where a similarity function is used to check the relevancy of database instances against the query instance. Hence, this variant is more suitable for a database application, as the sequential search could be replaced by a more efficient access method, for example, a MAM.

4.3.3 Modeling augmented by indexing

The previous variant assumes one-directional motivation, where a database research is motivated by an already formulated domain-specific retrieval problem. However, because the domain expert considered a future efficient access method only as optional, he/she naturally tended to design the retrieval problem as simple as possible, in order to minimize the cost used by sequential search. In consequence, this thinking stereotype pressurizes the domain expert to employ only cheap distances that are not sophisticated (exhibiting low precision and recall in the retrieval).

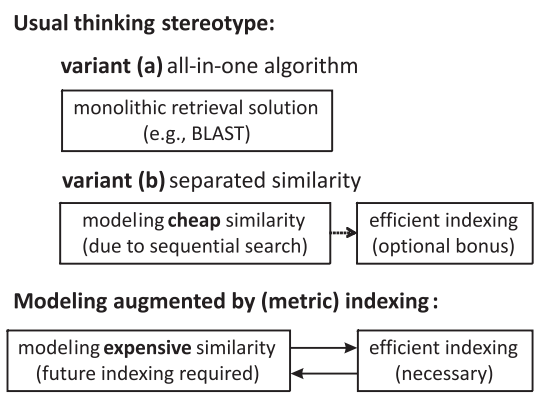


Figure 6: Bidirectional motivation.

Thus, we suggest a *bidirectional-motivation* thinking (see also Figure 6 bottom), that requires both worlds to tightly cooperate. The idea is based on "modeling augmented by indexing", where the domain expert banks on necessary indexing in her/his retrieval task. Hence, the prior knowledge of faster than sequential search enables the expert to model the similarity more generously, using expensive (metric) distance functions. The application of MAMs within such a framework is obvious, while the expensiveness of metric distances increases the likelihood of MAMs' success.

5. CONCLUSIONS

In this paper we have discussed benefits and the impact of metric access methods (MAMs) on the real-world applications and search engines. We asked six questions related to the correctness of intra-MAM research and to the applicability of MAMs in content-based retrieval. In the broad context of content-based retrieval, we have suggested that MAMs have to fight for their success, as waiting for an impulse of demand from outside the database community appears as rather naïve. Anyways, despite the scepticism intentionally (provocatively) invoked throughout the paper, we believe the metric access methods have solid foundations that promise successful applications of MAMs in many domains.

5.1 One more provocation at the end

At the very end, we would like to mention the topic of correct experimental comparison, however, a detailed analysis would deserve a standalone survey. Because the number of papers on various MAMs grows to a substantial volume, the credibility of experimental results needs to increase as well. While in the "ancient" times of only several papers on MAMs a particular result was rather easy to verify, nowadays, in the multitude of proposals such a verification is not easy due to increasing MAMs' complexity. At the same time, we often read claims that "our method beats the competitors by an order of magnitude", so one has a feeling that the power of modern MAMs, being transitively "by many orders of magnitude faster than the others", is almost infinite. Unfortunately, this is not true, and it points to the importance of correct experimentation, including *repeatable experiments* (renowned testbeds and algorithms, e.g., the SISAP library), *measuring realtime* (see Section 2.2), and *fair comparison* (optimization of the competing algorithms and not twisting the experimental setup to handicap the others).

Acknowledgments

This research has been supported in part by Czech Science Foundation project Nr. 201/09/0683.

6. REFERENCES

[1] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Res.*, 25:3389–3402, 1997.

[2] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing, 1999.

[3] S. Berretti, A. D. Bimbo, and P. Pala. Retrieval by shape similarity with perceptual distance and effective indexing. *IEEE Transactions on Multimedia*, 2(4):225–239, 2000.

[4] C. Böhm, S. Berchtold, and D. Keim. Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases. *ACM Computing Surveys*, 33(3):322–373, 2001.

[5] E. Chávez and G. Navarro. A Probabilistic Spell for the Curse of Dimensionality. In *ALENEX'01, LNCS 2153*, pages 147–160. Springer, 2001.

[6] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[7] L. Chen and X. Lian. Efficient processing of metric skyline queries. *IEEE Trans. on Knowl. and Data Eng.*, 21(3):351–365, 2009.

[8] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB'97*, pages 426–435, 1997.

[9] P. Ciaccia, M. Patella, and P. Zezula. Processing complex similarity queries with distance-based access methods. In H.-J. Schek, F. Saltor, I. Ramos, and G. Alonso, editors, *Advances in Database Technology - EDBT'98, 6th International Conference on Extending Database Technology, Valencia, Spain, March 23-27, 1998, Proceedings*, volume 1377 of *Lecture Notes in Computer Science*, pages 9–23. Springer, 1998.

[10] R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, 40(2):1–60, 2008.

[11] A. Eckhardt, T. Skopal, and P. Vojtáš. On fuzzy vs. metric similarity search in complex databases. In *Proc. 8th Conference on Flexible Query Answering Systems (FQAS'09)*, volume 5822 of *LNAI*, pages 64–75. Springer, 2009.

[12] R. Fagin. Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci.*, 58(1):83–99, 1999.

[13] M. L. Hetland. The basic principles of metric indexing. In *Swarm Intelligence for Multi-objective Problems in Data Mining*. Springer, 2009.

[14] M. L. Hetland. Ptolemaic indexing. *CoRR*, abs/0911.4384, 2009.

[15] G. Hjaltason and H. Samet. Incremental similarity search in multimedia databases, Comp. Science Dept. TR-4199, Univ. of Maryland, College Park, 2000.

[16] W. H. Hsu, L. S. Kennedy, and S.-F. Chang. Reranking methods for visual search. *IEEE Multimedia*, 14:14–22, 2007.

[17] E. Keogh, L. Wei, X. Xi, S.-H. Lee, and M. Vlachos. LB_Keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 882–893. VLDB Endowment, 2006.

[18] M. L. Mico, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. *Pattern Recogn. Lett.*, 15(1):9–17, 1994.

[19] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *IEEE Conference on Computer Vision and Pattern Recognition. CVPR '07*, pages 1–8, 2007.

[20] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[21] T. Skopal. Unified framework for fast exact and approximate search in dissimilarity spaces. *ACM Transactions on Database Systems*, 32(4):1–46, 2007.

[22] T. Skopal and B. Bustos. On Nonmetric Similarity Search Problems in Complex Domains. *ACM Computing Surveys 44(3)*, 2012, issue tentative, available at http://siret.ms.mff.cuni.cz/skopal/pub/nmsurvey.pdf.

[23] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[24] Wikipedia. List of content-based image retrieval engines `http://en.wikipedia.org/wiki/List_of_CBIR_engines`, June 16, 2010.

[25] L. Ye and E. Keogh. Time series shapelets: a new primitive for data mining. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 947–956, New York, NY, USA, 2009. ACM.

[26] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach (Advances in Database Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

Indexing

SISAP 2010

Dimension Reduction for Distance-Based Indexing

Rui Mao
Shenzhen University
3688 Nanhai Rd., Office Tower #342
Shenzhen,Guangdong, 518060, China
(+86)755 2655 8644
mao@szu.edu.cn

Willard L. Miranker
Yale University
227 Church Street, PH2E
New Haven, CT 06510, USA
(+1)203 432 7226
miranker@cs.yale.edu

Daniel P. Miranker
University of Texas at Austin
1 University station, C0500
Austin, TX 78712, USA
(+1)512 471 9541
miranker@cs.utexas.edu

ABSTRACT
Distance-based indexing exploits only the triangle inequality to answer similarity queries in metric spaces. Lacking of coordinate structure, mathematical tools in R^n can only be applied indirectly, making it difficult for theoretical study in metric space indexing. Toward solving this problem, we formalize a "pivot space model" where data is mapped from metric space to R^n, preserving all the pair wise distances under L^∞. With this model, it can be shown that the indexing problem in metric space can be equivalently studied in R^n. Further, we show the necessity of dimension reduction for R^n and that the only effective form of dimension reduction is to select existing dimensions, i.e. pivot selection. The coordinate structure of R^n makes the application of many mathematical tools possible. In particular, Principle Component Analysis (PCA) is incorporated into a heuristic method for pivot selection and shown to be effective over a large range of workloads. We also show that PCA can be used to reliably measure the intrinsic dimension of a metric-space.

Categories and Subject Descriptors
H.2.2 [**Database management**] Physical Design – *Access methods*; H.3.1 [**Information storage and retrieval**]: Content analysis and indexing — *indexing methods*; H.3.3 [**Information storage and retrieval**]: Information search and retrieval — *clustering*, *search process*

General Terms
Algorithms.

Keywords
Similarity query, metric space, dimension reduction, intrinsic dimension, pivot selection, pivot space model.

SISAP2010, September 18-19, 2010, Istanbul, Turkey.

1. INTRODUCTION
Distance-based indexing [7, 11, 20], also known as metric-space indexing, only requires a metric distance function to answer similarity queries. This, so-called, "black-box" model is advantageous for any application where the data cannot be effectively mapped to feature vectors, enabling a uniform programming model for problems that can be recast into metric spaces. The generality of the approach is also its challenge. What makes distance-based indexing difficult is the lack of coordinate structure. The dimension of the data is not explicit. As a result, mathematical tools developed for R^n are not directly applicable to distance-based problems.

A common method (Section 3) is to first map the metric space to low dimensional R^k, and then to answer the similarity query with geometric methods developed for R^k. To summarize and formalize this method, we propose the *pivot space model*. With this model, there are three steps to answer a similarity query in a metric space. In step 1, the data in the metric space is mapped into a subset of R^k, called the *pivot space*. Then, a region in R^k containing all the query results, named the *query cube*, is determined. In step 2, multi-dimensional methods are applied to retrieve all the points in the query cube. In step 3, all the points retrieved in step 2 are compared with the query object to remove the false positives.

It has been shown that any finite metric space of n points can be mapped isometrically into R^n, one dimension for each point, using the L^∞ norm [16]. In pivot space model we call the image of a dataset so constructed the *complete pivot space*. It will follow that the evaluation of a query in the complete pivot space may be accomplished using multidimensional indexing methods but will require a calculation for each of the n dimensions. Thus, dimension reduction is necessary. Next, we prove that any dimension reduction technique that creates new dimensions will still require a calculation for each of the n dimensions. Therefore, the only effective form of dimension reduction for the complete pivot space is one that selects a subset of the existing dimensions, i.e. pivot selection.

To demonstrate how a general dimension reduction technique can be applied to distance-based problems, we design a pivot selection algorithm based on PCA, a popular dimension reduction technique for multi-dimensional spaces (Section 5). The basic idea is to select existing dimensions that would best approximate the eigenvectors computed by PCA. Results show that our pivot selection heuristic outperforms a deterministic corner-selection and a non-deterministic incremental selection heuristic [5].

Several analytic studies of high dimensional query problems have concluded that the indexability of data is sensitive to its dimension and that the performance of tree-based indexing on fixed size vector data sets degrades to a sequential scan as the dimension of the data increases [2, 21, 23]. Based on the pivot space model, we propose to estimate the intrinsic dimension of a metric space dataset based on the eigenvalues computed by PCA on its complete pivot space. Empirical results show that our method gives more accurate estimates of intrinsic dimension (Section 6).

2. RELATED WORK

The development of distance-based indexing algorithms is commonly decomposed into to two sub-problems, pivot selection and partitioning methods. These are well discussed in a pair of surveys and a text [7, 11, 20].

Only a few methods have been investigated for pivot selection [3-5, 14, 15, 25]. Bustos et al. exploit sampling and select pivots that maximize the mean [5]. Further, they argue that good pivots are usually outliers, but that the reverse is not true. The Metric Tree (M-tree) bulkload algorithm selects pivots randomly but does not use any sampling [8]. SA-tree selects the centers of neighboring cells of a Voronoi diagram as pivots [18].

The farthest-first-traversal (FFT) k-center clustering algorithm is usually used to choose pivots. It is a fast and convenient way to identify corners, or outliers. FFT minimizes the maximum cluster diameter and gives a result at most twice the optimal diameter [12]. Its time and space complexities are both O(n). The use of FFT is based on Yianilos' observations made of uniformly distributed points in the unit square [25]. He proposes to select the corners of the data set as pivots for Vantage Point Tree (VPT). Multi-Vantage Point Tree (MVPT) [3] selects multiple corners.

Brin, in GNAT, suggests index trees be constructed with a variable number of pivots such that a larger number of pivots are used for higher populated clusters to maintain balance [4].

What determines the indexability of data is the intrinsic dimension, which is invariant and independent of data representation. Many authors have tried to define and quantify intrinsic dimension [7, 14]. We consider it to be k, where the data can be embedded into R^k with small distortion.

The following are two existing methods.

Method 1: Chavez et al. [7] define the intrinsic dimension of a metric space as $\rho = \mu^2/2\sigma^2$, where μ and σ^2 are the mean and variance of the pairwise distances.

Method 2: Mao et al. [14] measure how the volume of a hyper-ball, or the number of points in it, changes with respect to the radius. Let r be the range query radius, and n be the average number of range query results. Then, slope coefficient, given by linear regression on the logarithm of n and r, is an estimate of the intrinsic dimension.

Method 1 is simple. Ramakrishnan et al. use similar forms to that of Method 1 to determine the stability of workloads [2, 21]. Method 2 is actually a variation of the Box Dimension [9]. It is limited by values of r and the assumption of uniform distribution.

3. PIVOT SPACE MODEL

In this section, we summarize a generally used metric space indexing approach and propose the pivot space model. A theorem is given showing that the same pivot space can be produced from datasets in a metric space, or in R^n.

3.1 General Steps

In this subsection, let R^n denote a general real coordinate space of dimension n. There are three steps.

Step 1: (1) Map the data into R^n. (2) Map the query object into R^n. (3) Determine a region in R^n that completely covers the range query ball.

Pivot selection maps the data into R^n, i.e. to select particular points in the database as pivots and to represent each point by its distances to the pivots. We name the image of data so obtained the *pivot space*. Determining the distances between a range query object and pivots essentially maps the query into the pivot space.

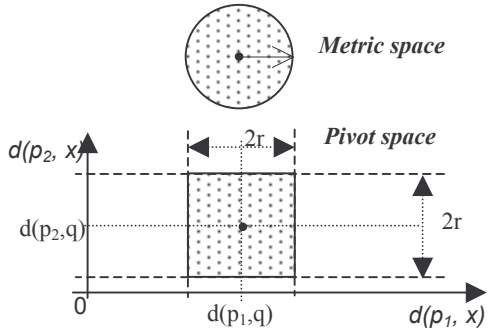


Figure 1. The query ball of a range query in a metric space is covered by a square in the pivot space

The shape of the image of the query ball of a range query (q, r) in a general metric space is not clear in a pivot space. However, it can be proved, from the triangle inequality, that the image of the query ball is completely covered by a hypercube of edge length 2r in the pivot space [7], which is actually a ball of radius r in the new metric space specified by the pivot space and the L^∞ distance. We call the hypercube the *query cube*. Figure 1 shows an example where 2 pivots are selected. All the points in the query ball are mapped into the query square in the 2-d pivot space plane. Points outside of the square can be discarded using the triangle inequality while points within the square cannot be discarded. To answer the query, one first retrieve all the points in the square. Then their distances to q are computed directly to determine whether their pre-images are in the query ball.

Step 2: Exploit multi-dimensional techniques to retrieve all the points in the region determined in Step 1.

The basic idea of Step 2 is divide-and-conquer based on the data coordinates. Many partition methods in multi-dimensional indexing can be applied here. For example M-tree [8, 22] partitions the data in a manner similar to an R-tree [10], while MVPT's partition is similar to a k-d tree [1].

Step 3: For each point retrieved in Step 2, compute its distance to the query object to remove false positives.

In Step 1, the query cube is a superset of the image of the query ball. Therefore, after all the points in the query cube are

retrieved, their distances to the query object have to be computed to remove the false positives. Step 1 is the focus in this paper.

3.2 Pivot Space Model

Let (M, d) be a metric space, where M is the space containing the data, and d is a metric distance function. Let $S = \{x_i \mid x_i \in M, i = 1, 2, \ldots, n\}$, be the database, $n \geq 1$. S is a finite indexed subset of M. Duplicates are not allowed.

Let $P = \{p_j \mid j = 1, 2, \ldots, k\}$ be a set of pivots. $P \subseteq S$. Duplicates are not allowed.

Definition 1 Pivot Space, $F_{P,d}(S)$: Given the set of pivots $p_j \in P$, each point in S can be mapped to a non-negative number, which is its distance to p_j. That is, the following mappings can be defined:

$$f_j: M \rightarrow R^+ \bigcup \{0\},\ f_j(x) = d(x, p_j),\ p_j \in P,\ j = 1, \ldots, k.$$

The ordered collection of these k mappings defines a vector valued mapping $F_{P,d}$ on M, which maps a point in M to a point in the non-negative orthant of R^k. The j-th coordinate of the image represents the distance to p_j:

$$F_{P,d}: M \rightarrow R^k:\ x_p \equiv F_{P,d}(x) \in F_{P,d}(M),$$

$$x_p = (f_1(x), \ldots, f_k(x)) = (d(x, p_1), \ldots, d(x, p_k)),$$

We say x_p, a k-dimensional vector in R^k, is the image of x. The *pivot space* of S is defined as the image set of S under $F_{P,d}$,

$$F_{P,d}(S) = \{x_p \mid x_p = F_{P,d}(x) = (d(x,p_1), \ldots, d(x,p_k)),\ x \in S\}.$$

Here and in what follows, the superscript p denotes objects in the pivot space. $F_{P,d}(x)$ can be written as F(x, P, d) and $F_{P,d}(S)$ as F(S, P, d) to show that these mappings are specified in term of three parameters. Pivot space shall also refer to R^k where $F_{P,d}(S)$ resides, since confusion will not result.

	p_1^p	p_2^p	…	p_k^p
x_1^p	$d(x_1, p_1)$	$d(x_1, p_2)$	…	$d(x_1, p_k)$
x_2^p	$d(x_2, p_1)$	$d(x_2, p_2)$	…	$d(x_2, p_k)$
…	…	…	…	…
…	…	…	…	…
x_n^p	$d(x_n, p_1)$	$d(x_n, p_2)$	…	$d(x_n, p_k)$

Figure 2. Point-pivot pairwise n×k distance matrix $D_{P,d}(S)$

Definition 2 Point-Pivot Pairwise Distance Matrix $D_{P,d}(S)$: $D_{P,d}(S)$, also written as D(S, P, d), is an n×k matrix (Figure 2), whose (i,j)-th element is the distance from the i-th data point to the j-th pivot. Each row vector (x_i^p) can be regarded as a point in the pivot space specified by all the distances from a database point to each of the pivots, and each column vector (p_j^p) can be regarded as a basis vector in the pivot space specified by the distances from all points to the j-th pivot:

$$D_{P,d}(S) = (x_1^p, x_2^p, \ldots, x_n^p)^T = (p_1^p, p_2^p, \ldots, p_k^p),$$

where the x_i^p s are the row vectors,

$$x_i^p \equiv F_{P,d}(x_i) = (d_{i1}, d_{i2}, \ldots, d_{ik}),$$

and p_j^p is the column vector, $p_j^p \equiv (d_{1j}, d_{2j}, \ldots, d_{nj})^T$. Moreover, $d_{ij} = d(x_i, p_j) \geq 0$, i = 1, 2, …, n, j = 1, 2, …, k.

Properties of the pivot space and the distance matrix include:

(1) Each point in $F_{P,d}(S)$ is a row in $D_{P,d}(S)$. $F_{P,d}(S)$ and $D_{P,d}(S)$ are representations of one another.

(2) Because different points in S can have the same distance to a pivot, there might be duplicates among the rows of $D_{P,d}(S)$. Therefore, the correspondence between the database points and points in the pivot space $F_{P,d}(S)$, or the rows of $D_{P,d}(S)$, is many-to-one. Similarly, there is a many-to-one correspondence between the pivots and the points in $F_{P,d}(S)$, or the columns of $D_{P,d}(S)$.

The following characterizes the case when all the database points are used as pivots.

Definition 3 Complete Pivot Space $F_d^c(S)$ and Complete Distance Matrix $D_d^c(S)$: $F_d^c(S) = F_{S,d}(S)$ and $D_d^c(S) = D_{S,d}(S)$.

That is, when all points in S are selected as pivots, the complete pivot space of S is the pivot space of S, and the complete distance matrix of S is the distance matrix of S.

The complete pivot space and the complete distance matrix have the following properties.

(1) The dimension of the complete pivot space $F_d^c(S)$ is n.

(2) Because duplicates are not allowed in S, there are no repetitions in the ordered set of distances from one point to all others. Therefore, there are no duplicates in $F_d^c(S)$.

(2.1) Each point in the complete pivot space has exactly one coordinate with value zero, while all other coordinates are positive. That is, all such points reside in the non-negative orthant of R^n.

(2.2) $D_d^c(S)$ is a symmetric n×n matrix, with zeros on the main diagonal and positive entries elsewhere.

(2.3) The correspondence between points in S and points in $F_d^c(S)$ (row vectors of $D_d^c(S)$) is one-to-one. The correspondence between points in S and the basis vectors in $F_d^c(S)$ (column vectors of $D_d^c(S)$) is one-to-one.

(3) $D_d^c(S)$ contains all and only the information provided by the distance oracle. Different data sets from various metric spaces can have the same complete pairwise distance matrix.

The following shows the identicalness between pivot spaces.

Theorem 1: Given metric space (M, d), database S and pivot set P, then F(S, P, d) = F(F(S, P, d), F(P, P, d), L^∞), and D(S, P, d) = D(F(S, P, d), F(P, P, d), L^∞).

Proof: It suffices to show: F(S,P,d) = F(F(S,P,d), F(P,P,d), L^∞).

The following 5 relations follow by definition:

$F(S,P,d) = \{x^p \mid x^p = F(x,P,d) = (d(x,p_1), \ldots, d(x,p_k)),\ x \in S\}$

$F(P,P,d) = \{p^p \mid p^p = F(p,P,d) = (d(p,p_1), \ldots, d(p,p_k)),\ p \in P\}$

$F(F(S,P,d), F(P,P,d), L^\infty) = \{ F(x^p, F(P,P,d), L^\infty)\}$

$F(x^p, F(P,P,d), L^\infty) = (L^\infty(x^p,p_1^p), \ldots, L^\infty(x^p,p_k^p)),\ x^p \in F(S,P,d)$

$L^\infty(x^p,p^p) = L^\infty[(d(x,p_1), \ldots, d(x,p_k)), (d(p,p_1), \ldots, d(p,p_k))]$

$= \max\{ \mid d(x, p_j) - d(p, p_j) \mid, j = 1, 2, \ldots, k \}$

Equality holds in one of the triangle inequalities

$|d(x,p_j) - d(p,p_j)| \leq d(x, p)$, j = 1, 2, …, k since for $p \in P \subseteq S$, there exists a t such that $p = p_t$, $t \in \{1, \ldots, k\}$.

Therefore, we deduce that $L^\infty(x^p, p^p) = d(x, p)$, and

$$F(x^p, F(P, P, d), L^\infty) = (d(x, p_1), \ldots, d(x, p_k)) = x^p.$$

Thus $F(S,P,d)=F(F(S,P,d), F(P,P,d),L^\infty)$, as required. □

In others word, Theorem 1 says that for any pivot space F(S, P, d) generated from a metric space (S, d) and a set P of k pivots, an identical pivot space can be generated from a subset of R^k, i.e. F(S, P, d), with pivots F(P, P, d) and the L^∞ distance. Therefore, when dealing with the pivot space, there is no difference whether the pivot space is created from a metric space dataset S or a real coordinate dataset F(S, P, d). Thus, we can assume that the original data is F(S, P, d), in R^k.

In the complete pivot space, Theorem 1 becomes:

$$F^c_{L^\infty}(F^c_d(S)) = F^c_d(S), \text{ and thus } D^c_{L^\infty}(F^c_d(S)) = D^c_d(S)$$

Note that it states the known fact that any finite metric space (size n) is isometric to a metric space formed by a subset of R^n (R^{n-1}, to be more precise) with the L^∞ distance [16].

In the following, Corollary 1 states that the mapping from the metric space to the complete pivot space can be applied recursively and the result remains invariant. Corollaries 2 and 3 describe the impact of Theorem 1 on distance-based indexing. The proofs are omitted as they are straightforward.

Corollary 1: Let $F^{c\ (1)}_{L^\infty}(F^c_d(S)) = F^c_{L^\infty}(F^c_d(S))$ and

$F^{c\ (n+1)}_{L^\infty}(F^c_d(S)) = F^c_{L^\infty}(F^{c\ (n)}_{L^\infty}(F^c_d(S)))$, $n \geq 1$. Then,

$$F^{c\ (n)}_{L^\infty}(F^c_d(S)) = F^c_d(S), n \geq 1.$$

Corollary 2: Since the data set S from any general metric space can be mapped isometrically into the complete pivot space with the L^∞ metric without loss of any distance information, instead of indexing S in a black-box metric space, it is equivalent to index $F^c_d(S)$ in the much more palpable vector space R^n.

Corollary 3: The intrinsic dimensions of S and $F^c_d(S)$ equal.

This is because that the intrinsic dimensions are specified by the pairwise distances and the pairwise distance matrices of S and $F^c_d(S)$ are the same.

4. DIMENSION REDUCTION FOR DISTANCE-BASED INDEXING

This section is dedicated to the problem "how to map a metric space to R^k"? We study the question from the perspective of dimension reduction. The discussion starts from the complete pivot space, the most straightforward case and the case with all the information. Issues under discussion include "can we evaluate similarity queries in the complete pivot space directly?", "how to perform dimension reduction for the complete pivot space?", "why is pivot selection important?", and "how to select pivots?"

4.1 Evaluating Similarity Queries in the Complete Pivot Space Directly

Theorem 2 answers the question "can we evaluate similarity queries in the complete pivot space directly?" It is straightforward and helps to understand Theorem 3.

Theorem 2: Evaluation of similarity queries in the complete pivot space degrades the query performance to linear scan.

Proof: Given a similarity query object, the computation of its coordinates in all dimensions of the complete pivot space is already a linear scan of the database. □

Therefore, to answer the similarity query posed in the metric space in R^n, dimension reduction in the complete pivot space is inevitable.

4.2 Pivot Selection: the Only Form of Effective Dimension Reduction

Theorem 2 establishes the necessity of dimension reduction in the complete pivot space. The underlying reason is the lack of uniform coordinate structure in general metric spaces. In the following, we discuss the type of dimension reduction that can be applied to the complete pivot space. A dimension reduction technique is termed "effective" if evaluation of similarity queries in the space generated by this technique does not degrade to a linear scan. Not all the dimension reduction for multi-dimensional indexing is effective for distance-based indexing.

Theorem 3: If a dimension reduction technique creates new dimensions based on all existing dimensions, evaluation of similarity queries in the space generated by this technique degrades to a linear scan.

Proof: Let $S = \{x_i \mid i = 1, 2, \ldots, n\}$, be the database, and let d be the distance oracle. Let y be an arbitrary point in the metric space, and let its coordinates in the complete pivot space $F^c_d(S)$ be $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$. Then, $y^{(i)} = d(y, x_i)$, $i = 1, 2, \ldots, n$. Let a new coordinate $y^{(n+1)} = G(y^{(1)}, y^{(2)}, \ldots, y^{(n)})$, be specified by some unspecified function G. For example, G might be a linear combination of its variables. Given a query object q, to compute coordinate $q^{(n+1)}$, coordinates $q^{(1)}, q^{(2)}, \ldots, q^{(n)}$ in the complete pivot space $F^c_d(S)$ have to be computed first. This already represents a linear scan of the database. □

Although the power of dimension reduction methods in multi-dimensional indexing is clear, the cost to set up the coordinate structure based on new dimensions is excessive. Therefore, dimension reduction for the complete pivot space should only select existing dimensions.

Therefore, pivot selection is the only effective form of dimension reduction for the complete pivot space. It is the only effective form of mapping from a metric space to R^k.

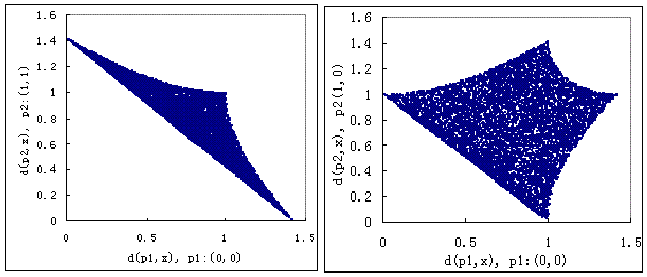
4.3 The Importance of Pivot Selection

To date, the major attention in this area has been on the partition problem in Step 2 (Section 3.1). The importance of pivot selection is not fully recognized, and most methods just try to mimic multi-dimensional indexing in metric spaces.

Pivot selection can be viewed as a process of information loss. The information available to Step 2 is limited by pivot selection. Pivot selection impacts the distribution of data in the pivot space, which is critical to the search performance of Step 2. Moreover, in a pivot space produced from a "better" set of pivots, data are

typically more distinguishable. Thus, false positives are reduced and Step 3 is faster.

An example is to select two pivots for points randomly and uniformly sampled from the unit square with the Euclidean norm. The common method is to use the FFT algorithm to select the two farthest opposite corners of the data [12]. For uniform vector data, these two corners are close to the two ends of the data's first principal component (with the largest eigenvalue). The distances from a point to the two farthest opposite points largely correlate with each other. If a point is close to the first pivot, usually it is far from the second pivot. As a result, if two points are not distinguishable by the first pivot (their distances to the first pivot are similar), they are not likely to be particularly distinguishable by the second pivot. Therefore, selecting these two pivots does not provide much more information than selecting just one pivot. An alternative is to select two corners on the 1st and 2nd principal components. Since the principal components are orthogonal to each other, the two pivots in this case define two less correlated dimensions. If two points are not distinguishable by one pivot, they can still be distinguished by the other pivot.



a) Pivots: opposite corners (0,0) and (1,1)

b) Pivots: neighboring corners (0,0), (1,0)

Figure 3 Pivot spaces (2 pivots) of points randomly sampled from the unit square, with different selection of

Figure 3 illustrates the case just discussed. The two pivots are opposite corners (0,0) and (1,1) in a) and are neighboring corners (0,0) and (1,0) for b). We can see that the pivot space in b) spreads out more widely (less density) than that in a). Therefore, data in pivot space in b) are more distinguishable. Although pivots are always corners, different corners can still have much different impact on query performance. This supports Bustos et al.'s observation that outliers might not be good pivots [5].

4.4 Adapt Dimension Reduction to Pivot Selection

We propose a heuristic to adapt general dimension reduction methods that create new dimensions to distance-based problems. The basic idea is to select existing dimensions that best approximate the new dimensions created by dimension reduction.

One way is to select the existing point with the maximum correlation, or the minimum angle, with the new dimensions created by dimension reduction. Bustos et al. [5] suggest that a good choice of pivots should maximize the mean of the pairwise distances in the pivot space. To increase the mean of the pairwise distance also, we consider the covariance instead of the correlation. That is, for each new dimension, select the point with the largest projection on that new dimension in the pivot space.

With this heuristic, a general dimension reduction technique can be adapted to pivot selection in two steps. First, run the dimension reduction on the complete pivot space to create the new dimensions. Second, select the pivots with largest projections on the new dimensions.

We show how to adapt PCA to pivot selection with this heuristic in the next section.

5. PCA IN DISTANCE-BASED INDEXING

In this section, we apply PCA to pivot selection and to the estimation of intrinsic dimension.

5.1 PCA for Pivot Selection

In Step 3 (see in Section 3.1), points of the query cube in the pivot space need to be checked by computing the distance with the query object directly. Therefore, it is of key importance to reduce the number of points in the query cube. We aim to maximize the variance of the data along the dimensions of the pivot space, which means the data points are more distinguishable among each other. Because PCA considers the distribution of the whole data set and maximizes the variance of the data along each principal component, we choose it for pivot selection.

A difficulty with PCA is the computational cost. Even a fast approximation usually takes $O(n^2)$ time, which is too expensive for large databases even if it is computed off-line. As Bustos et al. [5] point out, although good pivots are usually outliers, outliers are not always good pivots. The set of outliers forms a good candidate set for good pivots. PCA can be conducted on the candidate set. In other words, PCA is not performed on the complete pivot space, but on a pivot space with outliers as pivots. Since the size of the set of outliers is much smaller than the database, the computational cost of PCA is decreased.

```
PivotSelection ( (S, d):data set, k: number of pivot, c: constant)
{    //run FFT to create a candidate set of size k*c
1.    candidate = FFT(S, k*c);
      //generate the pivot space with candidate as the pivot set
2.    PS = F(S, candidate, d);
      //run PCA on PS
3.    PCSET = EMPCA(PS, k);
      // for each PC, find the point with the largest project on it
      Pivots = {};
4.    for each PC ∈ PCSET
            Pivots = Pivots U argmax_x( Proj(PC,x) );
      return Pivots;     }
```

Figure 4. Algorithm for pivot selection

Our pivot selection algorithm is shown in Figure 4. In step 1, a number of outliers of the data are found by FFT and taken to form a candidate set of pivots. Empirical results show that a good choice of the constant c is approximately 30. In step 2, the distances between the corners and the database points are computed to form the distance matrix of the candidate outlier pivot space, on which PCA is performed in step 3. The PCA algorithm applied is EMPCA [19], which can compute only a given number of principal components with the largest eigenvalues. Finally in step 4, for each principal component, the data point whose image in the pivot space has the largest projection on that principal component is selected as a pivot. Since k and ck is much smaller than the database size, the time

complexity is O(n). The algorithm is compared with FFT and Bustos et al.'s incremental sampling selection method [5]. Results show that the overall performance of our algorithm surpasses the other two.

5.2 Estimating the Intrinsic Dimension

According to previous work [16] and the pivot space model, the complete pivot space together with the L^∞ distance is isometric to the original metric space. Therefore, methods to estimate the intrinsic dimension of R^n [6, 13] are now applicable to a metric space. We introduce a third method to estimate the intrinsic dimension based on the relative change of eigenvalues of PCA in the complete pivot space, equivalently, of the complete distance matrix $D_d^c(S)$.

Method 3: Let $Q = \{q_1, q_2, \ldots, q_n\}$ be the principal components (PCs) of the data in the complete pivot space. Let $\lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ be the variances (eigenvalues) corresponding to each PC, $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n \geq 0$. Note that the λ_i are normalized so that they sum to 1. The intrinsic dimension is estimated as (1) $\hat{d} = \text{argmax}_i (\lambda_i / \lambda_{i+1})$, $i = 1, \ldots, n-1$, and (2) $\sum_{j=1}^{i} \lambda_j > 0.6$, and (3) $0.015 \leq \lambda_{i+1} \leq 0.035$.

Condition (1) indicates that the eigenvalues decreases relatively the most from λ_i to λ_{i+1}. Condition (2) guarantees that at least 60% of the variance of the data is maintained if they are reduced to dimension $\hat{d}$. Condition (3) ensures that principal components with tiny (<1.5%) variance (eigenvalue) will be excluded and principal components with larger (>3.5%) variance will be included.

The three methods are evaluated using a suite of workloads. The results in Section 6 show that all three methods are asymptotically correct, while Method 3 gives quantitatively more accurate estimates for data whose intrinsic dimension are known.

6. EMPIRICAL RESULTS

6.1 Organization of Empirical Study

The test suite consists of synthetic vector data, biological data, real vector data and an image dataset [17]. The synthetic vector consists of data of uniform, exponential and normal distributions. Different dimensions have independent identical distributions. Three types of biological data are considered: (1) the amino-acid sequence fragments of the yeast proteome with weighted-edit distance based on the metric PAM substitution matrix [24], (2) the DNA sequence fragments of the Arabidopsis genomes with Hamming distance, and (3) analytically determined peptide fragmentation spectra of human and E. coli proteins with a pseudo-semi-metric cosine distance. The real vector data consists of the US cartographic boundary data of Texas and Hawaii. The image dataset consists of images represented by 66 dimensional feature vectors with a linear combination of L^1 and L^2 norms. The suite is summarized in Table 1.

The index method used in this study is MVPT [3]. The partition algorithm is clustering partition [14]. The sizes of the databases are all 100k, except for those small workloads for which only limited amounts of data is available. The number of pivots is 2 for Texas and mass-spectra data, 4 for DNA and protein data, and 3 for all others. Based on each pivot, 3 partitions are generated. The maximum number of data points in each index leaf node is 100. Since distance evaluation in a metric space is usually costly, we use the average number of distance calculations, which is implementation independent, to answer 5000 range queries as the performance measure of each index. The queries are chosen sequentially from the beginning of the dataset files of each workload. The radii of range queries are chosen so that approximately 0.01% of the databases are retrieved.

Table 1. Summary of test suite

Workload	Total size	Distance oracle	Domain dimension
Vector (uniform)	1M	L^1, L^2, L^∞ norm	1-20
Vector(exponential)	100k		1-10
Vector(normal)	100k		1-10
Texas	190k		2
Hawaii	9k		2
Mass-spectra	<90k	Fuzzy cosine distance	40,000
protein	100k	Weighted edit distance	6-18
DNA	<256k	Hamming distance	9-18
Image	10221	L-norms	66

In the following, the query performance of PCA-based pivot selection is compared with FFT and incremental pivot selection. Then, we show the results on the intrinsic dimension of the test suite using the 3 methods in Section 6.3.

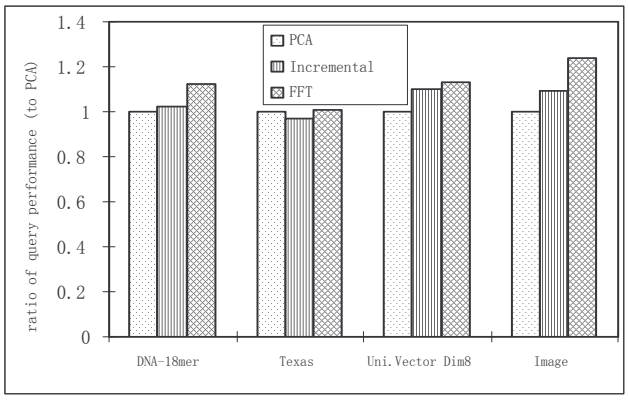


Figure 5. Query cost of three pivot selection methods

6.2 Comparison of pivot selection heuristics

We compare the three pivot selection heuristics, i.e. FFT, Bustos et al.'s incremental selection method [5], and PCA-base method. Indices are built with different heuristics and their query performances are compared.

To be fair, we make sure the index construction costs (measured by number of distance computations to build the indices) of the PCA-based method and the incremental method similar. There are two parameters, A and N, for the incremental method. According to Bustos et al. [5], we try to use a large value for A and a small value for N (Table 2). Moreover, since it is hard to make the two methods have exactly the same construction cost,

we always allow more construction cost for incremental method (Table 2).

The query performances of the three methods are also shown in Figure 5 (normalized by the value of the PCA method). Both Figure 5 and Table 2 show that FFT always yields the worst performance. For most of the workloads, the PCA based method yields the best query performance. The only case where the incremental method yields the best performance is for Texas data, where all the three methods do not differ much. We believe this is because Texas data are of low intrinsic dimension and so are easy to index.

relationship can be drawn from Methods 1 and 2. Method 3, although some constants are determined empirically, always gives d+1 (q+1) as the estimate except a few cases. Thus, Method 3 is more consistent and stable.

Moreover, Method 3 almost always yields d+1 for the vector data of dimension d, no matter what is the probability distribution of the data and the metric. This is consistent with the observation that if the distances from an unknown vector to d + 1 vectors are known, then the coordinates of the unknown vector can be computed accurately. The three methods are consistent with each other for image data.

Table 2. Comparison of pivot selection heuristics, measured by query cost per the number of distance calculations

Dataset	Radius	Selectivity	FFT	Incremental				PCA	
			query	build	query	A	N	build	query
DNA 18-mer fragments		0.017%	68357.2	60.9M	62417.7	9k	10	57.5M	60968.1
Texas	0.015	0.011%	27.7	66.6M	26.7	30k	10	58.8M	27.5
Uniform Vector dimension 8	0.65	0.015%	4952.5	66.3M	4822.6	30k	10	58.5M	4394.4
image	0.08	0.13%	1371.4	5.4M	1209.9	5k	15	4.6M	1106.1

Table 3. Estimates of intrinsic dimension of three methods

Workload	Domain dimension	Distance oracle	Intrinsic dimension		
			$\mu^2/2\sigma^2$	regression	$\text{argmax}_i(\lambda_i/\lambda_{i+1})$
Vector (uniform)	D=1-20	L^∞	1.72d - 1.81	0.73d + 0.88	d+1 (d≠3,4), 4, 7 (d=3, 4)
		L^1	d	0.75d + 0.84	d+1
		L^2	1.41d - 0.71	0.78d - 0.72	d+1
Vector (exponential)	D = 1-10	L^∞	0.244d + 0.446	0.676d + 0.62	d+1
		L^1	0.499d - 0.0006	0.737d + 0.482	d+1
		L^2	0.427d + 0.113	0.72d + 0.534	d+1
Vector (normal)	D = 1-10	L^∞	0.644d + 0.559	0.858d + 0.325	d+1
		L^1	0.875d + 0.002	0.863d + 0.32	d+1
		L^2	0.989d - 0.145	0.872d + 0.305	d+1
Texas	2	L^∞ / L^1 / L^2	1.29 / 1.42 / 0.87	1.54 / 1.54 / 1.51	3
Hawaii	2	L^∞ / L^1 / L^2	0.31 / 0.26 / 0.36	1.47 / 1.45 / 1.44	2
Protein q-gram	q = 6-18	Weighted edit distance	2.46q + 2.32	-0.08q + 4.16	q+1 (q<18), 17 (q=18)
DNA q-gram	q = 9-18	Hamming distance	1.27q + 0.37	0.14q + 2.52	q+1 (q<18), 21 (q=18)
Mass-spectra	40,000	Fuzzy cosine distance	0.62	1.23	2
Image	66	Linear combination of L-norms	5.26	4.85	5

6.3 Estimate of intrinsic dimension

Finally, we show results of estimated intrinsic dimension on all the workloads with the three methods. Wherever possible, we vary the domain dimension to see how the estimates change. If a linear relationship is observed, we compute the slope and intercept of the relationship by linear regression. For vector data, we use L^1 and L^∞ norms in addition to the L^2 norm. The estimates are shown in Table 3. First, we can see that as domain dimensions increase, estimates given by all the three methods increase linearly (one exception is the regression method and protein data). Therefore, we think all the three methods are asymptotically accurate. Methods 1 and 2 can be adjusted by constant factors to give accurate estimations for particular data and metric. However, they have different values for slope and intercept for different data and metrics. No generally applicable

7. CONCLUSIONS AND FUTURE WORK

The "power" of metric-space indexing is the generality of the abstraction and encapsulation; just provide a distance function. But, we (the community) have been powerless to do anything other than apply what we can make work with respect to R^n. To date, this has been done opportunistically. The theorems in this paper reveal that within certain boundaries we can in fact be methodical about this. We now even understand, for example, how to exploit PCA in this context. No surprise, the heuristic PCA inspired methods outperform heuristics inspired by ad-hoc observations.

We believe the objective function of incremental sampling is worthy but the associated algorithm is computationally expensive. It would be interesting to see if additional iterations of

incremental sampling, despite its cost, can achieve query performance similar to our PCA method. Further, incremental sampling might be made more efficient by sampling from a set of corners instead of the whole dataset. There are other dimension reduction methods for a vector space with annealing characteristics. We anticipate more research along this direction.

By virtue of considering the complete pivot space, our work has drawn a direct parallel between distance-based indexing and high-dimensional indexing. Both have to do dimension reduction and remove false positives. One may project a finite metric space to a reduced dimension coordinate system, but no proper subset of the pivots can faithfully recreate the geometry of the space. Thus, pivot selection results in information loss, typical of dimension reduction techniques. Dimension reduction is integral to indexing both finite metric-spaces and high-dimensional data.

We show that the intrinsic dimension of data in a metric space can be estimated by the intrinsic dimension of the complete pivot space. Thus, methods for vector spaces are now applicable to metric spaces. We have demonstrated how a PCA method can be applied to estimate the intrinsic dimension of the metric space. More work on this is expected.

Another interesting piece of future work is to determine the optimal number of pivots, and its relationship with the intrinsic dimension.

8. ACKNOWLEDGMENTS

We sincerely thank Gonzalo Navarro, Glen Nuckolls, and Piotr Indyk for their comments and suggestions.

This research was supported by a grant from the US National Science Foundation, DBI-0640923, a J. Tinsley Oden Faculty Research Fellowship at the Institute for Computational Engineering and Science, the University of Texas at Austin, USA, and the Initiating Fund for Member of the Chinese Academy of Sciences from Shenzhen University, China.

9. REFERENCES

[1] Bentley, J.L., *Multidimensional binary search trees used for associative searching.* Commun.ACM, 1975. **18**(9): p. 509-517.

[2] Beyer, K.S., J. Goldstein, R. Ramakrishnan, and U. Shaft. *When Is "Nearest Neighbor" Meaningful? the 7th International Conference on Database Theory*. 1999: Springer-Verlag.

[3] Bozkaya, T. and M. Ozsoyoglu, *Indexing large metric spaces for similarity search queries.* ACM Trans. Database Syst., 1999. **24**(3): p. 361-404.

[4] Brin, S. *Near Neighbor Search in Large Metric Spaces*. in *the 21th International Conference on Very Large Data Bases (VLDB'95)*. 1995: Morgan Kaufmann Publishers Inc.

[5] Bustos, B., G. Navarro, and E. Chavez, *Pivot selection techniques for proximity searching in metric spaces.* Pattern Recogn. Lett., 2003. **24**(14): p. 2357-2366.

[6] Camastra, F., *Data dimensionality estimation methods: a survey.* Pattern Recognition, 2003. **36**(12): p. 2945-2954.

[7] Chavez, E., G. Navarro, R. Baeza-Yates, and J. Marroqu, *Searching in metric spaces.* ACM Computing Surveys, 2001. **33**(3): p. 273-321.

[8] Ciaccia, P. and M. Patella. *Bulk loading the M-tree*. in *9th Australasian Database Conference (ADO'98)*. 1998.

[9] Clarkson K.L., Nearest-neighbor searching and metric space dimensions, In: Nearest-Neighbor Methods for Learning and Vision: Theory and Practice, MIT Press, 2006, pp. 15--59

[10] Guttman, A., *R-trees: a dynamic index structure for spatial searching*, in *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. 1984.

[11] Hjaltason, G.R. and H. Samet, *Index-driven similarity search in metric spaces.* ACM Transactions on Database Systems (TODS), 2003. **28**(4): p. 517-580.

[12] Hochbaum, D.S. and D.B. Shmoys, *A best possible heuristic for the k-center problem.* Mathematics of Operational Research, 1985. **10**(2): p. 180-184.

[13] Kegl, B., *Intrinsic dimension estimation using packing numbers.* Advances in Neural Information Processing Systems, 2003. **15**: p. 681-688.

[14] Mao, R., W. Xu, S. Ramakrishnan, G. Nuckolls, and D.P. Miranker. *On Optimizing Distance-Based Similarity Search for Biological Databases*. in *the 2005 IEEE Computational Systems Bioinformatics Conference (CSB 2005)*. 2005.

[15] Mao, R., W. Xu, N. Singh, and D.P. Miranker, *An Assessment of a Metric Space Database Index to Support Sequence Homology*. International Journal on Artificial Intelligence Tools (IJAIT), 2005: p. 867-885.

[16] Matousek, J., *Lectures on Discrete Geometry*. 2002: Springer-Verlag New York, Inc. 497.

[17] Test suite. http://aug.csres.utexas.edu/mobios-workload/.

[18] Navarro, G. *Searching in Metric Spaces by Spatial Approximation*. in *Proceedings of the String Processing and Information Retrieval Symposium & International Workshop on Groupware*. 1999: IEEE Computer Society.

[19] Roweis, S., *EM Algorithms for PCA and SPCA*. Neural Information Processing Systems 10 , 1997: p. 626-632.

[20] Samet, H., *Foundations of Multidimensional and Metric Data Structures*. 2006, Morgan-Kaufmann.

[21] Shaft, U. and R. Ramakrishnan. *When Is Nearest Neighbors Indexable?* in *Tenth International Conference on Database Theory (ICDT 2005)*. 2005: Springer

[22] Uhlmann, J.K., *Satisfying General Proximity/Similarity Queries with Metric Trees.* Information Processing Letter, 1991. **40**(4): p. 175-179.

[23] Weber, R., H.J. Schek, and S. Blott. *A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces*. in *International Conference on Very Large Data Bases*. 1998.

[24] Xu, W. and D.P. Miranker, *A Metric Model of Amino Acid Substitution.* Bioinformatics, 2004. **20**(8): p. 1214-1221.

[25] Yianilos, P.N. *Data structures and algorithms for nearest neighbor search in general metric spaces*. in *the fourth annual ACM-SIAM Symposium on Discrete algorithms*. 1993: Society for Industrial and Applied Mathematics.

On the Asymptotic Behavior of Nearest Neighbor Search Using Pivot-Based Indexes

Benjamin Bustos
PRISMA Research Group
Department of Computer Science
University of Chile
bebustos@dcc.uchile.cl

Nelson Morales
DELPHOS Lab
AMTC
University of Chile
nmorales@ing.uchile.cl

ABSTRACT

This paper presents an asymptotic analysis for the nearest neighbor search with pivot-based indexes. We extend a previous analysis based on range queries with fixed tolerance radius, because there is a probability that the nearest neighbor is missed. We introduce a probabilistic analysis and then we show the expected search cost for range-optimal algorithms. Finally, we also show the analysis of the proposed search algorithm taking into account the extra CPU time, which leads to further insights on the efficiency of different implementations of this algorithm.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content analysis and indexing—*indexing methods*

General Terms

Theory

Keywords

Nearest-neighbor search, pivot-based indexing, asymptotic analysis

1. INTRODUCTION

The concept of similarity search has applications in a vast number of fields. For example, content-based retrieval of similar objects in multimedia databases can be very useful for industrial applications, medicine, molecular biology, among others. Other applications for similarity search include machine learning and classification (where a new element must be classified according to its closest existing element); image quantization and compression (where only some vectors can be represented and those that cannot must be coded as their closest representable point); text retrieval (where we look for words in a text database allowing a small number of errors, or we look for documents which are similar to a given query or document); sequence comparison in computational biology (where we want to find a DNA or protein sequence in a database allowing some errors due to typical variations); etc.

All those applications have some common characteristics. There is a universe of *objects* and a *distance function* defined among them, which in many interesting cases satisfies the properties of a metric (strict positiveness, symmetry, and the triangle inequality). That is, the universe of objects together with the distance function form a *metric space*. The similarity between objects is given by their distance in the defined metric space: the smaller the distance between two objects, the more similar they are. Thus, for a given application there is a finite *dataset*, which is a subset of the universe of objects. Later, given a new object from the universe, one wants to retrieve similar elements found in the dataset.

An important similarity query type is the *nearest neighbor (NN) search*. In this type of query, one wants to retrieve from the dataset the most similar object to the query. Similarly, a k-NN search returns the k closest objects from the dataset to the query. Both queries can be easily done by a sequential scan over the dataset, computing the distance from the query to every object in the set, and returning the closest ones. However, this may be too expensive if the dataset is very large or if the distance function is expensive to compute. Thus, it is common to define the complexity of a similarity search in metric spaces as the number of distances computations required to answer the query.

For this reason, one usually preprocesses the dataset to build an index to avoid unnecessary distance computations at query time. For example, *pivot-based indexes* uses the distances between the objects from the dataset to some selected objects (the pivots) to discard objects from the similarity search without measuring the distance to the query (see Section 2.2 for details). In this way, the index can be used to return the nearest neighbor to the query, avoiding (hopefully) the sequential scan.

This paper presents a formal analysis of nearest neighbor search algorithms for pivot-based indexes. As noted by Navarro [13], asymptotic analysis is a very important tool that may lead us to better understand the behavior of index structures. We base our analysis on a range-optimal algorithm that works on pivot tables. Then, we extend our analysis by considering the extra CPU time (not expended in distance computations) required to perform the search, and we argue that this is necessary for making the analysis of NN search with pivot-based indexes meaningful.

SISAP'10, September 18-19, 2010, Istanbul, Turkey

Table 1: Notation used in this paper

$(\mathbb{U}, \delta)$	Metric space
$\mathbb{S} \subset \mathbb{U}$	Dataset
$n = \|\mathbb{S}\|$	Size of the dataset
$s, s_i \in \mathbb{S}$	An element of the dataset
$cost(\delta)$	Complexity of function $\delta(\cdot,\cdot)$
(q, r)	Range query with tolerance r
$\mathbb{P} \subset \mathbb{S}$	Set of pivots
$\theta \in P$	A pivot
$p = \|\mathbb{P}\|$	Number of selected pivots
$lb(q, s)$	Pivot-based lower bound distance
$\nu(r)$	Probability of not discarding an element in a (q, r) query
$f(x)$	Distance distribution of δ
$F(x) = \int_{-\infty}^{x} f(t)dt$	Cumulative distribution of $f(x)$
δ^*	Distance to the NN
$f^*(x)$	Distribution of δ^*
$F^*(x) = \int_{-\infty}^{x} f^*(t)dt$	Cumulative distribution of $f^*(x)$
$\pi = Pr(\delta^* > \mathbb{E}(\delta^*))$	Probability that δ^* is larger than its expected value

The contributions of this paper are:

- We improve a previous analysis of nearest neighbor search algorithms, to account the probability of failing to find the nearest neighbor.
- We present a framework for probabilistic analysis of nearest neighbor search, and we apply it to the case of uniform distance distribution.
- We present a nearest neighbor algorithm for pivot-based indexes, and we analyze it and prove that it is range-optimal.
- We take into account the extra CPU time in our analysis, that may be involved in the nearest neighbor search.

This paper is organized as follows. Section 2 presents the basics concepts of similarity search in metric spaces and explains the pivot-based indexing method. Section 3 present an analysis of pivot-based NN search algorithms that uses a fixed-search radius. Section 4 presents an introduction to probabilistic analysis for pivot-based NN search algorithms. Section 5 presents a range-optimal algorithms for pivot-based indexes and its analysis. Section 6 shows a model cost that includes the extra CPU time. Finally, Section 7 concludes the paper.

2. BASIC CONCEPTS

This section introduces the basic concepts on similarity search in metric spaces and pivot-based indexing. Table 1 shows the notation used in this paper.

2.1 Similarity search in metric spaces

Let $\mathbb{U}$ be the "universe" of valid objects, and let $\delta : \mathbb{U} \times \mathbb{U} \to \mathbb{R}^+$ be a metric distance function (i.e., it holds the properties of strict positiveness, symmetry, and the triangle inequality). The pair $(\mathbb{U}, \delta)$ is a metric space.

Let $\mathbb{S} \subset \mathbb{U}$ be a data collection (or dataset), and let $q \in \mathbb{U}$ be a query object. There are two typical similarity queries in metric spaces:

- *Range query.* Given a search radius $r \in \mathbb{R}^+$, the range query (q, r) reports all objects from the dataset that are within a distance r to q, that is, $(q, r) = \{s \in \mathbb{S}, \delta(s, q) \leq r\}$. The subspace $\mathbb{V} \subset \mathbb{U}$ defined by q and r (i.e., $\forall v \in \mathbb{V}\ \delta(v, q) \leq r$ and $\forall x \in \mathbb{X} - \mathbb{V}\ \delta(x, q) > r$) is called the *query ball*.
- *k nearest neighbors query (k-NN).* It reports the k closest objects from $\mathbb{S}$ to q. That is, it returns a set $\mathbb{C} \subseteq \mathbb{S}$ such that $|\mathbb{S}| = k$ and $\forall x \in \mathbb{C}, y \in \mathbb{S} - \mathbb{C}, \delta(x, q) \leq \delta(y, q)$. In this paper, we are interested in the case $k = 1$, which is known as the *nearest-neighbor* (NN) search.

Both types of similarity queries can be solved by performing a sequential scan of the dataset. However, this may be too slow for practical applications, where the dataset may contain millions of objects. Thus, efficient similarity search algorithms works on top of index structures (known as *metric access methods*), that try to discard objects from the dataset without computing their distance to q, thus avoiding the sequential scan. While there are several indexing techniques for metric spaces, in this paper we are specially interested in *pivot-based indexing techniques*.

2.2 Pivot-based indexing

Pivot-based indexes select a number of *pivot* objects from $\mathbb{S}$, and classify all the other objects according to their distance from the pivots. The canonical pivot-based range query algorithm works as follows: Given a range query (q, r) and a set of p pivots $\{\theta_1, \ldots, \theta_p\}, \theta_i \in \mathbb{S}$, by the triangle inequality it follows for any $u \in \mathbb{U}$ and $1 \leq i \leq p$ that $\delta(q, u) \geq |\delta(\theta_i, u) - \delta(\theta_i, q)|$. The objects $s \in \mathbb{S}$ of interest are those that satisfy $\delta(q, s) \leq r$, so one can exclude all the objects that satisfy $|\delta(\theta_i, s) - \delta(\theta_i, q)| > r$ for some pivot θ_i, without actually evaluating $\delta(q, s)$. This discarding criterion is known as the *pivot exclusion condition*.

The pivot-based index consists of the pn precomputed distances $\delta(\theta_i, s)$ between every pivot and every object of the data collection. At query time, the search algorithm computes the p distances between the pivots and q, $\delta(\theta_i, q)$. Those distance calculations correspond the *internal complexity* of the algorithm, and this complexity is fixed if there is a fixed number of pivots. Then, it tries to discard objects by applying the exclusion condition. The list of objects $\{s_1, \ldots, s_m\} \subseteq \mathbb{S}$ that cannot be discarded, known as the *object candidate list*, must be checked directly against the query. These additional distance calculations $\delta(s_i, q)$ correspond to the *external complexity* of the algorithm. The total complexity of the search is the sum of the internal and the external complexities.

Examples of pivot-based indexes are *Burkhard-Keller Tree* [5], *Fixed-Queries Tree* (FQT) [1], *Fixed-Height FQT* [1], *Fixed Queries Array* [8], *Vantage Point Tree* [16], *Multi Vantage Point Tree* [4], *Excluded Middle Vantage Point Forest* [17], *AESA* [15], *Linear AESA* [12], *Spaghettis* [7], *Maximum Pruning* [14], and *Dynamic SSS* [6].

NN search algorithms with pivot-based indexes have usually been built over the range query algorithm. These techniques include [9]:

- *Increasing radius*: the search performs several range queries with increasing value of r, until at least one object lies inside the ball (q, r). If more than one object is found, the closest one to q is the NN.
- *Backtracking with decreasing radius*: the search starts with $r = \infty$, and it reduces r every time it computes a distance $\delta(q, s)$ that is smaller than r, keeping the closest object to q found so far. The search finishes when all objects have been checked.

2.3 Standard cost model

The total time for answering similarity queries in metric spaces is defined as [9]

$$\#\ dist.\ comp. \times cost(\delta) + \ extra\ CPU\ time\ +\ \mathrm{I}/O\ time,$$

where $cost(\delta)$ is the complexity of function $\delta(\cdot, \cdot)$. It has been argued [9, 18] that the most important cost in a similarity search is the distance computations, thus the cost model is usually simplified to

$$\#\ dist.\ comp.$$

See Navarro [13] for a survey on formal analyses of indexing techniques and algorithms for similarity search in metric spaces.

3. ANALYSIS OF NN SEARCH USING FIXED-RANGE QUERIES

In this section, we revisit the analysis of NN search algorithms based on range queries with fixed radius, and we extend it. Let $\delta^*(q)$ be the distance to the nearest neighbor of q (from now on, let us write δ^* for short). A common way of studying the cost of search algorithms for 1-NN, is to take into account that if δ^* was known, then 1-NN reduces to the range query (q, δ^*), therefore the expected cost of a 1-NN search is that of a range query $(q, \mathbb{E}(\delta^*))$ [2]. However, such analysis does not consider that the cost of a query search is not linear in the radius, hence the expected cost of using a query search with radius δ^* is not the same as the cost of a query search with radius $\mathbb{E}(\delta^*)$. It also does not consider that searching with fixed radius $\mathbb{E}(\delta^*)$ may not find the nearest neighbor.

3.1 Range based Nearest-Neighbor Queries

Given a range-optimal algorithm (see Section 5) for finding the NN in multidimensional indexes, Böhm [2] suggests reducing its analysis to the analysis of a range query using a tolerance radius for the range query equal to the expected value $\mathbb{E}(\delta^*)$ of the distance to the nearest neighbor. Concretely, following the analysis by Navarro [13], we have that if the distances between the elements of the database and from these to the query are independent and identically distributed, and if we call T_R the cost of such an algorithm, we have that

$$\mathbb{E}(T_R) = p + n\,\nu(\mathbb{E}(\delta^*))^p, \tag{1}$$

where p is the number of pivots and $\nu(\cdot)$ is the probability of not discarding an object using the p pivots.

While this analysis may produce a good approximation of the expected cost of the algorithm, we observe that it does not consider the fact that any NN search algorithm resorting to use a range query with fixed value of r_0 will fail (return no elements) with probability $Pr(\delta^* > r_0)$. In particular, the procedure above will require to perform at least one more distance calculation with probability $\pi = Pr(\delta^* > \mathbb{E}(\delta^*))$.

As using a fixed radius r_0 (like $\mathbb{E}(\delta^*)$) may produce no results, the only way to guarantee that the nearest neighbor will be found is to update the query radius. Some of the possibilities that we visualize are:

- Doing a linear search in the case of failure. This has expected cost $p + n\big(\nu(\mathbb{E}(\delta^*)^p)\big) + \pi n$, therefore being of order n.
- Multiplying r_0 by some factor greater than one, do the search and continue updating or stopping depending on the results of the search.
- Updating r by setting $r = \mathbb{E}\big(\delta^* | \delta^* > \mathbb{E}(\delta^*)\big)$, and update consequently depending on the search result.

Note that any of these schemes (and any algorithm performing a range query with radius $\mathbb{E}(\delta)$) will have an expected cost that is strictly larger than the one from Eq. (1).

Notice also that, in particular, when the distances are not bounded (i.e., the distance distribution has an infinite support set), we have that any search algorithm using a finite query radius r_0 is not exact, as it will fail with probability $Pr(\delta^* > r_0)$.

4. PROBABILISTIC ANALYSIS

This section aims to present a brief introduction to the probabilistic analysis that would be required in order to properly understand the cost of NN algorithms for pivot-based indexes. We present some basic calculations and then apply them to the specific case of the uniform distribution.

4.1 Distribution and expected values of δ^*

We need first to calculate the probability density of δ^*, that is the distance to the nearest neighbor, when the pivots are chosen randomly and independently.

As the pivots are chosen randomly, the distance distribution to any pivot is $f(x)$ (the distance distribution to the objects in the database), and since $\delta^*(q) = \min_{i=1}^{n} \delta(q, s_i)$, we have that $\delta^*(q) \geq z \iff \delta(q, s_i) \geq z$ for any $i = 1, \ldots, n$. Using that $Pr(\delta(q, s_i) \geq z) = 1 - F(z)$, and independence, it follows that $Pr\big(\delta^* \geq z\big) = (1 - F(z))^n$. Therefore, $Pr(\delta^* \leq z) = F^*(z) = 1 - (1 - F(z))^n$. That is, $\delta^* \sim f^*(z)$ with

$$f^*(z) = n f(z)(1 - F(z))^{n-1}, \tag{2}$$

and

$$\mathbb{E}\big(\delta^*\big) \;=\; \int_{\infty}^{s} z n f(z)(1 - F(z))^{n-1} dz. \tag{3}$$

Notice that in the general case of k-NN, we have that

$$Pr\big(\delta^*(x) \geq z\big) = \binom{n}{k} F(z)^{k-1} (1 - F(z))^{n-k+1}.$$

Finally, recall that the probability of not discarding an element when performing a search with radius r is (see Navarro [13]):

$$\nu(r) = \int_0^1 f(s)\,[F(s+r) - F(s-r)]\,ds.$$

4.2 Uniform distribution case

When the distance δ follows a uniform distribution in $[0,1]$, $f(x) = 1, F(x) = x$ (within [0,1]) and therefore

$$f^*(z) = n(1-z)^{n-1},\ F^*(z) = 1 - (1-z)^n$$

and, integrating by parts

$$\mathbb{E}(\delta^*) \quad = \quad \int_0^1 zn(1-z)^{n-1}dz = \frac{1}{n+1}.$$

Also $\pi = Pr\left(\delta^* > \frac{1}{n+1}\right) = \left(\frac{n}{n+1}\right)^n \rightarrow \frac{1}{e}$ as n grows. This means that in about 1/3 of the opportunities that the algorithm is used, it will fail to find the nearest neighbor and therefore at least one additional range query will be needed.

We also obtain that $\nu(r) = r(2-r)$.

As $\nu(\cdot)$ is a concave positive function and $p \geq 1$, we have that $\mathbb{E}(\nu(\delta^*)) \leq \nu(\mathbb{E}(\delta^*))$. It follows that as an algorithm using $\mathbb{E}(\delta^*)$ as a search radius will have an expected cost $\mathbb{E}(T_R) \geq p + n\nu\left(\mathbb{E}(\delta^*)\right)^p$. We conclude that

$$\mathbb{E}(T_R) \geq p + n\nu\left(\mathbb{E}(\delta^*)\right)^p \geq p + n\mathbb{E}(\nu(\delta^*))^p \qquad (4)$$

The right side of Eq. (4) is actually the expected cost of an algorithm that *knows* δ^* and performs one range query with that radius, which is the definition of a range-optimal search algorithm.

5. RANGE-OPTIMAL ALGORITHMS

A NN search algorithm is *range-optimal* if it computes the same number of distances than the canonical range query algorithm using $r = \delta^*$ as search radius. This is an interesting property, because it means that for such an algorithm there is no inherent advantage of having or estimating δ^* a priori (without knowing the object that is actually the NN) to perform the search. An example of a range-optimal algorithm is the incremental search algorithm by Hjaltason and Samet [11], that works on hierarchical index structures. Berchtold et al. [3] proved that this algorithm is range-optimal.

The algorithms for NN search mentioned in Section 2.2 and Section 3 are not range-optimal. So, we now describe a NN search algorithm that uses the pivot-based index, and we prove that this algorithm is range-optimal.

The algorithm starts by computing the distances between all pivots and the query object q. The pivot whose distance to q is minimum (*mindist*) is the first NN candidate. Then, the algorithm computes the lower-bound distances from q to all objects $s \in \mathbb{S}$ (excluding the pivots). A lower-bound distance is computed as $lb(q,s) = \max_{i=1}^{p} |\delta(\theta_i, s) - \delta(\theta_i, q)|$. Next, the algorithm sorts the objects $s \in \mathbb{S}$ in ascending order according to their lower-bound distances to q. Starting with the the object u with smallest lower bound distance, the algorithm applies the pivot exclusion criterion using as tolerance radius the distance from the candidate NN to the query object. If s cannot be discarded, it computes the distance between s and q. If this distance is smaller than *mindist*, it sets u as the new NN candidate and updates *mindist*. The process ends when all the objects from $\mathbb{S}$ have been checked or if the lower bound distance of the next object in the list is greater than *mindist* (i.e., no other object can be closer to q than the actual NN candidate). Algorithm 1 shows the pseudocode for this algorithm.

Algorithm 1 Range-optimal pivot-based NN search algorithm

Require: $q \in \mathbb{U}$
Ensure: NN is the nearest neighbor of q in $\mathbb{S}$
1: $mindist \leftarrow \min_{i=1}^{p}\{\delta(\theta_i, q)\}$
2: $NN \leftarrow \theta_{\arg\min_{i=1}^{p}\{\delta(\theta_i,q)\}}$
{Computing lower bound distances}
3: **for all** $s_i \in \mathbb{S} - \mathbb{P}$ **do**
4: $lb(q, s_i) \leftarrow 0$
5: **for all** $\theta_j \in \mathbb{P}$ **do**
6: **if** $|\delta(s_i, \theta_j) - \delta(\theta_j, q)| > lb(q, s_i)$ **then**
7: $lb(q, s_i) \leftarrow |\delta(s_i, \theta_j) - \delta(\theta_j, q)|$
8: **end if**
9: **end for**
10: **end for**
{Sorting objects in $\mathbb{S} - \mathbb{P}$ by ascending lower bound}
11: $\mathbb{S}' \leftarrow Sort(\mathbb{S} - \mathbb{P})$
{Searching for NN}
12: **for** $i = 1$ to $n - p$ **do**
13: **if** $lb(q, s'_i) > mindist$ **then**
14: break
15: **end if**
16: **if** $\delta(q, s'_i) < mindist$ **then**
17: $mindist \leftarrow \delta(q, s'_i)$
18: $NN \leftarrow s'_i$
19: **end if**
20: **end for**
21: **return** NN

We now prove that Algorithm 1 is range-optimal.

LEMMA 5.1. *Algorithm 1 returns the correct NN.*

PROOF. By contradiction, suppose that for a query object q the algorithm returns an object $s \in \mathbb{S}$ such that $\delta(q,s) > \delta^*$, and let s^* be the real NN. This means that the algorithm stopped checking objects (lines 13 and 14 of the pseudocode) before finding s^*. Thus, $lb(q, s^*) > \delta^*$ because the algorithm checks the objects in ascending lower bound distance and the condition on line 13 had to be true before reaching s^*. But $lb(q, s^*) \leq \delta^*$, because $lb(q, s^*)$ is a lower bound distance of δ, which leads to the contradiction. □

LEMMA 5.2. *Algorithm 1 does not compute any distance from q to an object $s \in \mathbb{S}$ such that $lb(q,s) > \delta^*(q)$*

PROOF. The algorithm sorts the objects $s \in \mathbb{S} - \mathbb{P}$ by lower bound distance (line 11 of the pseudocode), it checks the objects in that order, and it stops the search as soon as $lb(q,s) > mindist = \delta^*$ (by Lemma 5.1). Thus, the algorithm only computes distances for those s such that $lb(q,s) \leq \delta^*$. □

THEOREM 5.3. *Algorithm 1 is range-optimal.*

PROOF. It follows directly from Lemmas 5.1 and 5.2. □

Notice that because a range-optimal search algorithm is equivalent to one doing a range search with radius $r = \delta^*$, we obtain that the cost of this algorithm is $T = p + n\nu(\delta^*)^p$ (with distribution f^* of δ^*), and its expected cost is the one given by Eq. (4). Therefore, if $\nu(\cdot)$ is concave (as we showed for uniform distance distribution), Algorithm 1 is more efficient than any possible strategy that uses a fixed-search radius for finding the nearest neighbor.

6. EXTENDED MODEL COST

In the last section, we showed that range-optimal algorithms computes the minimum number of distances (given a fixed set of pivots) needed to find the correct nearest neighbor. Thus, under the standard cost model for searching in metric spaces, any range-optimal algorithm is equivalent. However, we will show that this is far from true, and that it is important to consider the extra CPU time to obtain further insights from the theoretical analysis.

6.1 A CPU-expensive range-optimal NN algorithm

The algorithm we devise is very simple: We will determine the value of δ^* so we can perform a range query with radius $r = \delta^*$ and retrieve the nearest neighbor. The trick in the algorithm is that we will determine δ^* at cost zero and therefore the expected cost of such an algorithm will be the same of Eq. (4):

$$\mathbb{E}(T_R) = p + n\mathbb{E}\left(\nu(\delta^*)^p\right).$$

To determine δ^*, we use the following procedure. Let q be the query object. We start evaluating the distances $d_i = \delta(q, \theta_i)$ for $i = 1, \ldots, p$ and constructing the lower bounds $lb(q, s_j) = \max_i\{|d_i - \delta(\theta_i, s_j)|\}$. Without loss of generality, let us assume these values are ordered such that $j < \ell \Rightarrow lb(q, s_j) \leq lb(q, s_\ell)$.

During the search procedure, we will have a nearest neighbor candidate s_c (the first one being the closest pivot), and a search radius d such that $\delta(q, s_c) \geq d$ (the first one being $d = 0$).

While $\delta(q, s_c) > d$ we increase d, each time by ϵ (ϵ of the machine) until:

1. either $lb(q, s_j) = d$ for some element s_j, in which case evaluate $\delta(q, s_j)$ and we update our candidate $s_c := s_j$ if $\delta(q, s_j) < \delta(q, s_c)$, and continue to iterate,
2. either $\delta(q, s_c) = d$, in which case we can stop.

The only distances evaluated by this algorithm are those of elements such that $lb(q, s_j) < \delta^*$, therefore it is range-optimal. Thus, if one only considers distance computations for analyzing this algorithm, it is equivalent in complexity to Algorithm 1. Indeed, we observe that the algorithm described above strongly relies on the fact that the only cost involved in search algorithms is the one of evaluating the distance function while any other calculations are negligible. This is not very realistic: even if the distance computation is expensive, the proposed schema performs a huge amount of calculations (not involving distances), making the asymptotic analysis ineffective.

We now consider the extra CPU time for analyzing the described range-optimal algorithm. Then, we propose two variants of this algorithm, the last one being at least as efficient as Algorithm 1.

6.2 Analyzing the range-optimal algorithm

Considering the extra CPU time, the cost of Algorithm 1 is the cost of computing the lower bounds distances (p distances plus $O(n)$ extra CPU time), the cost of sorting the lower bounds ($O(n \log n)$), and the distance computations for the non-discarded objects. Its expected cost is

$$\mathbb{E}(T) = \left(p + n\mathbb{E}\left(\nu(\delta^*)^p\right)\right) \cdot cost(\delta) + O(n \log n) + n\mathbb{E}\left(\nu(\delta^*)^p\right). \quad (5)$$

This means that if $cost(\delta) = o(\log n)$ (reasonable for many practical applications), the most expensive part of the algorithm is the sorting instruction. Now, the question is how to reduce the extra CPU time.

6.3 CPU-efficient range-optimal algorithms

6.3.1 Using linear selection

One alternative to reduce the CPU time is to replace the sort instruction in Algorithm 1 by a linear selection instruction (using the selection in worst-case linear time algorithm, see Cormen et al. [10], Chapter 9.3) on each iteration of the last for cycle. The pseudocode is presented in Algorithm 2.

Algorithm 2 Range-optimal pivot-based NN search algorithm with linear selection

Require: $q \in \mathbb{U}$
Ensure: NN is the nearest neighbor of q in $\mathbb{S}$
1: $mindist \leftarrow \min_{i=1}^{p}\{\delta(\theta_i, q)\}$
2: $NN \leftarrow \theta_{\arg\min_{i=1}^{p}\{\delta(\theta_i,q)\}}$
{Computing lower bound distances}
3: **for all** $s_i \in \mathbb{S} - \mathbb{P}$ **do**
4: $lb(q, s_i) \leftarrow 0$
5: **for all** $\theta_j \in \mathbb{P}$ **do**
6: **if** $|\delta(s_i, \theta_j) - \delta(\theta_j, q)| > lb(q, s_i)$ **then**
7: $lb(q, s_i) \leftarrow |\delta(s_i, \theta_j) - \delta(\theta_j, q)|$
8: **end if**
9: **end for**
10: **end for**
{Searching for NN}
11: **for** $i = 1$ to $n - p$ **do**
12: $s' \leftarrow Select(i, \mathbb{S} - \mathbb{P})$
13: **if** $lb(q, s') > mindist$ **then**
14: break
15: **end if**
16: **if** $\delta(q, s') < mindist$ **then**
17: $mindist \leftarrow \delta(q, s')$
18: $NN \leftarrow s'$
19: **end if**
20: **end for**
21: **return** NN

The complexity of Algorithm 2 is the cost of computing the lower bounds distances (p distances plus $O(n)$ CPU extra time), the cost of the selection ($O(n)$ on each iteration of the last for cycle), and the distance computations for the non-discarded objects. Its expected cost is

$$\mathbb{E}(T) = \left(p + n\mathbb{E}\left(\nu(\delta^*)^p\right)\right) \cdot cost(\delta) + O(n) + O(n) \cdot n\mathbb{E}\left(\nu(\delta^*)^p\right). \quad (6)$$

Therefore, if $n\mathbb{E}\left(\nu(\delta^*)^p\right) = o(\log n)$, Algorithm 2 is more efficient than Algorithm 1. However, it has a worst case of $O(n^2)$ extra CPU time.

6.3.2 Using a heap

A better alternative than using linear selection is to use a heap to check the objects in order, according to their lower bound distance. Note that the heap does not require extra space, as it can be build over the array that stores the lower bound distances. The pseudocode is showed in Algorithm 3.

Algorithm 3 Range-optimal pivot-based NN search algorithm with a heap

Require: $q \in \mathbb{U}$
Ensure: NN is the nearest neighbor of q in $\mathbb{S}$
1: $mindist \leftarrow \min_{i=1}^{p}\{\delta(\theta_i, q)\}$
2: $NN \leftarrow \theta_{\arg\min_{i=1}^{p}\{\delta(\theta_i,q)\}}$
{Computing lower bound distances}
3: **for all** $s_i \in \mathbb{S} - \mathbb{P}$ **do**
4: $lb(q, s_i) \leftarrow 0$
5: **for all** $\theta_j \in \mathbb{P}$ **do**
6: **if** $|\delta(s_i, \theta_j) - \delta(\theta_j, q)| > lb(q, s_i)$ **then**
7: $lb(q, s_i) \leftarrow |\delta(s_i, \theta_j) - \delta(\theta_j, q)|$
8: **end if**
9: **end for**
10: **end for**
{Converting list of objects in $\mathbb{S} - \mathbb{P}$ in a heap}
11: $\mathbb{S}' \leftarrow MinHeapify(\mathbb{S} - \mathbb{P})$
{Searching for NN}
12: **for** $i = 1$ to $n - p$ **do**
13: $s' \leftarrow HeapExtractMin(S')$
14: **if** $lb(q, s') > mindist$ **then**
15: break
16: **end if**
17: **if** $\delta(q, s') < mindist$ **then**
18: $mindist \leftarrow \delta(q, s')$
19: $NN \leftarrow s'$
20: **end if**
21: **end for**
22: **return** NN

The complexity of Algorithm 3 is the cost of computing the lower bounds distances (p distances plus $O(n)$ CPU extra time), the cost of MinHeapify ($O(n)$), the cost of HeapExtractMin on each iteration of the last for cycle ($O(\log n)$), and the distance computations for the non-discarded objects. Its expected cost is:

$$\mathbb{E}(T) = (p+n\mathbb{E}\big(\nu(\delta^*)^p\big))\cdot cost(\delta)+O(n)+O(\log n)\cdot n\mathbb{E}\big(\nu(\delta^*)^p\big) \quad (7)$$

If $n\mathbb{E}\big(\nu(\delta^*)^p\big) = o(n)$, Algorithm 3 is always more efficient than Algorithm 1 (and it is clearly superior to Algorithm 2). In the worst case, both Algorithms 1 and 3 have the same time complexity.

7. CONCLUSIONS

This paper presented a formal analysis of nearest neighbor search algorithms for pivot-based indexes. We presented a brief probabilistic analysis, and then showed the expected cost for range-optimal search algorithms. We also showed that algorithms based on a fixed-search radius have a probability to fail returning the nearest neighbor, and that range-optimal algorithms are always more efficient if the probability of not discarding an object is a concave function (e.g., in the case of uniform distance distribution). Additionally, we extended our analysis by considering the extra CPU time, and showed that this is important to be able to distinguish between different range-optimal algorithms.

The analysis that considers the extra CPU time showed that the proposed range-optimal algorithms have a complexity of at least $O(n)$, because all of them must compute the lower bound distances for all objects in the dataset. Thus, if the distance function is not expensive (such as the Minkowski distances, which are widely used in vector spaces and are $O(d)$, with d the dimensionality of the space), the extra CPU time cost cannot be neglected. An interesting question that this analysis raises is if there is a range-optimal algorithm for pivot-based indexes that has $O(n^\alpha)$ extra CPU time, with $0 < \alpha < 1$.

Further work could consider extending the analysis of the nearest neighbor search to other indexing schema, like the ones based on partitioning the spaces into zones, and to include the I/O cost in the cost model (for indexes stored in secondary memory). We also observe that while we applied the probabilistic framework to the analysis of the uniform distribution, it would be interesting to do the same for some other distributions. In fact, we consider that determining theoretical distance distributions that are reasonable for the analysis is an interesting topic itself.

8. REFERENCES

[1] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Proc. 5th Combinatorial Pattern Matching (CPM'94)*, LNCS 807, pages 198–212, 1994.

[2] C. Böhm. A cost model for query processing in high dimensional data spaces. *ACM Transsactions on Database Systems*, 25(2):129–178, 2000.

[3] C. Böhm, S. Berchtold, and D. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, 2001.

[4] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proc. ACM International Conference on Management of Data (SIGMOD'97)*, pages 357–368, 1997. Sigmod Record 26(2).

[5] W. Burkhard and R. Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236, 1973.

[6] B. Bustos, O. Pedreira, and N. Brisaboa. A dynamic pivot selection technique for similarity search. In *Proc. 1st International Workshop on Similarity Search and Applications (SISAP'08)*, pages 105–112, 2008.

[7] E. Chávez, J. Marroquín, and R. Baeza-Yates. Spaghettis: an array based algorithm for similarity queries in metric spaces. In *Proc. String Processing and Information Retrieval (SPIRE'99)*, pages 38–46. IEEE CS, 1999.

[8] E. Chávez, J. Marroquín, and G. Navarro. Fixed queries array: A fast and economical data structure for proximity searching. *Multimedia Tools and Applications (MTAP)*, 14(2):113–135, 2001.

[9] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and

C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.

[11] G. Hjaltason and H. Samet. Ranking in spatial databases. In *Proc. 4th International Symposium on Advances in Spatial Databases (SSD'95)*, LNCS 951, pages 83–95. Springer, 1995.

[12] L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbor approximating and eliminating search (AESA) with linear preprocessing-time and memory requirements. *Pattern Recognition Letters*, 15:9–17, 1994.

[13] G. Navarro. Analyzing metric space indexes: What for? In *Proc. 2nd International Workshop on Similarity Search and Applications (SISAP'09)*, pages 3–10. IEEE CS Press, 2009. Invited paper.

[14] J. Venkateswaran, T. Kahveci, C. M. Jermaine, and D. Lachwani. Reference-based indexing for metric spaces with costly distance measures. *VLDB Journal*, 17(5):1231–1251, 2008.

[15] E. Vidal. An algorithm for finding nearest neighbors in (approximately) constant average time. *Pattern Recognition Letters*, 4:145–157, 1986.

[16] P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, pages 311–321, 1993.

[17] P. Yianilos. Excluded middle vantage point forests for nearest neighbor search. In *DIMACS Implementation Challenge, ALENEX'99*, Baltimore, MD, 1999.

[18] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach (Advances in Database Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

Enlarging Nodes to Improve Dynamic Spatial Approximation Trees *

Marcelo Barroso
Departamento de Informática
Universidad Nacional de San Luis
Argentina
mabarros@unsl.edu.ar

Nora Reyes
Departamento de Informática
Universidad Nacional de San Luis
Argentina
nreyes@unsl.edu.ar

Rodrigo Paredes
Departamento de Ciencias de la Computación
Universidad de Talca
Curicó, Chile
raparede@utalca.cl

ABSTRACT

The *metric space model* allows abstracting many similarity search problems. Similarity search has multiple applications especially in the multimedia databases area. The idea is to index the database so as to accelerate similarity queries. Although there are several promising indices, few of them are dynamic, i.e., once created very few allow to perform insertions and deletions of elements at a reasonable cost.

The *Dynamic Spatial Approximation Trees* (*DSA–trees*) have shown to be a suitable data structure for searching high dimensional metric spaces or queries with low selectivity (i.e., large radius), and are also completely dynamic. The performance of *DSA–trees* is directly related to the amount of backtracking in search time. To boost the performance in this data structure a sufficient condition is to maintain in the nodes elements close-to-each-other. In this work we propose to obtain a new data structure for searching in metric spaces, based on the *DSA–trees*, which holds its virtues and takes advantage of element clusters, which are present in many metric spaces, and can also make better use of available memory to improve searches. In fact, we use these element clusters to improve the spatial approximation.

Categories and Subject Descriptors

H.3.1 [**Information storage and retrieval**]: Content analysis and indexing—*indexing methods*; H.3.3 [**Information storage and retrieval**]: Information Search and Retrieval —*search process*

General Terms

Algorithms, Experimentation

*Partially Supported by PPUA Project 01-10-269 and Fondo de Desarrollo de Investigación, Universidad de Talca.

SISAP 2010, September 18-19, 2010, Istanbul, Turkey.

Keywords

Metric space searching, DSA–trees

1. INTRODUCTION

Similarity search is a suitable way to find in a database any kind of unstructured data and has applications in a vast number of fields. Some examples are next generation databases (e.g. storing images, fingerprints or audio clips), text searching, information retrieval, machine learning and classification, image quantization and compression, computational biology, and function prediction. These applications have some common characteristics, where one of the main ones is that the concept of exact search is of no use and we search instead for similar objects.

The similarity search problem can be formally defined through the concept of *metric space*, which provides a formal framework that is independent of the application domain. There is a universe U of objects and a nonnegative *distance function* $d : U \times U \longrightarrow \mathbb{R}^+$ defined among them. This distance satisfies the three axioms that make the set a *metric space*: *strict positiveness* ($d(x, y) = 0 \Leftrightarrow x = y$), *symmetry* ($d(x, y) = d(y, x)$) and *triangle inequality* ($d(x, z) \Rightarrow d(x, y) + d(y, z)$). The smaller the distance between two objects, the more "similar" they are. We have a finite database $S \subseteq U$, which is a subset of the universe of objects and can be preprocessed (to build an index, for example). Later, given an object from the universe (a query q), we must retrieve all similar elements in the database. There are two typical queries of this kind:

Range query: Retrieve all elements within distance r to q in S. This is, the set $\{x \in S, d(x, q) \leq r\}$.

Nearest neighbor query (k-NN): Retrieve the k closest elements to $q \in S$. That is, a set $A \subseteq S$ such that $|A| = k$ and $\forall\, x \in A, y \in S - A, d(x, q) \leq d(y, q)$.

In this paper we are devoted to range queries. Nearest neighbor queries can be rewritten as range queries in an optimal way [8], so we can restrict our attention to range queries. The distance is considered expensive to compute (think, for instance, in comparing two fingerprints). Hence, it is customary to define the complexity of the search as the number of distance evaluations performed, disregarding other components such as CPU time for side computations, and even I/O time.

Since, the computational cost of determining the similarity among objects is known to be a significant part of the total running time, a number of data structures and algorithms

have been devised to deal efficiently with large collections of data [13, 12, 5]. Given a database of $|S| = n$ objects, queries can be trivially answered by performing n distance evaluations. The goal is to build an index of the database to speed up queries, avoiding the exhaustive search and computing a minimal amount of distances. All those structures work on the basis of discarding elements using the triangle inequality, and most of them use the classical divide-and-conquer approach (which is a general algorithmic approach).

There are effective methods to search on D–dimensional spaces considering that the distance function belongs to the Minkowski's distance function family $L_p = (\sum_{1 \le i \le d} |x_i - y_i|^p)^{1/p}$, such as kd–trees [2, 3] but for roughly 20 dimensions or more those structures cease to work well. For a survey on these methods see [7]. We focus in this paper in general metric spaces, although the solutions are well suited also for D–dimensional spaces. It is interesting to notice that the concept of "dimensionality" is related to "easiness" or "hardness" of searching a D–dimensional space: higher dimensional spaces have a probability distribution of distances among elements whose histogram is more concentrated and with larger mean. This makes the work of any similarity search algorithm more difficult. We extend this idea, following [5], by saying that a general metric space is high dimensional when its histogram of distances is concentrated.

Among all the techniques for metric space indexing we are interested in the *dynamic* data structures, where the database is unknown beforehand and the objects arrive to the index at random times as well as the queries. Static data structures may benefit from the *full knowledge* of the database to select the best reference points for a particular data structure. A dynamic data structure cannot make such strong assumptions about the database and will not have statistics about all the database.

The *Dynamic Spatial Approximation Tree* (*DSA–tree*) is a recently proposed data structure for searching in metric spaces [11], based on a novel concept: rather than dividing the search space, approach the query spatially, that is, start at some point in the space and get closer and closer to the query [10]. The *DSA–tree* behaves better than the other existing data structures on metric spaces of high dimension or queries with low selectivity, which is the case in many applications. This index is fully dynamic and is incrementally built via insertions. As such, the tree root will be the first object arriving, and this is repeated recursively at every level in the tree. The *DSA–tree* supports insertion and deletion of elements and has proven to be competitive in all dimensionalities but unable of taking advantage of the available memory.

Unlike some other metric data structures [5, 4], the (*DSA–tree*) does not take advantage if the metric space has clusters, or can improve the search at the expense of using more memory. Our proposal, based on *DSA–tree* is still dynamic, competitive, and makes better use of the available memory. If the *DSA–tree* grouped objects that are very close to each other it could achieve better search performance by having to do less backtracking. We propose a new data structure that performs the spatial approximation on groups of objects or *clusters*, rather than individual objects, and thus reduces search costs.

2. PREVIOUS WORK

Algorithms to search in general metric spaces can be divided into two large areas: *pivot-based* algorithms and *compact partition-based* ones. Pivot-based algorithms are better suited for low dimensional metric spaces, while compact partitions ones deal better with high dimensional metric spaces. Although the former can improve by using more memory, they need more and more memory to beat the latter as dimension grows. On the other hand, indices based on compact partitions use a fixed amount of memory and cannot be improved by giving them more space. However, there are algorithms that combine ideas from both areas. See [12, 13, 5, 9] for more complete surveys.

Pivot-Based Algorithms.

The idea is to use a set of k distinguished elements ("pivots") $p_1 \dots p_k \in S$ and storing, for each dataset element x, its distance to the k pivots $(d(x, p_1) \dots d(x, p_k))$. Later, given the query q, its distance to the k pivots is computed $(d(q, p_1) \dots d(q, p_k))$. Now, if for some pivot p_i it holds that $|d(q, p_i) - d(x, p_i)| > r$, then we know by the triangle inequality that $d(q, x) > r$ and therefore do not need to explicitly evaluate $d(x, p)$. All the other elements that cannot be discarded using this rule are directly compared with the query.

Clustering Algorithms.

This second trend consists of dividing the space into zones as compact as possible, usually in a recursive fashion, and storing a representative point ("center") for each zone plus a few extra data that permit quickly discarding the zone at query time. Two criteria can be used to delimit a zone. The first one is the *Voronoi region*, where we select a set of centers and put every other point inside the zone of its closest center. The regions are bounded by hyperplanes and the zones are analogous to Voronoi regions in vector spaces. Let $\{c_1 \dots c_m\}$ be the set of centers. At query time we evaluate $(d(q, c_1), \dots, d(q, c_m))$, choose the closest center c and discard every zone whose center c_i satisfies $d(q, c_i) > d(q, c) + 2r$, as its Voronoi area cannot intersect the query ball. The second criterion is the *covering radius* $cr(c_i)$, which is the maximum distance between c_i and an element in its zone. If $d(q, c_i) - r > cr(c_i)$, then there is no need to consider zone i. The two criteria can be combined.

Combining Clustering with Pivots.

There are some indices that combine both ideas by dividing the space into compact zones and, at the same time, storing distances to some distinguished elements (pivots) [1].

3. DYNAMIC SPATIAL APPROXIMATION TREES

In this section we briefly describe *DSA–trees*, in particular the version called *timestamp with bounded arity* (reported in [11] as one of the better options for this dynamic tree), on top of which we build our approach.

3.1 Insertion Algorithm

The *DSA–tree* is built incrementally via insertions. The tree has a maximum arity A. Each tree node a stores a *timestamp* of its insertion time, $time(a)$, its *covering radius*,

Algorithm 1 Insertion of a new element x into a *DSA–tree* with root a using timestamping and bounded arity.

```
Insert(Node a, Element x)

1. R(a) ← max(R(a), d(a,x))
2. c ← argmin_{b∈N(a)} d(b,x)
3. If d(a,x) < d(c,x) ∧ |N(a)| < MaxArity Then
4.    N(a) ← N(a) : x
5.    N(x) ← ⟨⟩, R(x) ← 0
6.    T(x) ← CurrentTime
7.    CurrentTime ← CurrentTime + 1
8. Else Insert(c,x)
```

$R(a)$, and its set of children $N(a)$ (the *neighbors* of a). To insert a new element x, its point of insertion is sought starting at the tree root and moving to the neighbor closest to x, updating $R(a)$ in the way. We finally insert x as a new (leaf) child of a if **(1)** x is closer to a than to any $b \in N(a)$, and **(2)** the arity of a, $|N(a)|$, is not already maximal. In other case, we insert x in the subtree of the closest element $b \in N(a)$. Neighbors are stored left to right in increasing timestamp order. Note that each element is older than its children and than its next sibling. See Algorithm 1.

3.2 Search Algorithm

The idea for range searching is to replicate the insertion process of relevant elements. That is, we act as if we wanted to insert q but keep in mind that relevant elements may be at distance up to r from q. So in each decision for simulating the insertion of q we permit a tolerance of $\pm r$, so that it may be that relevant elements were inserted in different children of the current node, and backtracking is necessary.

We have to consider two facts. The first is that, at the time an element x was inserted, a node a in its path may not have been chosen as its parent because its arity was already maximal. So, at query time, instead of choosing the closest to x among $\{a\} \cup N(a)$, we may have chosen only among $N(a)$. Hence, we perform the minimization only among elements in $N(a)$. The second fact is that, at the time x was inserted, elements with higher timestamp were not yet present in the tree, so x could choose its closest neighbor only among elements older than itself.

Hence, we consider the neighbors $\{b_1, \ldots, b_k\}$ of a from oldest to newest, disregarding a, and perform the minimization as we traverse the list. This means that we enter into the subtree of b_i if $d(q, b_i) \leqslant \min\{d(q, b_1), \ldots, d(q, b_{i-1})\} + 2r$. That is, we always enter into b_1; we enter into b_2 if $d(q, b_2) \leqslant d(q, b_1) + 2r$; and so on. Let us stress again the reason: between the insertion of b_i and b_{i+j} there may have appeared new elements that chose b_i just because b_{i+j} was not yet present, so we may miss an element if we do not enter into b_i because of the existence of b_{i+j}.

Up to now we do not really need the exact timestamps but just to keep the neighbors sorted by timestamp. We make better use of the timestamp information in order to reduce the work done inside older neighbors. Say that $d(q, b_i) > d(q, b_{i+j}) + 2r$. We have to enter into the subtree of b_i anyway because b_i is older. However, only the elements with timestamp smaller than that of b_{i+j} should be considered when searching inside b_i; younger elements have seen b_{i+j} and they cannot be interesting for the search if they are inside b_i. As parent nodes are older than their descendants, as soon as we find a node inside the subtree of b_i with timestamp larger than that of b_{i+j} we can stop the search in that branch, because all its subtree is even younger.

Algorithm 2 Searching for q with radius r in a *DSA–tree* rooted at a, built with timestamping and bounded arity.

```
RangeSearch(Node a, Query q, Radius r,
        Timestamp t)

1. If T(a) < t  ∧  d(a,q) ≤ R(a) + r Then
2.   If d(a,q) ≤ r Then Report a
3.   d_min ← ∞
4.   For b_i ∈ N(a) Do // ascend. timestamps
5.     If d(b_i,q) ≤ d_min + 2r Then
6.       t' ← min{t} ∪ {T(b_j), j > i ∧
                          d(b_i,q) > d(b_j,q) + 2r}
7.       RangeSearch(b_i,q,r,t')
8.       d_min ← min{d_min, d(b_i,q)}
```

Algorithm 2 depicts the search process. Note that $d(a, q)$ is always known except in the first invocation, and the initial t is $+\infty$.

4. OUR PROPOSAL

As we mention previously, we build our proposal on *DSA–tree*. The new data structure, called *DSACL–tree* for short, performs the spatial approximation on groups or *clusters* of objects that are very close to each other, rather than individual objects. By this way it can reduce search costs, because it has to do less backtracking. The new data structure is still dynamic, competitive, and makes better use of the available memory.

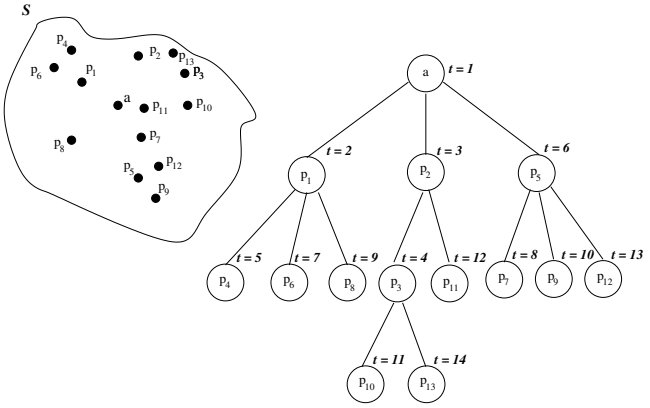
Therefore, in the *DSACL–tree* each node represents a cluster of very similar objects, for short we refer to it simply as *cluster*. Thus, we relate the clusters by their proximity in the metric space. So, each node of the tree would be able to store multiple database objects, reducing the number of nodes with respect to the original *DSA–tree*.

As in the *DSA–tree* we build the tree incrementally, considering a *maximum arity* and maintaining information of the *timestamp* (time of insertion of each element). We also register the *timestamp* $time(c)$ of each node c in the tree, that is the time when this node was created. Each node c keeps an object $center(c)$ as the *center* of the cluster and the k *nearest objects* ($cluster(c)$) seen in its subtree, and is connected with their clusters-neighbors $N(c)$. The cluster also have a *cluster radius* $rc(c)$, that is considering the objects in increasing order to the $center(c)$ the distance of the k-th object in the $cluster(c)$. Any object further away from the center than $rc(c)$ would become part of another tree node, which could be a new neighbor in some cases, since the arity is bounded in the same way as *DSA–tree*. Each node c also stores the maximum distance between the $center(c)$ and the farthest object in its subtree $R(c)$ (as *DSA–tree* does), called *covering radius* of the subtree of c.

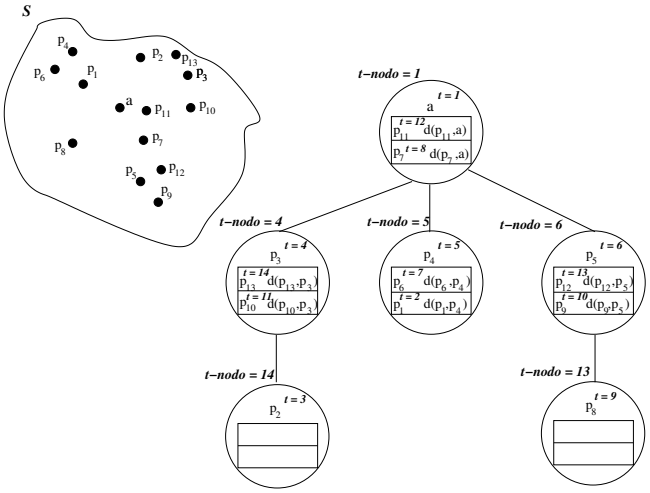
Since each node c represents a cluster centered in $center(c)$ with at most k objects within $cluster(c)$, we maintain the distances between $center(c)$ and all the objects in $cluster(c)$ ordered by increasing distance to the center. At search time, we can use these stored distances in order to minimize the number of distance computations using the triangle inequality. Besides, if $x_1, \ldots, x_k$ are the objects in $cluster(c)$ sorted by distances, the covering radius of the cluster will be $rc(c) = d(center(c), x_k)$. Therefore, for each object x_i inside the cluster, we stored its insertion moment $time(x_i)$

and the distance $d(center(c), x_i)$. It is clear that it is not necessary to really register $rc(c)$ because it can be obtained from the stored distances inside the node.

Figure 1 allows us to see graphically the differences between *DSA–tree* and *DSACL–tree* for the same metric space in $\mathbb{R}^2$, using Euclidean distance. The insertion sequence in both cases is $a, p_1, p_2, \ldots, p_{13}$ and the maximum arity is 3. For the *DSACL–tree* the maximum size of the cluster is 2.



(a) *DSA–tree*



(b) *DSACL–tree*

Figure 1: Comparison between a *DSA–tree* and *DSACL–tree* for the same metric space $S \subset \mathbb{R}^2$, using Euclidean distance.

Figure 2 depicts, for the same metric space used as example, how are grouped the elements in clusters within S and how they are related into the tree.

Now, we describe the insertion process.

4.1 Insertion

When we insert a new element x in a *DSACL-tree* we have to consider two cases. The first is when the new element becomes the center of a new node and the other is when x should be inserted into a cluster of a node c, that is x must belong to the $cluster(c)$, which means x is one of the k-nearest elements for the $center(c)$ at that current time.

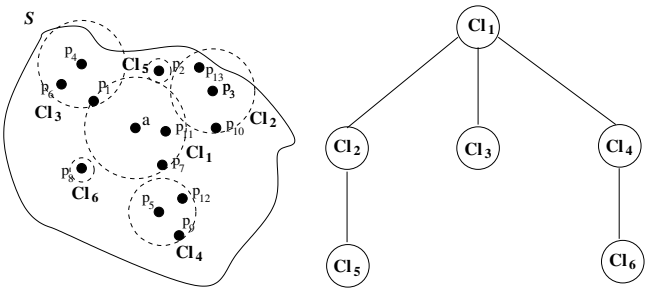


Figure 2: Clusters and their relations from the *DSACL–tree*, for the same example of Figure 1.

As we have already mentioned, to incrementally build the *DSACL–tree* we maintain the same considerations as *DSA–tree*, i.e. we set the maximum arity of the tree and store the insertion time of each object $timestamp(\cdot)$. We also maintain the creation time $time(c)$ for each node c in the tree and set the maximum size k of each cluster node. Moreover, given a node c, we store in its $cluster(c)$ the k-nearest elements seen in its subtree sorted by increasing distance to the $center(c)$, and for each of them we register its timestamp and the distance to the center.

When we insert a new element x in the cluster of a node c (that can occurs when $d(center(c), x) < rc(c)$) and we have achieved the maximum cluster size ($|cluster(c)| = k$), we need to remove an element from $cluster(c)$ to create a free slot for x. As objects within the cluster are sorted in increasing distance ($cluster(c) = x_1, \ldots, x_k$) we simple exclude x_k and include x. Of course we have to reinsert x_k searching its new insertion point in the subtree of c. Note that this also reduce the volume of $cluster(c)$.

When we insert a new element x into a *DSACL–tree* the whole insertion process is the following. It begins in the node which is the tree root, and in each center node c considered, we first evaluate if x must belong to $cluster(c)$, this is to achieve the smallest possible volume for the current cluster. Thus, we have the following cases:

1. If x should not belong to $cluster(c)$, we need to determine if x is closer to $center(c)$ than the centers of the neighboring nodes in $N(c)$:
   (a) If x is closer to $center(c)$ and the arity of c is not already the maximum, we insert x as the center of a new neighboring node of c. In this case we create a new node b with $center(b) = x$, and we add b in $N(c)$.
   (b) If x is not closer to the $center(c)$, or the arity of c has achieved the maximum value, we continue the insertion process from the c's neighbor node b ($b \in N(c)$) whose $center(b)$ is the closest to x. Recursively with b we have to check whether x must belong to the $cluster(b)$, x has to be inserted as a center of a new neighboring node, or we have to continue down the tree until we find the correct insertion point.
2. On the other hand, if there is a free slot in c or x is closer of the center than its current covering radius, then x must belong to $cluster(c)$. In the first case ($|cluster(c)| < k$), we simple insert x with its

distance $d(center(c), x)$ into the $cluster(c)$ preserving the increasing order of distances. In the second case ($d(center(c), x) < rc(c)$), we can reduce $rc(c)$ by excluding the farthest element within the cluster and inserting x (and its distance) into $cluster(c)$. Next, we need to reinsert the excluded element by searching its correct insertion point from the node c as we mention.

In the last case it is important to notice what happens when the cluster is full, but x can reduce $rc(c)$. As x must belong to $cluster(c)$, we remove x_k, the farthest current element in $cluster(c)$ from it former cluster and then we reinsert x_k in the tree. Therefore, x_k has to choose the nearest element between $center(c)$ and the centers of the neighboring nodes in $N(c)$. If the former $center(c)$ is the nearest one, we insert a new node b whose $center(b) = x_k$ in $N(c)$, if the arity is not maximal. Otherwise, we continue the search of the insertion point from the nearest center of a neighboring node in $N(c)$, following the same procedure previously detailed. This can recursively reduce the volume of other clusters.

Algorithm 3 illustrates the whole insertion process. The function is invoked as `InsertCl(a, x)`, where a is the root node of the tree and x is the element to be inserted. As can be seen, as in the *DSA–tree* we only have to follow a path from the tree root to the cluster, or to the node which will be its father node when this element have to be a new center of a neighboring node. $MaxArity$ is the upper bound of the arity, k is the maximum size of a cluster, and $CurrentTime$ is the current time, which increases before each insertion.

Algorithm 3 Insertion of a new element x into a *DSACL–tree* with a as tree root, using timestamping and bounded arity.

```
InsertCl (Node a, Element x)
 1. R(a) ← max(R(a), d(center(a), x))
  /* Let rc(c) be the distance from center(a)
     to the farthest element in cluster(a) */
 2. If ((|cluster(a)| < k) ∨ (d(center(a), x) < rc(a))) Then
 3.   cluster(a) ← cluster(a) ∪ {x}
 4.   d'(x) ← d(center(a), x)
 5.   timestamp(x) ← CurrentTime
 6.   If (|cluster(a)| = k + 1) Then
 7.     y ← argmax_{z∈cluster(a)} d'(z)
 8.     cluster(a) ← cluster(a) − {y}
 9.     InsertCl(a,y)
10. Else
11.   c ← argmin_{b∈N(a)} d(center(b), x)
12.   If d(center(a), x) < d(center(c), x) ∧ |N(a)| < MaxArity
        Then /* b is a new node, neighbor of a,
                with center(b) = x */
13.     N(a) ← N(a) ∪ {b}
14.     center(b) ← x
15.     N(b) ← ∅, R(b) ← 0
16.     cluster(b) ← ∅
17.     timestamp(x) ← CurrentTime
18.     time(b) ← CurrentTime
19.   Else
20.     InsertCl (c,x)
```

The tree can be built incrementally. The first element inserted x will create a single node a, which become the tree root, whose center will be $center(a) = x$ with $rc(a) = 0$, $cluster(a) = \emptyset$, $N(a) = \emptyset$, and $R(a) = 0$. Then, the following k insertions will be responsible to fill $cluster(a)$ completely. In this situation, the next insertion will create a new node in the neighborhood of a, whose center will be the farthest from $center(a)$.

As can be noticed, never an insertion may change the center of a node, it only could create at most a new node with one corresponding center. However, an insertion can affect many nodes, by changing its clusters.

Finally, we can observe that in *DSACL-tree* occurs the same as in *DSA–tree*, we cannot guarantee that a new element x will become a neighbor of the first node a satisfying that x does not belong to $cluster(a)$ and it is nearest to $center(a)$ than any other center of neighboring nodes $b \in N(a)$. The reason is that the node a had already reached the maximum arity, so x will be forced to choose the node in $N(a)$ whose center is the nearest one to continue the insertion process. This fact will have implications in searches.

4.2 Range Search

When performing a range query, we proceed in a similar way as *DSA–tree*, that is we perform the spatial approximation to the query via the centers of nodes. As we mentioned previously, the idea for range searching is to replicate the insertion process of the relevant elements to the query. That is, we act as if we wanted to insert q but keeping in mind that relevant elements may be at distance up to r from q, so in each decision we simulate the insertion of q permitting a tolerance of $\pm r$. So that it may be that relevant elements were inserted in a cluster, in different children of the current node, and backtracking is necessary.

We have two important facts to consider during searches. The first one is that at the time an element x was inserted, a node a in its path may not have been chosen as its parent because its arity was already maximal. So, at query time, instead of choosing the closest center to x among $\{a\} \cup N(a)$, we may have chosen only among $N(a)$. Hence, we perform the minimization only among elements in $N(a)$. The second fact is that, at the time x was inserted, elements with higher timestamp were not yet present in the tree, so x could choose its closest center of a neighbor only among elements older than itself.

Moreover, because we have clusters within the nodes, in each node a reached during the search we have to check if $cluster(a)$ intersects with the query (q, r) and its cluster using $rc(a)$. If there is no intersection we can discard all the elements in the cluster without any comparison with q. However, if there is intersection, we still can use $center(a)$ as a pivot. That is, we can avoid some distance computation considering the stored distances to the $center(a)$ and the triangle inequality to discard any element $x_i \in cluster(a)$ that satisfies $|d(center(a), x_i) - d(center(a), q)| > r$. In this case, non discarded elements must be compared directly with q.

It is very important to notice that, if the query is strictly contained inside the cluster of a node a reached during the search, we can stop the process on the subtree of a, because in no other place of this subtree we could find any relevant elements to the query.

We also can use the timestamps and covering radii information to prune searches, as *DSA–tree* does.

Algortithm 4 depicts the search algorithm on a *DSACL–tree*. It is initially invoked as `RangeSearchCl(`a, q, r`m`$CurrentTime$`)`, where a is the tree root.

Figure 3 shows graphically the different situations that can occur during the search process in a *DSACL–tree*. In Figure 3(a), we find that $d(center(a), q) \leq R(a) + r$, so we

Algorithm 4 Range Search of q with radius r in a *DSACL-tree* with tree root a.

```
RangeSearchCl (Node a, Query q, Radius r, Timestamp t)
 1. If time(a) < t ∧ d(center(a), q) ≤ R(a) + r Then
 2.   If d(center(a), q) ≤ r Then Report a
 3.   If (d(center(a), q) − r ≤ rc(a))∨
        (d(center(a), q) + r ≤ rc(a)) Then
 4.     For c_i ∈ cluster(a) Do
 5.       If |(d(center(a), q) − d'(c_i)| ≤ r Then
 6.         If d(c_i, q) ≤ r Then Report c_i
 7.     If d(center(a), q) + r < rc(a) Then Return
 8.   d_min ← ∞
 9.   For b_i ∈ N(a) in increasing order of timestamp Do
10.     If d(center(b_i), q) ≤ d_min + 2r Then
11.       k ← min{j > i, d(center(b_i), q) > d(center(b_j), q) + 2r}
12.       RangeSearchCl (b_i,q,r,time(b_k))
13.     d_min ← min{d_min, d(center(b_i), q)}
```

need to evaluate the subtree of the node a, but we do not enter into its cluster because $d(center(a), q) - r > rc(a)$, thus there are not relevant elements for the query. In Figure 3(b), $d(center(a), q) - r \leq rc(a)$, then we need to consider the elements in $cluster(a)$, but in this case we can use $center(a)$ as pivot. Finally, in Figure 3(c), after evaluate the elements in $cluster(a)$, since $d(center(a), q) + r < rc(a)$ we can conclude the search process in this subtree as the query is strictly contained into $cluster(a)$.

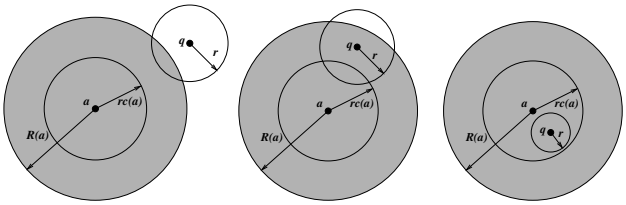


Figure 3: All the possible cases where it is satisfied that $d(center(a), q) \leq R(a) + r$.

5. EXPERIMENTAL RESULTS

For the experiments we have selected four widely different metric spaces. They are available from the SISAP Metric Library (`www.sisap.org`). Using these databases we can give a broad picture of the performance of our index.

***English*:** is a dictionary of 69,069 English words. We use the *edit distance* or *Levenshtein distance*, that is, the minimum number of character insertions, deletions and substitutions needed to make two strings equal.

***Documents*:** 1,265 documents under the Cosine similarity, from TREC-3 collection. In this model the space has one coordinate per term and documents are seen as vectors in this space. The distance we use is the angle among the vectors.

***NASA*:** 40,700 20-dimensional feature vectors, generated from NASA images, using Euclidean distance.

***Histograms*:** 112,682 8-D color histograms (112-dimensional vectors) from an image database. Euclidean distance is used for this metric space, because it is the simplest meaningful alternative, but any quadratic form can be used as a distance.

For the search experiments, we build the indices with 90% of the points and use the other 10% (randomly chosen) as queries. All our results are averaged over 10 index constructions using different permutations of the datasets. We have considered range queries retrieving on average 0.01%, 0.1% and 1% of the dataset. This corresponds to radii 0.140, 0.150 and 0.195 for the documents; 0.12, 0.285 and 0.53 for the images; and 0.051768, 0.082514 and 0.131163 for the histograms. Words have a discrete distance, so we used radii 1 to 4, which retrieved on average 0.003%, 0.037%, 0.326% and 1.757% of the dataset, respectively. The same queries are used for all the experiments on the same datasets. For economy of space, we place all the experimental results together in Figure 4.

For all the considered metric spaces we start by determining experimentally which is the best cluster size for searches, so we tested with sizes: 10, 50, 100, 150, 200, and 250. The best size is 50 elements in English space and 10 elements in the others. In the following results, we set the cluster size in 50 elements for all the experiments on *DSACL–trees* for English space and in 10 elements for Documents, NASA, and Histograms spaces.

Then, we have to select the best maximum arity for each space. We try several values of maximum arities, namely 2, 4, 8, 16, and 32, but we also evaluate how it is affected the search on our tree when we do not bound the arity. Figure 4, left column, shows the construction costs of the *DSACL–tree* with all the arities tested. As can be seen, construction costs increases as arity grows.

Additionally, the left column also shows the construction costs for the other indices which *DSACL-tree* is compared with (that performance comparison is given in Section 5.1).

Figure 4, middle column, depicts the results obtained in searches. As can be seen for the English space (Figure 4(b)) we select the option without bounded arity, for the Documents (Figure (e)) maximum arity of 4 is the best one, and for NASA space and for Histograms (Figures (h) and (k), respectively) the best arity is 8. The rule of thumb is that low arities are good for low dimensions or small search radii and viceversa (higher arities are better for high dimensions or low selectivity).

5.1 Comparison with other indices

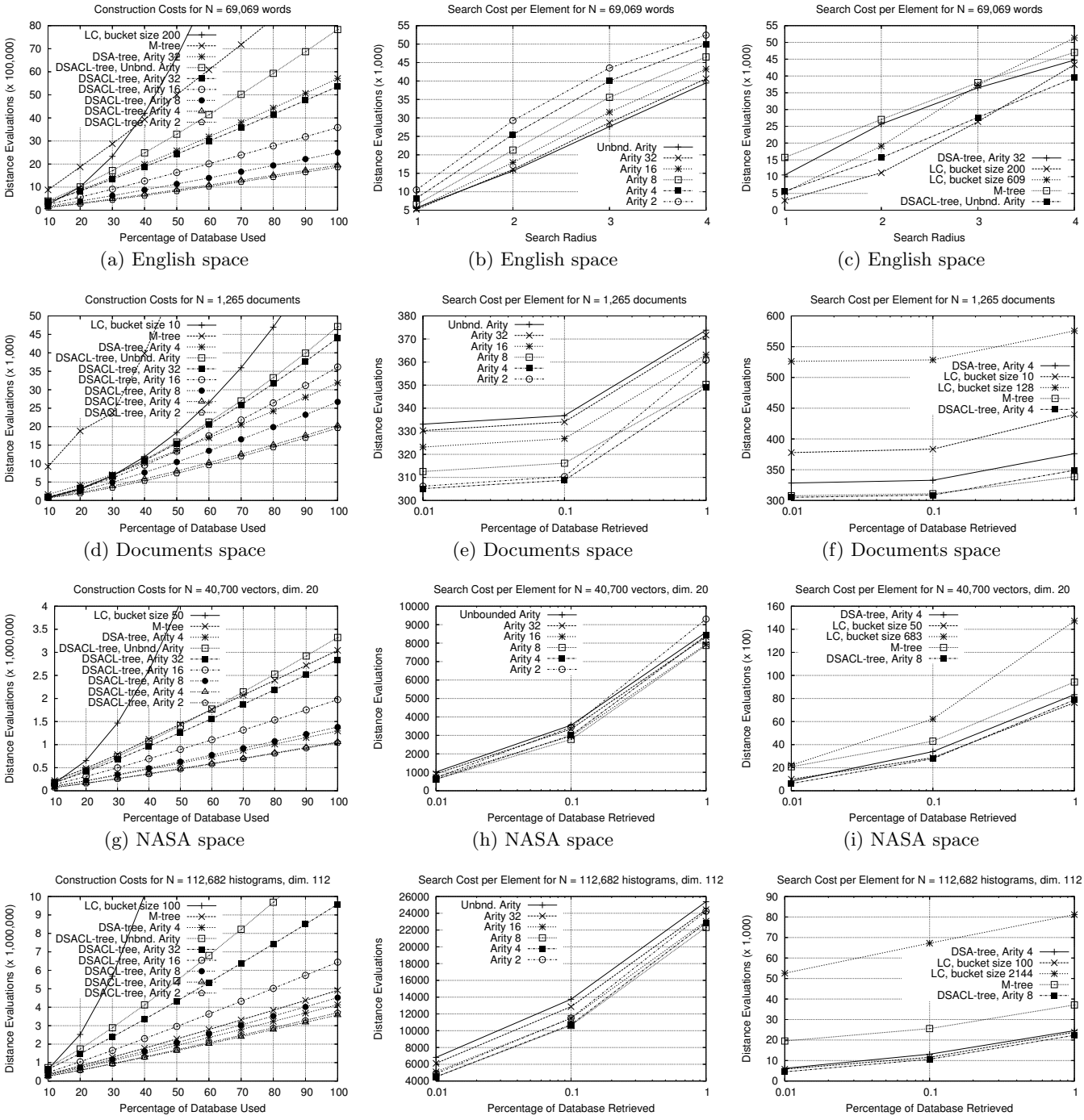
In order to evaluate the competitiveness of the *DSACL–tree*, we compare its search performance with three data structures. One of them is clearly the *DSA–tree*, the second one is the *M–tree* [6], that is the best-known dynamic secondary-memory data structure and its code is freely available[1], and the last one is the *List of Clusters* [4] because, despite it is not dynamic, it works well in high dimensional spaces. We choose these indices because they have demonstrated to be competitive and its codes are freely available[2].

For the *M–tree* we have used the parameter setting suggested by the authors[3]. For the *DSA–tree* we used the best

[1]At `http://www-db.deis.unibo.it/research/Mtree/`

[2]Codes of *DSA–tree* and *List of Clusters* are available at SISAP Metric Space Library (`www.sisap.org`)

[3]`SPLIT_FUNCTION = G_HYPERPL, PROMOTE_PART_FUNCTION = MIN_RAD, SECONDARY_PART_FUNCTION = MIN_RAD, RADIUS_FUNCTION = LB, MIN_UTIL = 0.2. We use PAGE SIZE = 4KB`



(a) English space (b) English space (c) English space

(d) Documents space (e) Documents space (f) Documents space

(g) NASA space (h) NASA space (i) NASA space

(j) Histograms space (k) Histograms space (l) Histograms space

Figure 4: In the first row we show the results for the English dictionary, using cluster size of 50 objects. In the following rows, we show results for the other spaces, using cluster size of 10 objects. In Plot (a), (d), (g), and (j), LC reaches $26.9 \cdot 10^6$, $73.3 \cdot 10^3$, $16.8 \cdot 10^6$, and $64.0 \cdot 10^6$ distance evaluations, respectively. In Plot (a) and (d), $M-tree$ reaches $10.6 \cdot 10^6$ and $145 \cdot 10^3$, distance evaluations, respectively. In Plot(j), *DSACL-tree* with unbounded arity reaches $12.7 \cdot 10^6$ distance evaluations. On the left column, we show construction costs both for the *DSACL–tree* with different arities and for the alternative indices. On the middle column, we show *DSACL–tree* search costs. On the right column, we compare the *DSACL–tree* search costs with the costs of all the alternative indices *DSACL–tree* is compared with.

parameters reported in [11] for each considered metric space. For the *List of Clusters* we show the best cluster size experimentally determined for each space, but in order to make a fair comparison we also show for each metric space the option with the cluster size that obtains similar construction costs than *DSACL–tree*.

Figures 4(a), (d), (g), and (j) show that, in general, the *DSATCL–tree* is a very economical alternative in terms of distance computations during index construction. In fact, the plots show that it is similar, and sometimes cheaper, than the basic *DSA–tree*.

In the those plots, we show the construction cost of a *LC* with best cluster size. As can be notice, constructing this particular *LC* is very costlier. In terms of space requirement, the *M-tree* is the index using more space, and the *DSACL–tree* is the second alternative. The *DSA–tree* and the *LC* use little space in order to index the objects.

Finally, in terms of object retrieval, Figures 4(c), (f), (i), and (l) show that the best suited *DSACL–tree* for each metric space is consistently the best or second best in searching performance for all the spaces considered.

6. CONCLUSIONS AND FUTURE WORKS

In this work we present the *DSACL–tree* which is a new index for searching metric spaces. This new index enhances the good features of the *DSA–tree* (spatial approximation and dynamism) by taking into account the element clusters present in the metric space. In fact, each node of the *DSACL–tree* not only stores a single object as the root of a tree, but also it maintains a bucket to store the elements which are the closest ones with respect to the current root. This allows us to reduce the backtracking in the tree improving the cost of retrieval relevant elements when performing a proximity query. We have shown some empirical evidence that our new index is competitive with other dynamic indices such as *DSA–tree* and *M–tree*.

Future work considers a secondary memory version of the *DSACL–tree*. We also pretend to perform an exhaustive experimental study of our data structure, both to understand its behavior in other metric spaces and also to evaluate the quality of the clusters produced in the *DSACL–tree*.

7. ACKNOWLEDGMENTS

We wish to thank Mauricio Marin and Gonzalo Navarro for their valuable comments.

8. REFERENCES

[1] D. Arroyuelo, F. Muñoz, G. Navarro, and N. Reyes. Memory-adaptative dynamic spatial approximation trees. In *Proc. 10th Intl. Symp. on String Processing and Information Retrieval (SPIRE'03)*, LNCS 2857, pages 360–368. Springer, 2003.

[2] J. Bentley. Multidimensional binary search trees used for associative searching. *Comm. of the ACM*, 18(9):509–517, 1975.

[3] J. Bentley. Multidimensional binary search trees in database applications. *IEEE Trans. on Software Engineering*, 5(4):333–340, 1979.

[4] E. Chávez and G. Navarro. A compact space decomposition for effective metric indexing. *Pattern Recognition Letters*, 26(9):1363–1376, 2005.

[5] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001.

[6] P. Ciaccia, M. Patella, and P. Zezula. M-tree: an efficient access method for similarity search in metric spaces. In *Proc. 23rd Conf. on Very Large Databases (VLDB'97)*, pages 426–435, 1997.

[7] V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.

[8] G. Hjaltason and H. Samet. Incremental similarity search in multimedia databases. Technical Report CS-TR-4199, University of Maryland, Computer Science Department, 2000.

[9] G. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Trans. on Database Systems*, 28(4):517–580, 2003.

[10] G. Navarro. Searching in metric spaces by spatial approximation. *The Very Large Databases Journal*, 11(1):28–46, 2002.

[11] G. Navarro and N. Reyes. Dynamic spatial approximation trees. *ACM Journal of Experimental Algorithmics*, 12:article 1.5, 2008. 68 pages.

[12] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[13] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach*. Advances in Database Systems, vol.32. Springer, 2006.

Approximate search

SISAP 2010

Indexing inexact proximity search with distance regression in pivot space

Ole Edsberg
edsberg@idi.ntnu.no

Magnus Lie Hetland
mlh@idi.ntnu.no

Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim, Norway

ABSTRACT

We introduce an inexact indexing scheme where, at index building time, training queries drawn from the database are used to fit one linear regression model for each object to be indexed. The response variable is the distance from the object to the query. The predictor variables are the distances from the query to each of a set of pivot objects. At search time, the models can provide distance estimates or probabilities of inclusion in the correct result, either of which can be used to rank the objects for an inexact search where the true distances are calculated in the resulting order, up to a halting point. To reduce storage requirements, the coefficients can be discretized at the cost of some precision in the promise values. We evaluate our scheme on synthetic and real-world data and compare it to a permutation-based scheme that has been reported to outperform other methods in the same experimental setting. We find that, in several of our experiments, the regression-based distance estimates give better query performance than the permutation-based promise values, in some cases even when the pivot set for the regression-based scheme is reduced in order to make its memory size equal to that of the permutation-based index. Limitations of our scheme include high index building cost and vulnerability to deviation from the model assumptions.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Indexing Methods*

General Terms

Algorithms, Performance

1. INTRODUCTION

Proximity search has a variety of applications, including k nearest neighbours classification and multimedia information retrieval. In this paper, we introduce an indexing scheme to facilitate faster proximity search, with the restriction that the indexes may not access the internal structure of the objects to be indexed, only the output of the distance function. This restriction makes the indexing scheme more generally applicable, and places it in the category *distance-based indexing*. This category encompasses both *metric indexing* [5, 8, 9], where the distance function is assumed to satisfy the metric properties, and *non-metric indexing* [16], where it is not. Our indexing scheme is applicable both to metric and non-metric spaces, and to both of the main query types: k-NN queries, which must return the k objects with the smallest distance to the query object, and range queries, which must return all objects whose distances to the query object do not exceed a radius r.

Indexes that are guaranteed to return the correct result every time without error are called *exact*, whereas those without such a guarantee are called *inexact*. Some inexact indexing schemes are approximate, meaning that they provide a guarantee of how wrong the returned objects can be, some are probabilistic, meaning that they provide a guarantee of the probability that the result will be correct, some are probabilistically approximate, and some provide no guarantee at all [13]. Our regression-based indexing scheme, like the permutation-based scheme we will compare it against, is inexact and provides no guarantees. Its performance must be assessed empirically.

The *Curse of Dimensionality* is a phenomenon that affects many types of distance-based applications, including indexing of proximity search. It manifests in dramatic performance degradation as data dimensionality increases. The curse is not triggered by tuple size in itself (which may not even apply for some data sets), but by the *intrinsic* dimensionality, which depends on the distance distribution [5]. For high enough dimensionality, exact indexes fail completely and must compute the distance from the query to every indexed object. If exactness is not an absolute requirement, it may still be possible to obtain a mostly correct answer without calculating more than a fraction of the possible distances by making a trade-off between search accuracy and computational cost [4]. The curse is dampened, not cancelled; as the intrinsic dimensionality increases, we will have to settle for worse computation/accuracy trade-offs.

When the distance function is expensive to compute, a simple way to perform an inexact proximity search with a trade-off between computation and accuracy is to visit the objects in the database in the order, or schedule, given by a cheaply computed *promise value function*, calculating the true dis-

SISAP 2010, September 18–19, 2010, Istanbul, Turkey.

tance for each object only as it is visited [3]. The objects after the halting point of the schedule are eliminated without calculating the true distance, incurring a risk of losing members of the true query result but also a saving some distance computations. By adjusting the halting point, we can obtain different computation/accuracy trade-offs. The promise value function may require a precalculated data structure, which constitutes an index for the database. When designing indexing schemes in this setting, a central problem is to choose a promise value function that will place the members of the true query result as early as possible in the ordering.

In this article, we introduce a new indexing scheme where distance regression is used to obtain promise values for inexact proximity search. We fit one regression model for each object to be indexed. We fit the models on training queries drawn from the database, with the true distance from the indexed object to the query as the response variable, and the true distances from the indexed object to a set of pivot objects as the predictor variables. We fit one model for each object to be indexed. The models can produce both distance estimates and inferences about the probability of the indexed object being part of the true query result. With n indexed objects and m pivots, the scheme requires the storage of $n(m+1)$ coefficients for the production of distance estimates, with an additional $2n+2$ values required for the probabilistic inferences. The coefficients can be discretized, at the cost of some precision, to reduce the memory size of the index. Building our indexes is computationally expensive: Distances between the n database objects and the n' training queries must be calculated, and n regression models must be fitted. We compare the performance of our scheme against the permutation-based indexing scheme proposed by Chávez et al. [3], which was reported to perform better than other methods in the same setting. We reuse data sets, plot types and parameters from their study in our evaluation. In several of our experiments, our distance estimates give better query performance than the permutation-based promise values, in some cases even when the pivot set for the distance estimates is reduced in order to equalize the memory size of the indexes.

2. TERMINOLOGY

Given a database $\mathbb{S} = \{u_1, u_2, \ldots, u_n\}$, which is a finite subset of a universe $\mathbb{U}$ of objects, and a distance function $d : \mathbb{U} \times \mathbb{U} \to \mathbb{R}$, our task is to facilitate fast proximity search (of the range and/or k-NN type) for queries drawn from $\mathbb{U}$.

The indexes we consider operate with reference to an ordered set of pivot objects $\mathbb{P} = (p_1, p_2, \ldots, p_k) \in \mathbb{U} \setminus \mathbb{S}$.[1] $\mathbb{P}$ gives rise to a vector space embedding $\Phi(o) : \mathbb{U} \to \mathbb{R}^k$, mapping each object $o \in \mathbb{U}$ to $(d(o, p_1), d(o, p_2), \ldots, d(o, p_k))$.

Search is performed by calculating $\Phi(q)$ for the query object q, and then visiting $\mathbb{S}$ in the ordering,[2] or schedule, given by a promise value function $\mathfrak{P}(u, \phi, r) : \mathbb{S} \times \mathbb{R}^k \times \mathbb{R} \to \mathbb{R}$. The parameter r in $\mathfrak{P}(u, \Phi(q), r)$ stands for the radius of a range query, and is only needed for probability-based promise values. For k-NN queries it can be estimated from $\mathbb{S}$. At each point u in the schedule, the true $d(u, q)$ is calculated. By halting the search before the schedule has been completed, we save some distance calculations at the cost of possibly discarding some members of the true query results.

[1]For normal operation of the indexes, $\mathbb{P}$ would be taken from $\mathbb{S}$. Our interest is to compare the schedules produced by different promise value functions. For this purpose, having $\mathbb{P}$ as part of $\mathbb{S}$ would be a distraction because $\mathbb{P}$ in effect would be prepended to every schedule.

[2]To order the objects, we used the incremental sort algorithm of Paredes & Navarro [12].

3. RELATED WORK

Indexing methods for inexact similarity search have been surveyed by Patella & Ciaccia [13]. Of particular relevance to our regression-based scheme is a line of work [2, 4] starting with the idea of shrinking the query radius by a multiplicative factor, sacrificing exactness in return for increased filtering power, especially in spaces with high intrinsic dimensionality. This idea was applied to a simple pivot-based LAESA-like [11] indexing scheme where the lower bound from the triangle inequality was used to filter the database. Instead of controlling the computation/accuracy trade-off by shrinking the radius, it can, as done by Chávez et al. [3], be controlled by sorting the database according to the lower bound and adjusting the size of the fraction (with the lowest bounds) of objects that are not filtered away. This is the promise value based search mechanism we use in this paper.

Using the promise value based search mechanism, Chávez et al. introduced a permutation-based indexing scheme, where the promise value function was the rank correlation (Spearman's rho) between two permutations of the pivots: the permutation according to distance to the query object and the permutation according to distance to the indexed object [3]. They found that this scheme outperformed several other schemes, including promise values based on the L_1, L_2 and L_∞ norms between $\Phi(q)$ and $\Phi(u)$. We will apply their scheme as a baseline in our experiments, and we label it $\mathfrak{P}_\Pi$.

BoostMap [1] is another index that performs promise value based search, and that operates with reference to a set of (candidate) pivot objects. At index-building time, BoostMap applies an AdaBoost-like algorithm to construct a mapping from objects in $\mathbb{U}$ to vector space embeddings. These embeddings are linear combinations of a number of one-dimensional embeddings, selected by the algorithm from a large number of randomly generated candidates. The 1D embeddings come in two types: one type embeds an object as its distance to a single pivot object, and the other type embeds an object as its projection onto the line between a pair of pivot objects. The algorithm is optimized with reference to a set of triplets of objects, and tries to maximize the degree to which the distances between embedded objects agree with the true distances on the question of whether the second or the third object in a triplet is closer to the first. Only triplets where the second object is a k-NN of the first (for k up to a parameter $k_{\max}$) are included, thus focusing the optimization on preserving the nearest-neighbour structure in the data set.

4. INDEXING INEXACT PROXIMITY SEARCH WITH DISTANCE REGRESSION IN PIVOT SPACE

A good promise value function is one that, among other criteria, (*i*) places the true query results early in the schedule, thus providing good computation/accuracy trade-offs, (*ii*) can be calculated quickly enough relative to the true distance, and (*iii*) requires little space for its index. Our regression-based indexing scheme is motivated by criterion (*i*) and stems from two lines of thought:

1. *The promise value ordering should approximate as closely as possible the ordering of the indexed objects according to their true distance to the query (ascending).* This line of thought undoubtedly lies behind the permutation based indexing scheme of Chávez et al. [3], with the reasoning that objects with a higher pivot permutation correlation to the query are more likely to have a smaller true distance to it. We take a more direct route, and instead reason that objects with a smaller estimated distance to the query are more likely to have a smaller true distance to it. We will show how linear regression models can provide the needed distance estimates. We call the resulting promise value functions $\mathfrak{P}_{\hat{d}}$.

2. *The promise value ordering should approximate as closely as possible the ordering of the indexed objects according to their probability of being part of the true query result (descending).* This will, for any given halting point in the search schedule, maximize the expected number of members of the true query result that will have been found at that point. The application of any distance-based index to a query object can in this way be viewed (explicitly or implicitly) as a problem of probabilistic inference based on the incomplete information stored in the index, as well as the query parameters and the distance calculations performed as part of the search. The probability-based promise values take into account that the reliability of the estimates may vary between the indexed objects, for example, as a consequence of their position in relation to the pivot objects. We will show how linear regression models can provide such inferences for use as promise values. We call the resulting promise value functions $\mathfrak{P}_t$.

To see how ordering according to probabilities could give a different outcome from ordering according to estimated distances, consider a situation where two indexed objects a and b are both estimated to fall outside the query radius, and where the estimated distance from the query to a is just a little bit smaller than that to object b. Assume additionally that the index is known to produce much more reliable distance estimates for a than for b. Then it may be the case that, given the information available to us, b has a higher probability of belonging to the query result than a, even though a has a smaller estimated distance to the query.

While the probability-based promise values are justified in terms of maximizing search accuracy, it is not a given that they will perform better than the distance-based ones in practice; for example, the inference may depend on assumptions that are not fully satisfied by the distance space.[3]

[3]In our experiments, $\mathfrak{P}_t$ never performs better than $\mathfrak{P}_{\hat{d}}$. We still include $\mathfrak{P}_t$ because of its interesting rationale.

4.1 The linear models

We apply linear regression [6] to model the relationship between $d(u,q)$ and $\Phi(q)$, for each $u \in \mathbb{S}$. We fit n models, one for each u. Given a query q, the model for u allows us to estimate $d(u,q)$, and also to make inferences about $\Pr(d(u,q) \leq r)$.

Let $\Phi^1(o)$ be the vector resulting from prepending 1 to $\Phi(o)$. We assume that the distance from an indexed object u to a query q is equal to the dot product between $\Phi^1(q)$ and $m+1$ coefficients β_u plus an additive error term ϵ_u. The errors are assumed to be i.i.d. with mean 0. We can write the model for u as:

$$d(u,q) = (\Phi^1(q))^T \beta_u + \epsilon_u \tag{1}$$

To estimate β_u, we select, based on an assumption that $\mathbb{S}$ is representative of the queries the index will be applied to, a random sample $\mathbb{S}' \subseteq \mathbb{S}$ of n' training queries. We calculate $\Phi^1(q')$ for each $q' \in \mathbb{S}'$. When fitting the model that will index u, we exclude u from $\mathbb{S}'$ to avoid using a training query with $d(u,q') = 0$. We denote the training queries that will be used to index u as $q'_{\langle u,1\rangle}, q'_{\langle u,2\rangle}, \ldots, q'_{\langle u,n'_u\rangle} = \mathbb{S}' \setminus u$. From these, we construct the n'_u-length vector $\mathbf{y}_u$ and the $n'_u \times (m+1)$ matrix $\mathbf{X}_u$:

$$\mathbf{y}_u = \begin{pmatrix} d(u, q'_{\langle u,1\rangle}) \\ d(u, q'_{\langle u,2\rangle}) \\ \cdots \\ d(u, q'_{\langle u,n'_u\rangle}) \end{pmatrix} \quad \mathbf{X}_u = \begin{pmatrix} \Phi^1(q'_{\langle u,1\rangle}) \\ \Phi^1(q'_{\langle u,2\rangle}) \\ \cdots \\ \Phi^1(q'_{\langle u,n'_u\rangle}) \end{pmatrix}$$

Using $\mathbf{y}_u$ and $\mathbf{X}_u$, we obtain a least squares estimate $\hat{\beta}_u$ of β_u via QR decomposition (using the `lm.fit` function in R [15]), by solving the following linear regression problem:

$$\mathbf{y}_u = \mathbf{X}_u \beta_u + \boldsymbol{\epsilon}_u \tag{2}$$

Deviation from the assumptions may degrade the accuracy of estimates and inferences based on the model [6]. Since the search is inexact to begin with, this can be acceptable. In any case, empirical evaluation will show how well an index performs in practice.

4.2 Calculating promise values

The model for the indexed object u provides a distance estimate that can be used as a promise value for u with respect to the query q:

$$\mathfrak{P}_{\hat{d}}(u, \Phi(q), r) = \hat{d}(u,q) = \Phi^1(q)\hat{\beta}_u \tag{3}$$

We have already assumed that the distance conforms to Equation 1, with the errors i.i.d. and having mean 0. To make the probabilistic inferences we need for $\mathfrak{P}_t$, we must make the further assumption that the errors are normally distributed. The variance of the distance estimate is:

$$\mathrm{Var}(\hat{d}(u,q)) = (\Phi^1(q))^T (\mathbf{X}_u^T \mathbf{X}_u)^{-1} \Phi^1(q) \sigma_u^2 \,, \tag{4}$$

where $\sigma_u^2 = \mathrm{Var}(\epsilon_u)$. We can estimate σ_u^2 with $\hat{\sigma}_u^2$ as follows:

$$\hat{\sigma}_u^2 = \frac{\sum_{i=1}^{n'_u} (d(u, q'_{\langle u,i\rangle}) - \hat{d}(u, q'_{\langle u,i\rangle}))^2}{n'_u - m - 1} \tag{5}$$

The following statistic is t-distributed with $n'_u - m - 1$ degrees of freedom:

$$T = \frac{d(u,q) - \hat{d}(u,q)}{\hat{\sigma}_u \sqrt{1 + (\Phi^1(q))^T (\mathbf{X}_u^T \mathbf{X}_u)^{-1} \Phi^1(q)}} \tag{6}$$

If the assumptions of the model hold, the probability that u is part of the true query result is

$$\Pr(d(u,q) \le r) = \mathrm{PT}\left(\frac{r - \hat{d}(u,q)}{\hat{\sigma}_u \sqrt{1 + (\Phi^1(q))^T (\mathbf{X}_u^T \mathbf{X}_u)^{-1} \Phi^1(q)}}\right), \quad (7)$$

where PT is the cumulative t-distribution with $n'_u - m - 1$ degrees of freedom. If $\mathbf{X}_u$ were the same for all $u \in \mathbb{S}$, the following promise value function would sort $\mathbb{S}$ according to $\Pr(d(u, \Phi(q)) \le r)$:

$$\mathfrak{P}_t(u, \Phi(q), r) = \frac{r - \hat{d}(u,q)}{\hat{\sigma}_u} \quad (8)$$

In our case, however, recall that when we fit the model for u, we exclude u from the $\mathbb{S}'$. Therefore $\mathbf{X}_u$ will have one row less for those u that are selected to be part of $\mathbb{S}'$. We choose to ignore these differences, and use Equation 8 to calculate the probabilistic promise values.

Because $\mathfrak{P}_t$ makes use of the query radius r, it cannot be applied directly to k-NN queries. However, for each $k \in 1 \ldots n$, we can calculate the radius r that would return on average exactly k of the indexed objects as results for the training queries we have already sampled. We store these radii as part of the index.

4.3 Storage requirements

Calculating $\mathfrak{P}_{\hat{d}}$ for n indexed objects with m pivots requires the storage of $n(m+1)$ coefficients. However, because the promise value function is only used to order $\mathbb{S}$, and because $\Phi^1(q)$ is the same for all $u \in \mathbb{S}$, applying the same linear transform to all $\hat{\beta}_u$ will not change the promise value orderings. Let us denote the highest and lowest value among all coefficients for all models as $\hat{\beta}^+$ and $\hat{\beta}^-$, respectively. A simple way to discretize the coefficient $\hat{\beta}_{\langle u,i \rangle}$ to a p bit representation is to replace it by

$$\left\lfloor \min\left\{ \frac{\hat{\beta}_{\langle u,i \rangle} - \hat{\beta}^-}{\hat{\beta}^+ - \hat{\beta}^-} \times 2^p, 2^{p-1} \right\} \right\rfloor . \quad (9)$$

Discretization reduces the precision of the coefficients, and this inaccuracy will carry over to the promise values and possibly degrade query performance.

To calculate $\mathfrak{P}_t$, we additionally need to store the denominator of Equation 8 for each u. If we discretize the model coefficients, we need to store $\hat{\beta}^+$ and $\hat{\beta}^-$, and incorporate these and $\Phi^1(q)$ into Equation 8 to scale r properly. To support k-NN queries with $\mathfrak{P}_t$, we need to store the n estimated radii. We do not discretize the additional $2n + 2$ values, as they are dominated by the coefficients in terms of size.

4.4 Computational cost of building the index

Building a regression based index for n objects with m pivots and n' training queries requires $n'(n+m)$ distance calculations and the solution of n linear regression problems, each with matrix size $n' \times (m+1)$ or $(n'-1) \times (m+1)$. In comparison, the permutation based method requires only nm distance calculations.

5. EXPERIMENTAL EVALUATION

To reduce the number of arbitrary choices we make in our evaluation, we imitate the setup used by Chávez et al. [3] as far as possible, given the data sets available on the SISAP site.[4] Following their example, we select the pivots randomly. Because the object of study is the effectiveness of the different schedules, we keep the pivots separate from the indexed objects. We summarize the results in plots with a measure of computational cost on one axis and a measure of query accuracy on the other axis. We measure computation cost with the wall clock time and/or the percentage of the database having been compared to the query with a distance computation at that point in the schedule. In all but one case, we measure query accuracy as the percentage of the true query result having been retrieved at that point. As a baseline for comparison, we use $\mathfrak{P}_\Pi$, the permutation-based indexing scheme found by Chávez et al. [3] to generally outperform the other methods they had selected.

Indexing involves a trade-off between the memory size of the index and the query speedup it offers. To index n objects with m pivots, $\mathfrak{P}_\Pi$ requires $nm\lceil \log_2(m) \rceil$ bits of memory to store its index [3]. The indexes for $\mathfrak{P}_{\hat{d}}$ require $n(m+1)p$ bits, where p is the number of bits used to store a single coefficient. To make a fair performance comparison between $\mathfrak{P}_\Pi$ and $\mathfrak{P}_{\hat{d}}$, we reduce the number of pivots available to $\mathfrak{P}_{\hat{d}}$ by an amount that equalizes the memory size of the two indexes. If $\mathfrak{P}_\Pi$ uses m pivots, a version of $\mathfrak{P}_{\hat{d}}$ using p bits per pivot can only use $\lfloor \frac{m\lceil \log_2 m \rceil}{p} \rfloor - 1$ pivots if we want the comparison to be fair to $\mathfrak{P}_\Pi$.[5] We include both fair and unfair versions of $\mathfrak{P}_{\hat{d}}$ in the evaluation, and mark the fair ones with asterisks. We evaluate four versions of our regression-based indexing scheme:

$\mathfrak{P}_{\langle \hat{d}, 16\,\mathrm{b} \rangle}$ uses $\hat{d}(u,q)$ as promise values, with the coefficients discretized to 16 bits. For 256 pivots, it uses twice the memory of $\mathfrak{P}_\Pi$.

$\mathfrak{P}^*_{\langle \hat{d}, 16\,\mathrm{b} \rangle}$ uses $\hat{d}(u,q)$ as promise values, with the coefficients discretized to 16 bits and with a reduced pivot set to memory-equalize it with $\mathfrak{P}_\Pi$.

$\mathfrak{P}^*_{\langle \hat{d}, 12\,\mathrm{b} \rangle}$ is identical to $\mathfrak{P}^*_{\langle \hat{d}, 16\,\mathrm{b} \rangle}$, except that it only uses 12 bits for each coefficient. This means that it can keep more pivots after memory-equalization, but may suffer a greater loss of precision from the discretization.

$\mathfrak{P}_{\langle t, 16\,\mathrm{b} \rangle}$ uses promise values from Equation 8, with the coefficients discretized to 16 bits. For 256 pivots, it uses slightly more than twice the memory of $\mathfrak{P}_\Pi$.

We programmed the index building in R [15], with query execution and distance calculation written in C. We used $n' = 2000$ training queries, except in the one case where the database was smaller than that. Except where otherwise noted, we averaged performance measures over 1000 queries with $q \notin \mathbb{S}$.

[4] `http://sisap.org`

[5] Note that we do not include the cost of calculating $\Phi(q)$ in the plots of our results, even though the memory-equalized methods will spend slightly less computation time here because they use fewer pivots.

5.1 Experiments on synthetic data

Unit cube with L_2 distance. Fig. 1 shows range query performance on L_2 spaces with 10 000 unit cube vectors with 128, 256, 512 and 1024 dimensions, and with 128 and 256 pivots, with radius set to capture 0.05 % of the objects on average. We measured percentage of the true result retrieved vs. the percentage of distance calculations required.

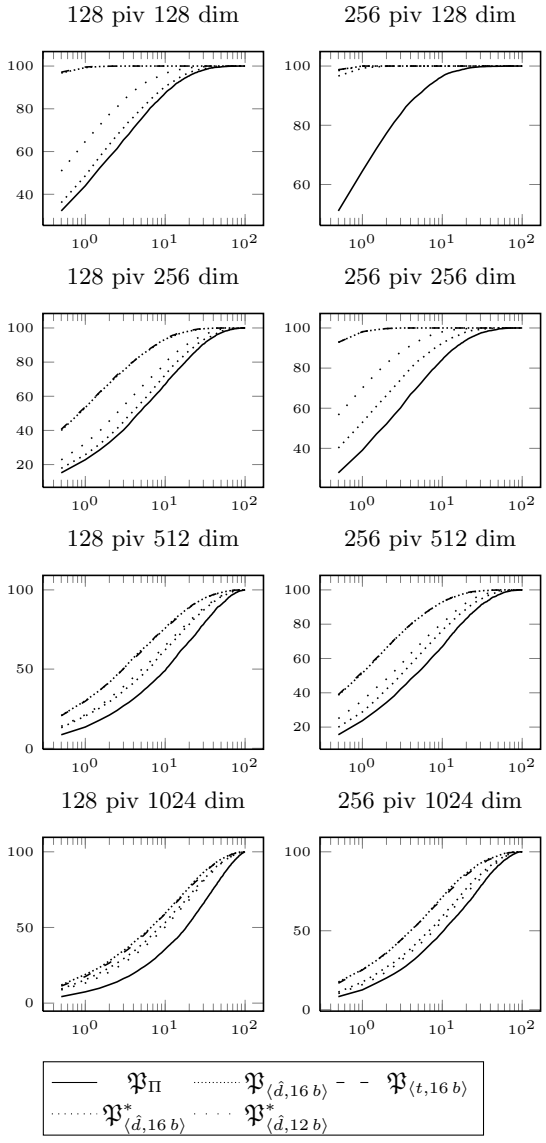


Figure 1: Index performance on range queries on L_2 distance between uniformly distributed vectors, with radius set to capture 0.05 % of the objects. On y axis: avg. % of true query result retrieved. On x axis: % of max distance calculations.

Clustered vectors with L_2 distance. Fig. 2 shows the performance of 1-NN queries on L_2 spaces with 10 000 1024-dimensional vectors generated from a uniform mixture of 32 multivariate normal distribution with variance 0.01 and the means for the distributions generated with another multivariate normal distribution with mean 0.5 and variance 0.09. We used 32 and 128 pivots.

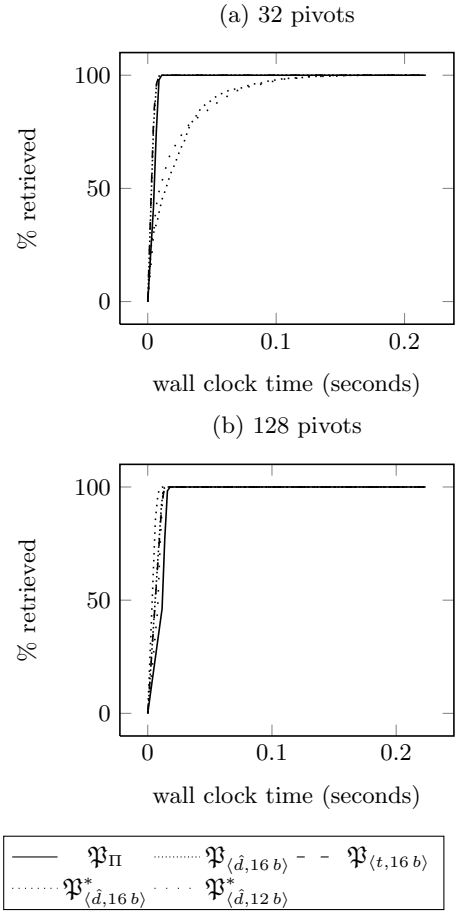


Figure 2: Index performance on 1-NN queries on L_2 distance between vectors generated from a mixture of 32 multivariate normal distributions.

Unit cube with fractional L_p distance (non-metric). Fig. 3 shows index performance on 2-NN queries on $L_{0.2}$ and $L_{0.8}$ spaces, with 10 000 uniformly generated vectors with 32, 64 and 128 dimensions, and with 128 and 256 pivots. Because of a coefficient discretization issue explained in section 5.3, we also included $\mathfrak{P}_{\hat{d}}$ with no discretization (using 8 bytes per coefficient) in this plot, and labelled it $\mathfrak{P}_{\langle \hat{d},\text{nodiscr}\rangle}$.

Index building time. We timed index building for a unit cube L_2 space with 1024 dimensions and 10 000 objects. We used 256 pivots. $\mathfrak{P}_{\Pi}$ spent about 50 seconds, and $\mathfrak{P}_{\langle \hat{d},16\,b\rangle}$ spent about 1900 seconds.

5.2 Experiments on real-world data

Face images (FERET) with L_2 distance. Fig. 4 shows performance of k-NN queries, with k ranging from 1 to 20, on images from FERET [14], a database of face images. We used the 761-coordinate eigenspace transformations of the images, because these were made available at the SISAP site. The database was divided into one section with 762 objects, and one with 254 objects. Of the 762 objects, we used 64 as pivots and indexed the remaining 698. We evaluated the indexes using the other 254 objects as queries, and measured the wall clock time and number of distance cal-

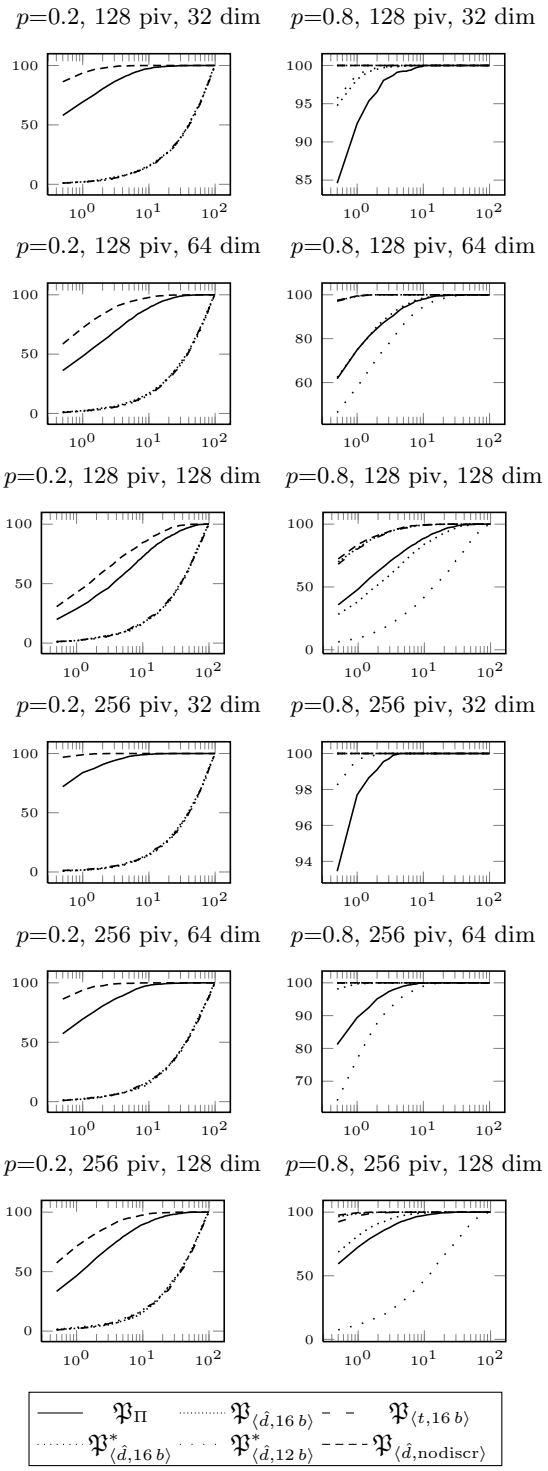


Figure 3: Index performance on 2-NN queries on uniformly generated L_p spaces with fractional p. On y axis: avg. % of true query result found. On x axis: % of max distance calculations.

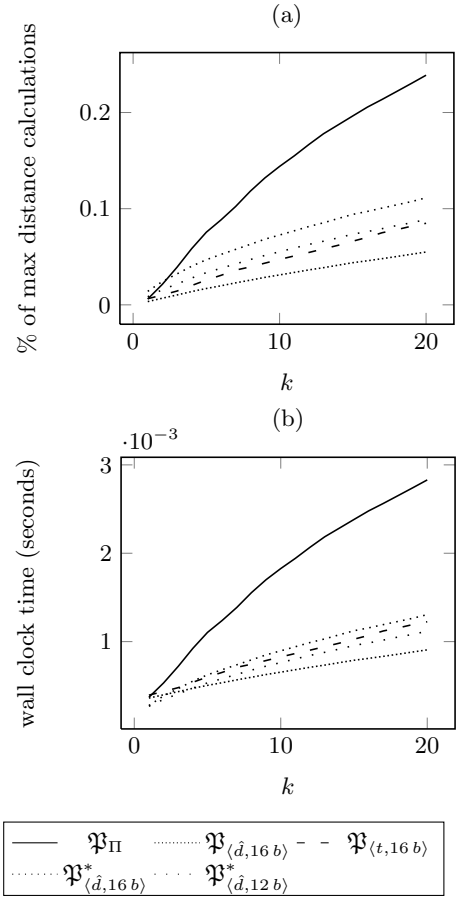


Figure 4: Index performance on L_2 space with 761-coordinate eigenspace transformations of face images. The plots show the amount of computation (y-axis) required to perfectly answer k-NN queries with varying values of k (x-axis).

culations necessary to perfectly answer k-NN queries with different values of k.

Documents (TREC). We also tested the methods on angle distance between vector representations of news documents from the Wall Street Journal, from the TREC data set [7]. There were 25 276 documents in the SISAP data. Of these, we used 128 as pivots, 1003 as queries, and indexed the remaining 24 145. The top part of Fig. 5 shows performance of range queries with a radius set to capture exactly 0.035 % of the database. The bottom part shows results for 5-NN queries.

Normalized edit distance (non-metric). Fig. 6 shows results of range queries (radius 1) on a space with normalized edit distance [10] between 40 000 words selected randomly from the Spanish dictionary file in the SISAP data. We used 32 and 128 pivots, and measured average percentage of the true query result found vs. percentage of the indexed objects having the true distance to the query calculated.

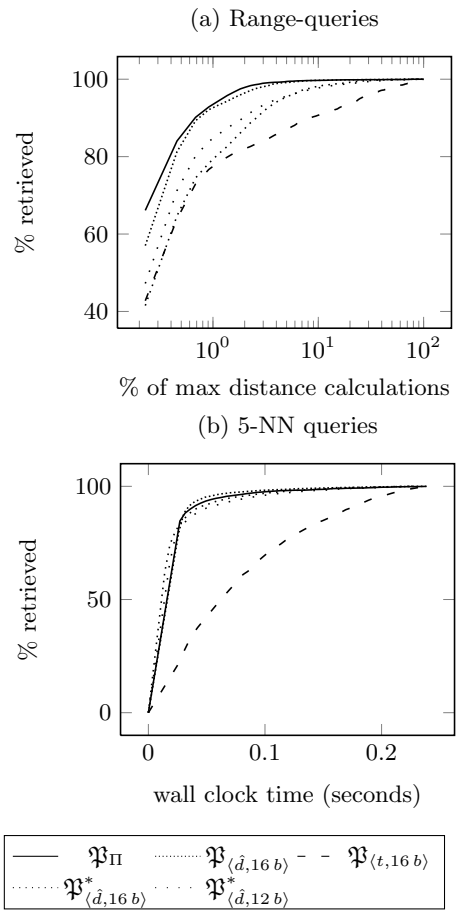


Figure 5: Index performance on angle distance between vector representations of news documents.

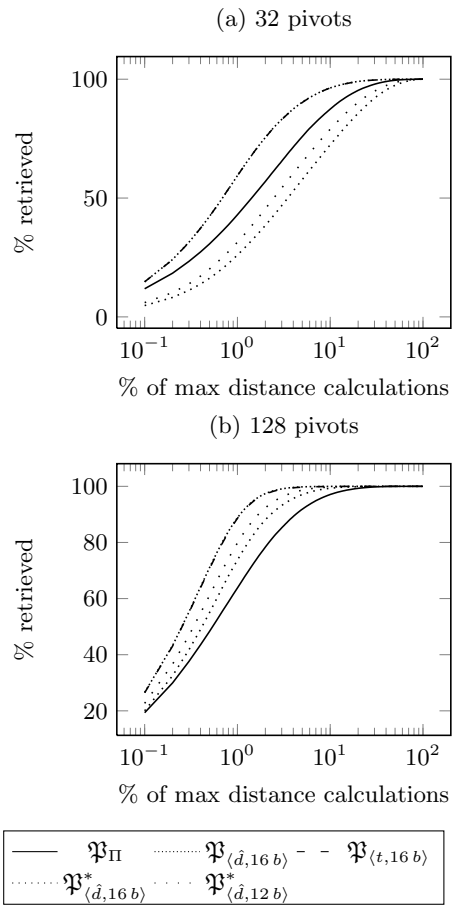


Figure 6: Index performance on range queries (radius 1) on normalized edit distance between 40 000 words from a Spanish dictionary.

5.3 Summary of results

The query performance of $\mathfrak{P}_{\langle \hat{d},16\,b\rangle}$, the unfair (memory-wise) version of $\mathfrak{P}_{\hat{d}}$, was better than that of the baseline $\mathfrak{P}_\Pi$ in the following experiments: unit cube L_2 (Fig. 1), unit cube $L_{0.8}$ (Fig. 3, right column), face images L_2 (Fig. 4) and normalized edit distance (Fig. 6). It was roughly equal to the baseline on clustered L_2 (Fig. 2) and in one experiment with document angle distance (Fig. 5, bottom). It was slightly worse than the baseline in the other experiment with document angle distance (Fig. 5, top). On unit cube $L_{0.2}$ (Fig. 3, left column) $\mathfrak{P}_{\langle \hat{d},16\,b\rangle}$, $\mathfrak{P}^*_{\langle \hat{d},16\,b\rangle}$, $\mathfrak{P}^*_{\langle \hat{d},12\,b\rangle}$ and $\mathfrak{P}_{\langle t,16\,b\rangle}$ all performed much worse than the baseline, and in fact behaved much like random schedules. To investigate, we looked at the coefficients of the models (before discretization) and found that the models invariably had a very large positive or negative value for one coefficient, with the rest of the coefficients being close to 0. This caused our discretization scheme to break down. We included the undiscretized, unfair (memory-wise) $\mathfrak{P}_{\langle \hat{d},\text{nodiscr}\rangle}$ in Fig. 3, and saw that it outperformed all other methods, including the baseline $\mathfrak{P}_\Pi$.

$\mathfrak{P}^*_{\langle \hat{d},16\,b\rangle}$ and $\mathfrak{P}^*_{\langle \hat{d},12\,b\rangle}$, the two fair (memory-wise) versions of $\mathfrak{P}_{\hat{d}}$, performed equal to or worse than $\mathfrak{P}_{\langle \hat{d},16\,b\rangle}$ in all cases. Their performance was better than the baseline on unit cube L_2 (Fig. 1), face images L_2 (Fig. 4) and normalized edit distance with 128 pivots (Fig. 6, bottom). On unit cube $L_{0.8}$ (Fig. 3, right column), their performance was sometimes better and sometimes worse than the baseline. Their performance was roughly equal to the baseline on clustered L_2 with 128 pivots (Fig. 2, bottom), and in one experiment with document angle distance (Fig. 5, bottom). They fared worse than baseline on clustered L_2 with 32 pivots, in the other experiment on document angle distance (Fig. 5, top) and on normalized edit distance with 32 pivots (Fig. 6, top). The poor performance of $\mathfrak{P}^*_{\langle \hat{d},16\,b\rangle}$ and $\mathfrak{P}^*_{\langle \hat{d},12\,b\rangle}$ in 32-pivot experiments is likely due to the fact that very few pivots will then remain after memory-equalization.

The probability-based, unfair (memory-wise) $\mathfrak{P}_{\langle t,16\,b\rangle}$ showed roughly the same performance as $\mathfrak{P}_{\langle \hat{d},16\,b\rangle}$ on unit cube L_2 (Fig. 1), on clustered L_2 (Fig. 2), on unit cube $L_{0.2}$ and $L_{0.8}$ (Fig. 3), and on normalized edit distance (Fig. 6). It fared worse than $\mathfrak{P}_{\langle \hat{d},16\,b\rangle}$ on face images L_2 (Fig. 4), where it beat the baseline, and on document angle distance (Fig. 5), where it was beaten by the baseline.

In the one experiment where we timed the index building, $\mathfrak{P}_{\langle \hat{d},16\,b\rangle}$ took about 38 times as long build its index as $\mathfrak{P}_\Pi$.

6. DISCUSSION

The regression-based indexing scheme has some downsides and limitations. First, the index is very expensive to build, as we have described in section 4.4. This may make our approach infeasible for some applications. If the index building can be amortized over a large number of queries, or scheduled to periods of downtime, high index building cost may be acceptable.

A second limitation is that the use of regression brings with it some dependencies on the properties of the distance space to be indexed. This may make regression-based indexes difficult to apply out-of-the-box in practice. Regression models will generally give more accurate predictions and inferences when the data they are fitted and tested on satisfies the underlying assumptions of the models (as described in section 4). It is not trivial to measure the extent to which the assumptions are satisfied, but some standard diagnostic plots are recommended [6]. Having tentatively looked at plots of residuals vs. fitted values and residuals vs. normal distribution quantiles, it appears to us that the document angle distance data (Fig. 5), where $\mathfrak{P}_{\langle \hat{d}, 16\,\mathrm{b}\rangle}$ fared tolerably and $\mathfrak{P}_{\langle t, 16\,\mathrm{b}\rangle}$ fared poorly, exhibit more deviation than the face image L_2 data (Fig. 4), where they both fared well compared to the baseline. This may indicate that $\mathfrak{P}_t$ is more vulnerable to deviation from the model assumptions than $\mathfrak{P}_{\hat{d}}$. A possible explanation for the higher vulnerability of $\mathfrak{P}_t$ is that $\mathfrak{P}_t$ depends on the additional assumption that the errors are normally distributed.

The importance of the satisfaction of model assumptions also means that we should be especially careful when drawing general conclusions from the performance of regression-based indexing on synthetic data sets.

7. CONCLUSION

We have introduced a regression-based indexing scheme that in several of our experiments, also in some with real-world data, provided better query performance than the permutation based scheme [3] we used as a baseline, in some cases even with our scheme's pivot set reduced in order to equalize memory consumption. This indicates that $\mathfrak{P}_{\hat{d}}$ can be a competitive indexing scheme in terms of query performance, and calls for a more thorough investigation.

Disadvantages of our scheme include costly computation involved in building the index and vulnerability to deviation from the model assumptions in the distance spaces to be indexed. The latter consideration highlights the need to evaluate indexing schemes, and especially regression-based ones, on diverse selections of real-world data sets.

8. ACKNOWLEDGEMENTS

The authors would like to thank Svein Erik Bratsberg and Jon Marius Venstad for assistance and discussions.

9. REFERENCES

[1] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. BoostMap: An embedding method for efficient nearest neighbor retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(1):89–104, 2008.

[2] E. Chávez and G. Navarro. A probabilistic spell for the curse of dimensionality. In *Proc. 3rd Workshop on Algorithm Engineering and Experimentation (ALENEX)*, 2001.

[3] E. Chávez, K. Figueroa, and G. Navarro. Effective proximity retrieval by ordering permutations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(9):1647–1658, 2008.

[4] E. Chávez and G. Navarro. Probabilistic proximity search: Fighting the curse of dimensionality in metric spaces. *Information Processing Letters*, 85(1):39–46, 2003.

[5] E. Chávez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[6] J. J. Faraway. *Linear Models with R*. Chapman & Hall/CRC Press, 2005.

[7] D. Harman. Overview of the third text retrieval conference. In *Proc. Third Text REtrieval Conf. (TREC-3)*, 1995.

[8] M. L. Hetland. The basic principles of metric indexing. In *Swarm Intelligence for Multi-objective Problems in Data Mining*. Springer-Verlag, 2009.

[9] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, 28(4):517–580, 2003.

[10] A. Marzal and E. Vidal. Computation of normalized edit distance and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):926–932, 1993.

[11] M. L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, 15(1):9–17, 1994.

[12] R. Paredes and G. Navarro. Optimal incremental sorting. In *Proc. 8th Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2006.

[13] M. Patella and P. Ciaccia. Approximate similarity search: A multi-faceted problem. *Journal of Discrete Algorithms*, 7(1):36–48, 2009.

[14] P. Phillips, H. Wechsler, J. Huang, and P. Rauss. The FERET database and evaluation procedure for face recognition algorithms. *Image and Vision Computing Journal*, 16(5):295–306, 1998.

[15] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2010. ISBN 3-900051-07-0.

[16] T. Skopal and B. Bustos. On nonmetric similarity search problems in complex domains. *ACM Computing Surveys*, to appear.

On Locality-sensitive Indexing in Generic Metric Spaces

David Novak
Masaryk University
Brno, Czech republic
david.novak@fi.muni.cz

Martin Kyselak
Masaryk University
Brno, Czech republic
xkyselak@fi.muni.cz

Pavel Zezula
Masaryk University
Brno, Czech republic
zezula@fi.muni.cz

ABSTRACT

The concept of Locality-sensitive Hashing (LSH) has been successfully used for searching in high-dimensional data and a number of locality-preserving hash functions have been introduced. In order to extend the applicability of the LSH approach to a general metric space, we focus on a recently presented Metric Index (M-Index), we redefine its hashing and searching process in the terms of LSH, and perform extensive measurements on two datasets to verify that the M-Index fulfills the conditions of the LSH concept. We widely discuss "optimal" properties of LSH functions and the efficiency of a given LSH function with respect to *kNN* queries. The results also indicate that the M-Index hashing and searching is more efficient than the tested standard LSH approach for Euclidean distance.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

Keywords

locality-sensitive hashing; metric space; similarity search; approximation; scalability

1. INTRODUCTION

The similarity indexing and searching is a wide research area that gives space to various different paths leading to a solution. A decade ago, Gionis et al. [13] proposed to apply the concept of Locality-sensitive Hashing (LSH) to the search in high-dimensional data and thus started a new research stream. Since then, the concept of LSH-based indexing was further developed and a number of hash functions that preserve the locality of the data have been defined. Current LSH functions are typically defined for specific data types together with given distance functions.

One of other strong streams in this field is indexing based purely on metric properties of the data space [17]. Majority of the structures of this kind are tree-based hierarchical indexes. Recently, M-Index [16] was introduced as a metric-based hash index that can be used both for efficient precise and approximate similarity search. Intuitively, the hashing function defined by the M-Index is locality-preserving and it's approximate algorithm follows the schema of multi-probe LSH searching [14].

In this paper, we focus just on locality-sensitive hashing in generic metric spaces using the M-Index. More specifically, we use the theoretical fundamentals of the LSH concept in order to rigorously study validity of the above-mentioned intuitive statement. The main contributions can be summarized as follows:

- we redefine the indexing and searching process of the M-Index in the terms of the LSH approach;
- we perform extensive measurements on two real-life datasets to verify that the M-Index fulfills the conditions of the LSH concept;
- we put under the same test a standard LSH technique for Euclidean distance and compare it with the M-Index;
- we widely discuss optimality of locality-preserving hash functions and we introduce a new relation that can shed light on the efficiency of a given LSH function with respect to *kNN* queries.

The paper is further organized as follows: In Section 2, we summarize the concept of LSH, as it developed during the last decade. In Section 3, we introduce the M-Index as an LSH technique applicable to a generic metric space. The key Section 4 contains majority of the contribution as specified above, and the work concludes with future work directions in Section 5.

2. LOCALITY SENSITIVE HASHING

The application of Locality Sensitive Hashing (LSH) for indexing and searching high-dimensional data was first proposed by Gionis et al. [13]. The basic idea of this approach is to construct several hash functions to map each (multi-dimensional) point v from domain $\mathcal{U}$ onto a scalar and for each of these hash functions build one hash index on the set of indexed objects $S \subseteq \mathcal{U}$.

The fundamental property of LSH functions $h : \mathcal{U} \rightarrow M$ (where M is a set of buckets) is that the close objects tend to be hashed to the same bucket (they have a higher probability of colliding), whereas objects distant from each other tend

SISAP'10 September 18–19, 2010, Istanbul, Turkey

to be hashed to different buckets. This property is formally defined as follows [13]:

DEFINITION 1. *Let $d : \mathcal{U} \times \mathcal{U} \rightarrow \mathbb{R}$ be a distance function and $\mathcal{B}(u, r) = \{v \in \mathcal{U} | d(u, v) \leq r\}$ for any $u \in \mathcal{U}$. A family $\mathcal{H}$ of functions $h : \mathcal{U} \rightarrow M$ is called (r_1, r_2, p_1, p_2)-sensitive for d if for any $u, v \in \mathcal{U}$:*

- *if $v \in \mathcal{B}(u, r_1)$ then $\mathcal{P}_{\mathcal{H}}[h(u) = h(v)] \geq p_1$,*
- *if $v \notin \mathcal{B}(u, r_2)$ then $\mathcal{P}_{\mathcal{H}}[h(u) = h(v)] \leq p_2$.*

In addition, the LSH function family should satisfy the inequalities $r_1 < r_2$ and $p_1 \gg p_2$.

In practice, achieving high probability of hash collisions for close objects is typically not very difficult; on the other hand, it might be tricky to guarantee that distant objects are hashed to different buckets. Therefore, several hash functions might be grouped to form a new hash function family. Formally, we define [13] a function family $\mathcal{G} = \{g : \mathcal{U} \rightarrow M^K\}$ such that $g(u) = (h_1(u), \ldots, h_K(u))$, where $h_i \in \mathcal{H}$. This concatenation technique thus amplifies the gap between p_1 and p_2.

When building an indexing structure, L functions $g_1, \ldots, g_L$ are selected at random from family $\mathcal{G}$. Then, each object $u \in S$ is hashed into $g_j(u)$ buckets, for all $j = 1, \ldots, L$. When processing a k-nearest neighbors query $kNN(q, k)$, buckets $g_1(q), \ldots, g_L(q)$ are accessed and all objects from these buckets form a candidate set C (duplicated objects are removed). For each $u \in C$, distance $d(q, u)$ is computed and the k most similar objects to q are returned.

As hash $g(q)$ consists of components $h_1(q), \ldots, h_K(q)$, the buckets corresponding to the hash keys that differ from $g(q)$ by ± 1 in one or several these K components are also likely to contain objects near the query point q. This *multi-probe* technique [14] can improve the way of forming the candidate set C and, at the same time, reduce the number of hash tables.

Specific LSH functions were proposed, for instance, for the following distance functions: ℓ_p distance [10] (Minkowski distance of order p), Hamming distance [13] (on binary vectors), Jaccard coefficient [6, 7] (for measuring the similarity of sets), or Arccos [8] (the angle between two vectors). In the following, we describe the LSH function for ℓ_p distances.

LSH Function for ℓ_p Distances

A family of LSH functions for real vectors with ℓ_p distance (denoted as $||\boldsymbol{v_1} - \boldsymbol{v_2}||_p$, $\boldsymbol{v_1}, \boldsymbol{v_2} \in \mathbb{R}^d$) for $p \in (0, 2]$ was proposed in [10]. The technique utilizes the properties of p-stable distributions [18]. First, a random vector $\boldsymbol{a} \in \mathbb{R}^d$ is generated such that its each entry is chosen independently from a p-stable distribution. Then, a dot product $\boldsymbol{a} \cdot \boldsymbol{v}$ projects each vector $\boldsymbol{v} \in \mathbb{R}^d$ to the real line. It follows from p-stability that, for two vectors $\boldsymbol{v_1}, \boldsymbol{v_2} \in \mathbb{R}^d$, the distance between their projections is distributed as $||\boldsymbol{v_1} - \boldsymbol{v_2}||_p X$ where X is the p-stable distribution.

If we "chop" the real line into equi-width segments of appropriate size w and assign hash values to vectors based on which segment they are projected onto, then it is intuitively clear that this hash function will be locality preserving in the sense described above.

Formally, each hash function $h_{\boldsymbol{a},b}(\boldsymbol{v}) : \mathbb{R}^d \rightarrow M$ maps vector $\boldsymbol{v}$ onto a set of integers. Each hash function in the family is specified by a choice of random vector $\boldsymbol{a} \in \mathbb{R}^d$ and a scalar $b \in \mathbb{R}$, where $\boldsymbol{a}$ has its entries chosen independently from a p-stable distribution and b is chosen uniformly from the range $[0, w)$. For given $\boldsymbol{a}, b$, the hash function $h_{\boldsymbol{a},b}$ is defined as $h_{\boldsymbol{a},b}(\boldsymbol{v}) = \lfloor \frac{\boldsymbol{a} \cdot \boldsymbol{v} + b}{w} \rfloor$.

Specifically, for ℓ_2 (Euclidean distance), the entries of vector $\boldsymbol{a}$ can be chosen, e.g. from the Gaussian (normal) distribution, which is 2-stable.

E²LSH Package

The package E²LSH (Exact Euclidean LSH) [1] utilizes the LSH approach described above to solve the randomized version of the r-near neighbor problem ($r > 0$) defined as follows: For a query point $q \in \mathcal{U}$, find all objects $v \in S$ such that $||q - v||_2 \leq r$. The main aim of this implementation is to propose a self-tuning algorithm to compute the optimal values of parameters K and L and build up a suitable LSH indexing structure.

To minimize the size of the candidate set while preserving a reasonable recall level (percentage of precise answer), the algorithm tries to find the parameter values in order to:

- keep K as high as possible (the larger the value of K, the smaller the number of objects in one bucket),
- keep L as low as possible (high values of L increase the search costs and result in large data replication).

In general, the self-tuning process computes these parameters as a function of the data set S, the radius r, and a set of query points. Therefore, every time one of these properties changes significantly, the parameters K and L should be recomputed and the indexing structure rebuilt in order to maintain the performance and recall level.

LSH Forest

The structure LSH Forest [4] was designed to avoid the need of tuning the L and K parameters and reconstructing the whole indexing structure. Instead of assigning hash values of fixed-length $g(u) = (h_1(u), \ldots, h_K(u))$ to objects $u \in S$, the hashes are of *variable length*; specifically, each object's hash value is made long enough to ensure that every point from S has a distinct value. Then a (logical) prefix tree (called LSH tree) can be constructed on the set of all hash values, with each leaf corresponding to an object. The *LSH forest* then consists of L LSH trees. See the original paper for more details [4].

PP-Index

PP-Index [11] is a recent metric-based technique for approximate search that is based on permutation of pivots (or their prefixes). Its keystone is a main memory tree similar to the M-Index cluster tree as introduced below. In contrast to the M-Index, this approach does not explicitly define any hash function and the author does not study the PP-Index as an LSH technique.

3. METRIC-BASED LSH WITH M-INDEX

The Metric Index (M-Index) [16] is an indexing structure for similarity search that is based on hashing principles. In the following, we introduce the M-Index approach to data indexing and searching in terms of the concept of LSH.

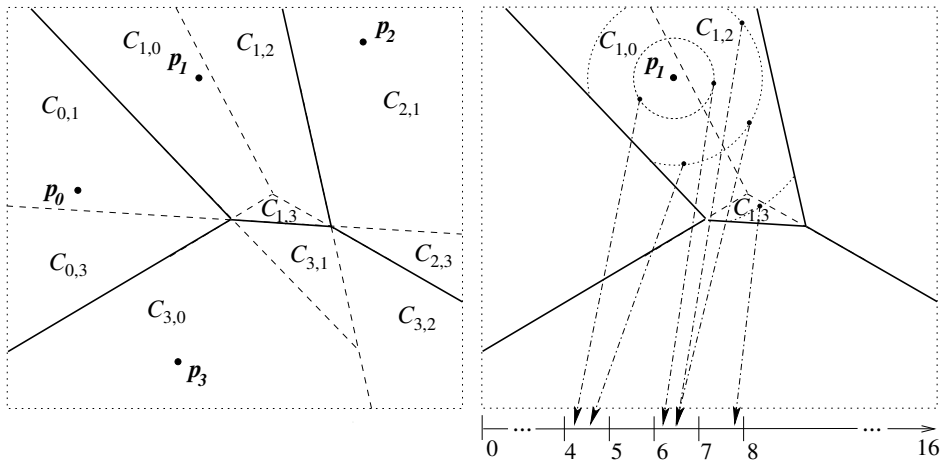


Figure 1: Principles of a two-level M-Index: partitioning (left) and mapping (right).

3.1 Fundamentals

The M-Index treats the data purely as a *metric space* $\mathcal{M} = (\mathcal{U}, d)$, where $\mathcal{U}$ is a *domain* of objects and d is a total *distance function* $d : \mathcal{U} \times \mathcal{U} \longrightarrow \mathbb{R}$ satisfying metric postulates (non-negativity, identity, symmetry, and triangle inequality) [17]. In the following, we assume that this distance function is normalized: $d : \mathcal{U} \times \mathcal{U} \longrightarrow [0, 1)$. In practice, this assumption can be always achieved by dividing the distances by a constant greater than the maximal value of d expected in the data domain.

The M-Index is designed to process standard metric-based similarity queries, in particular the *range query* $R(q, r)$, and *nearest neighbors query* $kNN(q, k)$. These queries can be either processed precisely, which is demanding, or in an approximate manner. In the latter case, the M-Index shares many principles with the locality-sensitive techniques.

3.2 M-Index Mapping

The M-Index defines a *universal mapping schema* from metric space $\mathcal{U}$ to a numeric domain. This mapping uses a fixed set of n reference objects (pivots) $\{p_0, p_1, \ldots, p_{n-1}\}$ and has the ability to preserve locality of the data. In order to formalize this schema, we need a preliminary definition: For an object $v \in \mathcal{U}$, let $(\cdot)_v$ be any permutation of pivot indexes $\{0, 1, \ldots, n-1\}$ such that

$$d(p_{(0)_v}, v) \leq d(p_{(1)_v}, v) \leq \cdots \leq d(p_{(n-1)_v}, v).$$

In other words, sequence $p_{(0)_v}, \ldots, p_{(n-1)_v}$ is ordered with respect to distances between the pivots and object v. The M-Index recursively partitions the data space in a Voronoi-like manner: On the first level, each object $v \in \mathcal{U}$ is assigned to its closest pivot p_i – clusters C_i are formed in this way (in other words, $(0)_v = i$ for all objects $v \in C_i$). On the second level, each cluster C_i is partitioned into $n-1$ clusters by the same procedure using set of $n-1$ pivots $\{p_0, \ldots, p_{i-1}, p_{i+1}, \ldots, p_{n-1}\}$ creating clusters $C_{i,j}$ where j is index of the second closest pivot to objects in cluster $C_{i,j}$, i.e. $(1)_v = j$. Figure 1 (left) shows an example of the M-Index partitioning for two levels. This partitioning process is repeated l-times in M-Index with l levels, for any l: $1 \leq l \leq n$.

The l-level M-Index further defines a mapping of the data space to a numeric domain $h : \mathcal{U} \longrightarrow [0, n^l]$ where the integral part of $h(v)$, $v \in \mathcal{U}$ results from a numbering schema of the clusters; specifically, cluster $C_{i_0,i_1,\ldots,i_{l-1}}$ is assigned number:

$$cluster(C_{i_0,i_1,\ldots,i_{l-1}}) = \sum_{j=0}^{l-1} i_j \cdot n^{(l-1-j)} \ . \qquad (1)$$

The fractional part of $h(v)$ is the distance between the object v and its closest pivot $d(p_{(0)_v}, v)$. Altogether:

$$h(v) = cluster(C_{(0)_v,(1)_v,\ldots,(l-1)_v}) + d(p_{(0)_v}, v) \ . \qquad (2)$$

Figure 1 (right) sketches this M-Index mapping principle (only for clusters $C_{1,j}$) – see the M-Index paper [16] for further details. The M-Index as described so far has a *static* partitioning and mapping for a given level l. Since neither the data distribution nor the space partitioning are uniform, the M-Index has a *dynamic* variant that further partitions only clusters that exceed certain data volume limit. In this case, the M-Index maintains a dynamic *cluster tree* to keep track of actual depth for individual clusters. The formula for $h(v)$ calculation is modified to take into account actual tree-depth [16] – the tree has an a priori given maximal level $1 \leq l_{\max} \leq n$. Such key-assignment approach can be considered a variant of *extensible hashing* [12]. The schema of this tree-like structure for $l_{\max} = 3$ is sketched in Figure 2.

Note that the M-Index clusters naturally correspond to LSH buckets. Taking the number of pivots n and the level $l_{\max}$ as parameters, such hash functions h form a family $\mathcal{H}^n_{l_{\max}}$ of M-Index mapping functions. Individual members

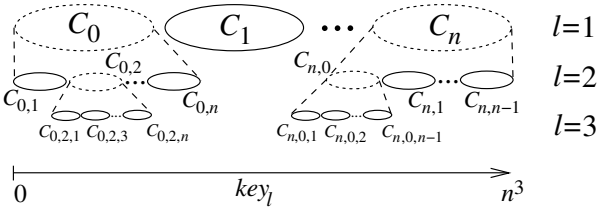


Figure 2: Dynamic cluster tree, $l_{\max} = 3$.

Symbol	Meaning
$\mathcal{U}$	domain of objects
S	set of indexed objects $S \subseteq \mathcal{U}$
M	set of buckets
d	distance function $d : \mathcal{U} \times \mathcal{U} \longrightarrow \mathbb{R}$
h	hash function $h \in \mathcal{H}$, $h : \mathcal{U} \rightarrow M$
K	number of concatenated functions/values
g	concatenated hash f. $g(u) = (h_1(u), \ldots, h_K(u))$
L	number of hash tables/hash functions
k	number of nearest neighbors for $kNN(q, k)$
C	set of candidate objects $C \subseteq S$
n	number of pivots
l	number of M-Index levels, $l \in \mathbb{N}$
$l_{\max}$	maximum level of dynamic M-Index
$\mathcal{H}^n_{l_{\max}}$	family of M-index hash functions

Table 1: Symbols used in this work.

of this family are defined by specific sets of pivots $p_0, \ldots, p_{n-1}$ selected randomly uniformly from the dataset. In the terminology introduced in Section 2, the M-Index does not use concatenation of several h function, i.e. $g(v) = h(v)$. For a better orientation in the symbols used throughout this paper see Table 1.

3.3 M-Index Searching

The M-Index precise evaluation strategies for similarity queries synergically exploit practically all known metric-based principles of data partitioning, pruning and filtering, which makes it relatively very efficient [16]. The approximate evaluation strategy for $kNN(q, k)$ queries follows the general schema of the multi-probe LSH with $L = 1$ (see Section 2): A set of query candidate objects is formed by data from cluster corresponding to key $h(q)$ (let us denote this cluster C_q) and an optional number of clusters that should contain objects "close to object q". These partitions are identified by a heuristic based purely on analysis of the $d(p_0, q), \ldots, d(p_{n-1}, q)$ distances. Cluster C_q gets the highest priority – it is assigned *penalty* equal to 0. Cluster C_q has the smallest sum of distances $d(p_{(0)_q}, q) + \cdots + d(p_{(l-1)_q}, q)$ and this sum grows for other clusters. This is reflected by the penalty in order to express the "proximity" of the cluster to q. Specifically:

$$penalty_q(C_{i_0,\ldots,i_{l-1}}) = \\ = \frac{1}{l} \cdot \sum_{j=0}^{l-1} \max\left\{d(p_{i_j}, q) - d(p_{(j)_q}, q), 0\right\}. \quad (3)$$

Note that the penalty is normalized by $1/l$ in order to make its values comparable for clusters on different levels.

By this mechanism, M-Index can construct the set of candidate objects of arbitrary size. In other words, some *virtual buckets* specific to given query object are created at search time. This is a straightforward way to effectively tune the result quality (answer recall) versus the searching costs (I/O or computational).

4. LSH PROPERTIES OF M-INDEX

Current LSH techniques are typically defined for a given data type with a specific distance function. The M-Index has the potential to be a universal LSH technique for *any metric space* with the option of multi-probe bucket access. The objective of this section is to verify that the M-Index matches the LSH theory and to compare its qualities with established LSH methods. We also discuss practical impacts of the LSH definition in context of *kNN* queries.

For LSH functions defined for a specific data space, there typically exists a formula to show that the probability of two points colliding in the hash index is decreasing with their mutual distance (Definition 1). We do not induce such a general rule for the M-Index, as we can hardly assume any properties of the dataset in a generic metric space.

Instead, we rather perform measurements to assess this collision probability on real-life datasets for various M-Index settings and we put a standard LSH technique under the same test.

Experiment Setting

We performed the experiments on two datasets: The *CoPhIR* dataset [5, 3] consists of five MPEG-7 [15] visual descriptors extracted from digital images (280 dimensions altogether). Each of the descriptors is compared by a metric function and these partial distances are aggregated into a single metric space [2]. We also use the set of *Color Structure* (CS) descriptors from CoPhIR as a separate dataset. The CS descriptor is a 64-dimensional vector and we use ℓ_2 metric as the distance function, so it can be indexed by standard LSH techniques. We test various sizes of these datasets up to one million objects.

In order to measure the probability of collisions, we have selected one hundred query objects uniformly at random from the dataset and, for a given structure, we always test hash collisions of these queries with all objects in the database. We report percentage of hash collisions depending on the objects mutual distances (the scale is quantized). In this way, the whole distance scale is covered by the measurements.

Collision Probability Curves

Let us analyze the collision probability of LSH function h using the notation from Definition 1, which defines an (r_1, r_2, p_1, p_2)-sensitive function. Figure 3 schematically depicts the relationship between these four parameters: The collision probability curve of an LSH hash function should run through the grey area.

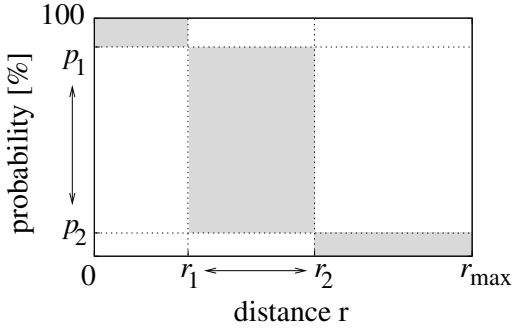


Figure 3: Collision probability case study.

Let us discuss an ideal shape of LSH function h: Probability p_1 should be as high as possible for such distances r_1 that will be used as query radii (distances to the k-th nearest neighbors, for $kNN(q,k)$ queries). Probability p_2 should be as low as possible for distances r_2 that are "not interesting for querying". If the collision probability is high for distances we are not interested in, then bucket $h(q)$ will contain many objects to be added to the candidate set and filtered out later, which increases the search costs. Thus, an ideal LSH function would decrease steeply between distances r_1 and r_2 (their difference would be small).

4.1 Basic LSH Properties

First, let us investigate the collision probability of the standard LSH scheme for ℓ_2 metric [10] (see Section 2). For this purpose, we utilize package E^{2}LSH [1] proposed by the authors. From this implementation, we only use the mechanism to generate member functions from the LSH family and to calculate hash keys for given vectors. We do not let E^{2}LSH set the K and L parameters for specific queries, because we want to study the LSH properties under various conditions and for various settings. Further on, we refer to this technique as E^{2}LSH, and we analyze it for various number of concatenated LSH functions (K) focusing on behavior of individual functions ($L = 1$). Figure 4 shows results for 500,000 objects from the Color Structure dataset.

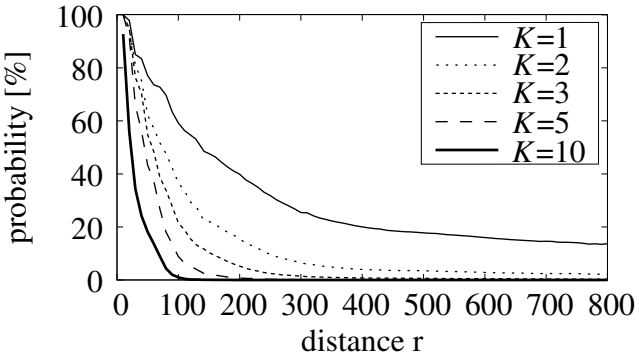


Figure 4: Collision probability of the E^{2}LSH indexes with 500,000 Color Structure objects.

The concatenation of more hash functions (increasing parameter K) results in shrinkage of the collision probability [13] – this effect can be observed well in the graph. We can see that for $K > 3$ the curves are very steep – only objects with distance up to 100 have a notable change to collide. Further in this section, we study what query radii are meaningful for this dataset and we analyze impact the collision probability on the *kNN* search efficiency. Each E^{2}LSH graph is averaged over five instances from the same LSH family – the results were stable among different instances.

Figure 5 presents results of the same experiment on the dynamic M-Index with $l_{\max} = 6$ for various number of pivots n. We can see that the collision probabilities are strictly monotonic with respect to mutual distance, which classifies the M-Index as an LSH technique according to Definition 1. The graphs are again averaged over several instances from family $\mathcal{H}^n_{l_{\max}}$ defined by specific sets of randomly chosen pivots. The dynamic hashing principle of the M-Index results in buckets of limited sizes – 1,000 objects, in our case. Due to this fact, the differences between individual curves

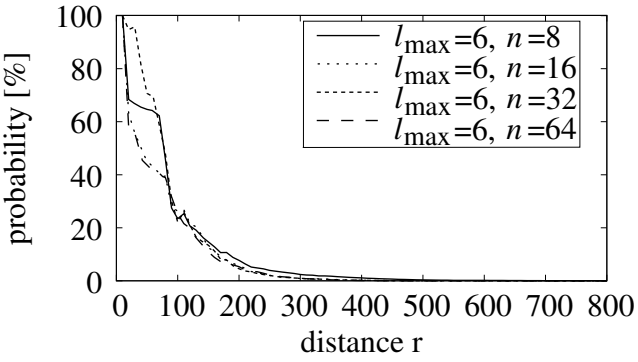


Figure 5: Collision probability of the M-Index with 500,000 Color Structure objects.

in Figure 5 are less significant than in static E^{2}LSH hashing, where higher values of K result in lower numbers of collisions (Figure 4). In the following measurements, we use M-Index functions from family $\mathcal{H}^{32}_6$ ($n = 32$) with cluster capacity of 1,000 objects.

The multi-probe approach, used by the M-Index search mechanism, defines virtual buckets of adjustable sizes (see Section 3.3). In Figure 6, we demonstrate collision probabilities of M-Index considering objects in a virtual bucket as colliding. We can see a significant increase of the probability for smaller mutual distances while the differences are smaller for more distant objects. This trend seems to be more convenient than the trend observed in Figure 4 – we elaborate on this topic more in the following section.

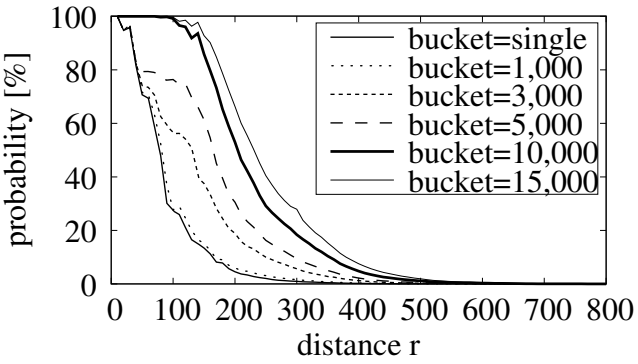


Figure 6: Collision probability of the M-Index with virtual buckets on 500,000 CS objects.

In order to fairly compare results of the collision experiment on the E^{2}LSH and the M-Index, we should put together settings with buckets of similar sizes, because these sizes directly influence the number of objects accessed during query execution. Table 2 shows average sizes of buckets hit by query objects during the measurements on E^{2}LSH indexes. The percentages in this table are stable for all set sizes, because the smaller datasets were created from the 1,000,000 collection as successive subsets by random uniform sampling. For the M-Index, the bucket size is directly tunable by the concept of virtual buckets.

The graph in Figure 7 compares E^{2}LSH with $K = 3$ (accessing 0.57 % of the 500,000 dataset $\approx$ 2,900 objects) with the M-Index accessing up to 3,000 objects. We can see a

concatenation parameter	$K=1$	$K=2$	$K=3$	$K=4$	$K=5$	$K=10$
average collision size	17.07 %	3.24 %	0.57 %	0.12 %	0.03 %	0.0007 %

Table 2: Average sizes of buckets hit by query objects on E^{2}LSH indexes (percentage of dataset size).

significant difference in the collision probability for lower distances; this probability for distances over 450 is zero for the M-Index and slightly above the zero level for the E^{2}LSH (the overall absolute number of collisions is the same for both curves). In the following we discuss the influence of this property on *kNN* search.

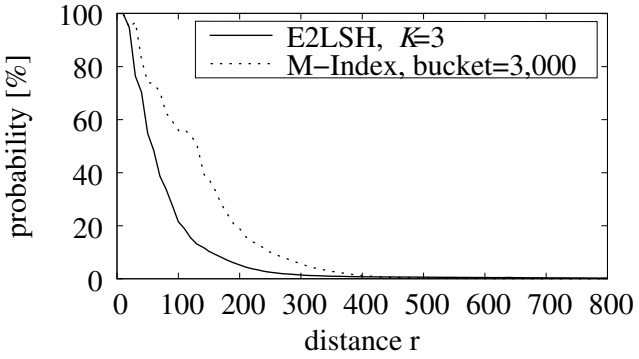


Figure 7: Collision probability of the M-Index and the E^{2}LSH with similar bucket sizes (3,000 objects) on 500,000 Color Structure objects.

4.2 LSH for Nearest-neighbors Search

This section tries to shed light on influence of the collision probability on the efficiency of $kNN(q,k)$ search using a specific LSH index. As we discussed in the previous section, an "ideal" shape of the collision probability curve depends on query radii – distances between the query objects and k-th nearest neighbors for $k \in \mathbb{N}$ of our interest. Figure 8 depicts average distances to the k-th nearest neighbor in the Color Structure dataset. Note the logarithmic scale on the x axis and the expected trend of the NN distances decreasing as the data space gets denser for larger data collections.

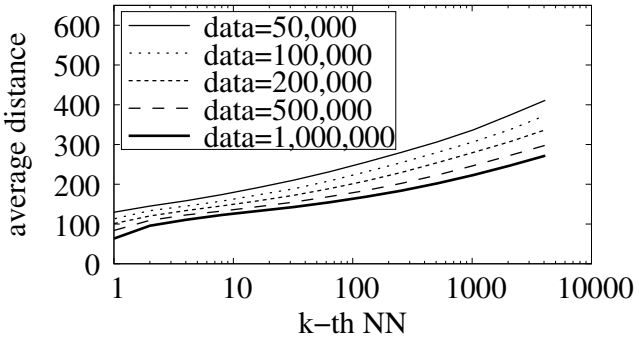


Figure 8: Average distances to k-th nearest neighbor for the Color Structure dataset.

This graph indicates that, roughly speaking, distances between 100 and 200 are interesting from the practical *kNN* point of view (k up to 100), which are exactly the distances for which the E^{2}LSH and the M-Index differ most significantly according to Figure 7. We would like to quantify this observation precisely, eliminating impreciseness that emerges from averaging. For this purpose, we introduce relation of two objects $p, q \in S$ being *k-th nearest neighbors* (there exist $k-1$ objects from S that are closer to q than object p, or vice versa). Note that the nearest neighbor rank of an object p from object q is not necessarily the same as of object q from object p.

Definition 2. *Given metric space $\mathcal{M} = (\mathcal{U}, d)$, a finite set of indexed objects $S \subseteq \mathcal{U}$ and two objects $p, q \in S$, let us denote $N_{p(q)} = \{u \in S | d(p,u) < d(p,q)\}$. We say that p, q are* k-th nearest neighbors *for $k \in \mathbb{N}$, iff*

$$\min\left\{|N_{p(q)}|, |N_{q(p)}|\right\} = k - 1.$$

In the following test, we measure probability of hash collisions of pairs of objects that are k-th nearest neighbors. In compliance with the previous measurements, we always consider objects from the whole dataset for each of one hundred randomly chosen query objects and, thus, the whole scale of k-th NN is covered. Graph in Figure 9 compares this collision probability on the E^{2}LSH and the M-Index for variable $k \in \mathbb{N}$. These results confirm the intuition that the E^{2}LSH and the M-Index collision probabilities differ significantly for distances important for searching in practice.

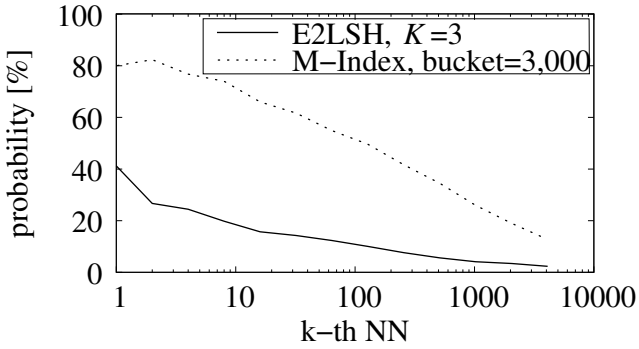


Figure 9: Collision probability for pairs of k-th nearest neighbors in the E^{2}LSH and the M-Index on 500,000 CS objects.

Let us inspect the scalability of the M-Index approach by fixing the size of the virtual bucket to 5,000 objects and observing the collision probability for dataset of variable sizes – see Figure 10. The probabilities are lower for larger datasets but, as seen from Figure 8, nearest neighbor distances shrink as datasets grow.

The exact influence of this trend is observable in the k-th nearest neighbor variant of this experiment in Figure 11. This graph actually corresponds to recall of $kNN(q,k)$ search with a single LSH index ($L = 1$) with constant search costs. For $k = 100$, the recall would be over 90 % on 50,000 dataset and about 70 % for one million Color Structure dataset.

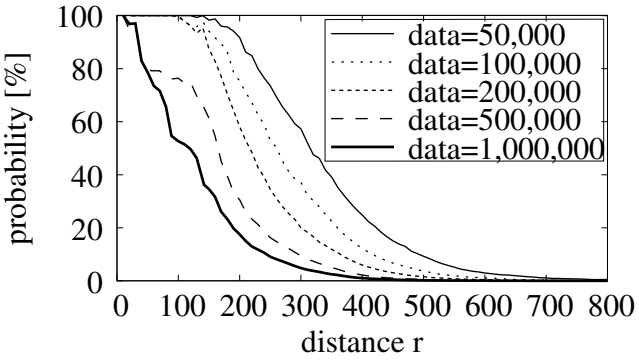


Figure 10: Collision probability of the M-Index with buckets with 5,000 objects on CS dataset.

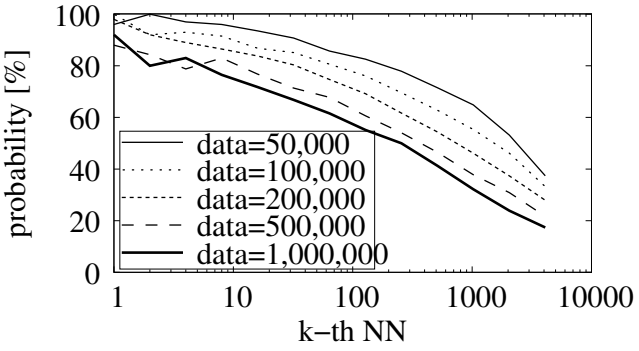


Figure 11: Collision prob. for pairs of k-th NN for CS dataset with virtual buckets of size 5,000.

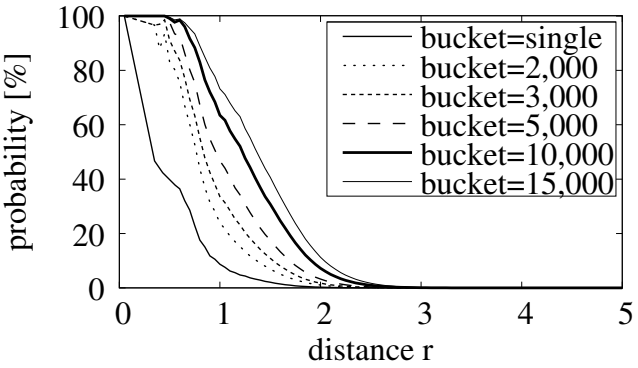


Figure 12: Collision probability for M-Index on CoPhIR dataset with 500,000 objects.

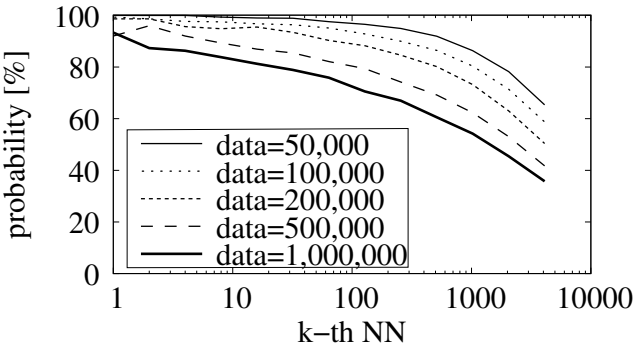


Figure 13: Collision prob. for pairs of k-th NN for CoPhIR dataset with virtual buckets of size 10,000.

4.3 M-Index on Complex Dataset

One of the main advantages of the M-Index in comparison with other LSH approaches is the ability to index any metric space. The distance function used with the CoPhIR dataset [5] is a complex, hierarchical weighted aggregation of several partial l_1 and l_2 functions (or their weighted modifications); such dataset cannot be directly processed by any standard LSH method and its intrinsic dimensionality [9] has been measured as 12.9 [3], which makes it rather difficult to index and search. Let us observe behavior of the M-Index on this data type.

We have tested several families of M-Index hash functions $\mathcal{H}^n_{l_{\max}}$, but we only report results for $\mathcal{H}^{32}_6$ with maximal bucket size of 1,000 objects – the differences between results for various families were only subtle (similarly as for the Color Structure dataset). Figure 12 shows the collision probability for various sizes of virtual buckets. We can see that the basic property of LSH functions according to Definition 1 is fulfilled and, intuitively, the shape of the curves is convenient for search efficiency.

Using Definition 2, we have measured collision probability of random k-th nearest neighbors with respect to $k \in \mathbb{N}$. Figure 13 presents the results using virtual buckets with 10,000 objects. Considering, for instance, the dataset with 1,000,000 CoPhIR objects, these curves correspond to the recall of $kNN(q,k)$ queries probing 1 % of the dataset.

4.4 Search Efficiency for Multiple Indexes

In this section, we study efficiency of the search (costs vs. recall) when multiple M-Indexes are used (parameter L). By means of the virtual buckets, we can precisely tune the number of accessed objects in particular index, thus influence the overall search costs, and observe the tradeoff between these costs and recall for various values of L.

Figure 14 presents the results of $kNN(q,30)$ search in 1,000,000 CoPhIR dataset using one, two, and three M-Indexes ($L = 1, 2, 3$). The x-axis indicates the *overall* number of accessed objects and the y-axis the recall averaged over hundred randomly selected query objects.

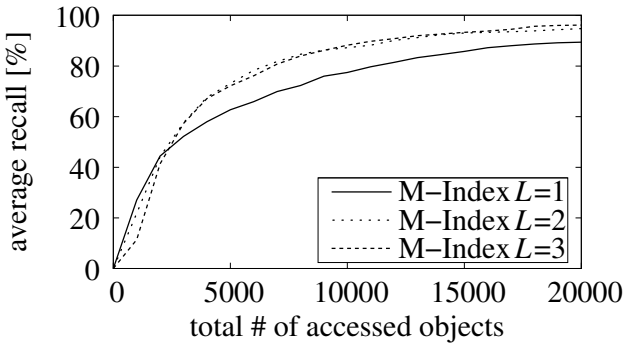


Figure 14: Average recall for various number of M-Indexes for 1,000,000 CoPhIR dataset.

We can see that the usage of multiple indexes started to be superior over one index at the level of approximately 50% recall. On the other hand, usage of more than two indexes does

not seem to improve the performance significantly. In the future, we would like to study influence of multiple indexes on recall variance and other statistical properties. Replication of the data in several indexes also offers the option of better fault tolerance and of parallel processing of the query.

5. CONCLUSION

The concept of Locality-sensitive Hashing has been successfully used for similarity indexing. In the last decade, several LSH functions were defined for specific distance measures. In this work, we have shown that the recently proposed structure M-Index [16] has the properties of an LSH technique while it is applicable to any data space with metric properties. The M-Index hashing is dynamic and the search algorithm implements the concept of multi-probe LSH.

We conduct experiments on real-life data in order to measure the collision probability of the M-Index hash function. This property is the basic LSH quality measure and we compare it with a standard LSH function for ℓ_2 distance. We widely discuss the shapes of these probability curves with respect to similarity queries in practice and we measure the collisions of pairs of objects being k-th nearest neighbors. This measure seems to reflect well the search efficiency of given LSH function and it indicates that the quality of the M-Index hashing and its multi-probe approach are superior to the tested ℓ_2 LSH function. Moreover, the M-Index can precisely tune the search costs at query time. When we increase the size of the dataset while keeping the costs constant, the query recall degrades only modestly.

In the future, we would like to study more the relationship between the multi-probe parameter (virtual bucket sizes) and various number of indexes (parameter L). Namely, we plan to focus on the variance and other statistical properties of the query recall with respect to search costs and the value of L. We would also like to compare the M-Index approach with standard multi-probe LSH techniques and with the LSH Forest. Using the distributed version of the M-Index, we plan to investigate the point of fault tolerance gained by data replication in a dynamic distributed system.

6. ACKNOWLEDGMENTS

This work has been supported by national research projects MSMT 1M0545, GACR 201/09/0683, GACR 103/10/0886, and GACR P202/10/P220. The hardware infrastructure was provided by the METACentrum under the research intent MSM6383917201.

7. REFERENCES

[1] A. Andoni and P. Indyk. E2LSH: Exact Euclidean locality-sensitive hashing, 2004. http://web.mit.edu/andoni/www/LSH/.

[2] M. Batko, F. Falchi, C. Lucchese, D. Novak, R. Perego, F. Rabitti, J. Sedmidubsky, and P. Zezula. Building a Web-scale image similarity search system. *Multimedia Tools and Applications*, 2009.

[3] M. Batko, P. Kohoutkova, and D. Novak. CoPhIR image collection under the microscope. In *SISAP '09: Proceedings of the 2009 Second International Workshop on Similarity Search and Applications*, pages 47–54, Washington, DC, USA, 2009. IEEE Computer Society.

[4] M. Bawa, T. Condie, and P. Ganesan. LSH forest: Self-tuning indexes for similarity search. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 651–660, New York, NY, USA, 2005. ACM.

[5] P. Bolettieri, A. Esuli, F. Falchi, C. Lucchese, R. Perego, T. Piccioli, and F. Rabitti. CoPhIR: A test collection for content-based image retrieval. *CoRR*, abs/0905.4627v2, 2009.

[6] A. Broder. On the resemblance and containment of documents. In *SEQUENCES '97: Proceedings of the Compression and Complexity of Sequences 1997*, page 21, Washington, DC, USA, 1997. IEEE Computer Society.

[7] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Comput. Netw. ISDN Syst.*, 29(8-13):1157–1166, 1997.

[8] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, NY, USA, 2002. ACM.

[9] E. Chávez and G. Navarro. Measuring the dimensionality of general metric spaces. Technical Report TR/DCC-00-1, Department of Computer Science, University of Chile, 2000.

[10] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, New York, NY, USA, 2004. ACM.

[11] A. Esuli. PP-Index: Using permutation prefixes for efficient and scalable approximate similarity search. In *Proceedings of LSDS-IR'09*, pages 17–24, 2009.

[12] R. Fagin, J. Nievergelt, N. Pippenger, and H. R. Strong. Extendible hashing – A fast access method for dynamic files. *ACM Transactions on Database Systems*, 4(3):315–344, 1979.

[13] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[14] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe LSH: Efficient indexing for high-dimensional similarity search. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 950–961. VLDB Endowment, 2007.

[15] MPEG-7. Multimedia content description interfaces. Part 3: Visual. ISO/IEC 15938-3:2002, 2002.

[16] D. Novak and M. Batko. Metric index: An efficient and scalable solution for similarity search. In *SISAP '09: Proceedings of the 2009 Second International Workshop on Similarity Search and Applications*, pages 65–73, Washington, DC, USA, 2009. IEEE Computer Society.

[17] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach.* Springer, 2006.

[18] V. M. Zolotarev. One-dimensional stable distributions. In *Vol. 65 of Translations of Mathematical Monographs.* American Mathematical Society, 1986.

On Locality Sensitive Hashing in Metric Spaces

Eric Sadit Tellez
sadit@lsc.fie.umich.mx
Universidad Michoacana, México

Edgar Chavez
elchavez@umich.mx
Universidad Michoacana / CICESE, México

ABSTRACT

Modeling proximity search problems as a metric space provides a general framework usable in many areas, like pattern recognition, web search, clustering, data mining, knowledge management, textual and multimedia information retrieval, to name a few. Metric indexes have been improved over the years and many instances of the problem can be solved efficiently. However, when very large/high dimensional metric databases are indexed exact approaches are not yet capable of solving efficiently the problem, the performance in these circumstances is degraded to almost sequential search.

To overcome the above limitation, non-exact proximity searching algorithms can be used to give answers that either in probability or in an approximation factor are close to the exact result. Approximation is acceptable in many contexts, specially when human judgement about closeness is involved.

In vector spaces, on the other hand, there is a very successful approach dubbed Locality Sensitive Hashing which consist in making a succinct representation of the objects. This succinct representation is relatively insensitive to small variations of the locality. Unfortunately, the hashing function have to be carefully designed, very close to the data model, and different functions are used when objects come from different domains.

In this paper we give a new schema to encode objects in a general metric space with a uniform framework, independent from the data model. Finally, we provide experimental support to our claims using several real life databases with different data models and distance functions obtaining excellent results in both the speed and the recall sense, specially for large databases.

Categories and Subject Descriptors

E.2 [**Data**]: DATA STORAGE REPRESENTATIONS—*Hash-table representations*; H.3.3 [**Information Systems**]: Information search and retrieval—*Search process*

SISAP 2010, September 18-19, 2010, Istanbul, Turkey.

1. INTRODUCTION

Proximity searching appears in a large number of applications ranging from pattern recognition, web searching, clustering, data mining, knowledge management and textual and multimedia information retrieval, to name a few.

The most general setup to state the proximity searching problem is the (dis)similarity space model. A (dis)similarity space is a pair $(\mathbb{X}, d)$ with $\mathbb{X}$ a set and $d(\cdot,\cdot)$ a positive real valued function between objects denoted by $d : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$. A finite subset $\mathbb{U} \subseteq \mathbb{X}$ is the database. The proximity searching problem of interest for us is to obtain the k-nearest neighbors of a query $q \in \mathbb{X}$ defined as $KNN(q) = \{x \in \mathbb{U} | d(x,q) \leq d(y,q) \forall y \in \mathbb{U}\}$ and $|KNN(q)| = k$. If $d(\cdot,\cdot)$ obeys the metric properties as follows: for any $x, y, z \in \mathbb{X}$, $d(x,y) > 0$, $d(x,y) = 0 \iff x = y$, $d(x,y) = d(y,x)$ (symmetry), and the triangle inequality: $d(x,z) + d(z,y) \geq d(x,y)$ then the pair $(\mathbb{X}, d)$ is called a *metric space*. A more restrictive setup consist in demanding the objects to have coordinates and the distance to be an ℓ_p metric, in this case the objects will be either vectors or a vector representation of the objects.

Proximity queries can be solved trivially by measuring the distance from the query to every database element. If the database is large and/or the distance function is expensive this sequential scan is no acceptable and an index should be used to obtain sublinear query times. As reported in many places [11, 7, 3], sublinear query times are only achievable in either low-dimensional vector spaces or intrinsically low dimensional metric spaces.

To alleviate the above described problem, authors have resorted to approximate or probabilistic solutions to the proximity problem as surveyed in [10]. In the probabilistic setup the query time can be speedup if instead of searching for the correct solution set $KNN(q)$ we accept a set $\widehat{KNN}(q)$ such that $\frac{KNN(q) \cap \widehat{KNN}(q)}{k} \geq \delta$ for a given $\delta < 1$. In other words, we accept up to a small number of false positives in exchange of speeding up the query process.

Probably the most well known probabilistic solution for the proximity searching problem in the context of high dimensional vector spaces is the *locality sensitive hashing* (LSH) (described below in 1.1) which consist in projecting the original high dimensional space to a lower dimensional space obtained as a random sample of the original coordinates. For the metric space searching model, a successful probabilistic approach is to use *permutations* (described below in 1.2) as object descriptors and to search in the representation space instead of searching in the original database, provided a distance between permutations is defined. The key difference

between both approaches is that a proximity searching is converted on the one hand, into a set of *exact* searches in the LSH approach, and on the other hand into a simpler proximity search in the permutations space.

A number of adhoc indexes have been designed to speedup the searching in the permutations space without a sequential scan. The pp-index approach consist in building a compact trie with the prefixes of the permutations, and obtaining as candidate answer to the query the subtree matching the prefix of the query, as described in [8]. The metric inverted index compute an approximation to the distance between permutations using an inverted index [1].

1.1 Locality Sensitive Hashing

The idea behind LSH is to pack a proximity searching problem into an exact searching problem using the following guideline:

Definition A family $\mathcal{H}$ of functions $h : R^d \to U$ is called $(P1, P2, r, cr)$-sensitive, if for any p, q:

- if $||p - q|| < r$ then $Pr[h(p) = h(q)] > P1$
- if $||p - q|| > cr$ then $Pr[h(p) = h(q)] < P2$

The above implies using exact matching only (comparing $h(q)$ and $h(p)$ with q the query and p the database elements), which can be accomplished in constant time, to solve a proximity searching. The most well studied case for LSH is the Hamming space which boils down to using random subsampling of bit-strings as described in [2]. A downside of LSH is the need to design a hashing function $\mathcal{H}$ with particular properties for each data model. In particular, for a general metric space, finding the bounding probabilities can be very hard.

To fix ideas, lets see an example with the binary case. Let $g_{xyz}(s)$ be a function copying and concatenating the x, y, z-th bits of s, and $\mathcal{H} = \{g_{152}, g_{743}\}$. Consider u, v, w in $\mathbb{U}$ in a d dimensional binary hamming space, the $Pr[g_x(u) = g_x(v)] = 1 - D(u, v)/d$ where D is the hamming distance. Then $Pr[g_{xyz}(u) = g_{xyz}(q)] = 1 - (D(u, v)/d)^3$ must follows the properties of definition 1.1. Let $u = 10010101$, $v = 11010000$, $w = 01001101$. Exemplifying, $D(u, v) = 3$, $D(u, w) = 4$, $D(v, w) = 5$, our hashing functions gives $g_{152}(u) = 100$, $g_{152}(v) = 101$, $g_{152}(w) = 011$, $g_{743}(u) = 010$, $g_{743}(v) = 010$, $g_{743}(w) = 000$. We can see that $g_{743}(u) = g_{743}(v) = 010$ whose are in fact the closer pairs in the triplet.

For vector spaces in L_p we can assume integer coordinates without loss of generality. This case reduces to the Hamming space by concatenating the coordinates in fixed sized unary representation per coordinate, and then use Hamming distance to compute the distance. The L_2 norm can be embedded into L_1 using projections. In general L_p can be projected in any other L_s with $s < p$ using random projections, as depicted in [2].

1.2 Overview of the Permutations Index

The motivation behind this indexing method [6] is to shift the problem of comparing directly the query object against every object in the database to comparing the *perspective* in which a set of elements is perceived. Each database element has an unique perspective of the *permutants* (defined below) and the query is only compared to those elements having similar perspective of the permutants.

Let $\mathbb{U}$ be the database of objects, and $\mathbb{P} \subseteq \mathbb{U}$ be a set of distinguished objects from the database, called *permutants*. Each $u \in \mathbb{U}$ defines a *permutation* Π_u, where the elements of $\mathbb{P}$ are written in increasing order of distance to u. Ties are broken using any consistent order, for example, the order of the elements in $\mathbb{P}$.

Definition Let $\mathbb{P} = \{p_1, p_2, \ldots, p_k\}$ and $u \in \mathbb{S}$. Then we define Π_u as a permutation of $(1 \ldots k)$ so that, for all $1 \leq i < k$ it holds either $d(p_{\Pi_u(i)}, x) < d(p_{\Pi_u(i+1)}, x)$, or $d(p_{\Pi_u(i)}, x) = d(p_{\Pi_u(i+1)}, x)$ and $\Pi_u(i) < \Pi_u(i + 1)$.

Each database element u is represented by a permutation Π_u. The query is represented by Π_q using the same definition. Elements that are close have similar permutations. We define what we mean by similar permutations as follows.

Sum the squares of differences in the relative positions of each element in both permutations. That is, for each $p_i \in \mathbb{P}$ we compute its position in Π_u and Π_q, namely $\Pi_u^{-1}(i)$ and $\Pi_q^{-1}(i)$, and sum up the squares of the differences in the positions [6].

Definition Given permutations Π_u and Π_q of $(1 \ldots k)$, Spearman Rho is defined as

$$S_\rho(\Pi_u, \Pi_q) = \sum_{1 \leq i \leq k} \left(\Pi_u^{-1}(i) - \Pi_q^{-1}(i)\right)^2.$$

Note that we can compute $S_\rho(\Pi_q, \Pi_u)$ by obtaining the inverse of both permutations and then computing the Euclidean distance of the inverse. It is also shown in [6] that we can use the sum of the absolute of the differences, without the squares, without noticeable penalization in the index recall.

The result is a table of n rows (one per database element) and k columns (one per permutant). Each cell needs $\lceil \log_2 k \rceil$ bits to store one permutation at each row. The indexing cost is kn distance computations plus $O(nk \log k)$ CPU time to sort all the permutations.

The search has two phases. The first sorts the database according to the permutation distance and selects as candidates the first elements. The second phase is to check the list. The permutation index allows KNN searches in pseudo-metric spaces, because the triangle inequality is not used explicitly. Our technique inherits this property allowing faster searches and smaller indexes.

Certain approximations to searching in the permutations space can be done as explored in [8, 1]. We describe below another alternative leading directly to a binary reduction of the problem which is suitable for LSH indexing with the Hamming distance.

1.3 Brief Permutations

A permutation can be represented in a binary string, using half a bit for each permutant. The idea for this representation is to discretize the *displacement* of each permutant from the canonical position (the identity permutation). This idea is exploited in [12] and described in algorithm 1 where the *Encode* function pack the permutation into a bit string. Notice that for large enough m (e.g $m \geq \frac{|P|}{2}$) the permutants in the center will be rarely set to 1. In order to use more effectively the space, we can pack a double number of permutants by swapping the central part, as described in 2.

Algorithm 1 Bit-encoding of the permutation P under the module m

Encode(Permutation P, Positive Integer m)

```
1: Let P^{-1} be the inverse P.
2: C ← 0 {Bit string of size |P|, initialized to zeros}
3: for all i from 0 to |P| − 1 do
4:    if |i − P^{-1}[i]| > m then
5:       C[i] ← 1
6:    end if
7: end for
8: return C
```

Algorithm 2 Bit-encoding permutations with swapping the central part

EncodePC(Permutation P, Positive Integer m)

```
1: Let P^{-1} be the inverse P
2: C ← 0^{|P|} {Bit string of size |P|, initialized to zeros}
3: M = ⌊|P|/4⌋
4: for all i from 0 to |P| − 1 do
5:    I ← i
6:    if ⌊I/M⌋ mod 3 ≠ 0 then
7:       I ← I + M
8:    end if
9:    if |I − P^{-1}[i]| > m then
10:      C ← C|(1 << i)
11:   end if
12: end for
13: return C
```

As a forming example to fix ideas, let $m = 2$, $u = (3, 6, 2, 1, 5, 4)$, $r = (5, 3, 1, 6, 2, 4)$ and $q = (6, 2, 3, 1, 4, 5)$. After the inverse $u^{-1} = (4, 3, 1, 6, 5, 2)$, $r^{-1} = (3, 5, 2, 6, 1, 4)$ and $q^{-1} = (4, 2, 3, 5, 6, 1)$. Applying algorithm 1 we have $\hat{u} = (|1-4| > m, |2-3| > m, |3-1| > m, |4-6| > m, |5-5| > m, |6-2| > m) = (1, 0, 0, 0, 0, 1)$, supposing $|a-b| > m$ evaluates to 1 for true and 0 for false. Similarly, we obtain $\hat{r} = (0, 1, 0, 0, 1, 0)$ and $\hat{q} = (1, 0, 0, 0, 0, 1)$. If H is the hamming distance, $H(\hat{u}, \hat{q}) = 0$ and $H(\hat{r}, \hat{q}) = 2$. Clearly, q is to u, and this can be verified using S_ρ as $S_\rho(u, q) = 8$, and $S_\rho(r, q) = 46$.

The rationale behind the brief permutations, or the binary encoded permutations, is to capture the proximity between the representations. The binary differences account for displacements larger than m in the permutant positions, which is the same behavior we observe in S_ρ.

It is interesting that even computing sequentially the Hamming distance is way faster, because we can use XOR bit-operation ($\oplus$) using the bit parallelism inherent in the computer integer operations computing 32 or 64 operations per instruction instead of the most expensive operations difference and product used in the S_ρ. Bit count can also be precomputed using a table.

With the above assumptions, $0 \oplus 0 = 0$ accounts for a small movement and $0 \oplus 1 = 1$ denotes a large movement. Since $1 \oplus 1$ can be interpreted either as a large or an small displacement (because the two permutations are compared against the identity , in order to encode in just one bit each permutant we choose only one, a small difference, which also coincide with the Hamming distance, and hence $1 \oplus 1 = 0$. If we choose $1 \oplus 1 = 1$ can be efficiently computed using $\oplus$ as OR instead of XOR. The third alternative is to use two bits, encoding left and right displacements. Experimentally all the above alternatives behave the same, except when the number of permutants is small, which is not very interesting in our setup.

2. LOCALITY SENSITIVE HASHING FOR METRIC SPACES

Obtaining the LSH for the general metric space setup is now very natural. The first step is to compute the brief permutation of all the database elements, and then compute a family of hashes for each binary string representing the objects. LSH can also be used with the regular (full) permutation representation with an ℓ_p hash family. With either alternative we can encapsulate finding a suitable LSH for each data model. Our approach is more general.

It is worth notice the nice properties of the permutation-space representation are preserved in the brief representation, namely the ability to predict proximity in the non-metric case, as described in [6], since the triangle inequality is not used explicitly.

To satisfy a query we first map q to the permutants space, and then retrieve a set of candidates using a suitable hash function (either in the full or brief representation). Once the candidates are chosen, there are two possibilities:

- If the original similarity function is expensive, we must refine LSH candidates using the S_ρ.
- If the original similarity function is cheap, we verify the entire set of candidates.

Finally, all the candidates are ranked with the original similarity function and presented as result.

In the full representation we need an embedding for L_1 or L_2 (Spearman footrule and Spearman ρ respectively) as detailed in [9, 2], this alternative produces good results but increase the searching and indexing costs; and since the permutations are just an approximation and not the final result we lean to using LSH over Hamming directly. If the original distance is not expensive, we don't store the mapping, keeping just the corresponding hashes.

In our experiments, We study three parameters:

- Number of permutants or references (*numrefs*). The number of permutants, randomly chosen at the preprocessing step, fixes the size of the bit-string. We stress we don't need to save bit-strings for most applications. As a rule of thumb, larger numrefs implies better recall.
- Number of hashing functions (*numhashes*). The number of hashing functions to be computed by LSH, affecting the total number of candidates and the recall: the more candidates, the bigger the recall and higher the cpu cost for checking candidates.
- Size of the bit sample (*samplesize*). Another LSH parameter, it controls the probability of collision. Larger *samplesize* values produces sparser tables, resulting in faster searches but with few candidates (most of the times meaning low recall).

It is difficult to obtain bounding probabilities for the hashing. Nevertheless, the problem is suitable for an experimental tuning of the three parameters described above. A good

tradeoff between speed and recall can be obtained by varying the parameters in a specific setting. We will devote the rest of the paper to show the experimental results in a number of example databases.

3. EXPERIMENTS

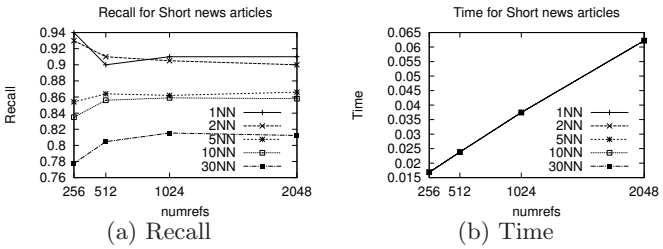
The test databases were taken from the *metric space* library[1] and the CoPhIR project [4]. Our implementation is available as open source from `www.natix.org`. All indexes share the same distance function's implementation. We must notice that all the test indexes were developed in C# under the mono 2.6.1 framework and linux 2.6.27 under Ubuntu/Linux 8.10 in a Xeon 2.33GHz with 8GiB of ram workstation. The indexes run in main memory and without parallelization. We choose four databases: documents and dictionaries for textual information retrieval, color's histogram and MPEG7 vectors for images in two different databases for multimedia information retrieval. Please note that the brief representation of the permutant space allows keeping the index in main memory, as well as the LSH tables.

The experiments were performed with brief indexes using modulus of 0.5 times the number of permutants, applying the central permutation technique as described in Algorithm 2. In our two level index we use several configurations, providing information to optimize in both recall and time axes. We provide a simple comparison against the sequential scanning to give an standard point of reference with other approaches. Additionally, the comparison against the sequential brief index is shown. The behavior of the BKT for the same set of queries is described too. The experimental results covers both execution time and number of distance computations, trying to give a complete idea of the performance from distinct points of view.

Our implementation uses a table of precomputed Hamming distances for two bytes integers. We considered all the distances as expensive, to give an idea of the worst case scenarios, and we computed the Hamming distance and sorted the candidates obtained from the LSH phase.

3.1 Documents

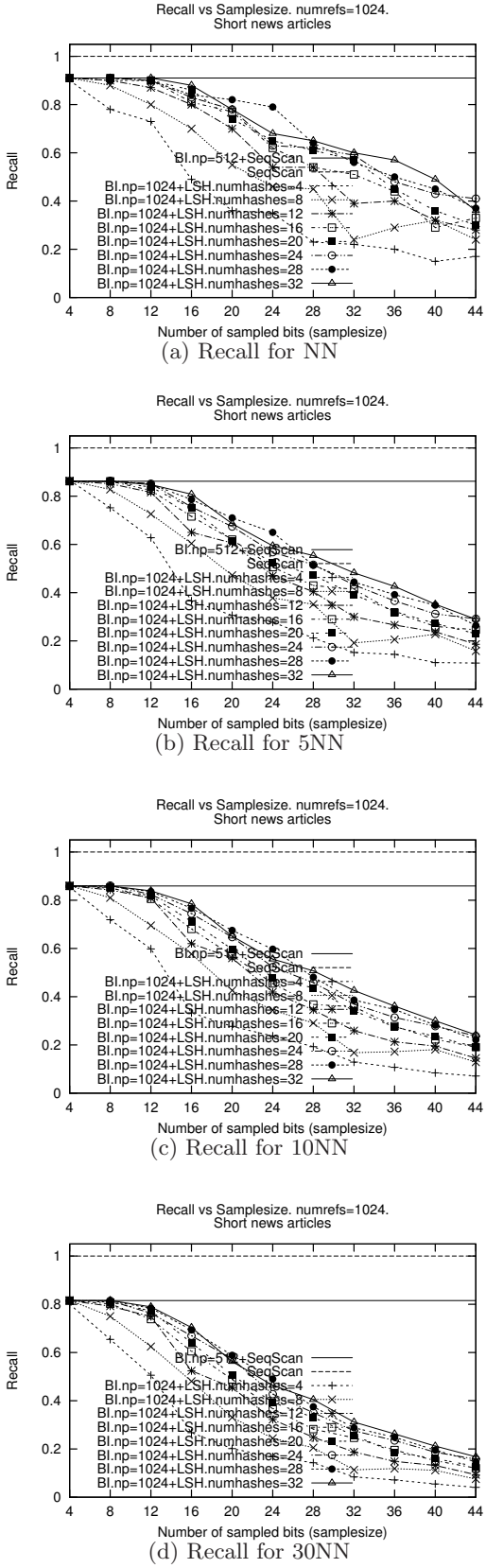
We used a collection of 25157 short news articles in $TF \times IDF$ format from Wall Street Journal $1987 - 1989$ files, specifically from TREC-3 collection. We use the standard metric of the angle between vectors as distance.



(a) Recall (b) Time

Figure 1: Recall and time behavior for brief index for different configurations of the database of news documents

[1]Metric space library `www.sisap.org`.



(a) Recall for NN

(b) Recall for 5NN

(c) Recall for 10NN

(d) Recall for 30NN

Figure 2: Experiment results for brief index against the two layer index using the documents $TF \times IDF$ collection and vector's angle as distance.

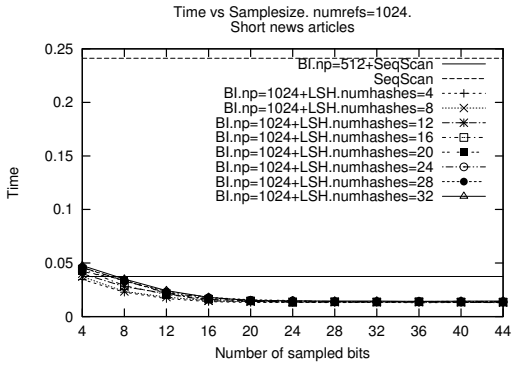


Figure 3: Experiment results for brief index against our two layer index using the documents $TF \times IDF$ collection and vector's angle as distance.

We extracted 100 random documents as queries, these documents were not indexed. We produce queries for 1, 5, 10, and 30 nearest neighbors (a metric index, like BKT using a ring width of 0.001 [7] needs to check up to 98% of the database for 30NN). We fix the number of distance computations to 1000 + the number of references, see Figure 2. Then for $30NN$ with recall bigger than 0.82 we need to review only the 8% of the database instead of the 98% in the alternative metric index (not shown for space constrains).

The recall and time behavior for the original brief permutation algorithm are shown 1. We can configure our index for several number of references, i.e. *numrefs* values, and we can reach at maximum the recall listed in Figure 1(a). Looking for a fast configuration with a good recall we chose to use 1024 permutants. Figure 1(b) shows the time necessary to solve queries. These times set the allowed upper bounds for each configuration. All curves are drawn as a single line because all of them uses the same resources.

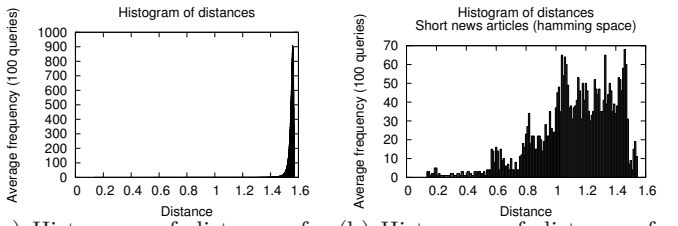
Figure 3 shows the average time per search for different configurations of permutants and LSH paramenters. For sampled bits bigger than 8, we can see that the two layer index is faster than the sequential brief index and always faster than the naïve solution of Figure 1(b), and in fact for $samplesize = 12$ we achieve faster times than the faster setup in the original algorithm. Configurations beyond these values are notoriously faster than any configuration in the original approach.

We claim that our mapped space is *easier* than the original one, since the high recall in the output (with a very small number of candidates) and the histogram of distances for every query set 4(b). The resulting space is more indexable by traditional metric indexes as shown in [5]. Using LSH as second layer, can be replaced by any other index, but LSH is faster and with a small overhead.

3.2 Dictionary

Searching in dictionaries for misspelled words, OCR errors, etc. is a common task in information retrieval or pattern recognition. We use the metric space library's English dictionary with 69069 words. English dictionary was selected to avoid encoding problems, but we expect the same behavior from other non-agglutinant languages. We used the plain edit distance.

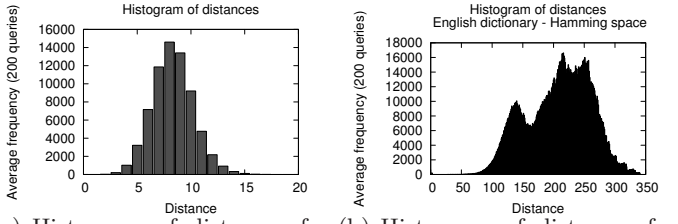
We took 200 randomly selected words from the database



(a) Histogram of distances for the original space (angle between vectors)

(b) Histogram of distances for the mapped space (Hamming using the brief index)

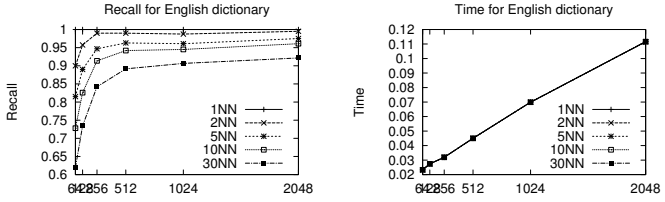
Figure 4: Histogram of distances for our query set, both in the original space and the mapped space, showing the behavior in the entire database.



(a) Histogram of distances for the original space

(b) Histogram of distances for the mapped space (Hamming using the brief index)

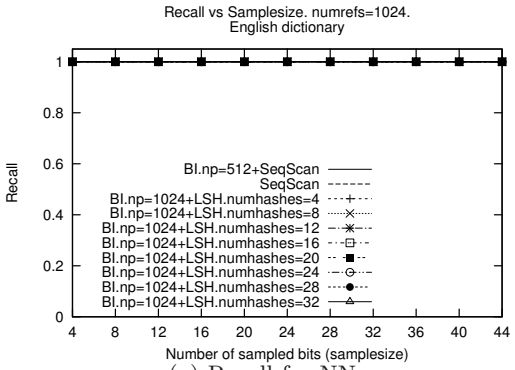
Figure 5: Histogram of distances for the English.dic dictionary, on the original space and the hamming space



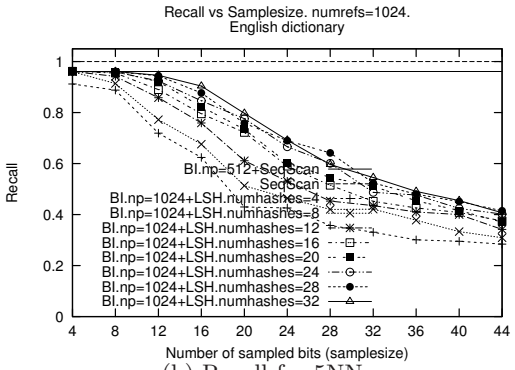
(a) Recall

(b) Time

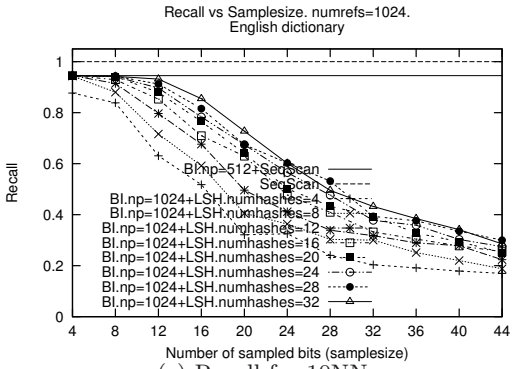
Figure 6: Recall and time behavior for brief index and different configurations over English dictionary



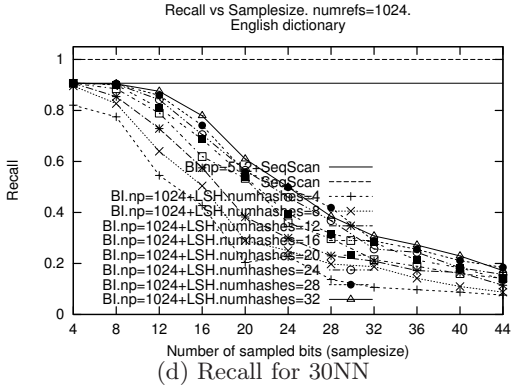
(a) Recall for NN



(b) Recall for 5NN



(c) Recall for 10NN



(d) Recall for 30NN

Figure 7: Results for brief index and our two layer index for the english dictionary, using edit distance.

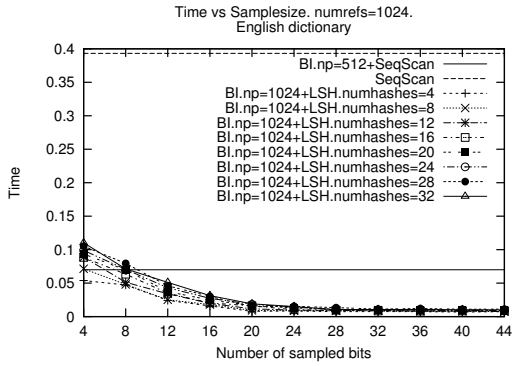


Figure 8: Time performance for brief index and the two layer index for color's histogram collection and L_2 distance.

as queries and searched for 30NN. For the NN (i.e the word as query) we have perfect recall of 1.0 using more than 128 permutants, and was close to one for 2NN also. To keep the comparison. in the same framework we completed the full 5, 10 and 30NN (Figure 7(d)). Please note that being a discrete distance for words, we have many ties in the KNN searches. The histogram of distances for our query set is shown in Figure 5, both the original distance and the mapped space (i.e. hamming) are depicted.

Just to compare, the BKT needs to review 56% of the database for the same task. The brief index needs to review 3% to get a recall of 0.97 for 2NN and 0.84 for 30NN. The average search time is shown in Figure 8, where our index is faster for any configuration with *samplesize* bigger than 8.

3.3 MPEG7 Vectors

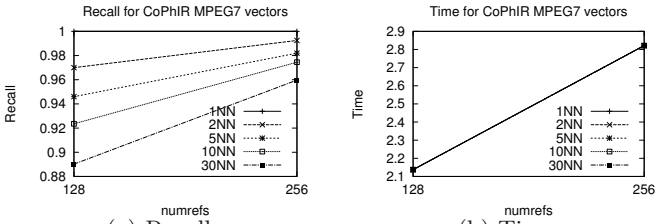
A database of 10 million 208-dimensional vectors from the CoPhIR project [4] were selected. It uses the L_1 distance. Every vector is one of five different MPEG7 vectors as described in [4]. We choose the first 200 vectors from the database as queries. Search for the 30NN were performed. Traditional metric indexes like BKT do not perform well with the this database size. This database is difficult because its size, since it avoids the use of traditional structures with high overhead.

Figure 10 shows the recall for different configurations, for a brief index with both sequential scanning and LSH layer. Here the step for *samplesize* is bigger than other databases because we want to discern between a larger number of objects. We fix the number of verified candidates to 50000, equivalent to review just the 0.5% of the database.

The brief index recall is shown in Figure 9(a), where the recall is close to 1. We only tested 128 and 256 permutants because of these good results. Figure 9(b) shows a very competitive time against the sequential scan of 18.6 seconds.

A dramatic improvement is observed in the time using the two layer index, see Figure 11. For index with $samplesize = 32$, queries are solved in close to 0.1 seconds. For $samplesize = 96$ over 0.002 seconds. These is achieved because the LSH layer never gives the entire set of requested candidates. For example $samplesize = 32$ achieves approximately 30000 candidates for the presented *numhashes*. Fixing $samplesize = 96$ we get close to 500, 1100 and 1600 candidates for 4, 8, 12 *numhashes* respectively. So, if we optimize the index for

performance it needs to review 0.016% of the database. A higher recall can be obtained reviewing 0.3% of the objects. The recall is affected by the small candidate set in the two layer index, as shown in Figure 10.



(a) Recall (b) Time

Figure 9: Recall and time behavior for brief index and different configurations over English dictionary

4. CONCLUSIONS AND FUTURE WORK

In this paper we presented a method to obtain a Locality Sensitive Hashing schema for general metric spaces and we showed experimentally how to obtain a good tradeoff between speed and recall. Our approach allows to create similarity indexes in a black box scheme without a deep knowledge about the data model and its similarity function.

The method is two layered. In the lower layer we have a permutation based index which can handle high dimensional spaces and can be used in similarity spaces as well. In the upper layer we use LSH allowing scalability. The entire setup reduces to a Hamming space to be searched for candidates and a subsequent candidate check with the original distance function.

The performance can be tuned by making large/small candidate lists to be verified and retrieved. Even with an small candidate set we found a competitive recall. In the storage side, we have presented two options. The first one for expensive similarity functions: we need to save 1 bit per permutant per each item in the database + hash tables, in the second (i.e. cheap similarity functions) we only need to save hash tables.

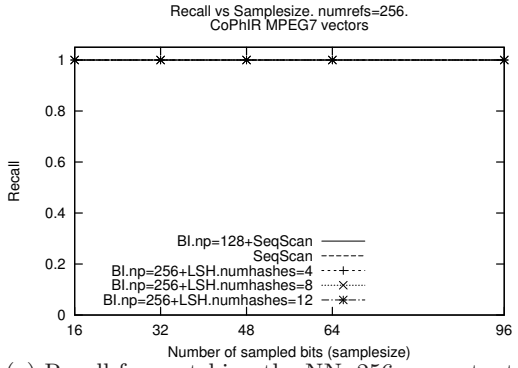
Very large databases show better the improvement in performance, as observed in the CoPhIR database. Smaller databases are improved but not significantly, they remain in the same order of magnitude even if the query time is improved two or three fold.

We are currently working in obtaining theoretical bounds on the distance from the exact proximity query and the approximated output we obtain with LSH and binary permutations.

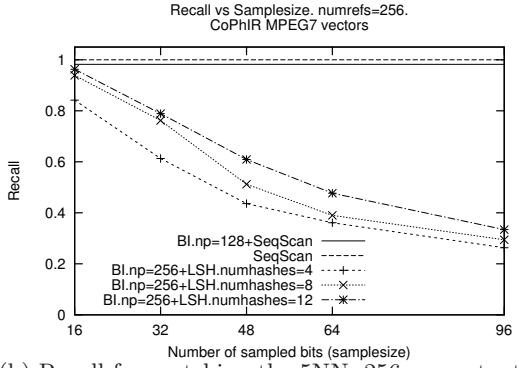
Future work includes secondary memory implementation and the inherent parallelism of the technique: the computation of permutations, searching in several hash functions, the verification stage, and the necessary sorting steps. Then a complete set of algorithms can be designed specially for multiple core processors and GPU architectures.

5. REFERENCES

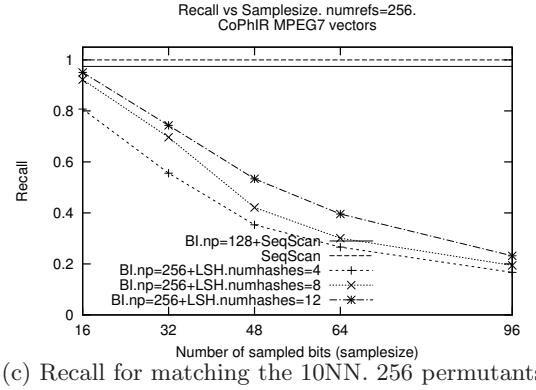
[1] Giuseppe Amato and Pasquale Savino. Approximate similarity search in metric spaces using inverted files. In *InfoScale '08: Proceedings of the 3rd international*



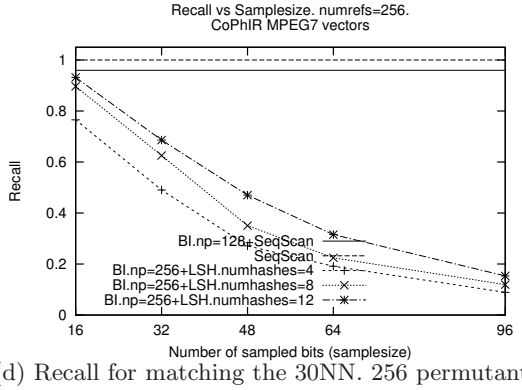
(a) Recall for matching the NN. 256 permutants



(b) Recall for matching the 5NN. 256 permutants



(c) Recall for matching the 10NN. 256 permutants



(d) Recall for matching the 30NN. 256 permutants

Figure 10: Results for brief index and our two layer index for CoPhIR 10M MPEG7 database.

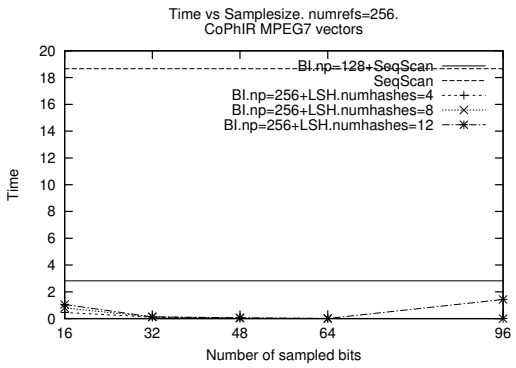


Figure 11: Time performance for brief index and the two layer index for MPEG7 vectors of 10M CoPhIR database

conference on Scalable information systems, pages 1–10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[2] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

[3] 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*, 33(3):322–373, 2001.

[4] Paolo Bolettieri, Andrea Esuli, Fabrizio Falchi, Claudio Lucchese, Raffaele Perego, Tommaso Piccioli, and Fausto Rabitti. Cophir: a test collection for content-based image retrieval. *CoRR*, abs/0905.4627v2, 2009.

[5] B. Bustos, G. Navarro, and E. Chávez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters*, 24(14):2357–2366, 2003.

[6] Edgar Chavez, Karina Figueroa, and Gonzalo Navarro. Effective proximity retrieval by ordering permutations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(9):1647–1658, September 2008.

[7] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33(3):273–321, 2001.

[8] Andrea Esuli. Pp-index: Using permutation prefixes for efficient and scalable approximate similarity search. In *LSDS-IR 2009 Workshop*, 2009.

[9] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[10] M. Patella and P. Ciaccia. Approximate similarity search: A multi-faceted problem. *Journal of Discrete Algorithms*, 7(1):36–48, 2009.

[11] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers, 2006.

[12] Eric Sadit Tellez, Edgar Chavez, and Antonio Camarena-Ibarrola. A brief index for proximity searching. In *Proceedings of 14th Iberoamerican Congress on Pattern Recognition CIARP 2009*, 2009.

CP-Index: Using Clustering and Pivots for Indexing Non-Metric Spaces

Victor Sepulveda
PRISMA Research Group
Department of Computer Science
University of Chile
vsepulve@dcc.uchile.cl

Benjamin Bustos
PRISMA Research Group
Department of Computer Science
University of Chile
bebustos@dcc.uchile.cl

ABSTRACT

Most multimedia information retrieval systems use an indexing scheme to speed up similarity search. The index aims to discard large portions of the data collection at query time. Generally, these approaches use the triangular inequality to discard elements or groups of elements, thus requiring that the comparison distance satisfies the metric postulates. However, recent research shows that, for some applications, it is appropriate to use a non-metric distance, which can give more accurate judgments about the similarity of two objects. In such cases, the lack of the triangle inequality makes impossible to use the traditional approaches for indexing. In this paper we introduce the CP-index, a new approximate indexing technique for non-metric spaces that combines clustering and pivots. The index dynamically adapts to the conditions of the non-metric space using pivots when the fraction of triplets that break the triangle inequality is small, but sequentially searching the most promising candidates when the pivots becomes ineffective.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing

Keywords

Similarity Search, Non-Metric Distance, Indexing, Clustering, Pivots.

1. INTRODUCTION

Nowadays, the amount of multimedia information available is growing at an increasing rate, and systems to manipulate and search for information in multimedia repositories become increasingly important. As technology improves, the ability to generate multimedia objects, as well as its complexity, increases. Indeed, it is estimated that about 95% of the web space (measured in bytes) consists of multimedia information [11]. Applications in this area include: multimedia information retrieval systems, biometrics, databases for medicine, geology, astronomy, and many others.

A main problem in these applications is to perform similarity search in a database, ie, finding the objects most similar to a query. However, when the objects of the database are complex, comparing their similarity can be computationally expensive, so it is desired to avoid as many comparisons of objects as possible. The traditional approach is to use metric distances for comparison, so the triangle inequality can be used to prune the search [12]. However, there are many cases where it is more appropriate to use distances that do not satisfy the metric properties, the so-called non-metric spaces.

There are a variety of reasons that justify the use of non-metric distances, eg, robustness to outliers, the use of machine learning models to make better comparisons, and modeling of human perception [11, 10, 7]. One of the most important reasons is to emulate the perceptual similarity. Certain studies show that human perception have no metric behavior [10, 7], since it do not uses a static set of criteria to judge similarity. Instead, it determines the most appropriate criteria for a particular comparison, and then use them to evaluate similarity. Thus, different pairs of objects are compared with different measures, and properties like the triangle inequality lose their meaning.

The triangle inequality is the base of most indexing techniques, so it is interesting to develop techniques to deal with the indexing in non-metric spaces. It is also desirable that the techniques are not dependent on a particular space, but based only on the pairwise distances and the information derived from them.

In general, the use of pivots is not considered in spaces that fail in the triangle inequality because it leads to false dismissals. However, in this paper we study the effect of clustering the data on the fails caused by pivots. We introduce the CP-index, a new approximate indexing technique that combines clustering and pivots for non-metric indexing. This technique adapts to the conditions of non-metric space, using pivots when the fraction of triplets that break the triangle inequality is small, but sequentially searching the most promising candidates when the use of pivots becomes useless, in a dynamic way. We experimentally tested this strategy with the L_p Fractional distance to represent spaces in which the triangle inequality is broken in different degrees.

The rest of this paper is organized as follows: in Section 2 we introduce the problem of similarity search, indexing in

SISAP'10, September 18-19, 2010, Istanbul, Turkey

Table 1: Notation Used

Symbol	Meaning
$\mathbb{U}$	Universe (descriptors of multimedia objects)
$\mathbb{S}$	Database of objects ($\mathbb{S} \subset \mathbb{U}$)
O_i, O_j	Database Objects ($O_i, O_j \in \mathbb{S}$)
Q	Query Object ($Q \in \mathbb{U}$)
$\sigma : \mathbb{U} \times \mathbb{U} \to \mathbb{R}$	Similarity function
$\delta : \mathbb{U} \times \mathbb{U} \to \mathbb{R}$	Distance function

metric and non-metric spaces, and give some examples of non-metric distances measures. In Section 3 we explain the main approaches proposed so far for indexing non-metric spaces. In Section 4 we introduce our proposal, the CP-Index. In Section 5 we present experimental results, and in Section 6 we conclude and present future work.

2. SIMILARITY SEARCH

In modern multimedia retrieval systems, the similarity search process has become its central task. Therefore, improving its efficiency and efficacy has become very important, since the complexity of multimedia objects and the volume of multimedia repositories grows up quickly.

First of all, we consider the similarity model. Table 1 shows the notation used in this paper. We have a distance function that measures the dis-similarity of two database objects as follows:

$$\delta(Q, O_i) < \delta(Q, O_j) \Longleftrightarrow \sigma(Q, O_i) > \sigma(Q, O_j).$$

This means that Q is more similar to O_i than to O_j. In the rest of this paper, The principal query modalities, range search and knn search, commonly used in multimedia retrieval systems, are defined as follows:

Definition 1. Let $Q \in \mathbb{U}$. The range query $range(Q, r)$, with tolerance radius $r \in \mathbb{R}^+$, returns the set:

$$Range = \{O_i \in \mathbb{S} \mid \delta(O_i, Q) \leq r\}$$

Definition 2. Let $Q \in \mathbb{U}$. The query for the K nearest neighbors $Knn(Q)$, with $K \in \mathbb{N}^+$, returns a set that satisfies the following:

$$|Knn| = K,\ \forall O_i \in Knn,\ \forall O_j \in \mathbb{S} - Knn,\ \delta(O_i, Q) \leq \delta(O_j, Q)$$

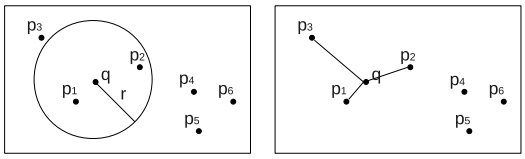


Figure 1: Example range query (left) and 3nn query (right). The first report the objects P_1 y P_2 and the second, the elements P_1, P_2 y P_3.

2.1 Metric Spaces

Traditionally, a metric distance function has been used as dis-similarity measure. The rationale behind this is to exploit the topological characteristics of a metric distance to facilitate indexing and to speed up the searches. Now we review the metric distance properties.

Definition 3. A distance function $\delta : \mathbb{U} \times \mathbb{U} \to \mathbb{R}$ is metric if and only if the following properties are satisfied:

Reflexivity	$\delta(O_i, O_j) = 0 \Longleftrightarrow O_i = O_j$
Non-negativity	$\delta(O_i, O_j) > 0 \Longleftrightarrow O_i \neq O_j$
Symmetry	$\delta(O_i, O_j) = \delta(O_j, O_i)$
Triangle Inequality	$\delta(O_i, O_j) + \delta(O_j, O_k) \geq \delta(O_i, O_k)$

2.2 Indexing

Generally, a distance function that appropriately models similarity between complex objects can be computationally expensive. Therefore, it is desirable to avoid as many distance calculations as possible while performing a similarity search. The traditional approach is to index the space or to classify the elements in some way to discard entire zones or groups of objects. The fundamental property for indexing the database and speed up the search is the triangle inequality, and most indexing techniques are based on it.

The different indexing strategies can be divided into two main approaches: partitioning the space or using pivots. The former divides the space into zones characterized by some of the objects contained in them, and the query object is compared with the elements that represent each zone, trying to discard entire zones if they are far enough. The latter uses a set of objects as pivots. The distances between these pivots and the rest of the database are computed and stored. At query time, the stored distances and the distances between the query and the pivots are used to compute lower bounds to the distance between query and objects, allowing the algorithm to discard non relevant objects.

2.3 Non-Metric Spaces

Any distance function that fails to satisfy any of the metric properties is a non-metric distance. There are a variety of reasons that justify using non-metric distances, like robustness to outliers, using machine learning models whose behavior can be difficult to predict, using black box distances whose exact behavior is unknown, or a better modeling of the human perception. Furthermore, restricting the domain expert to generate a similarity measure satisfying the metric postulates may be harmful to the effectiveness of retrieval, or even impossible.

Among the metric properties, failures in satisfying reflexivity or non-negativity (by distances that allow certain objects to be differently self-similar) and in symmetry (for example, if the similarity measure allows an object containing another one to be more different to it than vice versa) can be repaired relatively easily (e.g., adding a constant value to the distance). Distances that do not satisfy the triangle inequality tend to be much more difficult to treat, as this particularly affects the indexing techniques. Distance functions that satisfy the metric properties except the triangle inequality are called semi-metric and are of special interest. For the remainder of this paper, we will consider only the case of semi-metrics when referring to non-metric distances.

2.4 Examples of Non-metric Distances

We now present some relevant examples of distances that do not satisfy the metric properties, in particular, the triangle inequality. In addition to these and several other, any combination of non-metric distances can lead to a non-metric, and even certain combinations of metric distances may lead to non-metric distance measures.

2.4.1 Dynamic Partial Function

Dynamic Partial Distance Function [6] is based on the family of Minkowski distances, but uses only a few coordinates to calculate dis-similarity. The idea is to consider only a fraction of the most similar coordinates, assuming that objects that should be classified as "similar" may have a small number of very different components. DPF distance is defined as follows:

Let $c_i = |x_i - y_i|$ where x_i, y_i are the i-th coordinate of x and y. Now let Δ_m be the set of the m smallest c_i from $\{c_1, ...c_d\}$.

$$DPF_p(x,y) = \left(\sum_{c_i \in \Delta_m} {c_i}^p \right)^{1/p}$$

2.4.2 Cosine Distance

Cosine distance [2] measures the difference in the direction of two vectors, regardless of their magnitude. It is commonly used in text and documents retrieval, and is defined as follows:

$$\sigma_{cos}(x,y) = \frac{\sum_{i=1}^{d} x_i y_i}{\sqrt{\sum_{i=1}^{d} x_i^2 \sum_{i=1}^{d} y_i^2}}$$

σ_{cos} is a measure of similarity, but it can be made a distance by defining $\delta_{cos} = 1 - \sigma_{cos}$, which is a non metric distance function.

2.4.3 Dynamic Time Warping

Dynamic Time Warping [3] is a measure commonly used to compare the distance in time series. Let $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ be two time series, and M a $n \times m$ matrix with $M[i,j] = (x_i - y_j)^2$. A warping path $W = \{w_1, ...w_t\}$ is a list of components of M that satisfy the following properties: $w_1 = M[1,1]$ and $w_t = M[m,n]$, for $w_k = M[a,b]$ and $w_{k-1} = M[a',b']$ then $a - a' \leq 1$ and $b - b' \leq 1$, and for $w_k = M[a,b]$ and $w_{k-1} = M[a',b']$ then $a - a' \geq 0$ and $b - b' \geq 0$. DTW looks for the minimum warping path, so the distance is defined as follows:

$$DTW(x,y) = min_W \left\{ \sqrt{\sum_{k=1}^{t} w_k} \right\}$$

DTW fails to satisfy the triangle inequality so it is a semi-metric.

2.4.4 Fractional L_p Distances

The Minkowski family are perhaps the most widely used distances as a measure of dis-similarity in vector spaces. Minkowski distance with parameter p, L_p, is defined as follows:

Let $x, y \in \mathbb{R}^d$ be two vectors of dimension d and, $p \geq 1$.

$$L_p(x,y) = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p}$$

The fractional L_p distance [1] extends this concept by allowing a parameter $0 < p < 1$, but that results in a semi metric distance function, because it no longer satisfy the triangle inequality. As p gets close to 0, the space generated by the fractional L_p distance becomes "less metric", increasing the fraction of triplets that break the triangle inequality. Below is a graph showing the percentage of failure of the triangle inequality in a set of uniform distributed vectors in the unitary cube of dimension 8 for different values of p.

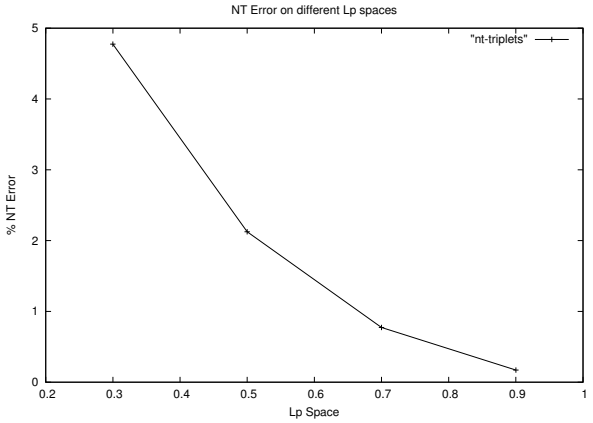


Figure 2: Failure rate of the Triangle Inequality for different fractional L_p norms.

This distance has the advantage of allowing us to control the rupture of the triangle inequality by changing the parameter p. Similar to the dimension in a synthetic vector space, with the Fractional L_p distance we can make tests on "more non-metric" spaces, where the fraction of triplets that break the triangle inequality is high, or closer to a metric in which the rate of the triangle inequality failure is low.

3. NON-METRIC INDEXING METHODS

Although there are indexing techniques for particular cases of non-metric spaces, such as the inverted index for documents, it is of interest to develop techniques that can deal efficiently the similarity search problem in generic non-metric spaces. Here, the pairwise distances are the only information used for indexing and querying. In the following, we review the main approaches proposed so far to address this problem.

3.1 Constant Shifting Embedding

There are simple ways to convert a semi-metric δ into a metric distance. One way is to add a suitable constant to the distance, to obtain a modified distance $\delta' = \delta + c$. The triangle inequality can be enforced with a large enough constant [9]. The problems of this approach are that the appropriate constant should be calculated considering all the data if there is no adequate analytical information, otherwise it might not be large enough for a new query object. A more difficult problem is that, in general, the appropriate constant to turn the semi-metric into a metric is too big and the new space becomes hard to index, because all distances become relatively too similar and the triangle inequality can no longer discard elements.

A more elaborated approach is the recently introduced Local Constant Embedding [4], where the space is divided into groups and the method looks for a suitable constant for each group. This allows the index to use smaller constants, permitting a better indexing of each group as an

independent metric space. However, the choice of appropriate groups is complex and computationally expensive: the proposed heuristic requires the whole distance matrix and it is cubic (in time) in the number of elements.

3.2 QIC

QIC [5] is an approach based on Filter&Refine, using several distance measures. Its application in non-metric spaces requires the use of a metric for building the index, that is a lower bound of the non-metric query distance. The search algorithm traverses the index using the lower bound distance, and then refines the result with the non-metric distance. This technique is implemented by modifying an M-tree to generate the so-called QIC-M-Tree. The disadvantage of this method is that finding a distance metric that is a lower bound of a complex non-metric can be difficult (or even impossible for black-box distances), and it also must be a tight lower bound for the search to be reasonably efficient.

3.3 TriGen

The TriGen algorithm [11] tries to modify the semi-metric by applying a concave or convex function to make it more or less metrical (reducing or increasing the triangular inequality error). In this way, the space becomes "less indexable" as it becomes more metric. The generation of the modifier function is performed automatically and the modified distance, turned into an almost metric, can be used in a metric access method for an approximate search.

3.4 Classification Techniques

Another approach that has been used for searching in non-metric spaces is to address the problem of similarity search as a classification problem. This approach uses statistical information or performs clustering of the data in some way to generate a set of classes or clusters. At query time, the query object is classified into one or more classes, and the corresponding clusters are searched for objects, assuming that the most similar objects to the query will be found in the most promising clusters.

An indexing technique using this approach is DynDex [6], which performs clustering with the CLARANS [8] algorithm in the construction phase. At query time, DynDex sequentially scans a fixed number of the most promising clusters.

4. CP-INDEX

4.1 Using Pivots in Non-Metric Spaces

Using pivots to speed up the search in a space where the triangle inequality fails may lead to false dismissals. As shown in the following example on the $L_{0.5}$ space, the commonly used method to compute a lower bound of the actual distance between a candidate and an query object fails. That is, the lower bound distance obtained using the pivot is not valid. Thus, the object would be incorrectly discarded and lost for the final result.

Figure 3 shows the example in 2-D and $L_{0.5}$ distance. There is an object p pivot for a candidate object x. The $L_{0.5}$ distance between the pivot and the candidate is 2. We have a query object q at an approximate distance of 5.82 of the pivot, and a query radius of 1. In this situation, the candidate would be dismissed because $|5.82 - 2| > 1$ (the lower bound distance is computed as $|\delta(p, x) - \delta(p, q)|$, see Zezula et al. [12] for details). But, the distance between the query object and the candidate is 1, thus it was relevant and incorrectly discarded.

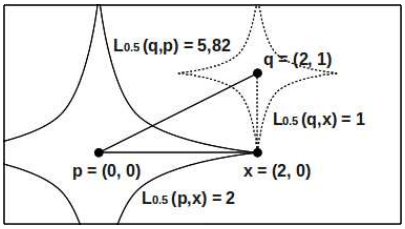


Figure 3: Example of a false dismissal on the 2-D and $L_{0.5}$ space.

However, there are cases in which this event is less frequent, depending on the chosen pivots and the relationship between the query object and the candidate to be discarded. Indeed, we experimentally discovered that when the database objects are separated into compact groups of close objects and the pivots are locally chosen and used at each group, less objects are incorrectly discarded than when using the same pivots but globally, for the whole database. Indeed, we conducted tests where we generated clusters of different qualities, and we studied the effectiveness of pivots in them. We partition the data into clusters and choose at random a certain percentage of elements as pivots. We used a set of random query objects, counting the number of times the triangle inequality fails (that is, the number of times a candidate is discarded incorrectly). We studied the effect of increasing the number of groups from one (i.e., the entire database), to groups with a small number of objects.

We observed that the number of false dismissals is reduced by having more clusters, because they are more compact. This was achieved by partitioning the database into clusters, by simply choosing centers at random, and by randomly choosing pivots in each cluster. Figure 4 shows the average number of false dismissals depending on the number of such clusters in a database of 10,000 vectors of dimension 112 in the $L_{0.5}$ space. Therefore, local pivots may be used for indexing some non-metric spaces, but it is not guaranteed that all relevant objects will be found.

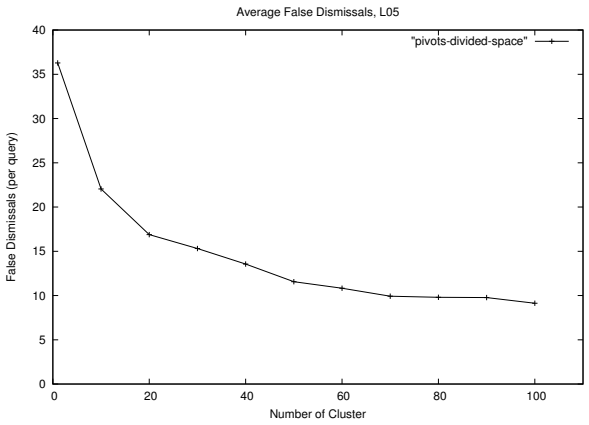


Figure 4: Reduction of false dismissals on the $L_{0.5}$ space by clustering the data.

4.2 Clustering&Pivots Index

The false dismissals arise from the relationship between the candidate to be discarded, the query object, and the pivots, that generate the breaking of the triangle inequality. However, there are spaces in which the triangle inequality fails too often. A better pivot selection technique can reduce failures, but testing in spaces where the failure rate of the triangle inequality is greater, such as the $L_{0.3}$, showed its inefficacy. In such spaces, most techniques that were tested fail or turn into sequential search. We then prepare a technique that uses pivots to reduce the work, but adapting to the characteristics of non-metric space, linear scanning when the pivots fails.

CP-Index uses a similar approach to that of DynDex, and considers the similarity search as a classification problem. During the construction phase a clustering process is performed using CLARANS, and in each cluster a set of pivots is chosen with some heuristics. Subsequently, the elements of each cluster are classified according to whether they can be correctly discarded by pivots or not, with the chosen pivots and a representative set of queries. At query time, we try to discard by pivots only those elements which have been classified as treatable by pivots. For the remaining elements, we consider that the pivots test will fail and calculate its actual distance to the query. In this way, the index adjusts and uses pivots only when it is safe. In spaces where the triangle inequality fails too often, such as $L_{0.3}$, sequentially searches the clusters while in more metric spaces, such as the $L_{0.7}$, uses pivots to discard candidates quickly. Finally, we set the amount of work (measured in distance computations) to be performed during the search. Thus, the CP-Index implements an approximate similarity search.

The CLARANS method is a clustering technique that uses only the pairwise distances. It does not depends on metric or vectorial properties of the space, but it does receives the number of clusters to generate. It relies on random search of centers that tend to improve the quality of clustering.

To explain the CLARANS method, we can use an analogy between the clustering process and the search in a graph. Suppose a graph $G_{n,k}$ in which each node represents a particular clustering. The Node N_m is formed by the set of k centroids $\{O_{m_1}...O_{m_k}\}$. Two nodes are neighbors (they are connected by an arc) if their sets of centroids differ by a single element. Also, each node has a cost equal to the sum of the distances of each element to the centroid of its cluster. This is a simple way of measuring the inverse of the quality of that particular clustering.

The method searches the graph for a node that minimizes the cost. However, instead of thoroughly searching each neighbor of each node, it starts from any node and looks only a few randomly chosen neighbors to continue. CLARANS stops when it has found enough neighbors of the current node that does not improve its quality. At this point, it assumes that the current node is good enough to be considered a local minimum in the graph. Then the process is repeated starting from a different node to find a certain number of local minimum and it keeps the best of them.

To test whether an object can be discarded by pivots, we test it with a set of queries and the pivots of its cluster. If any query leads to a false dismissal with those pivots, the object is marked as non treatable by pivots. We could make a more elaborate process, in which it would be necessary a certain number pivots that erroneously discard the element before marking it as untreatable, or which seeks to improve the set of pivots when the tests fail, but the simple test-and-mark process performed better than several heuristics.

The query set used in the test is important. If we had the future query objects, the test would mark as treatable by pivots only the elements that will not be erroneously discarded. However, if we assume that the query objects will have the same distances distribution of the elements in the database, it would suffice to choose appropriate elements of the same database for querying. Moreover, if the clustering process we have done was effective, then the representatives of the clusters will be good substitutes for the elements of the cluster. Therefore, we use only the representatives of the clusters as query objects to test and mark the database elements.

Here we present the pseudocode of the main algorithms involved in construction. The construction method (Algorithm 1) receives the dataset, the number of clusters and distance to be used. It makes the clustering of the data using the method MAKE_CLUSTERING, that can use the most appropriate algorithm for this. We use the CLARANS [8] algorithm, allowing it an appropriate amount of work to achieve a good trade-off between construction time and query efficiency. Then we use the method CHOOSE_QUERIES to choose a representative set of queries from the different clusters. In our tests we used the representatives of the clusters as query objects. Then, for each cluster we choose a set of pivots with some heuristic, and verify them with the method VERIFY_PIVOTS. In the tests we choose the pivots randomly, and the verification process is described in Algorithm 2. This method verifies that the triangle inequality is satisfied for the queries and the chosen pivots, for each element of the cluster. If the verification fails for any element of the cluster with any query, the element is marked as non-treatable by pivots. At query time, only the elements that have not been marked as intractable will be tested with the pivots. This test system allows that the decision of whether using pivots or not is taken dynamically depending on the characteristics of the space. If more triplets do not satisfy the triangle inequality, more elements will be directly checked against the query.

Algorithm 1 CONSTRUCTION

Require: Dataset $\mathbb{S}$, number of clusters K, Distance δ

```
1: Let ℂ be a set to store the clusters
2: MAKE_CLUSTERING(𝕊, K, δ, ℂ)
3: Let ℚ be a set to store the queries
4: CHOOSE_QUERIES(ℂ, ℚ)
5: for Each Cluster C of ℂ do
6:    PREPARE_PIVOTS(C, δ)
7:    VERIFY_PIVOTS(C, δ, ℚ)
8: end for
```

5. EXPERIMENTAL EVALUATION

5.1 Experimental Setting

For the tests we used a database of 112.682 color histograms of images of 112 components from the SISAP Library[1]. We used the distances $L_{0.3}$, $L_{0.5}$, $L_{0.7}$ and $L_{0.9}$ to

[1]http://sisap.org/Home.html

Algorithm 2 VERIFY_PIVOTS

Require: Cluster C, Distance δ, Query Set $\mathbb{Q}$

```
for Each Query Q of Q do
  for Each Pivot P of C do
    Compute δ(Q, P)
  end for
  for Each Object X of C do
    Compute δ(Q, X)
    for Each Pivot P of C do
      if |δ(Q, P) − δ(P, X)| > δ(Q, X) then
        Mark X as non-treatable by pivots
        break
      end if
    end for
  end for
end for
```

test the behavior of the technique in different spaces with different failure rates in the triangle inequality.

We also run tests in the space of Normalized Edit Distance with a English dictionary of 69069 words from the SISAP Library. Another experiment was conducted with a database of 12218 time series of 962 components, using the DTW distance.

We randomly choose 10% of each database as query objects, and the rest was used to build the indexes. For each query object, we linearly search for their 10 nearest neighbors in advance, and stored those results. Then, we performed the clustering process to index databases. We search in each index, allowing different amounts of work (measured in number of distance calculations), and verified the percentage of correct answers obtained on average for each of the 10NN queries. In each space we tested three versions of the index: one with 0 pivots (in a similar way to DynDex, but with limited work in place of a fixed number of clusters to search), one with pivots but without the test that verifies the feasibility of the pivots, and one with pivots and the verification test.

CLARANS clustering method divides the data set into a fixed number of clusters, and its cost is approximately $O(KNW)$ where N is the number of elements, K is the number of clusters, and W is a factor that determines the allowed work of the algorithm.

The parameter W indicates the number of neighbors of each node to be visited during the search of a good enough clustering (as described in section 4.2) and can be $O(1)$, $O(log(N))$, $O(N)$, or any other, which changes the quality of the result. In the tests we used $log(N)$, for a good trade-off between construction time and result. At query time, the index was allowed to make a number of distance calculations equal to 2% of the database, 4%, 6%, 8% and 10%. In each case, the search process is terminated when it reaches the quota of allowed work. Below are the results comparing the three versions.

The number of clusters is also important for construction time and query efficiency. If we assume that we will visit a fixed number of c clusters, and consider the average size of a cluster, then we can see that the number of distance calculations performed during the query will be:

$$Cost_{dist} = K + \frac{cN}{K}$$

The first part is to classify the query in one or more clusters (because there are K centroids), and the second is the cost of sequentially searching (in the worst case) each of them. If we seek for the optimal number of clusters, we obtain $K = O(\sqrt{N})$ clusters.

$$Cost'_{dist} = 1 - \frac{cN}{K^2} = 0 \Longrightarrow cN = K^2 \Longrightarrow K^{opt} = \sqrt{cN}.$$

However, that number of clusters is valid only for the case of linear scanning. It is difficult to find an optimal number of clusters considering the extra work required to use pivots, and the distance calculations saved by them. Still, we experimentally found that $K = \sqrt{N}$ clusters gives a good trade-off between construction time and efficiency for the cases tested. In practice, it may be necessary to perform some tests to find the best number of clusters for a particular instance.

5.2 Results

Figure 5 shows that in space $L_{0.3}$ the technique with verified pivots behaves almost indistinguishable to the linear scan of the most promising clusters. That is because in that space the triangle inequality fails too often, and the pivots test fails in most cases, and only a few objects can be checked to save distance calculating. It can be see that using pivots without verification in this space leads to no more than 42% of correct results, even with the maximum allowed work.

In the spaces $L_{0.5}$ and $L_{0.7}$ (Figure 6 and Figure 7), the technique with verified pivots performs better than the linear scanning, although using pivots without verification can improve the result in some ranges of allowed amount of work. That is, with few distance computations it gets a reasonable result, but it is unable to obtain higher percentages of correct answers because many objects were wrongly discarded.

In the space $L_{0.9}$ (Figure 8), using unverified pivots works better, because the space is almost metric. However, the technique with verified pivots works almost as good as the unverified technique, showing its adaptability and giving results almost identical (or even better) to the best of the other two versions in each space.

Figure 9 shows that in the space of Normalized Edit Distance, the technique with verified pivots and the unverified version behaves in a very similar way, a little better than the linear scanning. This can also be seen in Figure 10, for the space DTW.

So, CP-Index is able to dynamically adapt to the space, taking advantage of the pivots only when it is safe.

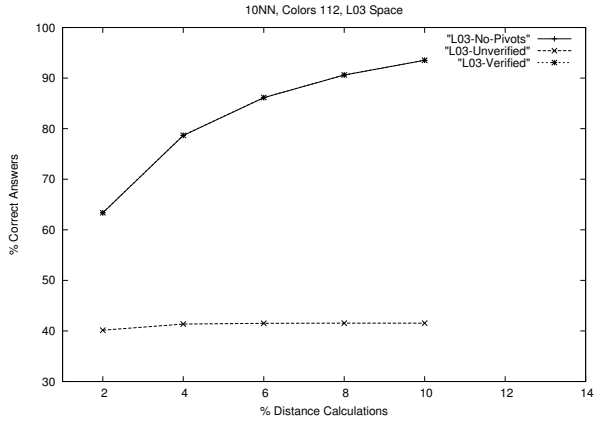


Figure 5: KNN in $L_{0.3}$ Space

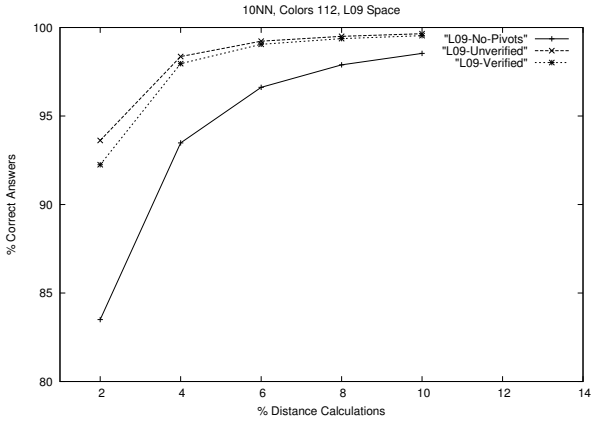


Figure 8: KNN in $L_{0.9}$ Space

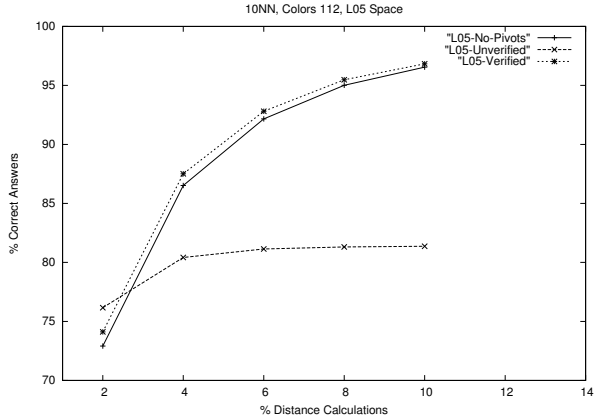


Figure 6: KNN in $L_{0.5}$ Space

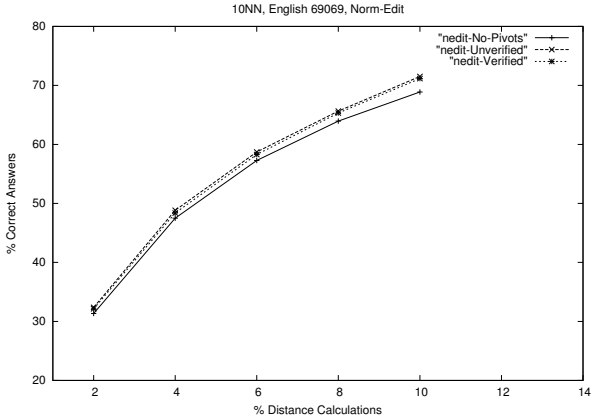


Figure 9: KNN in Normalized Edit Distance Space

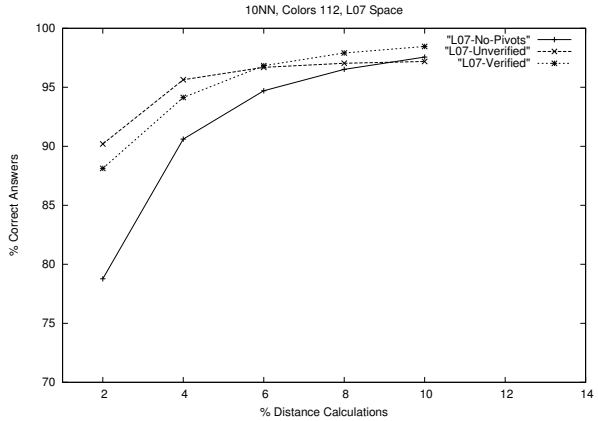


Figure 7: KNN in $L_{0.7}$ Space

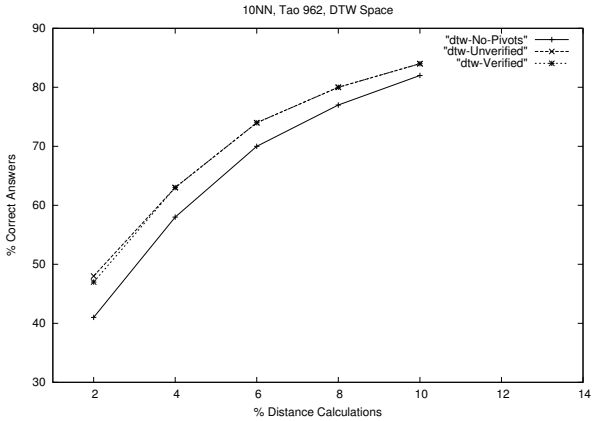


Figure 10: KNN in DTW Space

6. CONCLUSIONS

We have proposed a new approximate technique for indexing non-metric spaces which combines clustering and pivots. CP-Index adapts to the conditions of the non-metric space and tries using pivots, whenever it is possible and safe. However, when the breaking of the triangle inequality is large, it dynamically avoids using pivots preferring a sequential search of the most promising candidates. This combination allows the index to behave quite efficient in almost metric spaces, but it is capable of handling in a reasonable way most difficult spaces in which the triangle inequality fails often.

Future work, currently in development, focuses on improving the technique and its experimental comparison. On the one hand, it is under development a version that allows a finer selection of promising clusters, smaller but in greater number, to get a better response. We are also working on a dynamic version that allows insertions and deletions. On the other hand, we are working on the experimental comparison of these proposals with the major indexing techniques proposed so far for non-metric spaces.

7. REFERENCES

[1] C. Aggarwal, A. Hinneburg, and D. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *Proc. 8th International Conference on Database Theory (ICDT'01)*, London, UK, 2001. Springer-Verlag.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[3] D. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *Proc. AAAI Workshop On Knowledge Discovery in Databases*, pages 229–248, 1994.

[4] L. Chen and X. Lian. Efficient similarity search in nonmetric spaces with local constant embedding. *IEEE Transactions on Knowledge and Data Engineering*, 20(3):321–336, 2008.

[5] P. Ciaccia and M. Patella. Searching in metric spaces with user-defined and approximate distances. *ACM Database Systems*, 27(4):398–437, 2002.

[6] K. S. Goh, B. Li, and E. Chang. DynDex: A dynamic and non-metric space indexer. In *Proc. 10th ACM International Conference on Multimedia (MM'01)*, pages 466–475. ACM Press, 2002.

[7] D. Jacobs, D. Weinshall, and Y. Gdalyahu. Classification with nonmetric distances: Image retrieval and class representation. *IEEE Pattern Analysis and Machine Intelligence*, 22(6):583–600, 2000.

[8] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the 20th VLDB*, pages 144–155, 1994.

[9] V. Roth, J. Laub, J. M. Buhmann, and K. R. Müller. Going metric: Denoising pairwise data. In *Proc. International Conference on Neural Information Processing Systems (NIPS'02)*, pages 817–824, 2002.

[10] S. Santini and R. Jain. Similarity measures. *IEEE Pattern Analysis and Machine Intelligence*, 21(9):871–883, 1999.

[11] T. Skopal. Unified framework for fast exact and approximate search in dissimilarity spaces. *ACM Transactions on Database Systems*, 32(4):1–46, 2007.

[12] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach (Advances in Database Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

Applications

SISAP 2010

Improving the Similarity Search of Tandem Mass Spectra using Metric Access Methods

Jiří Novák, Tomáš Skopal, David Hoksza and Jakub Lokoč
SIRET Research Group
Department of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague
Malostranské nám. 25, 118 00 Prague, Czech Republic
http://siret.ms.mff.cuni.cz

ABSTRACT

In biological applications, the tandem mass spectrometry is a widely used method for determining protein and peptide sequences from an "in vitro" sample. The sequences are not determined directly, but they must be interpreted from the mass spectra, which is the output of the mass spectrometer. This work is focused on a similarity-search approach to mass spectra interpretation, where the parametrized Hausdorff distance (d_{HP}) is used as the similarity. In order to provide an efficient similarity search under d_{HP}, the metric access methods and the TriGen algorithm (controlling the metricity of d_{HP}) are employed. We show that similarity search using d_{HP} exhibits better correctness of peptide mass spectra interpretation than the cosine similarity commonly mentioned in mass spectrometry literature. Moreover, the search model using the d_{HP} distance could be extended to support chemical modifications in the query mass spectra, which is typically a problem when the cosine similarity is used. Our approach can be utilized as a coarse filter by any other database approach for mass spectra interpretation.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*indexing methods*; J.3 [**Life and Medical Sciences**]: Biology and Genetics

General Terms

Design, Performance

1. INTRODUCTION

Proteins – organic molecules made of amino acids – are the basis of all living organisms. The proteins are essential for construction of cells and for their proper functioning [22]. For bioinformatic purposes (i.e., in computerized biology), a protein can be understood as a linear sequence over 20-letter subset of the English alphabet[1], where each letter corresponds to an amino acid. A protein sequence must be determined from an "in vitro" protein sample, while tandem mass spectrometry is a very fast and popular method for this task. The proteins in the sample are split by enzymes into shorter pieces called *peptides*, and these are subsequently analyzed by the tandem mass spectrometer [8]. However, instead of direct production of the desired peptide sequences, the spectrometer outputs a set of experimental mass spectra[2] that have to be *interpreted* as peptide sequences some other way. In particular, the interpretation of an experimental spectrum may be accomplished by means of similarity search.

In order to interpret an experimental spectrum, a database P of known protein sequences (e.g., MSDB [17]) could be employed. The peptide sequences and their hypothetical spectra are generated from the database P, forming a database S of mass spectra. Then, the experimental spectrum is used as a query object q and the database S is searched for the nearest neighbor spectrum of q (the most similar spectrum from S). The experimental spectrum is then interpreted as a peptide sequence corresponding to the spectrum found as the nearest neighbor. As functions used to evaluate the similarity between two spectra, variations of the cosine similarity are popular.

1.1 Paper Contribution

We present the non-metric parameterized Hausdorff distance d_{HP}, which exhibits better correctness of mass spectra interpretation than the cosine similarity does. Moreover, we propose a technique for efficient search in a database of mass spectra indexed under d_{HP}, where for indexing we employ the metric access methods (MAMs). Since d_{HP} is a non-metric distance, the MAMs cannot be used directly, so prior to indexing we utilize the TriGen algorithm to control the metricity of d_{HP}. Among the number of MAMs, we have chosen the M-tree and the Pivot tables in our study. We also show that utilization of cosine similarity for the task of peptide mass spectra interpretation using MAMs is limited.

2. RELATED WORK

We briefly describe the structure of data captured by the mass spectrometer and two basic ways commonly used for mass spectra interpretation. The spectra may be interpreted directly using graph algorithms or by the search in a database of protein sequences.

[1]The letters B,J,O,U,X and Z are omitted.

[2]Each spectrum in the set corresponds to one peptide.

SISAP '10, September 18-19, 2010, Istanbul, Turkey.

2.1 Mass Spectrometry Fundamentals

The mass spectrum is a histogram of peaks corresponding to fragment ions (Fig. 1). A peak is represented by a pair $\left(\frac{m}{z}, I\right)$, where $\frac{m}{z}$ is the ratio of mass and charge, and I is the intensity of a fragment ion occurrence. The charge is supposed for our purposes $z = 1$, so the ratios $\frac{m}{z}$ are equal to the mass m of fragment ions in Daltons[3]. The precursor mass m_p (mass of peptide before splitting) and charge z_p are also provided as an additional information for each peptide spectrum captured by the spectrometer.

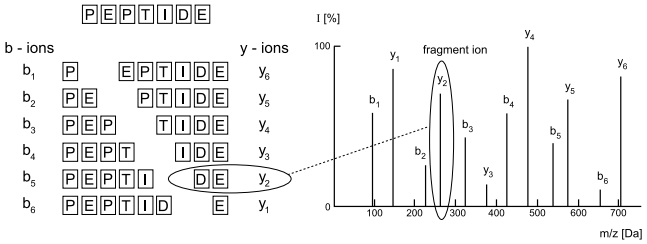


Figure 1: An example of a mass spectrum.

In a mass spectrum, there are several types of fragment ions that are the most important for a correct interpretation. The most frequent types of fragment ions with well predictable structure are y-ions and b-ions[4]. Each type of fragment ions forms a ion series, e.g., y-ions series or b-ions series (Fig. 1). The completeness of y-ions and b-ions series is crucial for correct spectra interpretation, because the mass difference between two neighboring peaks in one series, e.g., y_i and y_{i+1} corresponds to a mass of an amino acid.

Often, many of the y-ions or b-ions may never arise in the spectrometer and thus the number of missing y-ions and b-ions is too high to correct mass spectra interpretation. In fact, more than 85% of spectra captured by the spectrometer cannot be interpreted neither by an algorithm nor manually. However, there are more factors making the interpretation complex. Up to 80% of peaks in an experimental spectrum may correspond to fragment ions with very complicated or unpredictable chemical structure and they complicate the recognition of y-ions and b-ions. Such peaks are regarded as noise.

The interpretation of spectra may be also complicated due to chemical modifications of amino acids, because mass of amino acids are changed in that case and thus peaks are shifted. This may happen, e.g., during a sample preparation for mass analysis or in the spectrometer. The database UNIMOD [31] gathers discovered protein modifications for mass spectrometry. At the time of writing this paper, there were about 650 known modifications.

2.2 Graph-based Approaches

The mass spectra may be interpreted directly using graph algorithms (without any reference database). Such approaches are called *de novo* peptide sequencing [4] and they are based on detection of y-ions and b-ions series. A graph is constructed from an experimental spectrum, where a node corresponds to a peak (its mass m) and an edge is ranked with the mass difference between two nodes. The graph is traversed, while paths where weights of edges best fit the mass of amino acids are selected. A problem is that many paths and thus many peptide sequences can be assigned to an experimental spectrum, so the correctness of interpretation of such approaches is low (about 30%). This is due to noise, chemical modifications and the fact that some of y-ions or b-ions may never arise in the spectrometer. Some tools for mass spectra interpretation based on the *de novo* approach are, e.g., PEAKS [15], PepNovo [6] and Lutefisk [14].

[3]Dalton (Da) is a unit of the relative atom mass.

[4]In fact, more kinds of fragment ions with predictable structure may arise in the spectrometer, but many of them occur very rarely.

2.3 Similarity Search Approaches

The best way how the mass spectra may be interpreted is to search a database of already known or predicted peptide (protein, respectively) sequences [11, 26]. There are hypothetical mass spectra generated from peptide sequences, and an algorithm (mostly sequential) is used for similarity comparison of an experimental (query) spectrum with the hypothetical (database) spectra. The only difference is that fragment ions intensities cannot be generated from peptide sequences. The basic similarity functions for comparison of the experimental spectrum with hypothetical spectra generated from the database of protein sequences are, e.g., SPC [9] (shared peak count; in fact, the Hamming distance on boolean vectors, see Fig. 3), spectral alignment [23] (kind of dynamic programming distance on boolean vectors) [23], SEQUEST-like scoring [27]. The most common tools for mass spectra interpretation based on search in the database are SEQUEST [27], MASCOT [16], ProteinProspector [24], OMSSA [7], etc.

2.3.1 Metric Indexing

Since protein sequence databases grow rapidly and a sequential scan of the whole database becomes slow and inefficient, there is a need for utilization of index structures. A few methods for mass spectra interpretation based on metric access methods were proposed. Metric space approaches are usually based on variants of the cosine similarity (Sec. 4.1). One of them uses locality sensitive hashing to preprocess the database [5], another uses the MVP-tree [25]. The latter approach defines two alternatives of the cosine similarity. The first is called the fuzzy cosine distance, while the other is called the tandem cosine distance.

3. METRIC ACCESS METHODS

Since our approach to mass spectra interpretation is based on metric similarity search, we need to briefly summarize the main points concerning metric access methods (MAMs) [32] and their applicability. The MAMs were designed for efficient search in databases where a metric distance $d(x, y)$ is employed as the similarity function. The metric distance is a function that satisfies postulates of identity, symmetry, non-negativity and triangle inequality [32]. The metric postulates (especially the triangle inequality) are crucial for MAMs, in order to correctly organize database objects within metric regions and to prune irrelevant regions while searching. The MAMs usually support range and k-NN (k-nearest neighbor) queries. Among the vast number of MAMs developed so far, in our approach we have utilized the M-tree and Pivot tables.

3.1 M-tree

The *M-tree* [3] is a dynamic (updatable) index structure that provides good performance in secondary memory, i.e., in database environments. The M-tree index is a hierarchical structure, where some of the data objects are selected as centers (also called local *pivots*) of ball-shaped regions, while the remaining objects are partitioned among the regions in order to build up a balanced and compact hierarchy of data regions. When performing a query, the M-tree is traversed from the root, while the subtrees the regions of which overlap the query region must be searched as well, recursively.

3.2 Pivot Tables

A simple but efficient solution to similarity search represent methods called *pivot tables* (or distance matrix methods) [18]. In general, a set of l objects (so-called pivots) is selected from the database, while for every database object a l-dimensional vector of distances to the pivots is created. The vectors belonging to the database objects then form a distance matrix – the pivot table. When performing a kNN query, a distance vector for the query object q is determined the same way as for a database object. Then, the query is processed on the pivot table such that database object vectors which do not belong to the already retrieved kNN candidates are filtered out from further processing.

3.3 Intrinsic Dimensionality

The requirement on metric postulates is crucial for MAMs to index the database, however, the postulates alone do not guarantee an efficient query processing. The efficiency limits of any MAM also heavily depend on the distance distribution in the database S, and can be formalized by the concept of *intrinsic dimensionality* $\rho(S,d) = \frac{\mu^2}{2\sigma^2}$, where μ is the mean and the σ^2 is the variance of the distance distribution [2]. In other words, the intrinsic dimensionality is low if the data form tight clusters. Hence, the database can be efficiently searched by a MAM, because a query overlaps only a small number of clusters. On the other hand, a high intrinsic dimensionality (say, $\rho > 10$) indicates most of the data objects are more or less equally far from each other. Hence, in intrinsically high-dimensional database there do not exist clusters, while the search deteriorates to sequential search.

3.4 Non-metric and Approximate Search

The applicability of MAMs can be extended beyond the metric space model, so that MAMs could be used also for non-metric and/or approximate similarity search. In particular, given a *semi-metric distance* $d(x,y)$ (a metric distance violating the triangle inequality) and a database, the triangle inequality can be added to the semi-metric, so that we obtain a metric modification $f(d(x,y))$ that could be used for similarity search instead. Hence, the MAMs can be correctly used to index and search the database using the metric modification. Moreover, the enforcement of the triangle inequality could be only partial, where the "partial" metric distance could be used for approximate search by MAMs.

3.4.1 TriGen Algorithm

The TriGen algorithm [28] was proposed to keep a user-controlled amount of triangle inequality in a semi-metric distance. The idea is based on utilization of a T-modifier, which is either a concave or a convex increasing function f, such that f(0)=0. A concave function f, when applied on a semi-metric, increases the number of triplets $\langle f(d(x,y)), f(d(y,z)), f(d(x,z))\rangle$ that form a triangle (so-called *triangle triplets*), and so improves the triangle inequality fulfillment of $f(d)$. On the other hand, a convex T-modifier f does the opposite – it decreases the number of triangle triplets. Simultaneously, a concave modification $f(d)$ increases the intrinsic dimensionality, as it inhibits the differences between distances. Conversely, a convex modification $f(d)$ decreases the intrinsic dimensionality, as it magnifies the differences between distances. Formally, the proportion of triplets that are NOT triangular in a sample of examined triplets is called the *T-error*. Given a user-defined T-error tolerance, the TriGen algorithm was designed to find a T-modifier for which the intrinsic dimensionality $\rho(S, f(d))$ is minimized, while the T-error does not exceed the tolerance.

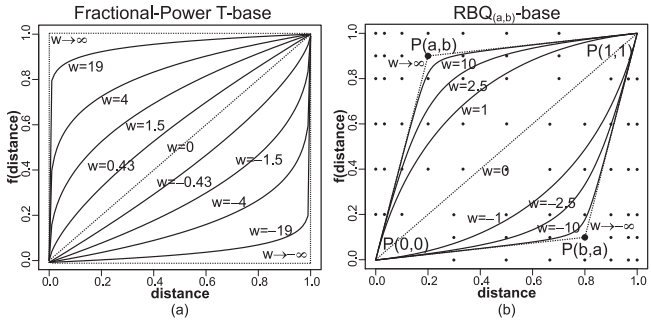


Figure 2: The FP-base and an RBQ$_{(a,b)}$-base.

In order to automate the search for the optimal T-modifier, the TriGen works with so-called *T-bases* $f(v,w)$. A T-base is a T-modifier with an additional parameter w, that aims to control to convexity or concavity of f. For $w > 0$, the f gets more concave, for $w < 0$ it gets more convex, and for $w = 0$ we get the identity $f(v,0) = v$. A simple T-base used by TriGen is the Fractional-Power base (FP-base) (1), while a more sophisticated T-base is the Rational-Bézier-Quadratic base (RBQ-base) (2), see Fig. 2. Actually, the TriGen uses multiple RBQ-bases – each for a particular pair (a,b).

$$\mathrm{FP}(v,w) = \begin{cases} v^{\frac{1}{1+w}} & \text{for } w > 0 \\ v^{1-w} & \text{for } w \le 0 \end{cases} \quad (1)$$

$$\mathrm{RBQ}_{(a,b)}(v,w) = \begin{cases} rbq(v,w,a,b) & \text{for } w > 0 \\ rbq(v,-w,b,a) & \text{for } w \le 0 \end{cases} \quad (2)$$

For description of the *rbq* function, see [28].

The modified distance $f(d)$ determined by TriGen can be then employed by any MAM for an exact but slower (T-error tolerance is zero, so ρ gets higher) or only an approximate but fast (T-error tolerance is positive, so ρ gets smaller) similarity search (metric or non-metric).

4. SIMILARITIES FOR MASS SPECTRA

Although the TriGen algorithm (Sec. 3.4.1) allows to use MAMs also with non-metric distances, it does not guarantee that a particular non-metric distance modified into metric will be suitable for indexing by MAMs. In particular, a highly non-metric distance (exhibiting high T-error) is modified by TriGen very aggressively to achieve zero T-error, which means the resulting metric will imply high intrinsic dimensionality of the database, thus making it not indexable.

This could be the case of non-metric distances mentioned in Section 2.3, where the T-error and intrinsic dimensionality was not observed as an important factor for indexability. Hence, when designing a new similarity that has to be indexable by MAMs, the attention should be given not only to the semantics of the similarity, but also to its indexability (low both the T-error and intrinsic dimensionality).

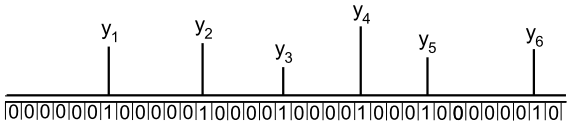


Figure 3: A high-dimensional boolean representation of a mass spectrum.

4.1 Cosine Similarity

The cosine similarity and its metric form, the angle distance, are commonly mentioned in mass spectrometry literature for peptide mass spectra interpretation [12, 29, 1, 25, 5]. The cosine similarity requires a representation of mass spectra as high-dimensional boolean vectors (Fig. 3). For example, let the range of $\frac{m}{z}$ values in a mass spectrum be 0-2,000 Da and let it be divided in subintervals of 0.1 Da. The mass spectrum is then represented by a 20,000-dimensional boolean feature vector having 1s at places corresponding to intervals for which the $\frac{m}{z}$ value is nonzero in the spectrum. Instead of storing the high-dimensional sparse vector x, there is usually a compact representation $\vec{x}$ used, where the positions of 1s in x (i.e., dimensions in which the values of x are nonzero) turn into values of the compact vector $\vec{x}$. The compact representation of vector x in Fig. 3 is $\vec{x} = \langle 7, 13, 18, 23, 27, 34\rangle$. We used a semimetric variant d_A of the angle distance based on the compact representation (4), where ξ is a mass error tolerance. Note that subintervals are not bounded as in Fig. 3 because the differences between $\frac{m}{z}$ values are computed.

$$d_a(\vec{x}_i, \vec{y}_j) = \begin{cases} 0, & \text{if } |\vec{x}_i - \vec{y}_j| > \xi \\ 1, & \text{else} \end{cases} \tag{3}$$

$$d_A(\vec{x}, \vec{y}) = \arccos\left(\frac{\sum_{x_i \in \vec{x}} \max_{y_j \in \vec{y}} \{d_a(\vec{x}_i, \vec{y}_j)\}}{\sqrt{\dim(\vec{x})\dim(\vec{y})}}\right) \tag{4}$$

However, the indexability of d_A showed to be inefficient due to the extremely high intrinsic dimensionality. In the experiments (Sec. 5.2) we show that even the utilization of TriGen algorithm with reasonably set T-error tolerance (say, up to 0.2) does not lead to an intrinsic dimensionality acceptable for indexing by MAMs. In particular, many mass spectra are maximally distant (having the angle $\frac{\pi}{2}$, i.e., $d_A = 1$), so that none of the T-modifiers can decrease the intrinsic dimensionality.

An approach for fast indexing under angle distance by MAMs was proposed in [25], where two semimetric alternatives of the cosine similarity have been defined. The first is called the fuzzy cosine distance and it has similar behaviour as d_A. The other is called the tandem cosine distance and it is, in fact, the combination of d_A with the precursor mass filter, i.e., the difference between the precursor mass m_p^x and m_p^y of the compared spectra was joined with d_A in order to reach good indexability (6), where c_1 and c_2 are constants and ξ' is the precursor mass error tolerance. A disadvantage is that the precursor mass filter could limit the capability of managing chemical modifications, because m_p of experimental spectra with modifications can differ by more than a few tens of Daltons from m_p of hypothetical spectra generated from the database of protein sequences.

$$d_{mp}(m_p^x, m_p^y) = \begin{cases} 0, & \text{if } |m_p^x - m_p^y| \le \xi' \\ |m_p^x - m_p^y|, & \text{else} \end{cases} \tag{5}$$

$$d'_A(\vec{x}, \vec{y}) = c_1 d_A(\vec{x}, \vec{y}) + c_2 d_{mp}(m_p^x, m_p^y) \tag{6}$$

4.2 Parametrized Hausdorff Distance

In this paper we employ the parameterized Hausdorff distance, that outperforms the angle distance d_A in both, the indexability by MAMs (i.e., efficiency) and correctness of mass spectra interpretation (i.e., effectiveness).

4.2.1 Logarithmic Distance

The first step towards d_{HP} was the logarithmic distance d_L (8) [19]. The d_L was defined for the compact representations $\vec{x}, \vec{y}$ of the high-dimensional boolean vectors x, y (Sec. 4.1). The logarithmic distance is more robust than the Euclidean distance (L_2). For our application (i.e., mass spectra interpretation) a distance is robust if two vectors $\vec{x}$ and $\vec{y}$ are closer if there are great differences in a small number of their components than if there are small differences in a large number of their components. For example, let us assume vectors $\vec{x} = \langle 200, 300, 400, 500\rangle$, $\vec{y_1} = \langle 200, 300, 460, 500\rangle$ and $\vec{y_2} = \langle 210, 305, 420, 475\rangle$. The missing number 400 in $\vec{y_1}$ with respect to $\vec{x}$ means that the corresponding peak in the mass spectrum is missing. The superfluous number 460 in $\vec{y_1}$ refers to a noise peak, so the vectors $\vec{x}$ and $\vec{y_1}$ should be closer than $\vec{x}$ and $\vec{y_2}$. However, the Euclidean distance (L_2) of the vectors $\vec{x}$ and $\vec{y_1}$ is higher. The d_L distance is lower and thus it models the similarity among mass spectra better ($d_L(\vec{x}, \vec{y_1}) \doteq 1.8$, $d_L(\vec{x}, \vec{y_2}) \doteq 4.4$, $L_2(\vec{x}, \vec{y_1}) = 60$, $L_2(\vec{x}, \vec{y_2}) \doteq 33.9$).

$$d_l(\vec{x}_i, \vec{y}_i) = \begin{cases} \log|\vec{x}_i - \vec{y}_i|, & \text{if } |\vec{x}_i - \vec{y}_i| > 1 \\ 0, & \text{else} \end{cases} \tag{7}$$

$$d_L(\vec{x}, \vec{y}) = \sum_i d_l(\vec{x}_i, \vec{y}_i) \tag{8}$$

4.2.2 Hausdorff distance

In general, the Hausdorff distance d_H [32] is a metric distance defined on sets A and B (10). The d_H is computed such that nearest neighbors from objects in one set are determined to all objects in the other set (both directions), while the maximal distance is reported as the result of d_H. The internal distance d_x operating on the objects of A, B could be any other distance (e.g., L_2 in case of point sets). If the d_x is a metric then the d_H is metric, too.

$$h(A, B) = \max_{a_i \in A}\left\{\min_{b_j \in B}\{d_x(a, b)\}\right\} \tag{9}$$

$$d_H(A, B) = \max(h(A, B), h(B, A)) \tag{10}$$

4.2.3 Parametrized Hausdorff Distance

The parametrized Hausdorff distance d_{HP} (13) is a semimetric, which combines advantages of the d_L and d_H [20]. The d_{HP} uses n^{th} root function instead of logarithm because of higher correctness of interpretation. Also, compact vectors of different dimensions can be used which is

desirable because the mass spectra have different numbers of peaks and thus the compared compact vectors have different lengths. A property inherited from d_H is its robustness – unlike d_L, the values in $\vec{x}, \vec{y}$ are compared based on their closeness, not the same position (dimension) in the vectors. Since the values in $\vec{x}$, $\vec{y}$ are ordered, the d_{HP} computation is of linear complexity (unlike the general d_H) [20]. Moreover, using of the time expensive n^{th} root function does not cause any problem, because the range of mass corresponding to generated peptide sequences is limited and thus a table of the roots can be precomputed.

$$d_e(\vec{x}_i, \vec{y}_j) = \begin{cases} |\vec{x}_i - \vec{y}_j|, & \text{if } |\vec{x}_i - \vec{y}_j| > \xi \\ 0, & \text{else} \end{cases} \quad (11)$$

$$h'(\vec{x}, \vec{y}) = \frac{\sum_{\vec{x}_i \in \vec{x}} \sqrt[n]{\left(\min_{\vec{y}_j \in \vec{y}} \{d_e(\vec{x}_i, \vec{y}_j)\}\right)}}{\dim(\vec{x})} \quad (12)$$

$$d_{HP}(\vec{x}, \vec{y}) = \max(h'(\vec{x}, \vec{y}), h'(\vec{y}, \vec{x})) \quad (13)$$

where $dim(\vec{x})$ is the dimension/length of the compact vector $\vec{x}$. The internal distance d_e measures the difference between two values, while only distances exceeding threshold ξ (mass error tolerance) are taken into account.

4.2.4 Modifying d_{HP} by TriGen

Although d_{HP} is generally a semi-metric distance, its T-error is very low (below 0.001) but its intrinsic dimensionality is very high (above 100). Thus, we have used TriGen to improve the intrinsic dimensionality, setting the T-error tolerances to the range 0.001 – 0.2. The FP-base and 454 different RBQ-bases (different points (a, b)) were used by TriGen. Note that d_A and d_{HP} must be normalized to $\langle 0, 1 \rangle$ in order to employ the TriGen. The d_A is normalized by $\frac{\pi}{2}$, while d_{HP} by $\sqrt[n]{d_e^{max}}$, where d_e^{max} is the maximal possible value in a compact vector (i.e., the dimension of the high-dimensional representation).

For all the T-error tolerances the TriGen found convex T-modifiers ($w < 0$), so the intrinsic dimensionality was reduced (down to 2 for T-error tolerance 0.2). The resulting modifiers determined by TriGen for the d_{HP} and $n = 50$ with lowest intrinsic dimensionality are shown in Tab. 1. The intrinsic dimensionality ρ using RBQ modifiers is slightly better than ρ using the FP modifier, however, testing many RBQ modifiers is time consuming.

	FP(v,w)			$RBQ_{(a,b)}$(v,w)				
T-err.tol.	ρ	T-err.	w	ρ	T-err.	a	b	w
0.001	18.6	0.001	-2.6	15.7	0.001	0.22	0.82	-3.1
0.01	7.6	0.013	-5.0	6.8	0.013	0.22	0.82	-11.3
0.02	6.0	0.023	-5.9	5.7	0.020	0.22	0.82	-20.5
0.04	4.5	0.042	-7.0	4.6	0.042	0.13	0.83	-7.6
0.06	3.8	0.062	-7.9	3.7	0.061	0.13	0.83	-10.4
0.08	3.3	0.082	-8.6	3.1	0.081	0.13	0.83	-15.3
0.1	3.0	0.102	-9.2	2.8	0.092	0.13	0.83	-20.4
0.12	2.8	0.120	-9.6	2.3	0.112	0.13	0.83	-54.9
0.14	2.6	0.138	-10.1	2.7	0.140	0.05	0.85	-6.4
0.16	2.4	0.154	-10.5	2.5	0.160	0.05	0.85	-7.1
0.18	2.3	0.173	-10.9	2.3	0.174	0.05	0.85	-7.5
0.2	2.2	0.191	-11.3	2.1	0.196	0.05	0.85	-8.4

Table 1: Empirically determined FP and RBQ modifiers and intrinsic dimensionality ρ for d_{HP}.

4.2.5 Precursor Mass Filter

As well as d_A, the d_{HP} can be extended with the precursor mass filter in order to reach much better indexability even if TriGen is not employed (14). A disadvantage is that later extension of d_{HP} for analysing of chemical modifications in the query mass spectra may be limited (Sec. 4.1).

$$d'_{HP}(\vec{x}, \vec{y}) = c_1 d_{HP}(\vec{x}, \vec{y}) + c_2 d_{mp}(m_p^x, m_p^y) \quad (14)$$

4.3 Interpretation using Similarity Search

The entire process of peptide mass spectra interpretation we propose can be summarized as follows:
1) Each protein sequence in the database is split to shorter peptide sequences. The rules for the splitting are determined by an enzyme. The most common enzyme is trypsin, which splits the protein chains after each amino acid K (lysine) and R (arginine) if they are not followed by P (proline) [21]. However, even if the splitting sites are well predictable, the process is not perfect in practice and there can some missed cleavage sites occur. The maximum number of missed cleavage sites $\max_{cs}$ is adjusted as a parameter.
2) The $\frac{m}{z}$ values of y- and b-ions are generated in ascending order for each peptide sequence, while each sequence corresponds to one indexed vector. The vector for the peptide sequence of the length l has the dimension $2(l-1)$, see Fig. 1.
3) The vectors are indexed by a MAM (e.g., by the M-tree or Pivot tables) under d_{HP} modified by the TriGen (Sec. 4.2.4).
4) The experimental spectrum is preprocessed before interpretation. The p peaks with highest intensity I from the experimental spectrum are selected and they form a query corresponding to a vector of their $\frac{m}{z}$ values.
5) A kNN query is processed by the MAM, while the correct peptide sequence corresponding to the spectrum could be obtained as the first neighbor in many cases. However, in real-world applications we need to provide more nearest neighbors, because an additional scoring algorithm could select a different peptide as the correct one from the kNN set. Such refining algorithm could be, e.g., SPC, spectral alignment, SEQUEST-like scoring (Sec. 2).

In the experimental results we suppose that a mass spectrum is successfully interpreted if the correct peptide sequence is among the k nearest neighbors (regardless of its position in the kNN result). Such an approach is often employed and the scoring is then handled separately. Hence, the overall setup of our method can be utilized as a coarse filter by any other database approach for mass spectra interpretation.

5. EXPERIMENTS

In our experiments, we used mass spectra from [10], which was obtained from 14 mass spectrometer runs on protein mixture A and 8 runs on protein mixture B. We used two query sets in our experiments. The first set Q_1 contained 119 spectra from the first run on mixture A and the second query set Q_2 contained 1941 spectra from all runs on both mixtures. The spectra split by trypsin were selected, interpreted by SEQUEST [27] and the results were manually checked by domain experts. Hence, we consider the SEQUEST-interpreted results as the confirmed ground truth.

The databases DB_1 and DB_2 were extensions of the file with correct protein sequences assigned to the mass spectra in [10]. The databases were extended with protein sequences from MSDB (release 08-31-2006) [17]. The DB_1 contained 100,000 protein sequences (5,600,747 peptide sequences) and DB_2 contained 500,000 protein sequences (29,460,880 peptide sequences).

In the experiments, the following values were measured:
a) The correctness of interpretation as a ratio of correctly assigned peptide sequences to all spectra from a query set.
b) The distance computations as the ratio of average number of distance computations per one mass spectrum interpretation to the cost of sequential scan.
c) The average query time per one mass spectrum interpretation.

All experiments were carried out on a machine with 2 processors Intel Xeon E5450 (8 cores × 3GHz) with 8 GB RAM and 64-bit OS Windows Server 2008 R2. We used a C++ parallel implementation of the M-tree and the Pivot tables for indexing and querying [13]. We have also implemented parallel version of the sequential scan, as a baseline access method. Our implementation was based on Intel's Threading Building Blocks (TBB) [30]. By default, the average query time per one mass spectrum interpretation was measured on 8 processor cores.

The following settings were used unless otherwise specified – the d_{HP} was computed with $n = 50$, the splitting enzyme was trypsin, the maximum missed cleavage sites ($\max_{cs}$) was set to 1, the mass error tolerance (ξ) was 0.4 Da, the precursor mass error tolerance (ξ') was 2 Da, 100 peaks with the highest intensity were selected from experimental spectra. The average length of indexed vectors (the compact representation) was about 28.8 and it could be halved if just y-ions were generated instead of both y- and b-ions, however, the correctness of interpretation would be much worse.

5.1 Sequential Scan

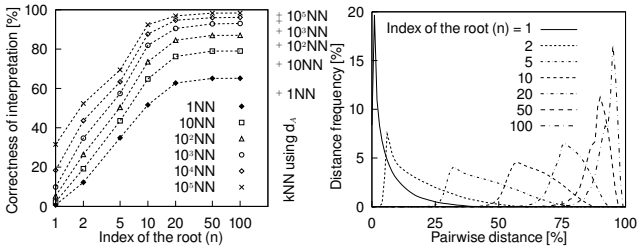


Figure 4: a) Correctness of interpretation (sequential scan), b) Distance distributions for different n.

First, the sequential scan was employed for the d_{HP} and d_A, while the correctness of interpretation and average query time were measured on DB_2 and Q_2. The correctness of interpretation is higher with increasing index of the root n (Fig. 4a). A comparison with the angle distance d_A is shown for different k in kNN queries (Fig 4a on the right). The d_{HP} has shown better correctness of interpretation than d_A. The application of the n^{th} root function in the d_{HP} has two main advantages. First, the similarity among mass spectra is modeled very well and second, the d_{HP} becomes almost a metric distance. In particular, the T-error for d_{HP} was about 0.83 for $n = 1$, but less than 0.01 for $n \geq 2$.

5.2 Improving the Indexability

A disadvantage of the n^{th} root function is that intrinsic dimensionality ρ increases with the increasing n, hence the difference between MAMs and sequential scan disappears for high n. In Fig. 4b see the distance distributions under d_{HP} (not modified by TriGen) for various n. The more the distribution is pushed to the right, the higher the intrinsic dimensionality.

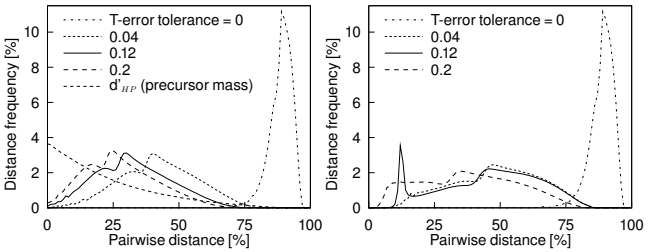


Figure 5: Distance distrib. of d_{HP} + a) FP, b) RBQ

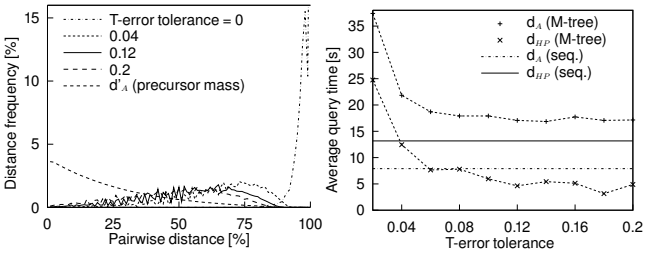


Figure 6: a) Distance distrib. of d_A + FP b) Average query time in M-tree (d_A and d_{HP}) + FP

In Fig. 5 observe the impact of T-error tolerance on the distance distributions obtained using the TriGen-modified d_{HP}, considering either FP-base or RBQ-bases. Obviously, a higher T-error tolerance leads to more convex T-modifier, and so to lower intrinsic dimensionality (distance distribution pushed to the left). In Fig. 6a, see the same for angle distance d_A, which shows its poor indexability by MAMs. In fact, about 35% of all pairwise distances are maximal $d_A = 1$ (not shown for all T-error tolerances in Fig. 6a for better readability). Note that these 35% distances cannot be fixed by the TriGen algorithm, as they are indistinguishable. In Fig. 5a and Fig. 6a, the distances d'_{HP} and d'_A (c_1=1, c_2=1) are shown for comparison (TriGen was not employed). Although theirs indexability is good, an extension of these distances for search of chemical modifications in the mass spectra may be too complicated to practical use.

The d_{HP} was compared with d_A on DB_2 indexed by M-tree, where 1000NN queries from Q_2 were used (Fig. 6b). While the d_A was 2.5× slower than sequential scan, the d_{HP} was 1.6× faster. The correctness of interpretation was 1.4× better for the d_{HP} than for d_A (compare with Fig. 7a). The average query time for d'_A and d'_{HP} was 0.4 s. The correctness of interpretation was 89.6% for d'_A and 85.7% for d'_{HP}.

5.3 Correctness of Interpretation

The following experiments were carried out on DB_2 and Q_2, while the M-tree was employed. The correctness of mass spectra interpretation is worse with increasing T-error tolerance. The kNN queries with higher k can be used to avoid this problem (Fig. 7a). The correct peptide sequences are not spread uniformly over all interval $\langle 1..k \rangle$ of a kNN query

result set but they are cumulated among a few nearest neighbors in many cases. As shown in Fig. 7b, the first nearest neighbor taken from the 100NN result was more likely to be correct than when taking the first nearest neighbor from 10NN result. The average query time of a kNN query and its distance computations ratio (wrt. sequential scan) increases with k (Fig. 8). The best results were obtained at T-error 0.06, while the correctness of interpretation was 75%, speed-up of the M-tree with respect to sequential scan was 1.7× and the distance computations ratio was 9.7%.

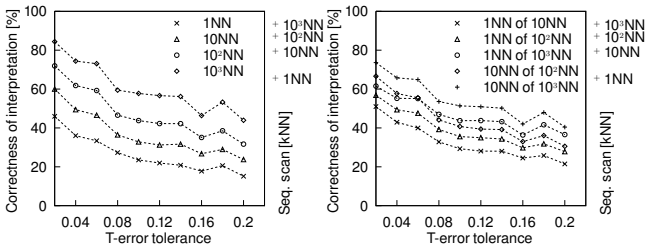


Figure 7: Correctness of interpretation (d_{HP})

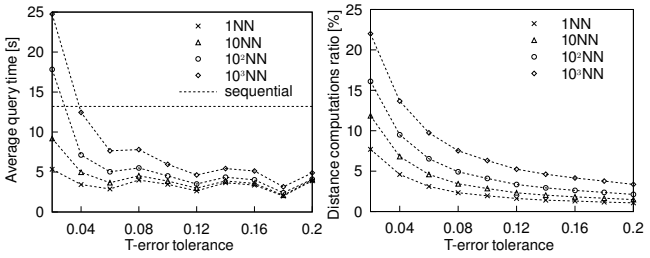


Figure 8: kNN query – a) average query time, b) distance computations ratio (wrt. sequential scan)

5.4 Comparison of M-tree and Pivot Table

We compared the efficiency of the d_{HP} (FP) indexed by M-tree and Pivot tables with the sequential scan. The experiments were made on DB_1 and Q_1, 2000NN queries were used, and 8 processor cores were employed. The Pivot table was constructed for 40 randomly selected pivots. The distance computations ratio is smaller (wrt. seq. scan) if the T-error tolerance is higher. The best results were obtained using the Pivot table for the T-error tolerance 0.04 and higher (Fig. 9b). However, the best average query time was obtained for the M-tree (Fig. 9a). Since the Pivot table was stored in the main memory, it was also fast, but the size of the Pivot table was almost 2 GB. The correctness of interpretation decreases with increasing T-error tolerance for both the M-tree and the Pivot table. (Fig. 10a).

We made comparison of the M-tree and Pivot tables on different number of processor cores (Fig. 10b) with the same settings. The M-tree on 8 cores was about 6.6× faster than M-tree on 1 core, the Pivot table was 4.9× faster and the sequential scan was 6.3× faster. If the mass spectra interpretation ran using the M-tree on 8 cores, the speed-up would be 40.1× against the sequential interpretation on 1 core.

The performance of the M-tree and Pivot tables (50 pivots) using d_{HP} was also examined on differently sized da-

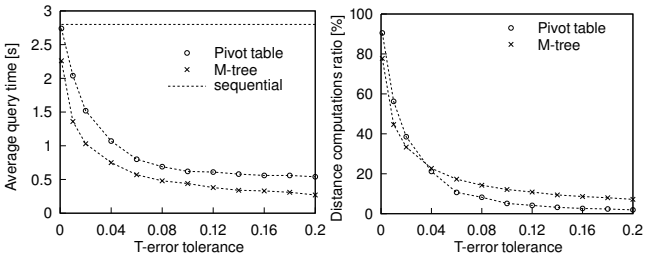


Figure 9: a) average query time, b) distance computations ratio

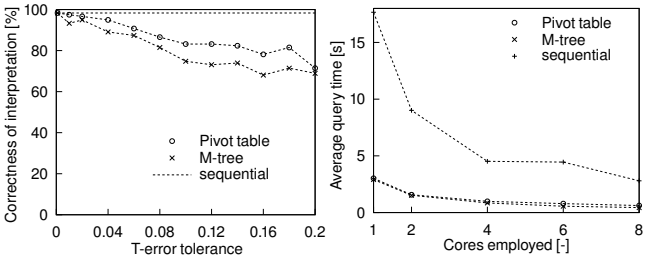


Figure 10: a) correctness of interpretation, b) number of processor cores

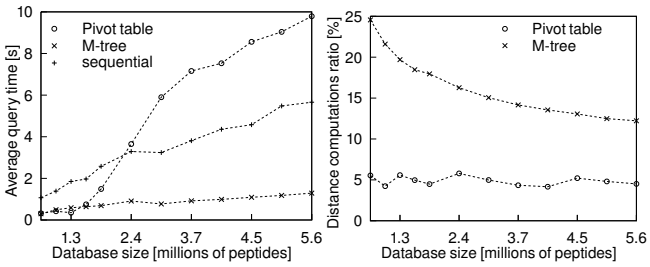


Figure 11: Database size - a) average query time, b) distance computations ratio

tabases of proteins from DB_1 (from 10 to 100 thousands of proteins; from 650 thousands to 5.6 millions of peptides or indexed vectors). The Fig. 11a shows the impact of database size on average query time, while the T-error tolerance was set to 0.1. The Pivot table is faster than M-tree as long as all its blocks are stored in main memory. If the main memory size is exceeded then Pivot table becomes inefficient. We had allocated 600 MB buffer in main memory and it was exceeded by Pivot table after 25 thousands of protein sequences (1.5 millions of peptides) were indexed. Moreover, for more than 40 thousands of protein sequences (2.4 millions of peptides) the sequential scan outperformed the Pivot tables. Fig. 11b shows that distance computations are misleading for Pivot tables when the size of main memory is exceeded.

6. CONCLUSIONS

The best way how to interpret the tandem mass spectra of peptides is to search a database of already known or predicted protein sequences. We have shown that M-tree and

parametrized Hausdorff distance (d_{HP}) is a powerful combination for tandem mass spectra interpretation. The d_{HP} models the similarity among spectra very well and it can be utilized by MAMs when TriGen algorithm is employed. In general, if the T-error is higher, then indexability of the d_{HP} by MAMs is much better, the search is faster and correctness of interpretation is a little lower.

The d_{HP} models the similarity among mass spectra better than angle distance, which is commonly mentioned in mass spectrometry literature. We verified the conclusions presented in [25] that the angle distance has its limitations for utilization by MAMs in terms of mass spectra interpretation and that the angle distance in combination with the precursor mass filter is indexable very well. Moreover, we have shown that the precursor mass filter can be easily joined with d_{HP} as well as with the angle distance. A disadvantage is that combination of the angle distance or d_{HP} with the precursor mass filter might not be applicable for the query mass spectra containing chemical modifications, which is in practice a relatively frequent phenomenon. In our future work we plan to use the PM-tree for mass spectra interpretation, which could speed-up the whole task by an order of magnitude and we plan to deal with chemical modifications in the query mass spectra.

7. ACKNOWLEDGMENTS

This research has been supported in part by Czech Science Foundation project Nr. 201/09/0683 and by the grant SVV-2010-261312.

8. REFERENCES

[1] Z. B. Alfassi. On the Normalization of a Mass Spectrum for Comparison of Two Spectra. *Journal of the American Society for Mass Spectrometry*, 15(3):385–387, 2004.

[2] E. Chávez and G. Navarro. A Probabilistic Spell for the Curse of Dimensionality. In *ALENEX'01, LNCS 2153*, pages 147–160. Springer, 2001.

[3] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proc. of 23rd Int. Conf. on VLDB*, pages 426–435, 1997.

[4] V. Dančík, T. A. Addona, K. R. Clauser, J. E. Vath, and P. A. Pevzner. De Novo Peptide Sequencing via Tandem Mass Spectrometry. *Journal of Computational Biology*, 6(3):327–342, 1999.

[5] D. Dutta and T. Chen. Speeding up Tandem Mass Spectrometry Database Search: Metric Embeddings and Fast Near Neighbor Search. *Bioinformatics Oxford Journal*, 23(5):612–618, 2007.

[6] A. Frank and P. Pevzner. PepNovo: de novo peptide sequencing via probabilistic network modeling. *Analytical Chemistry*, 77(4):964–973, 2005.

[7] L. Y. Geer and et al. Open Mass Spectrometry Search Algorithm. In *Journal of Proteome Research*, volume 3, pages 958–964, 2004.

[8] D. F. Hunt and et al. Protein Sequencing by Tandem Mass Spectrometry. In *Proc. Nati. Acad. Sci. USA*, volume 83, pages 6233–6237, 1986.

[9] N. C. Jones and P. A. Pevzner. *An Introduction to Bioinformatics Algorithms*. MIT Press, Cambridge, Massachusetts, 2004.

[10] A. Keller and et al. Experimental Protein Mixture for Validating Tandem Mass Spectral Analysis. *OMICS: A Journal of Integrative Biology*, 6(2):207–212, 2002.

[11] M. Kinter and N. E. Sherman. *Protein Sequencing and Identification Using Tandem Mass Spectrometry*. John Wiley & Sons, New York, USA, 2000.

[12] J. Liu and et al. Methods for peptide identification by spectral comparison. *Proteome Science*, 5(3), 2007.

[13] J. Lokoč. Parallel dynamic batch loading in the m-tree. In *SISAP 2009, IEEE*, pages 117–123, 2009.

[14] Lutefisk. http://www.hairyfatguy.com/Lutefisk/.

[15] B. Ma and et al. PEAKS: Powerful Software for Peptide De Novo Sequencing by MS/MS. http://www.bioinformaticssolutions.com/.

[16] MASCOT. http://www.matrixscience.com/.

[17] Mass Spectrometry Protein Sequence Database. http://www.proteomics.leeds.ac.uk/bioinf/msdb.html.

[18] G. Navarro. Analyzing metric space indexes: What for? In *SISAP 2009, IEEE*, pages 3–10, 2009.

[19] J. Novák and D. Hoksza. An Application of the Metric Access Methods to the Mass Spectrometry Data. In *CIBCB 2009, IEEE*, pages 220–227, 2009.

[20] J. Novák and D. Hoksza. Parametrised Hausdorff Distance as a Non-Metric Similarity Model for Tandem Mass Spectrometry. In *CEUR Proc. DATESO 2010*, pages 1–12, 2010.

[21] J. V. Olsen, S. Ong, and M. Mann. Trypsin Cleaves Exclusively C-terminal to Arginine and Lysine Residues. *Molecular and Cellular Proteomics*, 3:608–614, 2004.

[22] G. Petsko and D. Ringe. *Protein Structure and Function (Primers in Biology)*. New Science Press Ltd, London, UK, 2004.

[23] P. A. Pevzner, Z. Mulyukov, V. Dančík, and C. L. Tang. Efficiency of Database Search for Identification of Mutated and Modified Proteins via Mass Spectrometry. *Genome Research*, 11(2):290–299, 2001.

[24] ProteinProspector. http://prospector.ucsf.edu/.

[25] S. R. Ramakrishnan and et al. A Fast Coarse Filtering Method for Peptide Identification by Mass Spectrometry. *Bioinformatics*, 22(12):1524–31, 2006.

[26] R. G. Sadygov and et al. Large-scale Database Searching Using Tandem Mass Spectra: Looking up the Answer in the Back of the Book. *Nature Methods*, 1(3):195–202, 2004.

[27] SEQUEST. http://fields.scripps.edu/sequest/.

[28] T. Skopal. Unified Framework for Fast Exact and Approximate Search in Dissimilarity Spaces. *ACM Transactions on Database Systems*, 32(4):29, 2007.

[29] D. L. Tabb and et al. Similarity among Tandem Mass Spectra from Proteomic Experiments: Detection, Significance and Utility. *Anal. Chem.*, 75(10), 2003.

[30] Threading Building Blocks (TBB). http://www.threadingbuildingblocks.org/.

[31] UNIMOD. http://www.unimod.org/.

[32] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach (Advances in Database Systems)*. Springer, USA, 2006.

Case Study: An Inverted Index for Mass Spectra Similarity Query and Comparison with a Metric-space Method

Rui Mao
Shenzhen University
3688 Nanhai Road, Office Tower #342
Shenzhen, Guangdong, 518060, China
(+86)755 2655 8644
mao@szu.edu.cn

Smriti R. Ramakrishnan
University of Texas at Austin
1 University station, C0500
Austin, TX 78712, USA
(+1)512 471 9541
smriti@cs.utexas.edu

Glen Nuckolls
NetApp
1601 Trapelo Road
Waltham, MA 02451, USA
(+1)512 471 9541
glen.nuckolls@gmail.com

Daniel P. Miranker
University of Texas at Austin
1 University station, C0500
Austin, TX 78712, USA
(+1)512 471 9541
miranker@cs.utexas.edu

ABSTRACT
Query performance is a determining factor in the adoption of an indexing method for similarity query. Metric space indexing methods take great pride in their general applicability. However, it is usually hard for a general method to perform well for every domain. Therefore, it is of interest to investigate the performance of metric-space methods, comparing with domain specific methods, on a particular domain. This paper describes such an investigation for proteomic mass spectra. An inverted index method that exploits the sparsity of mass spectra binary format data and acts as a coarse filter before fine ranking is proposed and empirically compared with an existing metric-space indexing method. Results show that the inverted index method yields greater search efficiency and outperforms the metric-space method in query speed and index size.

Categories and Subject Descriptors
H.2.2 [**Database management**] Physical Design – *Access methods*; H.3.1 [**Information storage and retrieval**]: Content analysis and indexing — *indexing methods*; H.3.3 [**Information storage and retrieval**]: Information search and retrieval — *clustering*, *search process*

General Terms
Algorithms.

Keywords
Similarity query, metric-space indexing, mass spectra, sparse matrix, inverted index.

SISAP2010, September 18-19, 2010, Istanbul, Turkey.

1. INTRODUCTION
Tandem mass spectrometry (MS/MS) is the mainstream high-throughput technology for detecting proteins in complex samples containing thousands of proteins [10, 14]. A single MS/MS experimental run can generate tens of thousands of spectra, each containing a list of measured mass-to-charge ratios called peaks (real numbers), and the intensity of measurement per peak. Each spectrum represents a fragment of a protein sequence. The task is to identify which proteins are present in the sample, by labeling each spectrum with its correct amino-acid sequence. Multiple identified fragments from a given protein are evidence of its presence in the sample.

We focus on the spectrum identification step of the process, which can be formulated as a similarity search problem in which experimentally generated spectra are matched against a database of idealized "expected" spectra. The mapping is not exact due to several experimental and biological noise factors that necessitate an approximate search framework.

The exponential growth of protein database size [7] necessitates computational methods to analyze spectral data. Most of the analysis time is spent in the database search phase that matches experimental and theoretical spectra. Post-processing to compute peptide and protein scores takes only a few minutes in comparison. As a result, indexing method supporting rapid similarity search of mass spectra has become very popular.

A number of protein identification methods using mass spectra have been developed, such as MASCOT [11], ProFound [20], TruboSEQUEST [18] and clustering method [4]. The similarity measures include cosine distances based on shared peak count [12, 14] and Hausdorff distance [10].

Distance-based indexing [2, 6, 15,19], also known as metric-space indexing, does not make use of data domain information. The available information about the data is derived solely from an oracle that computes the distance between pairs of objects. The only requirement is that the distance function be a metric. The

data set can be clustered and a data structure compiled off-line. During an on-line search, the triangle inequality enables the elimination of clusters of data as possible solutions. As a consequence, on-line similarity queries may be quickly computed.

This, so-called, "black-box" model is advantageous for any application where the data cannot be effectively mapped to feature vectors. Even though one is not inclined to use distance-based indexing for multi-dimensional problems, distance-based indexing subsumes multi-dimensional indexing, enabling a uniform programming model for problems that can be recast into metric spaces.

In the context of the MoBIoS project [8], Ramakrishnan et al. propose a metric space method, MSFound, for similarity search of mass spectra [14]. They discuss common similarity measures, and define a cosine-distance based semi-metric distance to compare spectra. They use this distance function in conjunction with MVP Tree [1] to run range and k-NN queries. The system acts as coarse filter to generate a candidate set of results, which may then be ranked by a more discriminative confidence measure in a post-processing step [14]. The top match is regarded as the result.

Dutta and Chen [3] applied another metric-space indexing technique, Locality Sensitive Hashing (LSH), to mass spectra searches. The LSH algorithm [5] is an efficient hash-based indexing structure for nearest neighbor queries that also provides elegant probabilistic bounds on the error of the returned results [17].

The great generality of metric space indexing is also its challenge. It is not hard to imagine that a general method may not perform very well for a particular domain. It is of interest to investigate the performance of metric-space methods on a particular domain, comparing with a method that is specific to that domain. We propose a domain specific indexing method for mass spectra similarity search. This method is compared with MSFound [14] with the same tandem cosine distance.

Mass spectra can be represented in high dimensional binary vector format with sparsity as high as 99.9%. Our method is based on inverted index, a technique commonly used for sparse matrices. Experimental results demonstrate the high search efficiency of our method and its faster search time and smaller index size than MSFound [14].

2. MASS SPECTRA AND TANDEM COSINE DISTANCE

The representation of mass spectra and the tandem cosine distance are already elaborated by Ramakrishnan et al. [14]. They are briefly introduced here for the completeness of this paper.

Each mass spectra data record consists of a single precursor mass (M) and a list of real-valued m/z peaks, $P=\{p_i\}$.

Given the range of peaks $[M_1, M_2]$ Da, and a resolution $0 \le M_{res} \le 1.0$Da, the range can be divided into buckets of width M_{res}. Let the buckets be numbered by 0, 1, … Then P can be represented as a high dimensional binary vector S, where S[i] is 1 only if there exists $p \in P$ where p falls into bucket i [b06]. That is:

$$s[i] = \begin{cases} 1, \exists p \in P, i < \dfrac{(p - M_1)}{M_{res}} \le i+1 \\ 0, otherwise \end{cases}$$

For a common peak range [100, 5000] Da and a resolution 0.2 Da, S is a sparse vector of nearly 25000 dimensions.

This paper deals with mass spectra in its binary format. That is, each mass spectra data record is a pair {M, S}.

Given mass spectra data records $A = \{M_A, S_A\}$ and $B = \{M_B, S_B\}$, the shared peak count SPC(A, B) of A and B is the number of shared 1s of their binary peak vector [14]. That is, SPC(A, B) is the dot product of S_A and S_B:

$$SPC(A, B) = S_A \bullet S_B$$

To account for small peak shifts caused by measurement and calibration error of the mass spectrometer, shared peak count with tolerance, SPC_t, is defined. That is, two shared peaks may not fall into the same bucket, but can be within t buckets [14]:

$$SPC_t(A,B) = \sum_{i:S_A[i]=1} match(i,B)$$

$$match(i,B) = \begin{cases} 1, \exists j \in [i-t, i+t], S_B[j] = 1, \text{j is not matched with other i} \\ 0, otherwise \end{cases}$$

Note that $SPC_t(A, B)$ is bounded by the number of 1s in S_A or S_B.

Now, the fuzzy cosine distance of A and B, $D_{ms}(A, B)$, is defined as [14]:

$$D_{ms}(A,B) = \arccos\left(\frac{SPC_t(A,B)}{\|S_A\|\|S_B\|}\right).$$

Note that $0 \le D_{ms}(A, B) < \pi/2$.

The precursor mass distance $D_{pm}(A, B)$ computes the difference of precursor mass within a tolerance window [14]:

$$D_{pm}(A,B) = \begin{cases} 0, & |M_A - M_B| \le \tau_{pm} \\ |M_A - M_B|, & otherwise \end{cases}$$

A query and its correct match are expected to have common precursor mass and a set of common peak values. In reality, a distance metric must account for measurement errors since experimental peaks can differ from the theoretical database peaks by a few Da depending on the instrument. Therefore, the tandem cosine distance of mass spectra, the distance function advocated by MSFound [14], is defined as a linear combination of fuzzy cosine distance and precursor mass distance [b06]:

$$D_{tcd}(A, B) = C_1 D_{ms}(A, B) + C_2 D_{pm}(A, B).$$

Note that the tandem cosine distance is a semi-pseudometric distance [14].

3. AN INVERTED INDEX METHOD

The key idea of this domain specific method is to take use of the sparseness of the mass spectra binary peak vector. A binary peak

vector S can also be represented by a compressed vector $\overline{S}$ of the subscripts of non-zero entries of S in ascending order:

$$\overline{S} = [k_1, k_2, \ldots], S[k_i] = 1.$$

```
SPCt(S_A[1, 2, …], S_B[1, 2, …])
{
    if (|S_A[1] - S_B[1]|) ≤ t
        return 1 + SPCt(S_A[2, …], S_B[2, …])
    else if (S_A[1] < S_B[1])
        return SPCt(S_A[2, …], S_B[1, 2, …])
    else
        return SPCt(S_A[1, 2, …], S_B[2, …])
}
```

Figure 1. Recursive algorithm for SPC_t

Figure 1 gives a short recursive algorithm to compute the shared peak count with tolerance t between two compressed peak vectors $\overline{S_A}$ and $\overline{S_B}$. This algorithm counts the first possible match between two vectors and then eliminates both entries from contributing to a subsequence match. It is not hard to argue the correctness of this algorithm.

Since the above algorithm has to traverse both compressed vectors, one of the performance goals in designing the indexing method is to reduce the number of SPC_t computations.

3.1 Bulkloading the index

Since the computation of precursor mass distance is of low cost and the fuzzy cosine distance is bounded by $[0, \pi/2)$, it is more efficient to compute precursor mass distance first and prune some data before computing fuzzy cosine distance. Therefore, the database is first sorted by precursor mass during bulkload.

An inverted index is maintained to speed up the computation of fuzzy cosine distance. Let N be the dimension of the binary peak vectors and M be the total number of vectors, also let I index the dimensions and j index the vectors. Then an inverted index L is defined as:

$$L = \{L_i \mid L_i = \{j \mid S_j[i] = 1, j = 1, \ldots, M\}, i = 1, \ldots, N\}$$

Compressed vectors	Inverted index
$S_1 = [1, 4]$	$L_1 = [1, 2, 7]$
$S_2 = [1, 4, 5]$	$L_2 = [3, 4, 5]$
$S_3 = [2, 4]$	$L_3 = [\ ]$
$S_4 = [2]$	$L_4 = [1, 2, 3, 8]$
$S_5 = [2, 5]$	$L_5 = [2, 5, 6, 7]$
$S_6 = [5]$	
$S_7 = [1, 5]$	
$S_8 = [4]$	

Figure 2. An example of inverted index

That is, for each dimension of the binary peak vectors, a list is stored, consisting of the id of peak vectors whose such dimension is 1. Figure 2 shows an example of inverted index with N = 5 and M = 8. L can be organized as an array of lists to support random access of list heads.

3.2 Range query processing

We consider range query R(q, r) in this section. K nearest neighbor queries can be handled in essentially the same way.

Two pruning rules are exploited to reduce the number of SPC_t computations. They are given as theorems with brief proof in the following.

Theorem 1: (1) A is a query result of range query R(q, r) if $M_q - \max(\tau_{pm}, (r - C_1\pi/2)/C_2) \le M_A \le M_q + \max(\tau_{pm}, (r - C_1\pi/2)/C_2)]$ and $r - C_1\pi/2 > 0$; (2) A is not a query result of range query R(q, r) if $M_A > M_q + \max(\tau_{pm}, r/C_2)$, or $M_A < M_q + \max(\tau_{pm}, r/C_2)$.

Proof:

(1) is proved using the fact that $D_{ms} < \pi/2$; (2) is proved using the fact that $0 \le D_{ms}$. □

1. Prune data using bounds of precursor mass computed from Theorem 1. Put data satisfying Theorem 1 (1) in to a result set, and data satisfying Theorem 1 (2) into a candidate set, together with their precursor mass distance to q.
2. Search inverted index to compute $GSPC_t(q, A)$ for any database point A that $GSPC_t(q, A) > 0$, and A appears in the candidate set.
3. Prune data in the candidate set using Theorem 2.
4. For each element of the candidate set, compute its fuzzy cosine distance using algorithm in Figure 1 to answer the query.

Figure 3. Steps of range query processing

Definition: Given a query mass spectra q, Let L(q) be the set of lists in the inverted index that are within tolerance t to one of q's non-zero entry. That is:

$$L(q) = \{L_i \mid L_i \in L, \exists j, \mid i - j \mid \le t, S_q[j] = 1\}$$

Then the **gross shared peak count with tolerance t** of q and a mass spectra A in the database, $GSPC_t(q, A)$, is defined as the number of appearances of A in lists of (q).

$GSPC_t$ can be easily computed from the inverted index and is used in Theorem 2 for pruning.

Theorem 2: A is not a query result of range query R(q, r) if

$$\arccos\left(\frac{GSPC_t(q, A)}{\|S_q\|\|S_A\|}\right) > \frac{r - C_2 D_{pm}(q, A)}{C_1}$$

Proof:

The fact that $SPC_t(q, A) \le GSPC_t(q, A)$ is to be used. □

Figure 3 shows the steps in answering a range query R(q, r). Note that in step 1, data is sorted by precursor mass so that fast search methods can be applied. Any 1-dimensional index structure

supporting range queries can be applied here. For example, binary search tree for in-memory case and B-tree for large data sets. In step 2, the inverted index is organized as an array of lists, so that the head of each list can be reached in constant time. A hash map can be used to quickly count the gross shared peak count.

Besides query running time, there are some statistics serving as indicators of query efficiency. We show empirical results of them in Section 4.

(1) Precision

At the beginning of step 4, the candidate set contains false positives. Since it is costly to compute the fuzzy cosine distance, we want to minimize the number of positives. The precision is defined as:

$$\text{Precision} = \frac{\text{Number of results found in step 4}}{\text{Candidate set size before step 4}}$$

Larger values of the precision indicate higher query efficiency.

Let pruning efficiency be the percentage of data pruned by a prune rule. Then the pruning efficiency for the two theorems can be defined. Larger values of the pruning efficiency indicate better query performance.

(2) Pruning efficiency of Theorem 1:

$$PE_1 = 1 - \frac{\text{Candidate set size after step 1}}{\text{Database size}}$$

(3) Pruning efficiency of Theorem 2:

$$PE_2 = 1 - \frac{\text{Candidate set size before step 4}}{\text{Candidate set size after step 1}}$$

3.3 Cost analysis

Let N be the dimension of the binary peak vectors, M be the database size, and p be the probability that an entry of a binary peak vector is 1. Random distribution is also assumed.

The expected number of non-zero entries of each binary peak vector, or the expected length of each compressed peak vector, is Np.

The expected size of each list of the inverted index is Mp, and expected total number of ids stored in the inverted index is MNp.

For a given query q and t = 0, the expected number of points with positive gross share peak count to q is given by:

$$M(1-(1-p)^{NP})$$

For tiny values of p, it roughly equals to MNp^2.

(1) Bulkload time

The step to sort the data by precursor mass takes O(MlgM) time. Assuming data input is in compressed vector format, to create the inverted index, one just need to traverse all the non-zero entries. Thus the time is O(MNp). Overall, the bulkload time is O(MlgM).

(2) Space

The space for precursor mass if O(M) and for the inverted index is O(MNp).

Assume an integer takes 4 bytes and a float value takes 4 bytes. The precursor mass array takes 4M bytes. The inverted index takes 4MNp bytes. Therefore, for reasonable data parameters, the index is likely to fit in main memory.

(3) Query running time

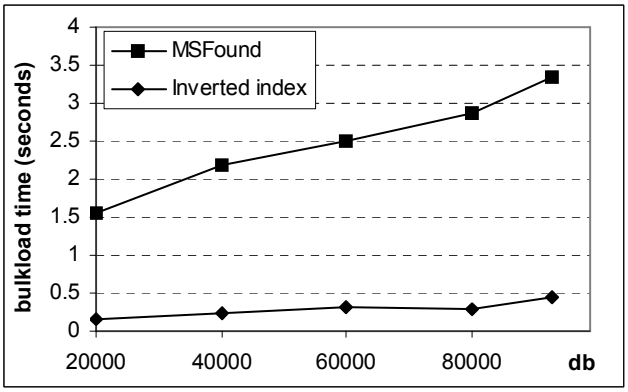
In step 1, determining the range of candidate and result in the sorted data array takes O(lgM) time. Putting candidates into candidate set and computing their precursor mass distances takes O(#candidate) time.

In step 2 for t = 0, the expected number (allow duplicates) of ids in L(q) is Mp*Np = MNp^2. So the total time is $O(MNp^2)$

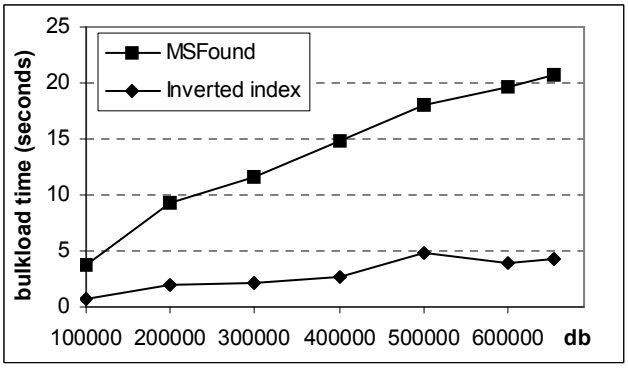
Step 3 takes O(#candidate) time.

In step 4, the shared peak count algorithm (Figure 1) running time is proportional to the length of compressed vectors. Therefore, it is O(NP), and the total time of step 4 is O(NP#computed)

Overall, since #candidate < M, assuming #computed is small (can be empirically verified) and ignoring the I/O cost, the over all cost is $O(MNp^2)$ for t = 0. For sparse vectors, the effect of N and M may be greatly reduced.



(a) Bulkload time of Dataset I: Ecoli



(b) Bulkload time of Dataset II: Human+Ecoli

Figure 4. Bulkload time of two datasets shows that inverted index has up to 10/5X speedups over MSFound

(4) I/O cost

The precursor mass array should be put into continuous disk blocks. The inverted index should also be put into continuous disk blocks while each list takes constant number of blocks to support fast access.

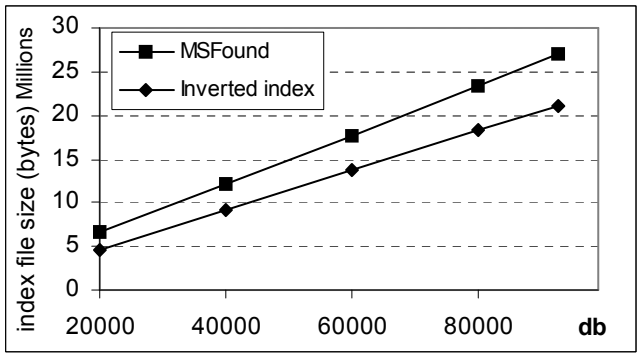
4. EMPIRICAL RESULTS

Two datasets are involved in the empirical study. Dataset I, with size 92768, contains MS/MS spectra from protein sequences of a seven-protein mixture from the Sashimi proteomics repository, concatenated with a control database of spectra from the Escherichia coli K12 (E. coli) genome. Dataset II, with size 654276, combines Database I with a larger control database of theoretical mass spectra from the human genome. A query set of 49 query objects is pre-determined. The test databases and query set in their original format are available from the Open Proteomics Database [13] and the Sashimi mass spectra repository [16].

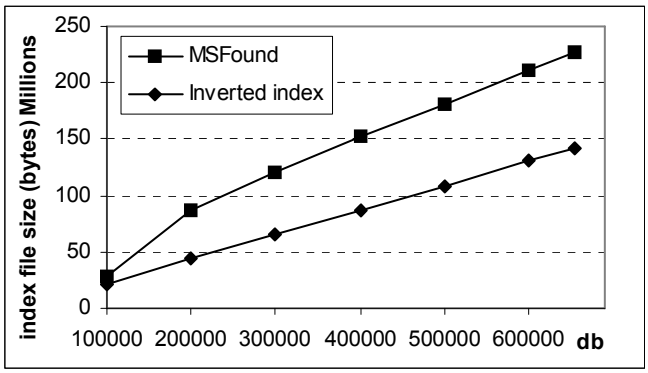
For both datasets, the mass range is [100, 5000] Da and resolution of 0.2 Da is used. So the dimension of the peak vector, N, is about 25000. It turned out that the sparsity of the binary vectors, p, is about 0.1%.

For each dataset, indexes of various sizes, on randomly select data, with both inverted index and general metric space method [14], are built and the bulkload time is collected.

The index file size on disk is also collected as an estimate of space consumption.



(a) Index file sizes of Dataset I: Ecoli



(b) Index file sizes of Dataset II: Human+Ecoli

Figure 5. Index file sizes of two datasets shows that inverted index takes up to 40% less space than MSFound

For each index, range queries of the pre-determined query set with various radii are executed and running time is collected. The running time of brute force linear scan to answer range queries is also collected as the comparison bottom line.

The values of constants are: $C_1 = C_2 = 1$, $\tau_{pm} = 2$, $t = 1$.

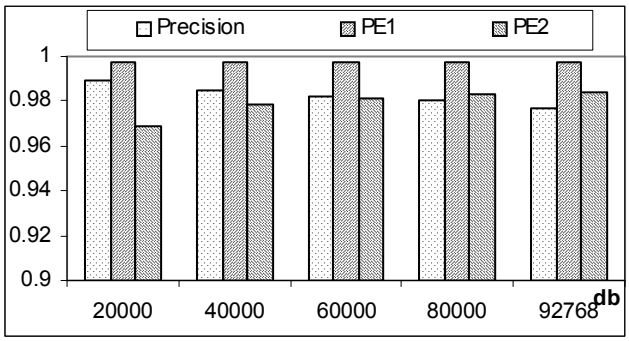
All the algorithms were implemented in JAVA on top of Sun JVM 1.5. All the experiments were carried out on an Apply Mac Pro running Mac OS X Server 10.4 with 8G memory and 2 dual core Intel Xeon CPU at 3 GHZ. The code and data in binary format are available at the MoBIoS repository [9].

4.1 Bulkload time

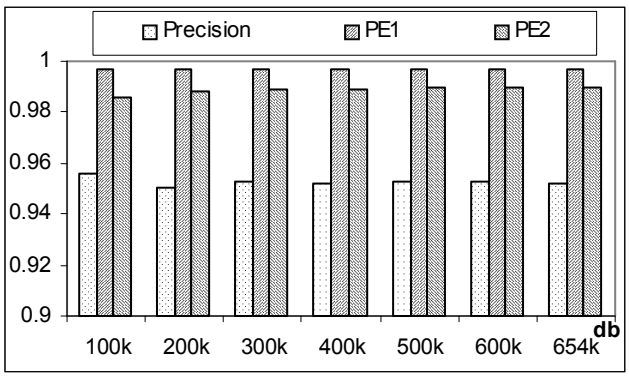
Figure 4 shows the bulkload time of inverted index method and MSFound on two datasets of various sizes. Comparing with MSFound, inverted index has up to 10X speedups on Dataset I and up to 5X speedups on Dataset II. Further, the figure indicates that both methods scales well with respect to bulkload time, consistent with the analysis in Section 3.3.

4.2 Size of index

The space consumption can be estimated by the size of the index file. Sizes of index files of both datasets with various database sizes are illustrated in Figure 5. A linear trend is clearly demonstrated between the index file size and database size, which is consistent with the analysis in Section 3.3. Further, inverted index takes up to 40% less space than MSFound.



(a) Query statistics of Dataset I: Ecoli



(b) Query statistics of Dataset II: Human+Ecoli

Figure 6. Range query statistics show that inverted index has high pruning/searching efficiency

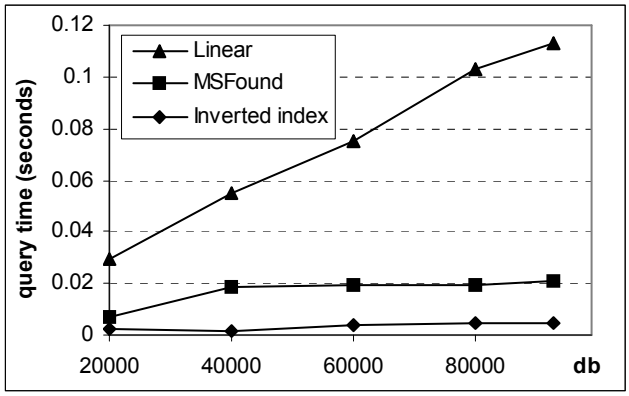
4.3 Range query time

Range queries with various radii are executed on indexes built above, and the query results are averaged on queries. In the following, the results from radius 1.45 are shown, which returns all the biologically correct query results [14].

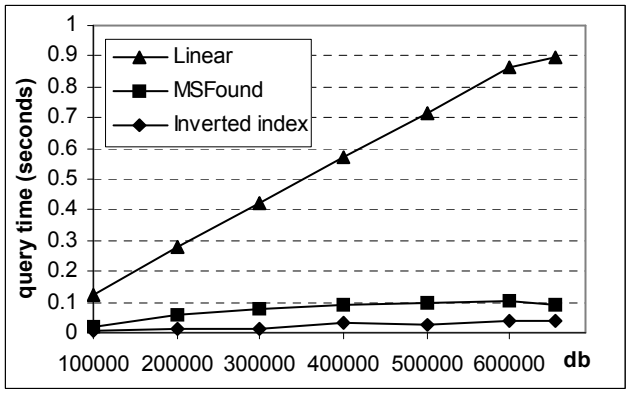
To show the query performance of inverted index, let's first take a look of the empirical results of those query statistics discussed in Section 3.2.

From Figure 6, we can see that the pruning efficiency of Theorem 1, PE_1, is about 99.7% for all the cases. That is, 99.7% of the database is pruned by Theorem 1. The pruning efficiency of Theorem 2, PE_2, is about 98% for Dataset I and about 99% for Dataset II. Therefore, the combined pruning efficiency, $1-(1-PE_1)(1-PE_2)$, of theorems 1 and 2 is as low as 99.994% for Dataset I and 99.997% for Dataset II.

From Figure 6, we can also see the precision is as high as 95% for Dataset I and 98% for Dataset II. That is, a majority of the data not pruned by theorems 1 and 2 is actually true positive. Most of the costly distance computations are performed between the query objects and the actually query results.



(a) Range query running time of Dataset I: Ecoli



(b) Range query running time of Dataset II: Human+Ecoli

Figure 7. Range query running time shows that inverted index outperforms MSFound

Next, let's take a look of the query running time. Figure 7 shows that both inverted index and MSFound are much better than linear scan. Actually, the speedup of inverted index over linear scan is about 20 for both datasets. For MSFound, the speedups over linear scan are about 5 for Dataset I and 9 for Dataset II. Further, the speedups of inverted index over MSFound are about 4 for Dataset I and about 2 for Dataset II, indicating that there are a lot for general metric space methods to catch up with domain specific methods, at least for the case studied here.

Last but not least, the speedups of MSFound over linear scan increases as database size increases for Dataset II. For both datasets, the speedups of inverted index over MSFound decreases as database size increases, indicating better scalability of MSFound.

5. CONCLUSIONS AND FUTURE WORK

The great general applicability of metric space indexing methods does not come for free. It is of interest to investigate the performance of general metric space methods on particular domains, comparing with domain specific method.

Taking use of the high sparsity of binary format mass spectra, we propose a domain specific indexing method, inverted index, for mass spectra to support similarity queries. Empirical results and statistics show that this method has scalable bulkload time and index file size, and great query efficiency.

The inverted index is compared with an existing metric space method, MSFound, and brute force linear scan. Empirical results show that both inverted index and MSFound are much better than linear scan. The inverted index possesses the fastest query time while MSFound has better scalability.

Query performance is one of the most important motivations of the application of metric space indexing method. It is not surprising to see that a domain specific method outperforms a metric space method, as long as the margin is not huge. Our goal is to reveal the cost to be general, and how much metric space methods should improve. Although results show that there is still large space for metric space methods to improve, their better scalability is yet very promising. This paper is just a case study of a particular domain specific method and a metric space method for a particular data type. The test data sets are tiny and we look forward to testing on larger data sets. Further, we expect more work along this direction on more domains, e.g. comparison with Locality Sensitive Hashing [3].

6. ACKNOWLEDGMENTS

This research is supported by NSF DBI-0640923, USA and the Initiating Fund for Member of the Chinese Academy of Sciences from Shenzhen University, China.

7. REFERENCES

[1] Bozkaya, T. and M. Ozsoyoglu, *Indexing large metric spaces for similarity search queries.* ACM Trans. Database Syst., 1999. **24**(3): p. 361-404.

[2] Chavez, E., G. Navarro, R. Baeza-Yates, and J. Marroqu, *Searching in metric spaces.* ACM Computing Surveys (CSUR), 2001. **33**(3): p. 273-321.

[3] D. Dutta and T. Chen. Speeding up tandem mass spectrometry database search: metric embeddings and fast near neighbor search. Bioinformatics, 23(5):612–618, 2007.

[4] Ari M. Frank, Nuno Bandeira, Zhouxin Shen, Stephen Tanner, Steven P. Briggs, Richard D. Smith, and Pavel A. Pevzner. Clustering Millions of Tandem Mass Spectra. J. Proteome Res. 2008 January; 7(1): 113–122.

[5] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In The VLDB Journal, pages 518–529, 1999.

[6] Hjaltason, G.R. and H. Samet, *Index-driven similarity search in metric spaces.* ACM Transactions on Database Systems (TODS), 2003. **28**(4): p. 517-580.

[7] D. Hoksza and T. Skopal. Index-based approach to similarity search in protein and nucleotide databases. CEUR Proc. Dateso 2007, vol. 235, pp. 67-80. 2007.

[8] Mir**a**nker, D.P., Xu W. **a**nd M**a**o, R. Mobios**:** a metric-space dbms to support biological discovery**.** Proceedings of the Intern**a**tion**a**l Conference on Scientific **a**nd St**a**tistic**a**l D**ata**b**a**se M**a**n**a**gement System, pp. 241-244, 2003.

[9] The MoBIoS repository: http://aug.csres.utexas.edu/sisap2010_ms

[10] J. Novák, D. Hoksza. Parametrised Hausdorff Distance as a Non-Metric Similarity Model for Tandem Mass Spectrometry. In the Proceedings of the Dateso 2010 Annual International Workshop on DAtabases, TExts, Specifications and Objects. Stedronin-Plazy, Czech Republic, April 21, 2010.

[11] Perkins,D. et al. Probability-based protein identification by searching sequence databases using mass spectrometry data. Electrophoresis, 20, 3551–3567, 1999.

[12] Pevzner,P. et al. Efficiency of database search for identification of mutated and modified proteins via mass spectrometry. Genome Res., 11, 290–299, 2001.

[13] J. Prince, M. Carlson, R. Wang, P. Lu, and E. Marcotte. The need for a public proteomics repository. Nature Biotechnology, 22(4):471–472, 2004.

[14] Ramakrishnan, S. R., Mao, R., Nakorchevskiy, A. A., Prince, J. T., Willard, W. S., Xu, W., Marcotte, E. M., and Miranker, D. P. 2006. A fast coarse filtering method for peptide identification by mass spectrometry. *Bioinformatics* 22, 12 (Jun. 2006), 1524-1531.

[15] Samet, H., *Foundations of Multidimensional and Metric Data Structures*. 2006, Morgan-Kaufmann.

[16] The Sashimi mass spectra repository: http://sashimi.sourceforge.net.

[17] G. Shakhnarovich, T. Darrell, and P. Indyk, editors. Nearest-Neighbor Methods g and Vision: Theory and Practice (Neural Information Processing).The MIT Press, March 2006.

[18] Yates III,J. et al. Method to correlate tandem mass spectral data of modified peptides to amino acid sequences in the protein database. Anal. Chem., 67, 1426–1436, 1995.

[19] P. Zezula, G. Amato, V. Dohnal and M. Batko. Similarity Search: The Metric Space Approach (Advances in Database Systems). Springer, New York, USA. 2006.

[20] Zhang,W. and Chait,B. ProFound—an expert system for protein identification using mass spectrometric peptide mapping information. Anal. Chem., 72, 2482–2489, 2000.

kNN based image classification relying on local feature similarity

Giuseppe Amato
ISTI-CNR
via G. Moruzzi, 1
Pisa, Italy
giuseppe.amato@isti.cnr.it

Fabrizio Falchi
ISTI-CNR
via G. Moruzzi, 1
Pisa, Italy
fabrizio.falchi@isti.cnr.it

ABSTRACT

In this paper, we propose a novel image classification approach, derived from the kNN classification strategy, that is particularly suited to be used when classifying images described by local features. Our proposal relies on the possibility of performing similarity search between image local features.

With the use of local features generated over interest points, we revised the single label kNN classification approach to consider similarity between local features of the images in the training set rather than similarity between images, opening up new opportunities to investigate more efficient and effective strategies. We will see that classifying at the level of local features we can exploit global information contained in the training set, which cannot be used when classifying only at the level of entire images, as for instance the effect of local feature cleaning strategies.

We perform several experiments by testing the proposed approach with different types of image local features in a touristic landmarks recognition task.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing; H.3.1 [**Information Storage and Retrieval**]: Information Search and Retrieval

Keywords

Image indexing, image classification, recognition, landmarks, local features

1. INTRODUCTION

A promising approach toward image content recognition is the use of classification techniques to associate images with classes (labels) according to their content. For instance, if an image contains a car, it might be automatically associated with the class *car* (labeled with the label *car*).

SISAP '10 , September 18-19, 2010, Istanbul, Turkey.

Traditional kNN classification algorithms decide about the class of an image by searching for the k images of the training set most similar to the image to be classified, and by performing a class weighted frequency analysis. The k closest images are identified relying upon a similarity measure between images.

As an alternative approach, in this paper we propose a new kNN based classification method relying on images represented by means of local features generated over interest points, as for instance SIFT [12] or SURF [6]. With the use of local features and interest points, kNN classification algorithms were revised to consider similarity between local features of the images in the training set rather than similarity between images, opening up new opportunities to investigate more efficient and effective strategies. In fact, direct use of similarity between local features is generally easier to be handled than sets of local features. In addition, we will see that classifying at the level of local features we can exploit global information contained in the training set, which cannot be used when classifying only at the level of entire images.

In this paper, we also study the effect of local feature cleaning strategies and we perform several experiments by testing the proposed approach with different types of image local features in a touristic landmarks recognition task.

The paper is organized as follows. In Section 4 we present an image similarity measures relying on local features to be used with a kNN classification algorithm. Section 5 proposes our new classification approach and defines four local feature classifiers. Section 6 discusses the use of the proposed approach for training local features cleaning. Finally, Sections 7 and 8 presents the experiments that we carried out to assess the proposed technique.

2. RELATED WORK

The first approach to recognizing location from mobile devices using image-based web search was presented in [16]. Two image matching metrics were used: energy spectrum and wavelet decompositions. Local features were not tested.

In the last few years the problem of recognizing landmarks have received growing attention by the research community. In [14] methods for placing photos uploaded to Flickr on the Wolrd map was presented. In the proposed approach the images were represented by vectors of features of the tags, and visual keywords derived from a vector quantization of the SIFT descriptors.

In [11] a combination of context- and content-based tools were used to generate representative sets of images for lo-

cation driven features and landmarks. SIFT features were used to establish links between different images which contain views of a single location. However, this information is combined with the textual metadata while we are only considering content-based classification.

In [18], Google presented its approach to building a web-scale landmark recognition engine. Most of the work reported was used to implement the Google Goggles service [1]. The approach makes use of the SIFT feature. The recognition is based on best matching image searching, while our novel approach is based on local features classification.

An important survey of local features detectors is [15]. However, the various local features are not compared. In this paper we decided to use for each local feature the detector proposed by the authors of each feature.

In [8] a survey on mobile landmark recognition for information retrieval is given. Classification methods reported as previously presented in the literature include SVM, Adaboost, Bayesian model, HMM, GMM. The kNN based approach which is the main focus of this paper is not reported in that survey.

In [10], various MPEG-7 descriptors have been used to build kNN classifier committees and test were performed on a slabs of stones dataset. In [7] the effectiveness of NN image classifiers has been proved and an innovative approach based on Image-to-Class distance that is similar in spirit to our approach has been proposed.

3. LOCAL FEATURES

The approach described in this paper focuses on the use of image local features. Specifically, we performed our tests using the SIFT [12] and SURF [6] local features. In this section, we briefly describe both of them.

3.1 SIFT

The Scale Invariant Feature Transformation (SIFT) [12] is a representation of the low level image content that is based on a transformation of the image data into scale-invariant coordinates relative to local features. Local feature are low level descriptions of keypoints in an image. Keypoints are interest points in an image that are invariant to scale and orientation. Keypoints are selected by choosing the most stable points from a set of candidate location. Each keypoint in an image is associated with one or more orientations, based on local image gradients. Image matching is performed by comparing the description of the keypoints in images. For both detecting keypoints and extracting the SIFT features we used the public available software developed by David Lowe [3].

3.2 SURF

The basic idea of Speeded Up Robust Features (SURF) [6] is quite similar to SIFT. SURF detects some keypoints in an image and describes these keypoints using orientation information. However, the SURF definition uses a new method for both detection of keypoints and their description that is much faster still guaranteeing a performance comparable or even better than SIFT. Specifically, keypoint detection relies on a technique based on a approximation of the Hessian Matrix. The descriptor of a keypoint is built considering the distortion of Haar-wavelet responses around the keypoint itself. For both detecting interest points and extracting the SURF features, we used the public available noncommercial software developed by the authors [4].

4. THE BASELINE

In this section we discuss how traditional kNN classification algorithms can be applied to the task of classifying images described by local features, as for instance SIFT or SURF. This will be later on compared to the new classification strategy that we propose in Section 5.

4.1 Single-label Distance-Weighted kNN

Given a set of documents D and a predefined set of *classes* (also known as *labels*, or *categories*) $C = \{c_1, \ldots, c_m\}$, *single-label document classification* (SLC) [9] is the task of automatically approximating, or estimating, an unknown *target function* $\Phi : D \rightarrow C$, that describes how documents ought to be classified, by means of a function $\hat{\Phi} : D \rightarrow C$, called the *classifier*, such that $\hat{\Phi}$ is an approximation of Φ.

A popular SLC classification technique is the *Single-label distance-weighted kNN*. Given a training set Tr containing various examples for each class c_i, it assigns a label to a document in two steps. Given a document d_x (an image for example) to be classified, it first executes a kNN search between the objects of the *training set*. The result of such operation is a list $\chi^k(d_x)$ of labeled documents d_i belonging to the *training set* ordered with respect to decreasing values of the similarity $s(d_x, d_i)$ between d_x and d_i. The label assigned to the document d_x by the classifier is the class $c_j \in C$ that maximizes the sum of the similarity between d_x and the documents d_i, labeled c_j, in the kNN results list $\chi^k(d_x)$

Therefore, first a score $z(d_x, c_i)$ for each label is computed for any label $c_i \in C$:

$$z(d_x, c_j) = \sum_{d_i \in \chi^k(d_x) \ : \ \Phi(d_i) = c_j} s(d_x, d_i) \ .$$

Then, the class that obtains the maximum score is chosen:

$$\hat{\Phi}^s(d_x) = \arg\max_{c_j \in C} z(d_x, c_j) \ .$$

It is also convenient to express a degree of confidence on the answer of the classifier. For the *Single-label distance-weighted kNN* classifier described here we defined the confidence as 1 minus the ratio between the *score* obtained by the second-best label and the best label, i.e,

$$\nu_{doc}(\hat{\Phi}^s, d_x) = 1 - \frac{\arg\max_{c_j \in C - \hat{\Phi}^s(d_x)} z(d_x, c_j)}{\arg\max_{c_j \in C} z(d_x, c_j)} \ .$$

This classification confidence can be used to decide whether or not the predicted label has an high probability to be correct.

4.2 Image similarity

In order the kNN search step to be executed, a similarity function between images should be defined. Global features, generally, are defined along with a similarity (or a distance) function. Therefore, similarity between images, is computed as the similarity between the corresponding global features. On the other hand, a single image has several local features. Therefore, computing the similarity between two images requires combining somehow the similarities between their numerous local features.

In the following we define a function for computing similarity between images on the basis of their local features that is derived from the work presented in [12]. In the experiments, at the end of this paper, we will compare the performance of the similarity function, when used with the *single-label distance-weighted kNN* classification technique, against the local feature based classification algorithm proposed in Section 5.

4.2.1 Local Feature Similarity

The computer vision literature related to local features, uses generally the notion of distance, rather than that of similarity. However in most cases a similarity function $s()$ can be easily derived from a distance function $d()$. For both SIFT and SURF the Euclidean distance is typically used as measure of dissimilarity between two features [12, 6]. Let $d(p_1, p_2) \in [0, 1]$ be the normalized distance between two local features p_1 and p_2. We define the similarity as:

$$s(p_1, p_2) = 1 - d(p_1, p_2)$$

Obviously $0 \leq s(p_1, p_2) \leq 1$ for any p_1 and p_2.

4.2.2 Local Features Matching

A useful aspect that is often used when dealing with local features is the concept of local feature matching. In [12], a distance ratio matching scheme was proposed that has also been adopted by [6] and many others. Let's consider a local feature p_x belonging to an image d_x (i.e. $p_x \in d_x$) and an image d_y. First, the point $p_y \in d_y$ closest to p_x (in the remainder $NN_1(p_x, d_y)$) is selected as candidate match. Then, the distance ratio $\sigma(p_x, d_y) \in [0, 1]$ of closest to second-closest neighbors of p_x in d_y is considered. The distance ratio is defined as:

$$\sigma(p_x, d_y) = \frac{d(p_x, NN_2(p_x, d_y))}{d(p_x, NN_1(p_x, d_y))}$$

Finally, p_x and $NN_1(p_x, d_y)$ are considered matching if the distance ratio $\sigma(p_x, d_y)$ is smaller than a given threshold. Thus, a function of matching between $p_x \in d_x$ and an image d_y is defined as:

$$m(p_x, d_y) = \begin{cases} 1 & \text{if } \sigma(p_x, d_y) < c \\ 0 & \text{otherwise} \end{cases}$$

In [12], Lowe proposed to use $c = 0.8$ reporting that this threshold allows to eliminate 90% of the false matches while discarding less than 5% of the correct matches. In Section 8 we report an experimental evaluation of classification effectiveness varying c that confirms the results obtained by Lowe. Please note, that this parameter will be used in defining the image similarity measure used as a baseline and in one of our proposed local feature based classifiers. However, the best performing classifier defined in Section 5.1.4 does not require this parameter because it is based on fuzzy matching.

4.2.3 Image Similarity Measure

A reasonable measure of similarity between two image d_x and d_y is the percentage of local features in d_x that have a match in d_y. We define the *Percentage of Matches* similarity function s^m as follows:

$$s^m(d_x, d_y) = \frac{1}{|d_x|} \sum_{p_x \in d_x} m(p_x, d_y)$$

where $m(p_x, d_y)$ is 1 if p_x has a match in d_y and 0 otherwise as defined in Section 4.2.2. For simplicity, we indicate the number of local features in an image d_x as $|d_x|$.

It is worth to say that this image similarity measure is not metric. In fact, it is not even symmetric. Thus, it could not be efficiently indexed by the various index structure defined for metric spaces (see [17]).

5. LOCAL FEATURE BASED IMAGE CLASSIFICATION

In the previous section, we considered the classification of an image d_x as a process of retrieving the most similar ones in the *training set* Tr and then applying a kNN classification technique in order to predict the class of d_x.

In this section, we propose a new approach that first assigns a label to each local feature of an image. The label of the image is then assigned by analyzing the labels and confidences of its local features.

This approach has the advantage that any access method for similarity search in metric spaces (see [17]) can be used to speed-up classification.

The proposed *Local Feature Based Classifiers* classify an image d_x in two steps:

1. first each local feature p_x belonging to d_x is classified considering the local features of images in Tr;
2. second the whole image is classified considering the class assigned to each local feature and the confidence of the classification.

Note that classifying individually the local features, before assigning the label to an image, we might loose the implicit dependency between interest points of an image. However, surprisingly, we will see that this method offers better effectiveness than the baseline approach. In other words we are able to improve at the same time both efficiency and effectiveness.

In Section 5.1 we define four distinct algorithms for local feature classification, i.e. step 1. All the proposed algorithms require searching for similar local features for each of the local features belonging to the image we have to classify. For the second step, we use the confidence-rated majority vote approach reported in Section 5.1.2.

In the following, we assume that the label of each local feature p_x, belonging to images in the training set Tr, is the label assigned to the image it belongs to (i.e., d_x). Following the notation used in Section 4,

$$\forall p_x \in d_x,\ \forall d_x \in Tr\ , \Phi(p_x) = \Phi(d_x).$$

In other words, we assume that the local features generated over interest points of images in the training set can be labeled as the image they belong to. Note that the noise introduced by this label propagation from the whole image to the local features can be managed by the local features classifier. In fact, we will see that when very similar training local features are assigned to different classes, a local feature close to them is classified with a low confidence. The experimental results reported in Section 8 confirm the validity of this assumption.

5.1 Local Features Classification

In the following we propose four different strategies for obtaining local feature (LF) classifiers and the corresponding confidence value.

As we said before, given $p_x \in d_x$, a classifier $\hat{\Phi}$ returns both a class $\hat{\Phi}(p_x) = c_i \in C$ to which it believes p_x to belong *and* a numerical value $\nu(\hat{\Phi}, p_x)$ that represents the confidence that $\hat{\Phi}$ has in its decision. High values of ν correspond to high confidence.

5.1.1 1-NN LF Classifier – $\hat{\Phi}^f(p_x)$

The simplest way to assign a label to a local feature is to consider the label of its closest neighbor in Tr. The *1-NN Local Features Classifier* assigns to a local feature p_x, the label of the closest neighbor in Tr. The confidence of the classification assigned is the similarity between p_x and is nearest neighbor. Formally:

$$\begin{cases} \hat{\Phi}^f(p_x) = \Phi(NN_1(p_x, Tr)). \\ \nu(\hat{\Phi}^f, p_x) = 1 - d(p_x, NN_1(p_x, Tr)) \end{cases}$$

Please note that this classifier does not require any parameter to be set. Moreover, the similarity search to perform over the local features training set is a simple 1-NN.

5.1.2 Weighted kNN LF Classifier – $\hat{\Phi}^k(p_x)$

This *Weighted kNN LF Classifier* is an extension of the *single-label distance-weighted kNN* reported in Section 4. First a kNN search on the local features of the Tr is executed. The result of such operation is a list of labeled features p_i belonging to Tr ordered with respect to decreasing values of the similarity $s(p_x, p_i)$. The label $\hat{\Phi}^k(p_x)$ assigned to the document d_x by the classifier is the class $c_j \in C$ that maximizes the sum of the similarity between p_x and the features p_i, labeled c_j, in the kNN results list $\chi^k(p_x)$. The confidence is then based on the ratio between second best and best class. Formally, we have to compute a score $z^k(p_x, c_j)$ for each class:

$$z^k(p_x, c_j) = \sum_{p_i \in \chi^k(p_x) \ : \ \Phi(p_i) = c_j} s(p_x, p_i)$$

Then the predicted label $\hat{\Phi}^k$ and the confidence ν are defined as follows:

$$\begin{cases} \hat{\Phi}^k(p_x) = \arg\max_{c_j \in C} z^k(p_x, c_j) \\ \nu(\hat{\Phi}^k, p_x) = 1 - \dfrac{\arg\max_{c_j \in C - \hat{\Phi}^k(p_x)} z^k(p_x, c_j)}{\arg\max_{c_i \in C} z^k(p_x, c_i)} \end{cases}$$

Note that for $k = 1$ we have $\hat{\Phi}^k(p_x) = \hat{\Phi}^f(p_x)$, while the measure of confidence is different. In fact, $\hat{\Phi}^k$ always assigns 1 as confidence when $k = 1$ is set while $\hat{\Phi}^f$ considers the first nearest neighbor similarity as measure of confidence. We will see in Section 5.2 that this difference is important for the whole image classification

This classifier requires the parameter k to be chosen.

5.1.3 LF Matching Classifier – $\hat{\Phi}^m(p_x)$

The *Local Feature Matching Classifier* decides the candidate label similarly to the *1-NN Local Features Classifier*, i.e.:

$$\hat{\Phi}^m(p_x) = \Phi(NN_1(p_x, Tr))$$

The very difference is the computation of the confidence value of the selected label which is evaluated using the idea of the distance ratio discussed in Section 4.2.2.

The confidence here plays the role of a matching function, where the idea of the distance ratio is used to decide if the candidate label is a good match:

$$\nu(\hat{\Phi}^m, p_x) = \begin{cases} 1 & \text{if } \dot{\sigma}(p_x, t_r) < c \\ 0 & \text{otherwise} \end{cases}$$

The distance ratio $\dot{\sigma}$ is computed considering the nearest local feature to p_x and the closest local feature that has a label different than the nearest local feature. This idea follows the suggestion given by Lowe in [12], that whenever there are multiple training images of the same object, then the second-closest neighbor to consider for the distance ratio evaluation should be the closest neighbor that is known to come from a different object than the first. Following this intuition, we define the similarity ratio $\dot{\sigma}$ as:

$$\dot{\sigma}(p_x, T_r) = \frac{d(p_x, NN_2^*(p_x, Tr))}{d(p_x, NN_1(p_x, Tr))}$$

where $NN_2^*(p_x, Tr)$ is the closest neighbor that is known to be labeled differently than the first as suggested in [12].

Note that searching for $NN_2^*(p_x, Tr)$ can not be directly translated in a standard k nearest neighbors search. However, the kNN implementation in metric spaces is generally performed starting with an infinite range and reducing it during the evaluation, considering at any time the actual NN_k. The very same approach can be used for searching $NN_2^*(p_x, Tr)$. In fact, while k is not known in advance, the actual NN_2^* during the similarity search, can be used to reduce the range of the query. Thus, the similarity search needed for the evaluation of $\dot{\sigma}(p_x, T_r)$ can be implemented slightly modifying the standard algorithms developed for metric spaces (see [17]).

The parameter c used in the definition of the confidence is the equivalent of the one used in [12] and [6]. We will see in Section 8 that $c = 0.8$ proposed in [12] by Lowe is able to guarantee good effectiveness. It is worth to note that c is the only parameter to be set for this classifier considering that the similarity search performed over the local features in Tr does not require a parameter k to be set.

5.1.4 Weighted LF Distance Ratio Classifier – $\hat{\Phi}^w(p_x)$

The *Weighted LF Distance Ratio Classifier* is an extension of the *LF Matching Classifier* defined in the previous section. However, the confidence is not binary but is a fuzzy measure derived from the distance ratio. Given that the greater the confidence the better the matching, we define the assigned label and the confidence as:.

$$\begin{cases} \hat{\Phi}^w(p_x) = \Phi(NN_1(p_x, Tr)) \\ \nu(\hat{\Phi}^w, p_x) = (1 - \dot{\sigma}(p_x, t_r))^2 \end{cases}$$

The intuition is that it could be better not to filter non-matching features on the basis of the distance ratio, but to use $1 - \dot{\sigma}(p_x, t_r)$ as a measure of confidence to be used during the classification of the whole image. Then, the value is squared to emphasize the relative importance of greater distance ratios.

Please note that for this classifier we do not have to specify neither a distance ratio threshold c or k. Thus, this classifier has no parameters at all.

5.2 Whole Image Classification

As we said before, the local feature based feature classification is composed of two steps (see Section 5). In previous section we have dealt with the issue of classifying the local feature of an image. Now, in this section, we discuss the second phase of the local feature based classification of images. In particular we consider the classification of the whole image given the label $\hat{\Phi}(p_x)$ and the confidence $\nu(\hat{\Phi}, p_x)$ assigned to its local features $p_x \in d_x$ during the first phase.

To this aim, we use a confidence-rated majority vote approach. We first compute a score $z(p_x, c_i)$ for each label $c_i \in C$. The score is the sum of the confidence obtained for the local features predicted as c_i. Formally,

$$z(d_x, c_i) = \sum_{p_x \in d_x, \hat{\Phi}(p_x)=c_i} \nu(\hat{\Phi}, p_x) \ .$$

Then, the label that obtains the maximum score is chosen:

$$\hat{\Phi}(d_x) = \arg\max_{c_j \in C} z(d_x, c_j) \ .$$

As measure of confidence for the classification of the whole image we use ratio between the predicted and the second best class:

$$\nu_{img}(\hat{\Phi}, d_x) = 1 - \frac{\arg\max_{c_j \in C - \hat{\Phi}(p_x)} z(d_x, c_j)}{\arg\max_{c_i \in C} z(d_x, c_i)} \ .$$

This whole image classification confidence can be used to decide whether or not the predicted label has an high probability to be correct. In the experimental results section 8 we will show that the proposed confidence is reasonable.

6. LOCAL FEATURES CLEANING

The number of local features obtained from an image is typically high. For instance, from a 500x500 pixels image we obtained an average of about 1,000 key points. Thus, it would be very important to reduce the number of local features in the training set, still maintaining an high effectiveness. This reduction allows better efficiency and reduce memory occupation. Moreover, deleting worst interest points could also results in better effectiveness.

The strategy that we used to decide which local features of Tr can be eliminated, was to assign a score to each local features of Tr considering Tr itself as training set with one of the local feature classifiers defined Section 5.1.4.

The discrepancy between pre-assigned label in the training set and the label predicted by the classifier together with the confidence expressed in predicting the label, was used as an indication for eliminating the local feature (remember that as reported in Section 5 the label pre-assigned to a local feature is the label assigned to the image it belongs to).

Given the good results reported in Section 8, we decided to test the *Weighted Ratio Based Matching*. We use the *Leave-One-Out* technique for classifying each point p_x in each image $d_x \in Tr$. During the classification we leave out the image d_x. Thus, the classification of the features in d_x is only based on the local features in $Tr - d_x$.

The discrepancy between the predicted label and the pre-assigned label is measured by using a score $e(p_x)$ computed as follows:

Figure 1: Example images taken from the dataset

$$e(p_x) = \begin{cases} \nu(\hat{\Phi}^w, d_x) & \text{if } \hat{\Phi}^w(p_x) = \Phi(p_x) \\ -\nu(\hat{\Phi}^w, d_x) & \text{otherwise} \end{cases}$$

The absolute value of the score of p_x is $\nu(\hat{\Phi}^w, d_x)$ that is the confidence of the predicted label. The sign is positive if the feature was correctly classified and negative otherwise. If the predicted label has an high confidence and it is wrong with respect to the pre-assigned label, then the score is very low and it is a good candidate to be eliminated.

In the experiment section we use a threshold on the score to control the amount of promising local features considered.

7. EVALUATION SETTINGS

For evaluating the various classifiers we need at least: a data set, an interest points detector, a local feature extractor, some performance measures. In the following, we present all the evaluation setting we used for the experimentation.

7.1 The Dataset

The dataset that we used for our tests is composed of 1,227 photos of landmarks located in Pisa and was used also in [5]. The photos have been crawled from Flickr, the well known on-line photo service. The dataset we built is publicly available. The IDs of the photos used for these experiments together with the assigned label and extracted features can be downloaded from [2]. In the following we list the classes that we used and the number of photos belonging to each class. In Figure 1 we reported an example for each class in the same order as they are reported in the list below:

- *Leaning Tower* (119 photos) – leaning campanile
- *Duomo* (130 photos) – the cathedral of St. Mary
- *Battistero* (104 photos) – the baptistery of St. John
- *Camposanto Monumentale (exterior)* (46 photos)
- *Camposanto Monumentale (field)* (113 photos)
- *Camposanto Monumentale (portico)* (138 photos)
- *Chiesa della Spina* (112 photos) – Gothic church
- *Palazzo della Carovana* (101 photos) – building
- *Palazzo dell'Orologio* (92 photos) – building
- *Guelph tower* (71 photos)

- *Basilica of San Piero* (48 photos) – church of St. Peter
- *Certosa* (53 photos) – the charterhouse

In order to build and evaluating a classifier for these classes, we divided the dataset in a *training set* (Tr) consisting of 226 photos (approximately 20% of the dataset) and a *test set* (Te) consisting of 921 (approximately 80% of the dataset). The image resolution used for feature extraction is the standard resolution used by Flickr i.e., maximum between width and height equal to 500 pixels.

The total number of local features extracted by the SIFT and SURF detectors were about 1,000,000 and 500,000 respectively.

7.2 Performance Measures

For evaluating the effectiveness of the classifiers in classifying the documents of the *test set* we use the micro-averaged *accuracy* and micro- and macro-averaged *precision*, *recall* and F_1.

Micro-averaged values are calculated by constructing a global contingency table and then calculating the measures using these sums. In contrast macro-averaged scores are calculated by first calculating each measure for each category and then taking the average of these. In most of the cases we reported the micro-averaged values for each measure.

Precision is defined as the ratio between correctly predicted and the overall predicted documents for a specific class. *Recall* is the ratio between correctly predicted and the overall actual documents for a specific class. F_1 is the harmonic mean of *precision* and *recall*.

Note that for the *single-label* classification task, micro-averaged *accuracy* is defined as the number of documents correctly classified divided by the total number of documents in the *test set* and it is equivalent to the micro-averaged *precision*, *recall* and F_1 scores.

8. EXPERIMENTAL RESULTS

In this section we report the experimental results obtained for both the image similarity based and local feature based classifiers. We also show that our measure of confidence can be used to improve effectiveness on classified images accepting a small percentage of not classified objects. Finally, we report preliminary results of our local feature cleaning approach.

8.1 The baseline

We first perform an evaluation of the baseline technique ($\hat{\Phi}^s$) based on image similarity. This evaluation will be used to assess the optimal parameters c and k for the baseline approach, to make a fair comparison against the proposed methods in next section.

The image similarity approach ($\hat{\Phi}^s$) uses the matching function defined in Section 4.2.2 that requires a threshold for the distance ratio c to be fixed in advance. In Figure 2 we report the performance obtained varying the matching threshold c. For each matching threshold c we report the best result obtained for k between 0 and 100. As mentioned in Section 4.2.1, in the paper where SIFT [12] were presented, Lowe suggested to use 0.8 as distance reatio threshold (c). The results confirm that the threshold proposed in [12] is the best for both SIFT and SURF and that the algorithm is stable around this values.

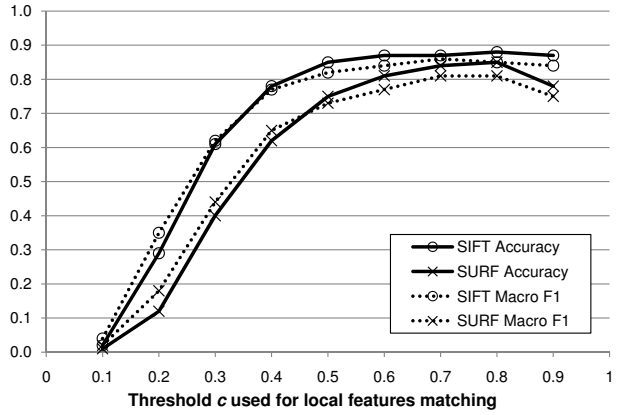


Figure 2: Accuracy and Macro F_1 obtained for various matching threshold by the similarity based approach.

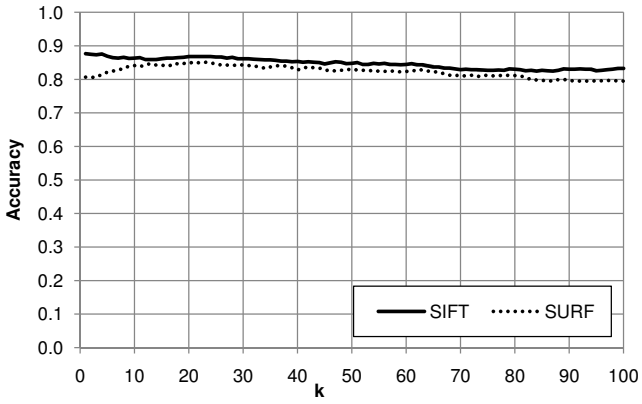


Figure 3: Accuracy obtained for various k using the image similarity based approach.

In Figure 3 we report the *accuracy* obtained for k between 1 and 100 by both SIFT and SURF for $c = 0.8$. The parameter k determines the number of closest neighbors retrieved in order to classify a given image using the *single-label distance-weighted kNN* technique (see Section 4).

The results show that SIFT performs generally better than SURF. Moreover, the k for which the best performance was obtained is typically much higher for SURF than SIFT. In other words, the test image closest neighbor in the training set is more relevant using SIFT than using SURF while the results obtained for higher k are almost the same. Specifically, the best result was obtained with $k = 1$ in case of SIFT and with $k = 20$ in case of SURF.

8.2 Local Feature Based Image Classification

In this section we compare the baseline approach against the proposed method for local feature based image classification. In this comparison we use the optimal settings of the parameters for the baseline approach, discussed in previous section. Specifically, we set c to 0.8, for both SIFT and SURF, while we use $k = 1$ for SIFT and $k = 20$ for SURF. We use $c = 0.8$ also for the *Local Feature Matching Classifier* ($\hat{\Phi}^m$).

									best	baseline
	classifier	$\hat{\Phi}^f$	$\hat{\Phi}^1$	$\hat{\Phi}^5$	$\hat{\Phi}^{10}$	$\hat{\Phi}^{25}$	$\hat{\Phi}^{50}$	$\hat{\Phi}^m$	$\hat{\Phi}^w$	$\hat{\Phi}^s$
Accuracy	**SIFT**	0.901	0.901	0.855	0.818	0.756	0.691	0.945	**0.952**	0.877
	SURF	0.883	0.881	0.841	0.794	0.714	0.668	0.927	**0.928**	0.851
F_1 Macro	**SIFT**	0.806	0.883	0.809	0.748	0.657	0.575	0.940	**0.947**	0.864
	SURF	0.791	0.866	0.804	0.727	0.606	0.542	0.915	**0.922**	0.828

Figure 4: Accuracy and Macro F_1 for the proposed local feature based classifiers and for the baseline Φ^s.

In Figure 5, we report *accuracy* and macro-averaged F_1 obtained by the various classifiers using both SIFT and SURF together with the results obtained by the image similarity based approach ($\hat{\Phi}^s$).

The first observation is that all the local feature based approaches perform significantly better then the baseline ($\hat{\Phi}^s$). In particular, both the *Local Features Matching* ($\hat{\Phi}^m$) and the *Weighted LF Distance Ratio* ($\hat{\Phi}^w$) classifiers outperform all the others. This is true using both SIFT and SURF features. The best overall performance were obtained by $\hat{\Phi}^w$ which is slightly better than its non-weighted counterpart ($\hat{\Phi}^m$). Moreover, it has another and probably even more important advantage – it does not require any parameter to be set.

The performance measures obtained by the *Weighted kNN Local Features Classifier* for various k ($\hat{\Phi}^1$, $\hat{\Phi}^5$, $\hat{\Phi}^{10}$, $\hat{\Phi}^{20}$ and $\hat{\Phi}^{50}$) show that the best results are obtained when considering only the closest neighbor (i.e., $k = 1$). Moreover, $\hat{\Phi}^k$ for $k = 1$ outperforms the *1-NN Local Features Classifier* ($\hat{\Phi}^f$) in terms of F_1 while the *accuracy* values are equivalent.

Even if in this paper we did not consider the computational cost of classification, we can make some simple observations. In fact, it is worth saying that the local feature based classifier are less critical from this point of view. First, because closest neighbors of local features in the test image are searched once for all in the Tr and not every time for each image of Tr. Second, because it is possible to leverage on global spatial index for all the features in Tr, to support efficient k nearest neighbors searching. In fact, the similarity function between two local features is the Euclidean distance, which is a metric. Thus, a metric data structure (see [17] and [13]) could be used to improve efficiency.

Regarding the local features used and the computational cost, we underline that the number of local features detected by the SIFT extractor is twice that detected by SURF. Thus, on one hand SIFT has better performance while on the other hand SURF is more efficient.

Let us now consider the confidence ν_{img} assigned to the predicted label of each image (see Section 5.2). This confidence can be used to obtain greater *accuracy* at the price of a certain number of false dismissals. In fact, a confidence threshold can be used to filter all the label assigned to an image with a confidence ν_{img} less than the threshold. In Figure 5 we report the *accuracy* obtained by the Φ^w classifier using SURF, varying the confidence threshold between 0 and 1. We also report the percentage of images in Te that were not classified together with the percentage of images that where actually correctly classified but that were filtered because of the threshold. Note that for $\nu_{img} = 0.5$ the *accuracy* of classified objects rise from 0.928 to 0.982

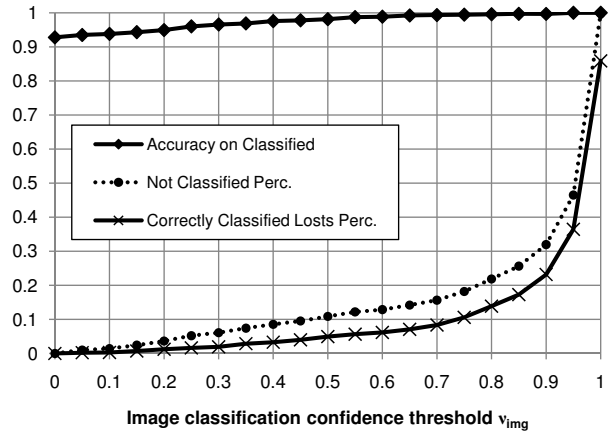


Figure 5: Accuracy on classified obtained by the $\hat{\Phi}^w$ classifier using SURF, for various image classification confidence thresholds.

obtained for $\nu_{img} = 0$. At the same time the percentage of correctly predicted images that are filtered (i.e., the classifier does not assign a label because of the low confidence threshold ν_{img}) is less than 5%.

This prove that the measure of confidence defined is meaningful. However, the best confidence threshold to be used depends on the task. Sometimes it could be better to try to *guess* the class of an image even if we are not sure, while in other cases it might be better to assign a label only if the classification has an high confidence.

8.3 Local Features Cleaning

We now consider the local features cleaning algorithm defined in Section 6. The goal of this process is the reduction of the total number of local features while maintaining good results. In Figure 6 we report the performance measures obtained by the $\hat{\Phi}^w$ classifier on using only local features from the *training set* that were evaluated higher than various local features evaluation thresholds e. Together with *accuracy* and macro-averaged F_1, we report the percentage of the local features in Tr that were used, i.e., obtained an evaluation of more than e. Note that, as defined in Section 6, the local features were evaluated only considering Tr while the performance is evaluated on Te.

Removing the local features with an evaluation threshold $e < 0$ we maintain just 40 percent of the total images in Tr, while both *accuracy* and macro F_1 only slightly decreases. Moreover, for $e < 0.2$ we maintain just 20 percent of the total features in Tr while the accuracy decreased of about 0.1. We consider this result to be very promising.

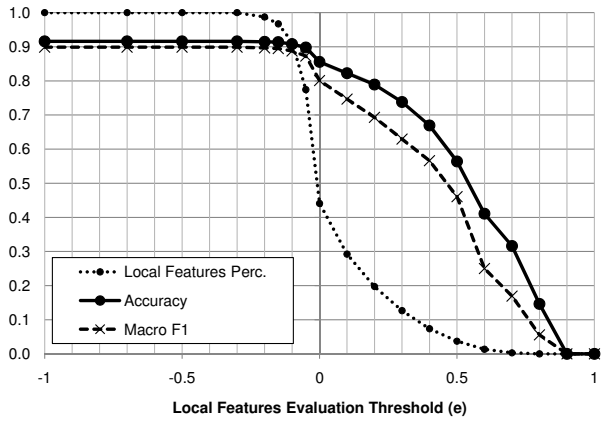


Figure 6: Accuracy and Macro F_1 obtained by $\hat{\Phi}^w$ using only SURF local features from the *training set* that were evaluated more than e.

9. CONCLUSIONS

In this paper, we defined a novel image classification approach, derived from the kNN classification strategy that classify images in two steps: first each local feature is classified considering the local features of a training set; second the whole image is classified considering the class assigned to each local feature and the confidence of these classifications. Four algorithms for the classification of local features were defined and tested.

The experimental results proved that this novel approach outperforms traditional approaches based on image similarity functions. Moreover, the confidence measure for the assigned label proved to be meaningful and useful in improving the accuracy of the classification.

It worths noting that the best results have been obtained by the Weighted LF Distance Ration Classifier ($\hat{\Phi}^w$). The nice feature of this classifier, in addition to be the best performing, is that it does not require any parameter to be set and tuned. Therefore, it is also the most easy and intuitive to be used, and the less prone to tuning errors.

Finally, we also defined a local features cleaning algorithm based on a features evaluation that can be used to reduce the number of local features in a training set at the price of slight efficacy degradation.

10. ACKNOWLEDGMENTS

This work was partially supported by the VISITO Tuscany project, funded by Regione Toscana, in the POR FESR 2007-2013 program, action line 1.1.d, and the MOTUS project, funded by the Industria 2015 program.

11. REFERENCES

[1] Google goggles. `http://www.google.com/mobile/goggles/`. last accessed on 30-March-2010.

[2] Pisa landmarks dataset. `http://www.fabriziofalchi.it/pisaDataset/`. last accessed on 30-March-2010.

[3] SIFT keypoint detector. `http://people.cs.ubc.ca/~lowe/`. last accessed on 30-March-2010.

[4] SURF detector. `http://www.vision.ee.ethz.ch/~surf/`. last accessed on 30-March-2010.

[5] G. Amato, F. Falchi, and P. Bolettieri. Recognizing landmarks using automated classification techniques: an evaluation of various visual features. In *in Proceeding of The Second Interantional Conference on Advances in Multimedia (MMEDIA 2010)*, pages 78–83. IEEE Computer Society, 2010.

[6] H. Bay, T. Tuytelaars, and L. V. Gool. Surf: Speeded up robust features. In *In ECCV*, pages 404–417, 2006.

[7] O. Boiman, E. Shechtman, and M. Irani. In defense of nearest-neighbor based image classification. In *CVPR*. IEEE Computer Society, 2008.

[8] T. Chen, K. Wu, K.-H. Yap, Z. Li, and F. S. Tsai. A survey on mobile landmark recognition for information retrieval. In *MDM '09*, pages 625–630. IEEE Computer Society, 2009.

[9] S. Dudani. The distance-weighted k-nearest-neighbour rule. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-6(4):325–327, 1975.

[10] T. Fagni, F. Falchi, and F. Sebastiani. Adaptive committees of feature-specific classifiers for image classification. In *Image Mining. Theory and Applications. Proceedings of the 2nd International Workshop on Image Mining Theory and Applications (IMTA-09)*, pages 113–122. INSTICC Press, 2009.

[11] L. S. Kennedy and M. Naaman. Generating diverse and representative image search results for landmarks. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 297–306, New York, NY, USA, 2008. ACM.

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

[13] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Computer Graphics and Geometric Modeling. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[14] P. Serdyukov, V. Murdock, and R. van Zwol. Placing flickr photos on a map. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 484–491, New York, NY, USA, 2009. ACM.

[15] T. Tuytelaars and K. Mikolajczyk. Local invariant feature detectors: a survey. *Found. Trends. Comput. Graph. Vis.*, 3(3):177–280, 2008.

[16] T. Yeh, K. Tollmar, and T. Darrell. Searching the web with mobile images for location recognition. In *CVPR (2)*, pages 76–81, 2004.

[17] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Springer-Verlag, 2006.

[18] Y. Zheng, M. Z. 0003, Y. Song, H. Adam, U. Buddemeier, A. Bissacco, F. Brucher, T.-S. Chua, and H. Neven. Tour the world: Building a web-scale landmark recognition engine. In *CVPR*, pages 1085–1092. IEEE, 2009.

Similarity Matrix Compression for Efficient Signature Quadratic Form Distance Computation

Christian Beecks, Merih Seran Uysal, Thomas Seidl
Data Management and Data Exploration Group
RWTH Aachen University, Germany
{beecks, uysal, seidl}@cs.rwth-aachen.de

ABSTRACT

Determining similarities among multimedia objects is a fundamental task in many content-based retrieval, analysis, mining, and exploration applications. Among state-of-the-art similarity measures, the Signature Quadratic Form Distance has shown good applicability and high quality in comparing flexible feature representations. In order to improve the efficiency of the Signature Quadratic Form Distance, we propose the similarity matrix compression approach which aims at compressing the distance's inherent similarity matrix. We theoretically show how to reduce the complexity of distance computations and benchmark computation time improvements. As a result, we improve the efficiency of a single distance computation by a factor up to 9.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Multimedia databases, Query processing; H.3.3 [**Information Search and Retrieval**]: Retrieval models, Search process

General Terms

Theory, Experimentation, Performance

Keywords

Signature Quadratic Form Distance, Similarity Matrix Compression, Content-Based Multimedia Retrieval, Similarity Search, Efficient Query Processing

1. INTRODUCTION

Content-based multimedia retrieval [6, 9, 13, 15, 17] is a widespread interdisciplinary field attracting both academia and industry. Retrieving multimedia objects sharing the most similar contents with regard to the query object is frequently performed by computing distance values among the objects' feature representations. This fundamental task has to be performed effectively and efficiently in order to satisfy users aiming at retrieving the most relevant objects in low query response times.

SISAP '10, September 18–19, 2010, Istanbul, Turkey.

In particular for multimedia data where similarity measures have to cope with adaptable content representations, the *Signature Quadratic Form Distance* [2, 5] has shown good applicability and high quality in comparing flexible feature representations, so-called *feature signatures*, with each other leading to high retrieval performance [3]. By adapting the cross-dimension concept of the traditional *Quadratic Form Distance* [8, 16], this adaptive similarity measure is not only defined for the comparison of feature histograms exhibiting the same length and structure but also for feature signatures of different size and structure.

Although it is shown that the Signature Quadratic Form Distance is a generalization of the Quadratic Form Distance [5], approximation techniques for efficient query processing [1, 10, 12, 16] applicable to the traditional Quadratic Form Distance are generally not applicable to the Signature Quadratic Form Distance [4]. Thus, the present paper introduces the *similarity matrix compression* approach in order to reduce the computational effort spent for the computation of the Signature Quadratic Form Distance. By making use of feature signatures partially sharing the same inherent information, we compress the distance's similarity matrix and show how to compute the Signature Quadratic Form Distance thereon. Our approach improves the efficiency of a single distance computation by a factor up to 9 and is thus applicable for efficient query processing in voluminous multimedia databases.

The organization of the present paper is as follows: in Section 2, we review the feature extraction process and the Signature Quadratic Form Distance on feature signatures. Additionally, we argue the inapplicability of existing techniques. Section 3 is devoted to the similarity matrix compression approach which is evaluated in Section 4. We conclude our work in Section 5 with an brief outlook on future work.

2. RELATED WORK AND BACKGROUND

In this section, we first study the feature extraction process and the resulting feature representation forms of multimedia data, *feature histograms* [8, 11] and *feature signatures* [3, 14]. Then, we present the Signature Quadratic Form Distance defined to compare feature histograms as well as feature signatures. Last, we briefly discuss efficient query processing techniques existing for the Quadratic Form Distance and argue their inapplicability to the Signature Quadratic Form Distance.

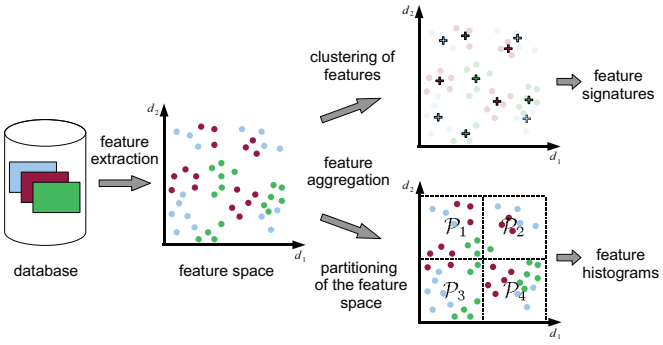


Figure 1: The feature extraction process in two steps: the objects' properties are extracted and mapped in some feature space and then represented in a more compact form.

The goal of the feature extraction process is to digitize and store the data objects' inherent properties in a compact way. Figure 1 illustrates the two main steps of the feature extraction process.

In the first step of the process, *feature extraction*, each data object is mapped onto a set of features in an appropriate feature space $\mathcal{FS}$ which comprises various dimensions, such as position, color, or texture. In this way, the content of each data object is exhibited via its feature distribution in the feature space. The features in the figure belonging to the same data object are shown via the same color, i.e. the blue, red, and green data objects are expressed by the blue, red, and green points in the feature space, respectively. More details about features for content-based multimedia retrieval can for instance be found in the work of Deselaers et al. [7] and Veltkamp et al. [18].

The second step of the process, *feature aggregation*, aggregates the extracted data objects' feature distributions via global partitioning of the feature space or clustering of features for each data object individually, as can be seen on the right-hand side in Figure 1. For each data object o, a single *feature histogram* of form $h^o = (h^{o_1}, \ldots, h^{o_n})$ can be generated where each entry $h^{o_i} \in \mathcal{R}^{\geq 0}$ of the feature histogram corresponds to the number of features located in the corresponding global partition $\mathcal{P}_i$. Thus a global partitioning of n partitions will lead to n-dimensional feature histograms. In the figure, we illustrate the aggregation of features according to four global partitions $\mathcal{P}_1, \ldots, \mathcal{P}_4$, which are fixed for all database objects. In contrast to feature histograms, each *feature signature* aggregates its data object features by individual clustering. As can be seen in the figure, the resulting feature signatures consist of clusters of the objects' features with the corresponding weights and centroids, visualized by pluses with different colors for each data object. The resulting feature signatures of the red and blue objects contain four centroids while the green object's feature signature comprises three centroids. In this way, each data object is represented by a single feature signature which is formally defined below.

Figure 2: An example image and the corresponding feature signature visualization.

DEFINITION 1. *Feature Signature*
Given a feature space $\mathcal{FS}$ and a clustering $\mathcal{C} = \mathcal{C}_1, \ldots, \mathcal{C}_n$ of the features $f_1, \ldots, f_k \in \mathcal{FS}$ of object o, the feature signature S^o is defined as a set of tuples from $\mathcal{FS} \times \mathcal{R}^+$ as follows:

$$S^o = \{\langle c^{o_i}, w^{o_i} \rangle, i = 1, \ldots, n\},$$

where $c^{o_i} = \frac{\sum_{f \in \mathcal{C}_i} f}{|\mathcal{C}_i|}$ and $w^{o_i} = \frac{|\mathcal{C}_i|}{k}$ represent the centroid and weight, respectively.

According to Definition 1, a feature signature S^o is a set of centroids $c^{o_i} \in \mathcal{FS}$ with the corresponding weights $w^{o_i} \in \mathcal{R}^+$ of the clusters $\mathcal{C}_i$. As the clustering per individual data object is not restricted to predefined partitions in the feature space, a feature signature represents the feature distribution of its data object more meaningfully than a feature histogram. Due to the individual clustering of features, feature signatures furthermore achieve a better balance between expressiveness and efficiency than feature histograms. Moreover, each feature histogram can be expressed as a corresponding feature signature by substituting the clustering with global partitioning.

An example image and its feature signature visualization are given in Figure 2 where the feature signature of the image is visualized from a five dimensional feature space (two position and three color dimensions) in a two-dimensional position space. Each circle and its radius correspond to a cluster and the weight of the cluster, respectively. The figure indicates that the corresponding feature signature approximates the visual content of an image very well. In this example, the sky, beach, and plants can be matched to their pairs of centroids and weights, accordingly.

Based on the previously defined feature representation forms, namely feature histograms and feature signatures, we will continue with presenting the Signature Quadratic Form Distance [2, 5], an adaptive similarity measure for content-based multimedia retrieval [3]. As this distance generalizes the traditional Quadratic Form Distance [8, 16], we give the latter's definition and explanation first.

DEFINITION 2. *Quadratic Form Distance*
Given two feature histograms h^q and h^p and a similarity matrix $A \in \mathcal{R}^{|h^q| \times |h^p|}$, the Quadratic Form Distance QFD_A between h^q and h^p is defined as:

$$QFD_A(h^q, h^p) = \sqrt{(h^q - h^p) \cdot A \cdot (h^q - h^p)^T}.$$

The similarity matrix A of a Quadratic Form Distance QFD_A realizes the cross-dimension concept by modeling similarities among all dimensions of the feature histograms. While the Quadratic Form Distance given in Defintion 2 is limited to compare feature histograms of the same size and structure, the Signature Quadratic Form Distance is able to compare flexible feature signatures of different size and structure with each other. For this purpose, the cross-dimension concept of the Quadratic Form Distance is adapted to the Signature Quadratic Form Distance. The distance computation is shifted from comparing all dimensions of the feature histograms to comparing all centroids of the feature signatures. For this purpose, the Signature Quadratic Form Distance makes use of a similarity function f_s modeling the similarity between two centroids. The formal definition of the Signature Quadratic Form Distance is given below.

DEFINITION 3. *Signature Quadratic Form Distance*
Given two feature signatures $S^q = \{\langle c^{q_i}, w^{q_i}\rangle |\ i = 1, \ldots, n\}$ *and* $S^p = \{\langle c^{p_i}, w^{p_i}\rangle |\ i = 1, \ldots, m\}$, *and a similarity function* $f_s(c_i, c_j) \mapsto \mathcal{R}$, *the Signature Quadratic Form Distance* $SQFD_{f_s}$ *between* S^q *and* S^p *is defined as:*

$$SQFD_{f_s}(S^q, S^p) = \sqrt{(w^q| - w^p) \cdot A_{f_s} \cdot (w^q| - w^p)^T},$$

where $A_{f_s} \in \mathcal{R}^{(n+m)\times(n+m)}$ *is the similarity matrix arising from applying the similarity function* f_s *to the corresponding centroids, i.e.* $a_{ij} = f_s(c_i, c_j)$. *Furthermore,* $w^q = (w^{q_1}, \ldots, w^{q_n})$ *and* $w^p = (w^{p_1}, \ldots, w^{p_m})$ *form weight vectors, and* $(w^q| - w^p) = (w^{q_1}, \ldots, w^{q_n}, -w^{p_1}, \ldots, -w^{p_m})$ *denotes the concatenation of* w^q *and* $-w^p$.

The similarity matrix A_{f_s} is dynamically determined for each comparison of two feature signatures and reflects similarities among all centroids of both feature signatures. $A_{f_s} = [a_{ij}]$ is computed by similarity functions [2, 5], such as $a_{ij} = e^{-L_2^2(c_i,c_j)/2}$. In this way, the Signature Quadratic Form Distance between two feature signatures S^q and S^p is computed by taking into account the centroids' weights and positions in the feature space.

The Signature Quadratic Form Distance is able to to determine a distance value between feature signatures and also feature histograms of any lengths and structure, by adapting the cross-dimension concept. Although it generalizes the Quadratic Form Distance, we argue for the inapplicability of existing efficient query processing techniques in the remainder of this section.

Over the last couple of years, numerous works addressing efficient query processing with the Quadratic Form Distance on feature histograms have been presented. Ellipsoid query processing using index structures [10, 16], reduction of dimensionality [12], and approximation techniques with sphere and box approximations [1] are some examples of these techniques. By examining these techniques, we have seen that they all rely on a fixed similarity matrix which means that the matrix will not change during a single query process. In the case of feature histograms, the entries of a similarity matrix model similarities among individual global partitions. This induces a fixed similarity matrix as the partitions typically do not change. In the feature signature case, the similarity matrix models similarities among centroids, which are based on an individual clustering, and thus changes from computation to computation. Thus, we have to apply the existing techniques for each distance computation individually, which is inapplicable in terms of efficiency. This briefly describes the inapplicability of existing techniques to the Signature Quadratic Form Distance in order to improve the efficiency of query processing. More details can be found in the work of Beecks et al. [4].

In the next section, we will present the *similarity matrix compression* approach in order to reduce the computational effort spent for single Signature Quadratic Form Distance computations.

3. SIMILARITY MATRIX COMPRESSION

In this section, we present the *similarity matrix compression* approach for the efficient query processing regarding the Signature Quadratic From Distance.

The idea of the proposed approach is to compress the similarity matrix of the Signature Quadratic Form Distance by making use of feature signatures partially sharing the same inherent information.

In order to formalize the compression in form of a specific Signature Quadratic Form Distance, we first define global and local components of feature signatures. Local components express individual information of a feature signature whereas global components capture the information shared among all feature signatures.

Without loss of generality, we define global and local components between *two* feature signatures in below. In this particular case, the global and local components of a feature signature only depend on the other feature signature. The definition can be easily extended to a larger set of feature signatures.

DEFINITION 4. *Global and Local Components*
Given two feature signatures $S^q = \{\langle c^{q_i}, w^{q_i}\rangle, i = 1, \ldots, n\}$ *and* $S^p = \{\langle c^{p_i}, w^{p_i}\rangle, i = 1, \ldots, m\}$, *we define the global components* S_g^q *and* S_g^p *and the local components* S_l^q *and* S_l^p *of the feature signatures* S^q *and* S^p *as follows:*

$$\begin{aligned} S_g^q &:= \{\langle c, w\rangle \in S^q \mid \exists w' \in \mathcal{R}^+ : \langle c, w'\rangle \in S^p\}, \\ S_g^p &:= \{\langle c, w\rangle \in S^p \mid \exists w' \in \mathcal{R}^+ : \langle c, w'\rangle \in S^q\}, \\ S_l^q &:= S^q \backslash S_g^q, \\ S_l^p &:= S^p \backslash S_g^p. \end{aligned}$$

The global components S_g^q and S_g^p of the corresponding feature signatures contain the tuples from $\mathcal{FS} \times \mathcal{R}^+$ whose centroids appear in both feature signatures as *global centroids*. In contrast, the local components S_l^q and S_l^p comprise tuples whose centroids are contained in exactly one feature signature.

In other words, global and local components are themselves feature signatures reflecting shared and individual information, respectively. By rearranging the weights of these components, the sorted weight vectors, as given in the following definition, are implied.

DEFINITION 5. *Sorted Weight Vectors*
Given two feature signatures $S^q = \{\langle c^{q_i}, w^{q_i}\rangle, i = 1, \ldots, n\}$ *and* $S^p = \{\langle c^{p_i}, w^{p_i}\rangle, i = 1, \ldots, m\}$ *with* $k \leq \min\{n, m\}$ *global centroids, i.e.* $|S_g^q| = |S_g^p| = k$*, we define the sorted weight vectors* $\widetilde{w}^q, \widetilde{w}^p$ *as follows:*

$$\begin{aligned}\widetilde{w}^q &:= (\widetilde{w}_g^q|\widetilde{w}_l^q) := (\widetilde{w}^{q_1}, \ldots, \widetilde{w}^{q_k}|\widetilde{w}^{q_{k+1}}, \ldots, \widetilde{w}^{q_n}),\\ \widetilde{w}^p &:= (\widetilde{w}_g^p|\widetilde{w}_l^p) := (\widetilde{w}^{p_1}, \ldots, \widetilde{w}^{p_k}|\widetilde{w}^{p_{k+1}}, \ldots, \widetilde{w}^{p_m}),\end{aligned}$$

under the following conditions:

$$\begin{aligned}&(i) \quad \forall i\ 1 \leq i \leq k,\ \exists c.\\ &\qquad \langle c, \widetilde{w}^{q_i}\rangle \in S_g^q \wedge \langle c, \widetilde{w}^{p_i}\rangle \in S_g^p,\\ &(ii) \quad \forall i\ k+1 \leq i \leq n,\ \exists c.\ \langle c, \widetilde{w}^{q_i}\rangle \in S_l^q,\\ &(iii) \quad \forall i\ k+1 \leq i \leq m,\ \exists c.\ \langle c, \widetilde{w}^{p_i}\rangle \in S_l^p,\\ &(iv) \quad \forall i, j\ 1 \leq i, j \leq n,\ i \neq j,\ \exists c, c'.\\ &\qquad \langle c, \widetilde{w}^{q_i}\rangle \in S^q \wedge \langle c', \widetilde{w}^{q_j}\rangle \in S^q \wedge c \neq c',\\ &(iv) \quad \forall i, j\ 1 \leq i, j \leq m,\ i \neq j,\ \exists c, c'.\\ &\qquad \langle c, \widetilde{w}^{p_i}\rangle \in S^p \wedge \langle c', \widetilde{w}^{p_j}\rangle \in S^p \wedge c \neq c'.\end{aligned}$$

Sorting and thus aligning the feature signatures' weight vectors w^q and w^p (see Definition 3 in Section 2) according to the global and local components given in Definition 4, we defined the sorted weight vectors $\widetilde{w}^q$ and $\widetilde{w}^p$ which are necessary for the similarity matrix compression. It can be shown that the computation of the Signature Quadratic Form Distance yields the same distance value whether the weight vectors w^q, w^p or the sorted weight vectors $\widetilde{w}^q, \widetilde{w}^p$ are used, i.e. the computed distance value of the Signature Quadratic Form Distance is independent of the centroids order in which they appear in the feature signatures.

Based on the definitions of global and local components (Definition 4) and sorted weight vectors of feature signatures (Definition 5), we formalize the similarity matrix compression approach in the remainder of this section. For this purpose, we illustrate the similarity matrix compression approach by visualizing the structure of the similarity matrices without compression and with compression in Figure 3. We denote the original uncompressed similarity matrix depicted on the left-hand side by A, instead of A_{f_s}, and the compressed similarity matrix depicted on the right-hand side by A'. The structures of both similarity matrices are given by the corresponding global and local centroids denoted as c_g^q, c_g^p, c_l^q, and c_l^p. While the original uncompressed similarity matrix exhibits a 4×4 block structure, the compressed one will be reduced to a similarity matrix comprising a 3×3 block structure.

Considering the structure of the original uncompressed similarity matrix A, the goal of the similarity matrix compression is to compress the blocks marked green and orange as they contain the same similarity information which is implied by the global centroids. In this way, the similarity matrix A is compressed to a smaller similarity matrix A' comprising the same similarity information as A. This compression can be used to simplify the computation of the Signature Quadratic Form Distance in order to speed up the computation, as shown in the following theorem.

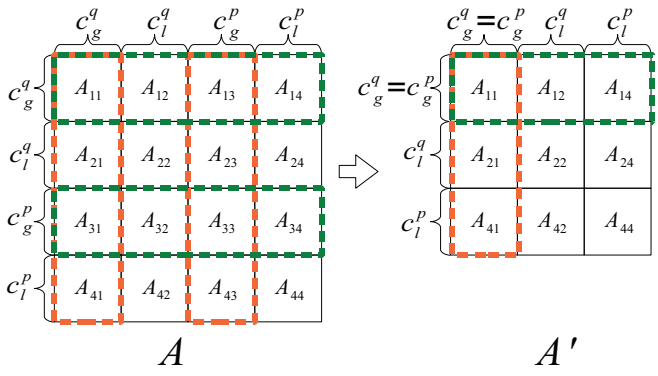


Figure 3: Similarity matrix compression: the original uncompressed similarity matrix A and the compressed similarity matrix A'.

THEOREM 1. *Similarity Matrix Compression*
Given two feature signatures $S^q = \{\langle c^{q_i}, w^{q_i}\rangle, i = 1, \ldots, n\}$ *and* $S^p = \{\langle c^{p_i}, w^{p_i}\rangle, i = 1, \ldots, m\}$*, it holds that*

$$\begin{aligned}&SQFD_A^2(S^q, S^p)\\ &\quad = (\widetilde{w}_g^q - \widetilde{w}_g^p|\widetilde{w}_l^q| - \widetilde{w}_l^p) \cdot A' \cdot (\widetilde{w}_g^q - \widetilde{w}_g^p|\widetilde{w}_l^q| - \widetilde{w}_l^p)^T,\end{aligned}$$

where $A \in \mathcal{R}^{(n+m)\times(n+m)}$ *and* $A' \in \mathcal{R}^{(n+m-k)\times(n+m-k)}$ *are the uncompressed and compressed similarity matrices as depicted in Figure 3.*

PROOF.

$$\begin{aligned}&SQFD_A^2(S^q, S^p)\\ =\ & ((\widetilde{w}_g^q|\widetilde{w}_l^q)| - (\widetilde{w}_g^p|\widetilde{w}_l^p)) \cdot A \cdot ((\widetilde{w}_g^q|\widetilde{w}_l^q)| - (\widetilde{w}_g^p|\widetilde{w}_l^p))^T\\ =\ & \underbrace{\widetilde{w}_g^q A_{11} \widetilde{w}_g^{qT} - \widetilde{w}_g^q A_{13} \widetilde{w}_g^{pT} - \widetilde{w}_g^p A_{31} \widetilde{w}_g^{qT} + \widetilde{w}_g^p A_{33} \widetilde{w}_g^{pT}}_{A_{11}=A_{13}=A_{31}=A_{33} \Rightarrow (\widetilde{w}_g^q - \widetilde{w}_g^p) \cdot A_{11} \cdot (\widetilde{w}_g^q - \widetilde{w}_g^p)^T}\\ +\ & \underbrace{\widetilde{w}_g^q A_{12} \widetilde{w}_l^{qT} - \widetilde{w}_g^p A_{32} \widetilde{w}_l^{qT} - \widetilde{w}_g^q A_{14} \widetilde{w}_l^{pT} + \widetilde{w}_g^p A_{34} \widetilde{w}_l^{pT}}_{A_{12}=A_{32}, A_{14}=A_{34} \Rightarrow (\widetilde{w}_g^q - \widetilde{w}_g^p) \cdot A_{12} \cdot \widetilde{w}_l^{qT} - (\widetilde{w}_g^q - \widetilde{w}_g^p) \cdot A_{14} \cdot \widetilde{w}_l^{pT}}\\ +\ & \underbrace{\widetilde{w}_l^q A_{21} \widetilde{w}_g^{qT} - \widetilde{w}_l^q A_{23} \widetilde{w}_g^{pT} - \widetilde{w}_l^p A_{41} \widetilde{w}_g^{qT} + \widetilde{w}_l^p A_{43} \widetilde{w}_g^{pT}}_{A_{21}=A_{23}, A_{41}=A_{43} \Rightarrow \widetilde{w}_l^q \cdot A_{21} \cdot (\widetilde{w}_g^q - \widetilde{w}_g^p)^T - \widetilde{w}_l^p \cdot A_{41} \cdot (\widetilde{w}_g^q - \widetilde{w}_g^p)^T}\\ +\ & \widetilde{w}_l^q A_{22} \widetilde{w}_l^{qT} - \widetilde{w}_l^q A_{24} \widetilde{w}_l^{pT} - \widetilde{w}_l^p A_{42} \widetilde{w}_l^{qT} + \widetilde{w}_l^p A_{44} \widetilde{w}_l^{pT}\\ =\ & (\widetilde{w}_g^q - \widetilde{w}_g^p) \cdot (A_{11}(\widetilde{w}_g^q - \widetilde{w}_g^p)^T + A_{12} \widetilde{w}_l^{qT} - A_{14} \widetilde{w}_l^{pT})\\ +\ & \widetilde{w}_l^q \cdot (A_{21}(\widetilde{w}_g^q - \widetilde{w}_g^p)^T + A_{22} \widetilde{w}_l^{qT} - A_{24} \widetilde{w}_l^{pT})\\ +\ & \widetilde{w}_l^p \cdot (A_{41}(\widetilde{w}_g^q - \widetilde{w}_g^p)^T + A_{42} \widetilde{w}_l^{qT} - A_{44} \widetilde{w}_l^{pT})\\ =\ & (\widetilde{w}_g^q - \widetilde{w}_g^p|\widetilde{w}_l^q| - \widetilde{w}_l^p) \cdot A' \cdot (\widetilde{w}_g^q - \widetilde{w}_g^p|\widetilde{w}_l^q| - \widetilde{w}_l^p)^T.\end{aligned}$$

□

According to Theorem 1, the computation of the Signature Quadratic Form Distance is carried out via the compressed similarity matrix A' comprising 9 blocks which can be seen in Figure 3 on the right-hand side.

Let us now examine these blocks of the compressed similarity matrix A' and the resulting Signature Quadratic Form Distance computation shown in Theorem 1 in terms of efficient query processing. The query processing is carried

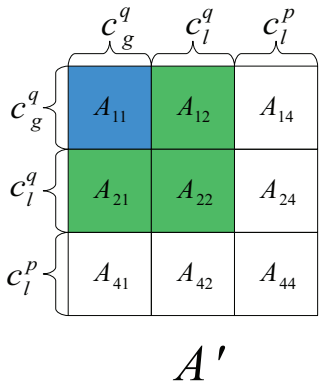


Figure 4: Precomputing the blocks of the compressed similarity matrix A': the blue block can be computed before the query is issued, while the green blocks can be computed once the query is issued.

out by computing the Signature Quadratic Form Distance between the query feature signature S^q and each database object's feature signature S^p and ranking the database objects according to increasing distance values to the query.

On the premise that the global centroids c_g^q and c_g^p of the global components S_g^q and S_g^p of a specific database are known in advance and that the query process makes use of a predetermined similarity function, we can precompute the entries of the corresponding similarity matrix block A_{11} before a query is given.

At query time the entries of the similarity matrix blocks A_{12}, A_{21}, and A_{22} can be computed. They contain similarity information about the query feature signature's local components and thus solely rely on the query feature signature. Furthermore, the remaining similarity matrix blocks A_{14}, A_{24}, A_{41}, A_{42}, and A_{44} have to be computed for each distance computation since these blocks can only be computed by taking into account the local centroids of database objects' feature signatures.

We depict the precomputation possibilities of a compressed similarity matrix in Figure 4 where the computation of the similarity matrix' entries of block A_{11} is indicated by the blue color. These entries can be computed in advance, without knowing the query. The green colored blocks indicate blocks of the similarity matrix which can be computed once the query is issued.

To sum up, we have introduced the similarity matrix compression approach compressing a similarity matrix of the Signature Quadratic Form Distance, which comprises a 4×4 block structure, to a smaller similarity matrix comprising a 3×3 block structure. We have shown in Theorem 1 that the Signature Quadratic Form Distance can be efficiently computed using the compressed similarity matrix. Additionally, we briefly described which blocks of the compressed similarity matrix can be precomputed in order to improve the efficiency of the query process even more. In the following section, we evaluate the similarity matrix compression approach in terms of efficiency.

Table 1: Average computation time values in milliseconds needed to compute the Signature Quadratic Form Distance by using similarity matrix compression *with precomputation*.

	size of feature signatures				
$c_g(\%)$	100	200	400	800	s_f
80	3.14	12.71	50.78	206.05	9.0
60	6.95	27.67	110.45	445.04	4.1
40	11.24	44.93	179.43	720.15	2.6
20	16.09	64.49	257.65	1033.54	1.8
0	28.81	114.96	459.89	1844.17	

Table 2: Average computation time values in milliseconds needed to compute the Signature Quadratic Form Distance by using similarity matrix compression *without precomputation*.

	size of feature signatures				
$c_g(\%)$	100	200	400	800	s_f
80	10.37	41.86	166.12	663.7	2.8
60	14.21	56.54	228.09	905.16	2.0
40	18.35	73.76	297.93	1181.04	1.6
20	23.25	93.39	374.31	1493.24	1.2
0	28.81	114.96	459.89	1844.17	

4. EXPERIMENTAL EVALUATION

In this section, we evaluate our approach in terms of efficient query processing by measuring the computation time needed to compare two feature signatures using the Signature Quadratic Form Distance.

For this purpose, we generated feature signatures with different size, i.e. with different number of centroids and weights, where we assumed 7-dimensional centroids reflecting position, color, and texture information. We varied the number of centroids of the feature signatures to be compared between 100 and 800, and the ratio c_g of global centroids between 0% and 80%.

In our experiments, we computed the Signature Quadratic Form Distance using the similarity function $a_{ij} = e^{-L_2(c_i,c_j)/2}$ where L_2 denotes the Euclidean distance. We ran all experiments on a 2.4GHz Intel Core 2 Duo machine and implemented our approach in JAVA 1.6.

The results of the computation time values averaged over 100 Signature Quadratic Form Distance computations are shown in Table 1 and Table 2. The first table reports the measured computation time values by applying the similarity matrix compression approach and all the possible precomputations as described in the previous section. The computation time values, given in milliseconds, indicate that the efficiency of the proposed similarity matrix compression approach depends on the ratio of global centroids c_g, i.e. the size of the global components S_g^q and S_g^p. The larger the size of global components, the higher the efficiency of our approach. Consequently, the highest speed-up factor s_f of 9 was reached when we set the ratio c_g of global centroids to 80%.

Table 2 gives the computation time values by using the similarity matrix compression approach without precomputation. As a result, the time needed to compute the Signature Quadratic Form Distance increases and the highest speed-up factor s_f of 2.8 was also reached by setting the ratio c_g of global centroids to 80%.

To sum up, we have shown that the proposed similarity matrix approach reduces the computation time of the Signature Quadratic Form Distance. Combining the similarity matrix compression with the possible precomputations, we reach a speed-up factor of 9. Thus our approach can successfully be applied to rank multimedia databases efficiently.

5. CONCLUSIONS

In this paper, we presented the similarity matrix compression approach for improving the efficiency of Signature Quadratic Form Distance computations. The experimental results indicate that a speed-up factor of 9 is obtained when our approach is applied to the Signature Quadratic Form Distance.

As future work, we plan to involve users in the evaluation process of the similarity matrix compression approach on different image, video, and audio databases. Furthermore, we plan to study the effectiveness of the proposed approach in different content-based retrieval applications.

6. REFERENCES

[1] M. Ankerst, B. Braunmüller, H.-P. Kriegel, and T. Seidl. Improving Adaptable Similarity Query Processing by Using Approximations. In *Proc. 24th Int. Conf. on Very Large Data Bases*, pages 206–217, 1998.

[2] C. Beecks, M. S. Uysal, and T. Seidl. Signature Quadratic Form Distances for Content-Based Similarity. In *Proc. 17th ACM Int. Conf. on Multimedia*, pages 697–700, 2009.

[3] C. Beecks, M. S. Uysal, and T. Seidl. A comparative study of similarity measures for content-based multimedia retrieval. In *Proc. IEEE Int. Conf. on Multimedia and Expo (Workshop on Visual Content Identification and Search)*, pages 1552–1557, 2010.

[4] C. Beecks, M. S. Uysal, and T. Seidl. Efficient k-Nearest Neighbor Queries with the Signature Quadratic Form Distance. In *Proc. 4th Int. Workshop on Ranking in Databases*, pages 10–15, 2010.

[5] C. Beecks, M. S. Uysal, and T. Seidl. Signature Quadratic Form Distance. In *Proc. ACM Int. Conf. on Image and Video Retrieval*, pages 438–445, 2010.

[6] R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image Retrieval: Ideas, Influences, and Trends of the New Age. *ACM Comp. Surv.*, 40(2):1–60, 2008.

[7] T. Deselaers, D. Keysers, and H. Ney. Features for Image Retrieval: An Experimental Comparison. *Information Retrieval*, 11(2):77–107, 2008.

[8] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and Effective Querying by Image Content. *Journal of Intelligent Information Systems*, 3(3/4):231–262, 1994.

[9] P. Geetha and V. Narayanan. A Survey of Content-Based Video Retrieval. *Journal of Computer Science*, 4(6):474–486, 2008.

[10] J. Hafner, H. S. Sawhney, W. Equitz, M. Flickner, and W. Niblack. Efficient color histogram indexing for quadratic form distance functions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(7):729–736, 1995.

[11] R. Hu, S. M. Rüger, D. Song, H. Liu, and Z. Huang. Dissimilarity measures for content-based image retrieval. In *Proc. IEEE Int. Conf. on Multimedia and Expo*, pages 1365–1368, 2008.

[12] H.-P. Kriegel and T. Seidl. Approximation-Based Similarity Search for 3-D Surface Segments. *Geoinformatica*, 2(2):113–147, 1998.

[13] M. S. Lew, N. Sebe, C. Djeraba, and R. Jain. Content-Based Multimedia Information Retrieval: State of the Art and Challenges. *ACM TOMCCAP*, 2(1):1–19, 2006.

[14] Y. Rubner, C. Tomasi, and L. J. Guibas. The Earth Mover's Distance as a Metric for Image Retrieval. *Int. Journal of Computer Vision*, V40(2):99–121, 2000.

[15] N. Sebe, M. Lew, X. Zhou, T. Huang, and E. Bakker. The state of the art in image and video retrieval. *Proc. ACM Int. Conf. on Image and Video Retrieval*, pages 7–12, 2003.

[16] T. Seidl and H.-P. Kriegel. Efficient User-Adaptable Similarity Search in Large Multimedia Databases. In *Proc. 23rd Int. Conf. on Very Large Data Bases*, pages 506–515, 1997.

[17] A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(12):1349–1380, 2000.

[18] R. Veltkamp, M. Tanase, and D. Sent. Features in content-based image retrieval systems: A survey. *State-of-the-art in content-based image and video retrieval*, pages 97–124, 2001.

Demonstrations

SISAP 2010

www.MMRetrieval.net: A Multimodal Search Engine

Konstantinos Zagoris Avi Arampatzis Savvas A. Chatzichristofis
Department of Electrical and Computer Engineering
Democritus University of Thrace
Xanthi, Greece
{kzagoris,avi,schatzic}@ee.duth.gr

ABSTRACT

We introduce an experimental search engine for multilingual and multimedia information, employing a holistic web interface and enabling the use of highly distributed indices. Modalities are searched in parallel, and results can be fused via several selectable methods. The engine also provides multistage retrieval, as well as a single text index baseline for comparison purposes. Initial impressions on its effectiveness are positive, while its efficiency may easily be improved.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

1. INTRODUCTION

As digital information is increasingly becoming multimodal, the days of single-language text-only retrieval are numbered. Take as an example Wikipedia where a single topic may be covered in several languages and include non-textual media such as image, sound, and video. Moreover, non-textual media may be annotated with text in several languages in a variety of metadata fields such as object caption, description, comment, and filename. Current search engines usually focus on limited numbers of modalities at a time, e.g. English text queries on English text or maybe on textual annotations of other media as well, not making use of all information available. Final rankings are usually results of fusion of individual modalities, a task which is tricky at best especially when noisy or incomplete modalities are involved.

In this paper we present the experimental multimodal search engine `http://www.mmretrieval.net` (Fig.1), which allows multimedia and multilingual queries in a single search and makes use of the total available information in a multimodal collection. All modalities are indexed separately and searched in parallel, and results can be fused with different methods depending on *a)* the noise and completeness characteristics of the modalities in a collection, and *b)* whether the user is in a need of initial precision or high recall. Beyond fusion, we also provide 2-stage retrieval by first thresholding the results obtained by secondary modalities, targeting recall, and then re-ranking the results based on the primary modality.

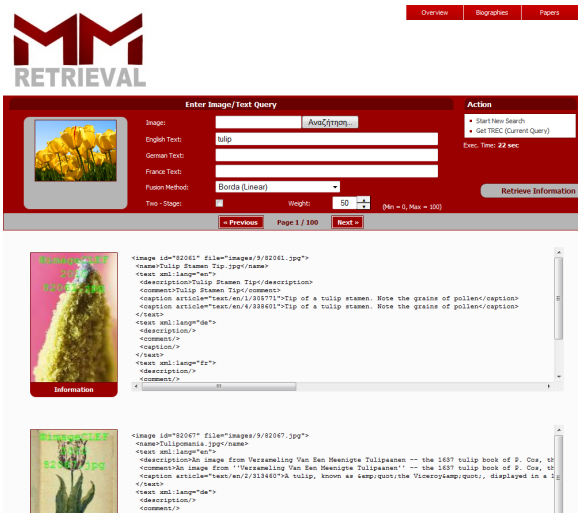


Figure 1: The www.MMRetrieval.net search engine.

The engine demonstrates the feasibility of the proposed architecture and methods on the ImageCLEF 2010 Wikipedia collection.[1] The primary modality is image, consisting of 237434 items, associated with noisy and incomplete user-supplied textual annotations and the Wikipedia articles containing the images. Associated modalities are written in any combination of English, German, French, or any other unidentified language.

2. INDEXING

To index the images, we consider the family of descriptors known as Compact Composite Descriptors (CCDs). CCDs consist of more than one visual features in a compact vector, and each descriptor is intended for a specific type of image. We index with two descriptors from the family, i.e., the Joint Composite Descriptor (JCD) [4] and the recently proposed Spatial Color Distribution (SpCD) [3]. JCD is developed for color natural images, while SpCD is considered suitable for colored graphics and artificialy generated images. Thus, we have 2 image indices.

[1] `http://www.imageclef.org/2010/wiki`

SISAP '10 September 18-19, 2010, Istanbul, Turkey.

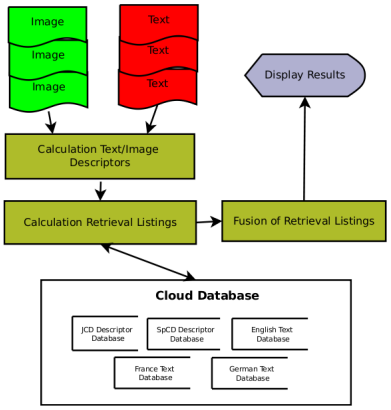


Figure 2: System's architecture.

The collection of images comes with XML metadata, consisting of a description, a comment, and multiple captions, per language (English, German, and French). Each caption is linked to the wikipedia article where the image appears in. Additionally, a raw comment is supplied which contains all the per-language comments and any other comment in an unidentified language; we do not use this field due to its great overlap with the per-language comments. Any of the above fields may be empty, noisy, or incomplete. Furthermore, a name field is supplied per image containing its filename. We do not use the supplied <license> field.

For text indexing and retrieval, we employ the Lemur Toolkit V4.11 and Indri V2.11 with the tf.idf retrieval model.[2] In order to have clean global (DF) and local statistics (TF, document length), we split the metadata per language and index them separately preserving the fields. Lemur allows searching within fields and we use this facility, as we will see below, resulting to many modalities. This, together with a separate index for the name field, results in 4 indices. Additionally, as a brute-force baseline, we also provide a single text index of all metadata and associated articles where no metadata fields or language information is used.

3. SEARCHING

The web application is developed in the C#/.NET Framework 4.0 and requires a fairly modern browser as the underlying technologies which are employed for the interface are HTML, CSS and JavaScript (AJAX). Fig.2 illustrates an overview of the architecture. The user provides image and text queries through the web interface which are dispatched in parallel to the associated databases. Retrieval results are obtained from each of the databases, fused into a single listing, and presented to the user.

Users can supply no, single, or multiple query images in a single search, resulting in $2 * i$ active image modalities, where i is the number of query images. Similarly, users can supply no text query or queries in any combination of the 3 languages, resulting in $5 * l$ active text modalities, where l is the number query languages: each supplied language results to 4 modalities, one per field described in the previous section, plus the name modality which we are matching with any language. The current beta version assumes that the user provides multilingual queries for a single search, while operationally query translation may be done automatically.

The results from each modality are fused by one of the supported methods. Fusion consists of two components: score normalization and combination. We provide two linear normalization methods, MinMax and Z-score, the ranked-based Borda Count in linear and non-linear forms, and the non-linear KIACDF. KIACDF is similar to the normalization introduced in [1], except that know-item queries are used (instead of historical) in estimating score transfer functions. We provide combination of scores across modalities with summation, multiplication, and maximum. In all fusion methods, except for where the max is used for combination, the user may select a weigh factor w, which determines the percentage contribution of the image modalities against the textual ones.

Beyond fusion, the system provides baseline searches on the single text index in two flavors: metadata only, and metadata including associated articles. In baseline searches, multilingual queries are concatenated and issued as one. Search can also be performed in a two-stage fashion. First, the text-only results of the baseline search on metadata plus articles are obtained. Then, the top-K results are re-ranked using only the image modalities which are fused by a selected method. We estimate the optimal K for maximizing the recall-oriented T9U measure, i.e. 2 gain per relevant retrieved and 1 loss per non-relevant retrieved, via the score-distributional method of [2].

4. FIRST IMPRESSIONS & OUTLOOK

In initial experiments, fusion methods using multiplication or summation seem to favor (in this order) initial precision at an expense of recall. Combination with max seems to favor recall, while two-stage retrieval seems to work best overall. Moreover, in theory, combination with max is more suitable than multiplication when descriptions are noisy or incomplete, while summation seems to provide in practice the most robust method.

We are currently planning controlled experiments in order to obtain a more concrete comparative evaluation of the effectiveness of the implemented methods. For enhancing efficiency, the multiple indices may easily be moved to different hosts.

5. REFERENCES

[1] A. Arampatzis and J. Kamps. A signal-to-noise approach to score normalization. In *Proceedings CIKM*. ACM, 2009.

[2] A. Arampatzis, J. Kamps, and S. Robertson. Where to stop reading a ranked list? Threshold optimization using truncated score distributions. In *Proceedings SIGIR*, pages 524–531. ACM, 2009.

[3] S. A. Chatzichristofis, Y. S. Boutalis, and M. Lux. SpCD - Spatial Color Distribution Descriptor - A fuzzy rule-based compact composite descriptor appropriate for hand drawn color sketches retrieval. In *Proceedings ICAART*, pages 58–63, 2010.

[4] S. A. Chatzichristofis, Y. S. Boutalis, and M. Lux. Selection of the proper compact composite descriptor for improving content based image retrieval. In *Proceedings SPPRA*, pages 134–140, 2009.

[2] `http://www.lemurproject.org`

SHIATSU: Annotating Your Videos the Easy Way![*]

Ilaria Bartolini
DEIS, University of Bologna, Italy
i.bartolini@unibo.it

Corrado Romani
DEIS, University of Bologna, Italy
corrado.romani@unibo.it

ABSTRACT

In this demonstration we present SHIATSU, an automatic semantic-based video tagging system which relies on shot boundary detection and hierarchical annotation. More in details, in SHIATSU, shots obtained from video segmentation are first automatically labelled and such labels are then propagated at the video level. The approach is novel and appealing because: 1) it only considers the visual content of each video in order to automatically suggest description labels, 2) predicted tags can be reviewed/accepted by the user and persistently stored in the system in order to be exploited when searching for video of interest, and 3) the provided GUI makes annotation of videos extremely intuitive and usable.

1. INTRODUCTION

Searching for needed information from large video collections based on visual content and genre still represents a main open problem. This requires the definition of effective and efficient automatic video characterization and annotation techniques, so as to allow domestic/professional users to correctly retrieve videos of interest [4].

In this demo we present SHIATSU (Semantic HIerarchical Automatic Tagging of videos by Segmentation Using cuts), a semantic-based hierarchical video annotation system which first segments each video in scenes that are *similar* in term of visual features and then automatically provides compact descriptions by means of tags (at both shot and video level) which are useful to categorize the video content [2]. The annotation process exploits labels of pre-annotated key frames which are *similar* to the video to be labelled. A demonstration of SHIATSU on the TRECVID benchmark is provided by using the knowledge base supplied by the *TRECVID-2007 High-Level Feature* task [5].

[*]This work is partially supported by the CoOPERARE MIUR Project.

SISAP 2010, September 18-19, 2010, Istanbul, Turkey.

2. SYSTEM OVERVIEW

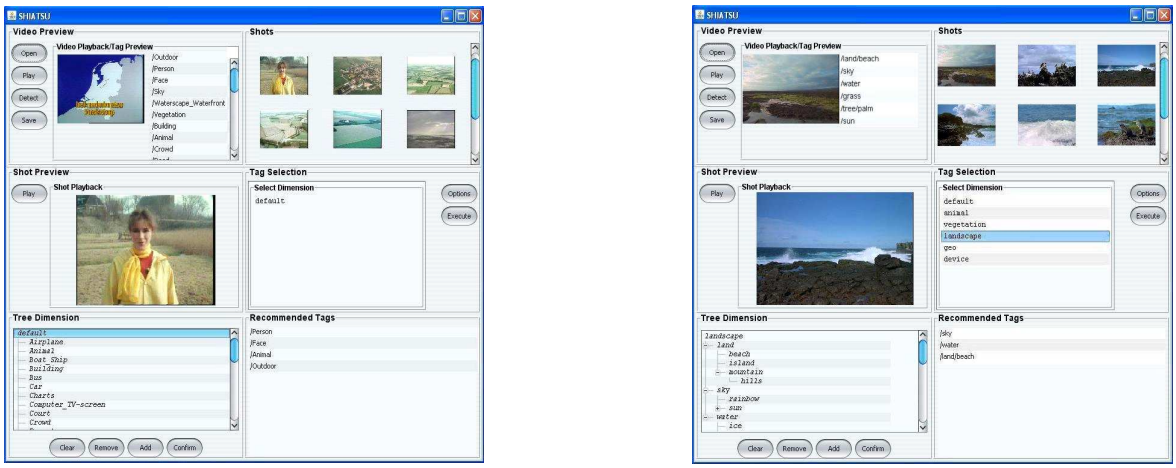
The core components of the SHIATSU system are a *video shot detector*, which fragments a video into coherent frame sequences, and a *video annotator* which attaches semantic concepts to such sequences. Shot tags are then propagated to the whole video, so as to obtain semantic indices for both the video and its shots: this allows the realization of a hierarchical, two-level, browsing platform.

When a video is processed, the shot detection component analyzes its frames and computes its shot boundaries, marking their timestamps (beginning and end of each shot). In details, color histograms and object edges are computed for every frame and such features are used to compare consecutive frames by applying two different distance metrics: this is because a shot transition usually produces a change in both the color and the texture structure of the frames. The shot selection process is done with a double dynamic threshold system which takes into account video content in order to adapt to different video types: frames are filtered on their color features and then on their edge features.

Every detected shot is then analyzed by the annotator component, which automatically assigns tags depending on the visual content of the shot. The user can then review the proposed tags and possibly modify the annotation results. After processing all the shots, the module selects the most appropriate tags for the whole video and saves all the information into a database. In details, the tagging phase exploits the Imagination system [1] and uses a set of pre-annotated images as a knowledge base. The system extracts a set of visual features from each image and saves the information in a database, indexing them efficiently with an implementation of the M-tree metric index [3].

The semantic concepts in the knowledge base can be organized either into a tree-shaped taxonomy named *dimension*, where terms are linked with a parent/child relationship, e.g., the term `landscape/land/beach` of the dimension `landscape` (see Figure 1 (b)) or as a flat structure, that we call `default` dimension, where all terms are at the same level (see Figure 1 (a)). Such dimensions can be easily modified and expanded.

When provided with a key frame to be labelled, SHIATSU first extracts its visual features. The key idea of tagging is to suggest those tags that are assigned to key frames in the knowledge base that are similar to the target frame (see [1] for more details). To efficiently perform similarity queries to key frames features, we exploit the M-tree index. Using the cuts timestamps, SHIATSU extracts a set of key frames representatives of each shot sequence, computes their vi-



(a) (b)

Figure 1: The SHIATSU interface: video tagging results for two videos when using the `default` dimension (a) and a specific dimension (i.e., `landscape`) (b).

sual features and compares them with those contained in the knowledge base. The annotator component suggests a set of concepts for each key frame: only terms recurring in the majority of key frames are selected as suitable concepts to describe the whole shot sequence. The number of key frames processed for every shot depends on the shot length. To avoid producing an overwhelming number of tags for each shot, only the most frequent tags retrieved for each key frame in the sequence are maintained (the total number of tags for each shot is also limited). The proposed tags can then be reviewed by the user.

Shot tags are useful to browse sequences across different videos, but they could be too specific to index a whole video, especially if this contains a wide range of different visual content. A simple criterion to select video tags from the set of shot tags is to weigh every tag depending on its frequency and the length of the shot it is associated to. Tags are ordered by descending values of weight and the first 10 tags (if available) become video tags. The rationale behind the proposed propagation method relies on the fact that concepts extracted from long shot sequences and/or that appear in several shots are probably more relevant, to describe the content of a whole video, than concepts occurring rarely or in short sequences. Finally, both shot and video tags are stored in the SHIATSU database, thus the user can exploit both of them when searching for her videos of interest.

3. DEMONSTRATION

Let us illustrate a possible usage scenario of SHIATSU (see Figure 1 for two real examples). First of all, the user opens a video by means of the "Open" button. The selected video can be played within the *Video Preview* frame by pressing the "Play" button, or segmented in shots trough the "Detect" option. In the latter case, the video is first processed by the video shot detector component and detected shots are then shown within the *Shots* frame.

At this point, the user can enable the tagging facilities by simply clicking on a specific shot. If this is the case, the user can play the shot within the *Shot Preview* frame by selecting the "Play" button. Then the user can select the dimension she prefers to consider during the annotation process; if user selection falls into the unstructured `default` dimension (SHIATSU default choice), as shown in the usage example of Figure 1 (a), all tags in the knowledge base are used as candidates for tagging. Coming back to our example, SHIATSU predicts tags `Person`, `Face`, `Animal`, and `Outdoor` which are, with exception for `Animal`, all relevant with respect to the shot content. On the other hand, if a specific dimension is selected (this is the case of the `landscape` dimension of the scenario depicted in Figure 1 (b)) the tags predicted by the annotator component will only contain concepts in that dimension. Continuing our running example, all predicted labels for the selected shot (i.e., `sky`, `water`, `land/beach`) are relevant.

Among tags predicted by the system for the shots, the user can refine them by deleting wrong labels and/or adding missing tags, depending on the precision of the provided results. When she is satisfied with the final result, she confirms it to the system which locally maintains provided annotations. Note that, in case a selected shot is already tagged, SHIATSU returns to the user the associated labels.

Any time a shot tagging is performed, SHIATSU propagates the labelling results at the video level by showing video tags within the *Video Preview* frame. Completing the running examples of Figure 1, video tags predicted by the system after annotating all the video shots are `Outdoor`, `Person`, `Face`, `Sky`, `Waterscape-Waterfront`, `Vegetation`, `Building`, `Animal` (not relevant), `Crowd` (not relevant), `Road` and `land/beach`, `sky`, `water`, `grass`, `tree/palm`, `sun` for the examples (a) and (b), respectively. At the end of the session, the user can save the result of the whole video annotation process for future use by selecting the "Save" button.

4. REFERENCES

[1] I. Bartolini and P. Ciaccia. Imagination: Accurate Image Annotation Using Link-analysis Techniques. In *AMR 2007*, pages 32–44, 2007.

[2] I. Bartolini, M. Patella, and C. Romani. SHIATSU: Semantic-Based Hierarchical Automatic Tagging of Videos by Segmentation using Cuts. In *AIEMPro 2010*. To appear.

[3] P. Ciaccia, M. Patella, and P. Zezula. M-tree: an Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB 1997*, pages 426–435, 1997.

[4] P. Geetha, and V. Narayanan. A Survey of Content-based Video Retrieval. In *Journal of Computer Science*, 4(6), pages 474–486, 2008.

[5] TRECVID Video Retrieval Evaluation: http://trecvid.nist.gov.

Audio Similarity Retrieval Engine

Pavel Jurkas
Masaryk University
Brno, Czech Republic
xjurkas@fi.muni.cz

Milan Štefina
Masaryk University
Brno, Czech Republic
xstefina@fi.muni.cz

David Novak
Masaryk University
Brno, Czech Republic
xnovak8@fi.muni.cz

Michal Batko
Masaryk University
Brno, Czech Republic
batko@fi.muni.cz

ABSTRACT

This paper briefly describes an audio similarity retrieval engine included in the MUFIN project. The engine uses low-level audio descriptors defined by MPEG-7 standard for calculation of similarity measure between audio recordings. The core of the engine is implemented in Java with the use of the MESSIF framework that provides support for metric-based indexing and searching. The presentation layer of the engine is provided by the MUFIN interface.

Keywords

similarity search, metric space, audio, MPEG-7, MESSIF

1. INTRODUCTION

The presented audio similarity retrieval engine is based on the data concept of metric spaces. The specific space for this purpose is defined by descriptors extracted from audio recordings and a similarity measure on these descriptors. Indexing techniques and algorithms for processing similarity queries have already been implemented within the MUFIN project [1] with the aid of the MESSIF framework [2].

2. AUDIO DESCRIPTORS

MPEG-7 Audio is a standard describing audio metadata [5] that can be generally divided into low-level and high-level. Low-level audio descriptors are more general and applicable on any type of input audio signal. They can be used for various tasks and use cases. High-level audio descriptors are more specifically focused and are not used in the engine.

Low-level audio descriptors are typically formed by time series of samples computed from original audio signal. Samples are located in equidistant points in time (typical sampling rate is 100 Hz). The engine uses the following four low-level descriptors from the MPEG-7 Audio standard:

- Audio Power (AP) – one-dimensional sequence capturing the power of original audio signal, see Figure 1b;

SISAP'10 September 18–19, 2010, Istanbul, Turkey

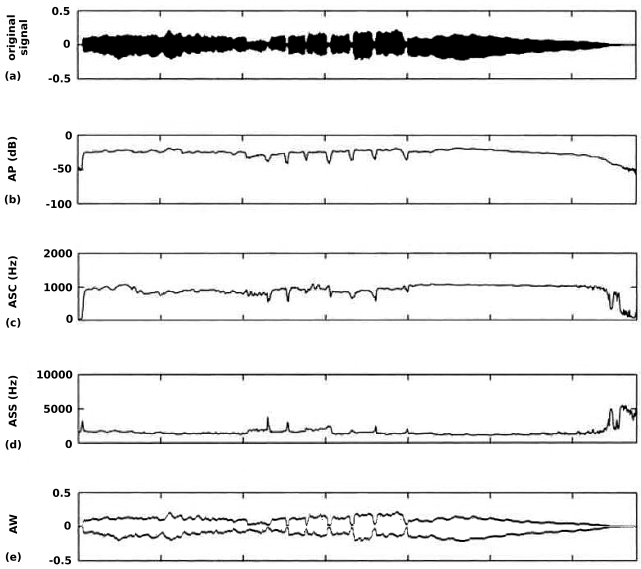


Figure 1: Example of the used MPEG-7 audio descriptors: (a) original 44.1 kHz sampled audio signal; (b) Audio Power; (c) Audio Spectrum Centroid; (d) Audio Spectrum Spread; (e) Audio Waveform.

- Audio Spectrum Centroid (ASC) – one-dimensional sequence which capture the central frequency in original audio signal, see Figure 1c;
- Audio Spectrum Spread (ASS) – one-dimensional sequence which capture the frequency spread in original audio signal, see Figure 1d;
- Audio Waveform (AW) – two-dimensional sequence capturing the minimal and maximal value of amplitude of original audio signal, see Figure 1e.

3. SIMILARITY MEASURE

A metric space $\mathcal{M} = (\mathcal{U}, d)$ is defined by a domain $\mathcal{U}$ of objects and a metric function $d : \mathcal{U} \times \mathcal{U} \rightarrow \mathbb{R}$ that determines the (dis)similarity between pairs of objects. In our case, the domain consists of particular audio descriptors extracted from the original audio recordings. The Euclidean distance L_2 was used as a metric function for AP, ASC and ASS descriptors. AW descriptor can be viewed as two one-dimensional sequences – one for minimum values and one for maximum values of amplitude, see Figure 1e. L_2 distances

are computed for both one-dimensional sequences separately and arithmetic mean is used as a final distance.

We have used the following data concept [3] in order to be able to search subsequences in the audio database. Each audio descriptor sequence in the database is split into overlapping subsequences of constant length w using the sliding window approach. Subsequences are associated with an offset k determining their position in the original sequence. M-Index is used for indexing all these subsequences [4].

The query sequence is split into disjoint subsequences of the same length w associated with index l (tail of the query sequence shorter than w is ignored). Separate nearest neighbor queries (kNN) are processed for each of these subsequences. The results of these queries consist of matching pairs of query-data subsequences that form a candidate set of possibly similar audio recordings.

This candidate set has to be refined. For each matching pair, we have corresponding offsets k and l determining how the query sequence matches the data sequence. Final similarity is computed as the distance between the longest overlapping subsequences of the query and the matching data sequence and is normalized by the number of pairs of samples in the overlapping subsequences.

Such indexing structure was separately created for all the mentioned audio descriptors using L_2 metric as well as for aggregation 1:1:1:1 of them. Aggregation function is computed as a weighted sum of all audio descriptors.

4. USER INTERFACE

A web application integrated in the MUFIN project is used as a user interface to the engine. The main web page is by default loaded with a random selection of audio recordings from the database. An audio recording can be played directly in the web browser, downloaded, or user can search for similar recordings. The kNN search is realized as described above and the results are displayed. User can also uploaded their own query audio recording.

A prototype version of this engine is running on a standard PC and is available at http://mufin.fi.muni.cz/audio.

5. DATA COLLECTIONS

Data collections from servers Partners In Rhytme[1] and The Freesound Project[2] were used. Audio recordings were divided into the following sets according their origin – Animals, House, Human, Instruments, Machines, Misc, and Natural. The whole database contains approximately 1,000 audio recordings. The four mentioned audio descriptors were extracted from each audio recording.

In Figure 2, we can see results of a $6NN$ query from the Animal collection (bird singing) for individual audio descriptors and aggregation 1:1:1:1 of all four descriptors. Using the AP descriptor, engine did not find any relevant result (the evaluation was done by human judges). Using the ASC, one relevant result was found, the ASS found two results, and the AW one result. Using the aggregation of all the descriptors, the results were all relevant.

All performed experiments showed that any audio descriptor is not sufficient itself. Different audio descriptors are suitable for different types of queries but aggregation of all descriptors gives at least as good results as usage of the most suitable audio descriptor separately. In the future, we would like to extend our audio database and add support for melody-based similarity search using high-level descriptors.

[1]http://www.partnersinrhytme.com/

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

Query: Animals\birdies\59360_dobroide_birdies.01.wav		
AP	Animals\birdies\59360_dobroide_birdies.01.wav	0.0
	Human\girl_Cough\45578_J.Zazvurek_Girl_Cough.wav	0.00136
	Human\Human_voice_Clock\80300_Corsica_S_04.wav	0.00147
	Human\Human_voice_Clock\80301_Corsica_S_05.wav	0.00152
	Human\Human_voice_Clock\80304_Corsica_S_08.wav	0.00153
	Human\Human_voice_Clock\80298_Corsica_S_02.wav	0.00154
ASC	Animals\birdies\59360_dobroide_birdies.01.wav	0.0
	Animals\Snake\snakehiss2.wav	0.03725
	Misc\Sword\1467_lostchocolatelab_10SWORD04.aif	0.04071
	Animals\birdies\59361_dobroide_birdies.02.wav	0.0418
	Misc\Sword\1453_lostchocolatelab_06SWORD06.aif	0.04201
	Misc\Sword\1464_lostchocolatelab_10SWORD01.aif	0.04332
ASS	Animals\birdies\59360_dobroide_birdies.01.wav	0.0
	Natural\AoE2\tf3.wav	0.02073
	Human\Baby_Noises\62070_NoiseCollector_vanessa10.wav	0.02076
	Animals\birdies\59373_dobroide_birdies.14.wav	0.02338
	Misc\Sword\1453_lostchocolatelab_06SWORD06.aif	0.02354
	Human\Baby_Noises\62079_NoiseCollector_vanessa8.wav	0.02397
AW	Animals\birdies\59360_dobroide_birdies.01.wav	0.0
	Human\Human_voice_Clock\80300_Corsica_S_04.wav	0.02293
	Animals\birdies\59376_dobroide_birdies.17.wav	0.02393
	Human\Human_voice_Clock\80301_Corsica_S_05.wav	0.02439
	House\Metal_Pan\16373_JonathanJansen_Metaal_1.wav	0.026
	House\Metal_Pan\16383_JonathanJansen_Metaal_19.wav	0.026
1:1:1:1	Animals\birdies\59360_dobroide_birdies.01.wav	0.0
	Natural\AoE2\tf4.wav	0.10712
	Animals\birdies\59361_dobroide_birdies.02.wav	0.11945
	Animals\birdies\59370_dobroide_birdies.11.wav	0.11964
	Natural\AoE2\tf3.wav	0.121
	Animals\birdies\59363_dobroide_birdies.04.wav	0.12325

Figure 2: Example of 6NN query of bird singing for individual audio descriptors and their aggregation 1:1:1:1 together with distances from the query.

6. ACKNOWLEDGMENTS

This work has been supported by national research projects MSMT 1M0545, GACR 201/08/P507, GACR 201/09/0683, GACR 103/10/0886, and GACR P202/10/P220.

7. REFERENCES

[1] M. Batko, V. Dohnal, D. Novak, and J. Sedmidubsky. Mufin: A multi-feature indexing network. In *Proceedings of SISAP '09*, pages 158–159, Washington, DC, USA, 2009. IEEE Computer Society.

[2] M. Batko, D. Novak, and P. Zezula. MESSIF: Metric similarity search implementation framework. In *First International DELOS Conference, Pisa, Italy, Revised Selected Papers*, volume 4877 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2007.

[3] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. *SIGMOD Rec.*, 23(2):419–429, 1994.

[4] D. Novak and M. Batko. Metric index: An efficient and scalable solution for similarity search. In *Proceedings of SISAP '09*, pages 65–73, Washington, DC, USA, 2009. IEEE Computer Society.

[5] P. Salembier and T. Sikora. *Introduction to MPEG-7: Multimedia Content Description Interface*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

Improving the Image Retrieval System by Ranking

Petra Budikova
budikova@ics.muni.cz

Michal Batko
batko@fi.muni.cz

Pavel Zezula
zezula@fi.muni.cz

Faculty of Informatics, Masaryk University, Brno, Czech Republic

Keywords

image retrieval system, ranking, relevance-feedback

1. INTRODUCTION

With the rapid growth of multimedia data, a lot of attention has been recently devoted to the development of multimedia retrieval systems. The research has followed two main directions: The first one applies existing text-search mechanisms to retrieve multimedia data based on its descriptive annotations, the second approach retrieves data by content. In case of text-based searching, the quality of results depends on the quality of text metadata, which is often not very high (especially in large general-purpose collections such as web image galleries). In the content-based approach, data objects are indexed and searched using features extracted from the data that describe their important characteristics. However, this solution suffers from the well-known *semantic gap* problem, i.e. the discrepancy between the similarity as computed using the descriptors and human understanding of similarity.

In our approach, we propose to bridge the semantic gap by combining both the orthogonal views. This method has already been proved to be very successful in the text-based searching – some of the major search engines (Google[1], Bing[2]) recently launched a new type of searching based on visual similarity of images. Both solutions exploit visual ranking of search results acquired by text retrieval [1, 5]. Result post-processing has also been employed in some content-based strategies to filter out less interesting objects from the result, usually by means of result clustering [4]. However, the existing content-based approaches do not employ additional measures of similarity. In our system, we provide a novel solution for large-scale content-based retrieval which enables to obtain high quality results by incorporating several measures of similarity into the search process.

[1]http://images.google.com/
[2]http://www.bing.com/images

SISAP'10 September 18–19, 2010, Istanbul, Turkey

2. RANKING

Ranking is often considered an integral part of the search process – search engines deliver ranked results. However, we model searching as a two-phase process, as depicted in Figure 1. During the initial search, suitable candidates are selected from the dataset and submitted to the ranking phase, where the more relevant objects from the candidate set are pushed to the top of the list. The ranking can be done either automatically, using the properties of candidate objects and statistics, or in cooperation with users that can actively participate in the process of defining the ranking function.

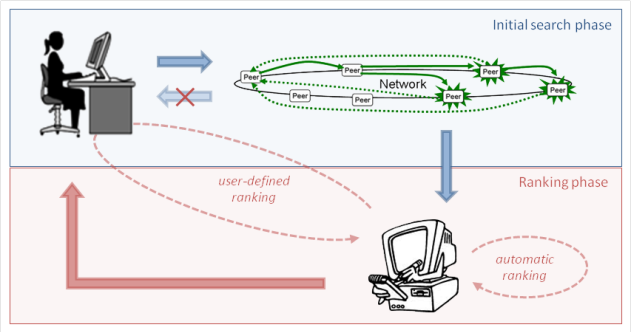


Figure 1: The two-phase search schema

In our scenario, we use image descriptors that form a metric space $\mathcal{M} = (\mathcal{D}, d)$. The initial search $F_{initial}$ is performed by any standard metric search query operation, e.g. the k-nearest neighbor search. In the ranking phase, a function $F_{rank} : \mathcal{D} \mapsto \mathbb{N}$ is applied on the result of $F_{initial}$ to establish a new rank of each object. The ranking function depends on the context in which it is evaluated and its computation may contain additional context-derived parameters.

Even though a user is interested in the first k objects, with k typically ranging from 10 to 100, the initial search should provide significantly more objects in order to allow the ranking to show interesting new data. The choice of the initial result size k' needs to balance the following three factors: the costs of the initial search for k' best objects, the cost of ranking the k' objects, and the probability that there are at least k relevant objects in the initial result of size k'.

We define several different types of ranking functions that are orthogonal to the content-based similarity. As our target data are images from a commercial microstock site with rich annotations, the ranking mainly exploits the text information. However, any other metadata such as time, location or popularity of a given object could be used as well.

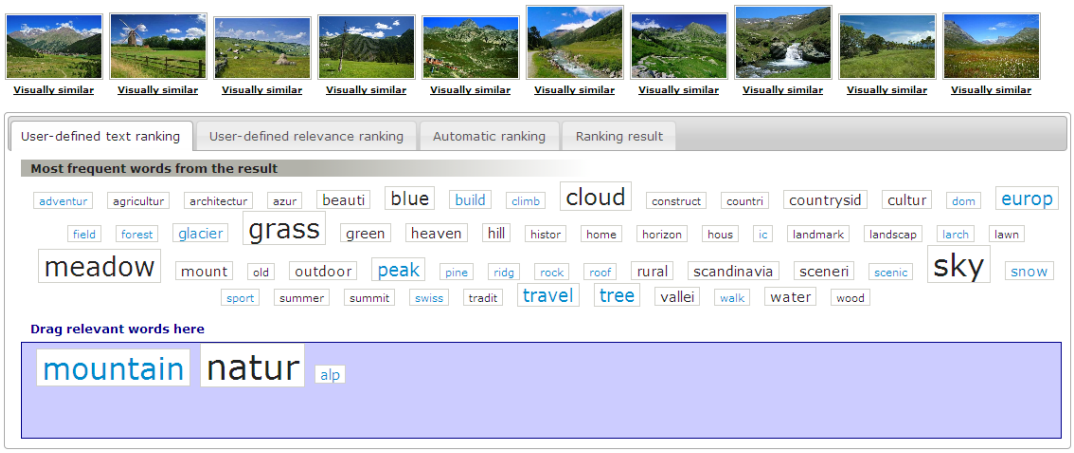


Figure 2: Interface of the demonstrated application

Keyword ranking Inversely to the search model applied by the common web search engines that combine text-based retrieval and visual ranking, we propose to rank the content-based search result with respect to keywords of the query image. The similarity between two sets of keywords is measured by the Jaccard coefficient.This ranking method is intended for data with rich and reliable annotations and can be enhanced using WordNet semantic relationships.

Word cloud ranking For data with sparse and erroneous text metadata, the keyword ranking is not applicable. In this case, we exploit the keywords of all objects in the initial result and compute their frequencies. We call the resulting set of keywords and frequencies the *word cloud*. Finally, the ranking employs the most frequent words from the cloud.

Combined visual and text ranking In the previous methods, we have only used the textual (keyword) information for the ranking, ignoring the initial ranking of the visual (content-based) search. However, it may also be useful to include it into the final ranking – we provide both the keyword and word cloud variants.

Selected descriptors ranking In the initial search, similarity of images is evaluated using a combination of several visual descriptors which refer to image's color, shape and texture. In the ranking phase, it is possible to choose the most important descriptor.

Relevance feedback ranking We propose to implement the relevance feedback as a ranking function. Users choose relevant objects from the initial result and the ranking function defines the final rank as a function on the content-based similarity to each of the objects marked as relevant.

User-defined keyword ranking Keywords may provide a strong ranking tool but automatic approaches may not always guess the optimal set of words. This method allows users to define the relevant keywords themselves.

3. APPLICATION

We demonstrate our solution over a dataset of commercial microstock photos with systematic annotations, which contains 8.3 million images. The content-based similarity of images is defined as a combination of several MPEG-7 descriptors [2] and each image is annotated by about 25 keywords on average.

We use an instance of the Multi-Feature Indexing Network (MUFIN) [3] for the initial search. The capabilities of MUFIN allow us to retrieve the results very fast even for large collections of data and its interfaces made it possible to plug-in the proposed ranking algorithms seamlessly into the web user interfaces. For each query image, 200 nearest images are submitted to the ranking phase and processed by a selected ranking function. The 10 most relevant objects are then shown to the user.

The web interface of the application (see Figure 2) enables easy selection of the preferred ranking function and a simple drag-and-drop marking of the relevant images or keywords in case of user-defined ranking. Users are then shown both the initial result and the ranked one. The application is available at **http://mufin.fi.muni.cz/ranking/**.

Acknowledgments

This work has been partially supported by the national research projects GACR 201/08/P507, GACR 201/09/0683 and by Brno PHD Talent Scholarship. The hardware infrastructure was provided by the METACentrum under the research intent MSM6383917201.

4. REFERENCES

[1] Y. Jing and S. Baluja. VisualRank: Applying PageRank to large-scale image search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1877–1890, 2008.

[2] MPEG-7. Multimedia content description interfaces. Part 3: Visual. ISO/IEC 15938-3:2002, 2002.

[3] D. Novak, M. Batko, and P. Zezula. Generic similarity search engine demonstrated by an image retrieval application. In *SIGIR '09*, page 840, 2009.

[4] R. H. van Leuken, L. Garcia, X. Olivares, and R. van Zwol. Visual diversification of image search results. In *WWW '09*, pages 341–350, New York, NY, USA, 2009. ACM.

[5] L. Wang, L. Yang, and X. Tian. Query aware visual similarity propagation for image search reranking. In *ACM Multimedia '09*, pages 725–728, New York, NY, USA, 2009. ACM.

Visual Video Retrieval System Using MPEG-7 Descriptors

Vojtěch Zavřel
Faculty of Informatics
Masaryk University
Brno, Czech Republic
xzavrel@fi.muni.cz

Michal Batko
Faculty of Informatics
Masaryk University
Brno, Czech Republic
batko@fi.muni.cz

Pavel Zezula
Faculty of Informatics
Masaryk University
Brno, Czech Republic
zezula@fi.muni.cz

Keywords

video retrieval system, content-based searching, metric space

1. INTRODUCTION

Video is one of the most popular media these days. However, the volume of the video content grows very fast, e.g. about 24 hours of video content is uploaded every minute to the most famous web site YouTube, and the necessity to search in this content is apparent. Generally, most of the video search systems are based on annotations or use additional text information, which is not always of high quality or lack precision. We present a system that allows user to search videos according to their visual content.

2. VIDEO SEARCHING BY CONTENT

There are several searching techniques commonly used by most of the video retrieval systems. Some of them are systems fully dependent on the text located near to the video like Google. Others (like YouTube) are catalog retrieval systems that usually use some additional information like description, categorization, relevance, relationships, comments and so. Such information can be used when looking for similar videos.

Different systems sometimes called *content based* (e.g. ANVIL[1], VARS[2] or VCode & VData[3]) use annotations to describe individual scenes by visual, audio, time and meta parameters. All those types of systems are highly dependent on the quality of the additional data, which is normally prepared by people [6].

Another possibility is to use automatic annotation and retrieval systems that use different techniques to build the describing information. Some of them are based on MPEG-7 descriptors or machine learning. Those are often restricted to a specific type of use – they are sometimes called *domain specific systems* [2, 4]. Others (e.g. SAPIR[4]) use metric space based similarity search principles where searching is based on an example query [3]. Basic idea is to search not on the level of the actual multimedia object but rather using characteristic features extracted from these videos. Features can be implemented as multidimensional vector-space descriptors and the similarity of the video content is then defined using these descriptors.

Although video contains several information carriers like image, audio, text, or time, for many people the image is still the most important part of the video. Therefore we demonstrate a system that allows users to search according an example video or a video scene. Visual part of the video in our system is represented by *keyframes*, i.e. still images from the video that best characterize a fraction of the video – typically one camera shot.

2.1 Keyframes extraction

We have prepared our own tool for keyframe extraction following the ideas of MPEG-7 standard's video summarization technique. This tool is based on MPEG-7 reference implementation library (eXperimental model) and its Summarization Client application [5].

The tool browses all the frames in the source video file and computes the frame change level according the specified precision parameters. Whenever a significant change is detected (and the given parameters are satisfied), a keyframe is extracted. Thus, each video is split into a sequence of keyframe images.

For just one minute of the video (even static) it is necessary to process 1500 frames (PAL format has 25 fps). The keyframe extraction allows us to reduce this number to 1-10% depending on the dynamicity of the source video. On our collection, we have observed about 4% reduction on average. It is still quite high amount of images, but using a scalable image retrieval system such as MUFIN [1], we have a potential to index millions of short videos. Moreover, the extraction parameters can be adjusted to further lower the ratio at the cost of losing some precision.

2.2 Keyframes indexing

To define the similarity between the videos (or the respective video scenes), we use the extracted key frames. Five MPEG-7 global visual descriptors – color structure (CS), color layout (CL), scalable color (SC), edge histogram (EH), and homogeneous texture (HT) [5] – are taken from each key frame. The dissimilarity of two key frames is defined as a

[1]http://www.anvil-software.de/
[2]http://sourceforge.net/projects/vars/
[3]http://social.cs.uiuc.edu/projects/vcode.html
[4]sapir.insti.cnr.it/index

SISAP'10 September 18–19, 2010, Istanbul, Turkey

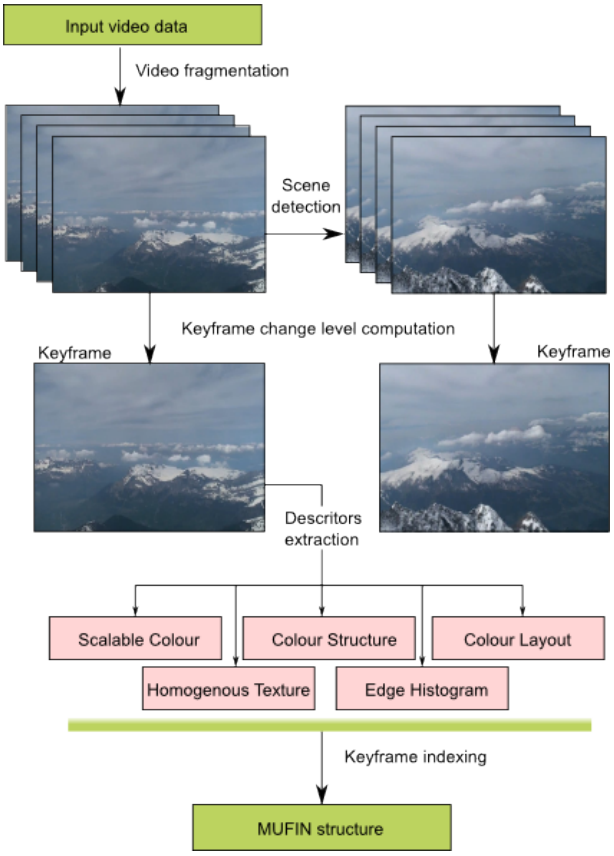


Figure 1: System architecture

weighted sum of the distance of each of the five descriptors in both the key frames:

$$d(o_1, o_2) = 3 * d_{CS}(CS(o_1),CS(o_2))+ 2 * d_{CL}(CL(o_1), CL(o_2))+3 * d_{SC}(SC(o_1), SC(o_2))+ 4 * d_{EH}(EH(o_1), EH(o_2))+0.5 * d_{HT}(HT(o_1), HT(o_2))$$

Then, the similarity of two videos is defined as a Jaccard coefficient [7], where the cardinality of the intersection is the number of key frames that have a similar key frame in the other video. This is defined as the nearest-neighbor key frame that has a distance below a given threshold. Moreover, only images within a given time window are considered, so the similar scenes in different parts of the videos are not compared and also the number of distance computations was reduced. Both the parameters were set experimentally by testing the results on a small sample set.

2.3 Demo collection

Our test database consists of more than 2000 videos. During the extraction phase more than 4 million of image frames had to be processed. The indexed key frames collection is about 60 000 images. The whole process of extraction on a personal computer with Intel Core2 family CPU took 1275 hours of CPU time. The most time-consuming part of the process was the extraction of key frames that took about 1120 hours of CPU time and the extraction of visual descriptors lasted about 85 hours. The video resolution is mainly 720 x 576 pixels.

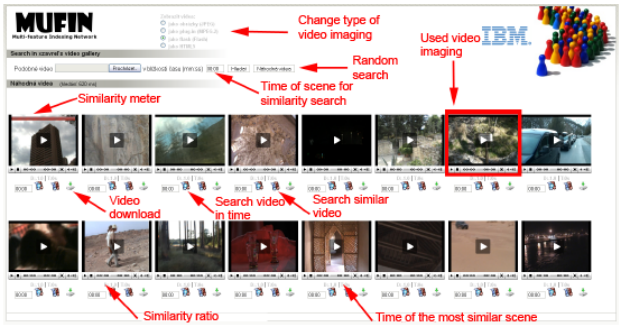


Figure 2: User interface example

2.4 Video visualization

Our system offers several ways of video visualization. The videos can be shown as images (timeline of key frames), as an embedded player, Flash player or HTML5 video tag.

Users have several possibilities of searching for videos. A user can upload his/her own video and search for a similar video as a whole or for a particular scene. Also a single image can be used instead of a whole video to search for key scenes. Then it is possible to use an iterative search of currently found videos - also based on a whole video or a single image specified using the video time. Finally, the interface allows to get a random selection of videos. The demo is available on webpage: **http://mufin.fi.muni.cz/videos**

3. ACKNOWLEDGMENTS

This work has been partially supported by the national research projects GACR 201/08/P507 and GACR 201/09/0683.

4. REFERENCES

[1] M. Batko, V. Dohnal, D. Novak, and J. Sedmidubsky. Mufin: A multi-feature indexing network. *Similarity Search and Applications, International Workshop on*, 0:158–159, 2009.

[2] A. Dorado, J. Calic, and E. Izquierdo. A rule-based video annotation system. *Circuits and Systems for Video Technology, IEEE Transactions on*, 14(5):622 – 633, may 2004.

[3] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: The qbic system. *Computer*, 28(9):23–32, 1995.

[4] Y. Liu and F. Li. Semantic extraction and semantics-based annotation and retrieval for video databases. *Multimedia Tools Appl.*, 17(1):5–20, 2002.

[5] B. S. Manjunath, P. Salembier, and T. Sikora, editors. *Introduction to MPEG-7: Multimedia Content Description Language*. Wiley, April 2002.

[6] H. Miyamori and S. ichi Iisaku. Video annotation for content-based retrieval using human behavior analysis and domain knowledge. *Automatic Face and Gesture Recognition, IEEE International Conference on*, 0:320, 2000.

[7] P. Zezula, G. Amato, V. Dohnal, and M. B. atko. *Similarity Search: The Metric Space Approach*, volume 32 of *Advances in Database Systems*. Springer-Verlag, 2006.

Sub-image Searching Through Intersection of Local Descriptors

Tomas Homola
Faculty of Informatics
Masaryk University
Botanicka 68a, 602 00 Brno
Czech Republic
xhomola@fi.muni.cz

Vlastislav Dohnal
Faculty of Informatics
Masaryk University
Botanicka 68a, 602 00 Brno
Czech Republic
dohnal@fi.muni.cz

Pavel Zezula
Faculty of Informatics
Masaryk University
Botanicka 68a, 602 00 Brno
Czech Republic
zezula@fi.muni.cz

1. INTRODUCTION

The complexity of search in current business intelligence systems, academic research, or even the home audiovisual databases grows up rapidly. Users require searching by the content of their data. For example, the user sees a cathedral while watching a movie and by taking a snapshot, his or her private collection of holiday photos can be searched for images containing that cathedral, as demonstrated in Figure 1. In practice, it is not sufficient to store data and search in it by exact match but rather by means of similarity, i.e. retrieving data items similar to a query item. Similarity searching is especially requested in multimedia databases, digital right management systems, computer aided diagnosis, but also in natural sciences and psychology. In these fields, theoretical primal background for querying by similarity is already defined.

In this demonstration, we focus on similarity searching from the perspective of identifying and locating an object depicted in a query image in a collection of images. In particular, we have implemented an algorithm that is based on local affine-invariant descriptors and that is capable of retrieving database images containing the object in the query image. The algorithm's capabilities are demonstrated on a collection consisting of 15,337 company logos from which 2×10^6 SIFT local descriptors were extracted.

2. SUB-IMAGE SEARCH ALGORITHM

Local descriptors, such as SIFT [5], extracted from images usually consist of three distinguished parts, called a *locator*, *descriptor* and *scale*. To avoid misunderstandings, we denote these local descriptors (SIFTs) as *features*. The locator defines the position of the feature within the image, while the descriptor itself summarizes the position's proximity in terms of color, texture, edges, etc. The scale attribute represents the importance of the feature (e.g., the spatial size of the surroundings the descriptor characterizes). This leads to the idea that for performance reasons we do not need to check all features extracted from an image, but for many applications it is sufficient to inspect several most-important ones to obtain high recall. Spurious false positives can later be filtered out using the complete sets of features. Finally, each feature has also a *unique identification* of the corresponding image.

In the following, we introduce an algorithm for locating a query object in the database. A formal specification of this algorithm has been proposed in [3] but for the needs of this demonstration it has been modified for performance reasons. The pseudo-code in Alg. 1 specifies the procedure of locating a query Q in database images. After extracting the most-important features from Q, several range queries with the radius ε (Line 4) are executed to obtain a candidate set. Notice that throughout the algorithm we use pairs of features, because the spatial correspondence of local features in the query and a database image is important for final ranking (in our case) or for filtering out non-matching images. After removing 'duplicate' pairs (more pairs between the same images), candidates containing fewer pairs of features than $limit$ get eliminated. Each set in the result corresponds to a database image and it contains the pairs of matching features. So, by taking the image's features from the pairs, the specific location of the query in the image is formed by computing a bounding rectangle over these features' positions. The final ranking is done on Lines 17-23 by checking the order of features in both the coordinates (x, y) independently.

3. DEMONSTRATION

We demonstrate the effectiveness of Algorithm 1 on a collection of $15,337$ company logos from which on average 152 local features (SIFT) were extracted. Thus, $2,359,839$ features were obtained in total. These features along with original image IDs were organized in the M-tree [2], so the range queries (see Line 4 of Alg. 1) were evaluated efficiently. Next, a sequential file ordered by image ID was created to store other meta-data, such as filename, about all the images. The algorithm is implemented in Java 6 with the implementation of M-tree available at http://mufin.fi.muni.cz/trac/mtree/. The demo is available at **http://mufin.fi.muni.cz/subimages/** and it is run on a single server equipped with two quad-core Intel Xeon 2 GHz CPUs, 14 GiB RAM and a six-disk RAID5 array. To speed up the retrieval in the M-tree, eight queries were executed concur-

Figure 1: Example of a sub-image searching for Sacré-Cœur cathedral.

SISAP '10, September 18-19, 2010, Istanbul, Turkey.

Algorithm 1 Locating Sub-images `locateQuery`

Input: a query image Q, a number nof of important features, a threshold ε, and a limit on the number of features ($limit$).
Output: $Result$ consisting of images containing the query.

1: $Result = \emptyset$ and $Temp = \emptyset$.
2: $F_Q = \{q_1, \ldots, q_{nof}\}$. // Extract local features from query
3: **for all** $q_i \in F_Q$ **do**
4: $RQ = rangeSearch(q_i, \varepsilon)$.
5: Add the pairs (q_i, f_k) for each $f_k \in RQ$ to $Temp$.
6: **end for**
7: // Group the pairs in $Temp$ by image identification of f_ks:
8: $Result = \{G_{\mathbb{E}_1}, \ldots, G_{\mathbb{E}_n}\}$, where $G_{\mathbb{E}_i} = \{(q_v, f_w), \ldots, (q_x, f_y)\}$ and all f_s have the same image ID.
9: **for all** $G_{\mathbb{E}_j} \in Result$ **do**
10: **for all** $q_i \in F_Q$ **do**
11: $G = \{(q_i, f_x) | (q_i, f_x) \in G_{\mathbb{E}_i}\}$. // Take pairs with q_i
12: Remove G from $G_{\mathbb{E}_i}$.
13: Add (q_i, f_k) to $G_{\mathbb{E}_i}$, where $(q_i, f_k) = argmin_{(q_i, f_x) \in G}(distance(q_i|\text{Desc}, f_x|\text{Desc}))$.
14: **end for**
15: **end for**
16: Remove all $G_{\mathbb{E}_i}$ from $Result$ which contain $< limit$ pairs.
17: **for all** coordinates α of the position coordinates (x,y) **do**
18: **for all** $G_{\mathbb{E}_i} \in Result$ **do**
19: Sort the pairs $(q_v, f_w) \in G_{\mathbb{E}_i}$ by the coordinate α of f_s.
20: Compute the α-rank of $G_{\mathbb{E}_i}$ // Number of swapped features w.t.r. the query (ORD function in [3])
21: **end for**
22: **end for**
23: Sort $Result$ by $\frac{\sum_\alpha \alpha\text{-rank of } G_{\mathbb{E}_i}}{nof}$.

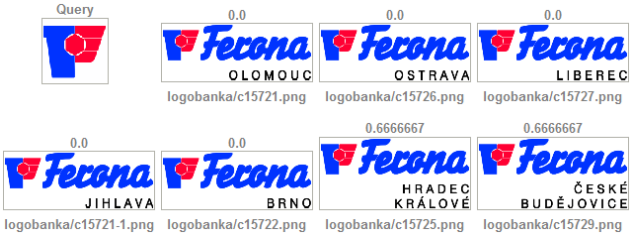


Figure 2: Result of search for the example logo (left-top image).

rently, so all cores were loaded. Figure 2 depicts a retrieval example, where the numbers above the images depict their dis-similarity to the query (as computed on Line 23 of Alg. 1). Below the images, there are the corresponding file names, which exemplifies the fact that no textual information was used during retrieval. As for the algorithm's parameters, we used $\varepsilon = 250$, $limit = 6$ and $nof = 18$, where these values were empirically obtained as good ones. In the demonstration, these parameters can be modified. The average query execution time is below 3 seconds, where nearly 2 seconds. are spent in the M-tree by evaluating all 18 queries and 1 second is needed to compare the feature pairs and ranks the images obtained.

4. RELATED WORK

There are many systems for content-base image retrieval, for example MUFIN, ALIPR, ImBrowse, idée and Gazopa. However, these systems focus on searching for images similar to a query image as whole, thus searching for sub-images is not supported.

Approaches to search for sub-images are based on locality sensitive hashing or encoding image features to visual words, which is hashing in fact too. Many approaches, e.g. [4], compare only the hash codes of features' descriptors without tackling their spatial distribution, which leads to false positives in the result set. The Geometric min-Hash (GmH) [1] converts image features to visual words too, but it extends the resulting hash code by including also image features in close neighborhood. From the searching point of view, it identifies regions in an image in which the identical or almost-identical groups of features with respect to the query image occur. It eliminates the features that are encoded to one visual word, which can cause problems when processing repeating patterns or when the queried sub-image occurs in the original image multiple times. By analogy to our approach, GmH optimizes the retrieval process by selecting some important features (anchors). A disadvantage of GmH is that only small spatial surrounding is considered instead of taking into account spatial order of features. So, it is limited to small query images only.

In contrary to the approaches mentioned above, the authors of [6] does not use local descriptors for the searching, but MPEG-7 global features are rather taken. In particular, database images as well as the query image are segmented to chunks of pre-defined size and the global features are extracted. Searching procedure then finds correspondence between the database images and the query image.

5. CONCLUSIONS

By exploiting the spatial information (locator property) of local image features, an effective algorithm for searching for objects contained in the query image has been implemented. An online application demonstrates the properties of this algorithm on a collection of 15 thousand images of company logos from which more than 2 million SIFT local features were extracted. The system exhibits query response time under three seconds while it operates on a moderate HW infrastructure.

Acknowledgements

This research was supported by Czech Science Foundation – projects GACR P103/10/0886, GACR 201/09/0683 – and a national project MUNI/A/0922/2009. The access to the METACentrum supercomputing facilities provided under the research intent MSM-6383917201 is highly appreciated too. We used this infrastructure for feature extraction. We also thank the LogoOpen company for providing the logo data set.

6. REFERENCES

[1] O. Chum, M. Perdoch, and J. Matas. Geometric min-hashing: Finding a (thick) needle in a haystack. *CVPR*, pages 17–24, 2009.

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

[3] T. Homola, V. Dohnal, and P. Zezula. Proximity-based order-respecting intersection for searching in image databases. In *Proceedings of the* 8^{th} *International Workshop on Adaptive Multimedia Retrieval (AMR 2010)*, 2010.

[4] Y. Ke, R. Sukthankar, L. Huston, Y. Ke, and R. Sukthankar. Efficient near-duplicate detection and sub-image retrieval. In *In ACM Multimedia*, pages 869–876, 2004.

[5] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 2004.

[6] M.-S. Ryu, S.-J. Park, and C. S. Won. Image retrieval using sub-image matching in photos using mpeg-7 descriptors. In *AIRS*, pages 366–373, 2005.

Posters

SISAP 2010

On Applications of Parameterized Hyperplane Partitioning

Jakub Lokoč, Tomáš Skopal
SIRET Research Group
Department of Software Engineering, Faculty of Mathematics and Physics, Charles University in Prague
Malostranské nám. 25, 118 00 Prague, Czech Republic
http://siret.ms.mff.cuni.cz

ABSTRACT

The efficient similarity search in metric spaces is usually based on several low-level partitioning principles, which allow filtering of non-relevant objects during the search. In this paper, we propose a parameterizable partitioning method based on the generalized hyperplane partitioning (GHP), which utilizes a parameter to adjust "borders" of the partitions. The new partitioning method could be employed in the existing metric indexes that are based on GHP (e.g., GNAT, M-index). Moreover, we could employ the parameterizable GHP in the role of a new multi-example query type, defined as a partition determined by an available query object and several "anti-example" objects. We believe that both applications of parameterizable GHP can soon take place in metric access methods and new query models.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: [Retrieval models]

1. INTRODUCTION

The similarity search using the metric space model proved to be a general approach applicable in various domains. A database $\mathbb{S}$ is supposed to consist of unstructured raw data, while the only available information is a metric distance δ defined for each pair of objects from $\mathbb{S}$. From the database management point of view, the search performance is crucial, hence indexes allowing efficient filtering are necessary. During the last two decades, there emerged many metric access methods (MAMs) [1], [3] that utilize metric postulates to filter non-relevant (sets of) objects during query processing. One of the most fundamental rules used in the design of efficient MAMs is *metric partitioning*, which divides the database into separate classes of similar objects. The general metric space model offers two basic types of partitioning – the *ball partitioning* and the *generalized hyperplane partitioning* (GHP). The ball partitioning employs a selected object (so-called *pivot*) and a radius, dividing the space in two partitions (e.g., in M-tree) or more, when a combination of balls is used (e.g., in MVP-tree or PM-tree). The GHP uses up to n pivots to divide the space into n voronoi-like partitions, where the i^{th} partition consists of objects that are closer to the i^{th} pivot than to any other pivot (e.g., in GNAT or M-index). In this paper, we propose a parameterized extension of the GHP, which employs a parameter to adjust "borders" of the partitions. We also prove several lemmas necessary for correct filtering rules.

SISAP 2010, September 18-19, 2010, Istanbul, Turkey.

2. PARAMETERIZED GENERALIZED HYPERPLANE PARTITIONING

For the lack of the space, we will define partitioning just for two pivots, but the definition and all the presented lemmas can be simply extended for an arbitrary number of pivots.

DEFINITION 1. *The parameterized generalized hyperplane partitioning (pGHP) using two pivots a, b and a parameter c divides a database $\mathbb{S}$ into two subsets A, B as follows:*

$$A = \{x | x \in \mathbb{S} \wedge \delta(a,x) < \delta(b,x) + c\}$$
$$B = \{x | x \in \mathbb{S} \wedge \delta(a,x) \geq \delta(b,x) + c\}$$

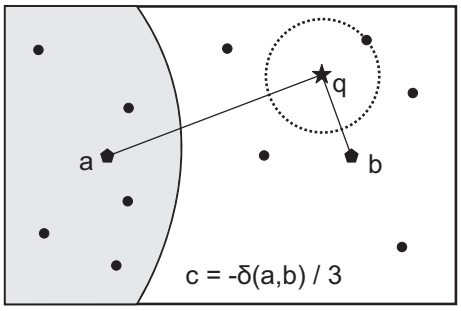


Figure 1: Parameterized generalized hyperplane partitioning.

Since the ball-region is the most frequent shape of a query, we also propose a lemma giving the relation between pGHP and the ball region (see also Figure 1).

LEMMA 1. *Let (q, r) be a range query and a, b be pivots used for the pGHP with parameter c, then (a) if $\delta(a,q) - r \geq \delta(b,q) + r + c$ then $\forall x$ where $\delta(x,q) \leq r$ holds $\delta(a,x) \geq \delta(b,x) + c$ and (b) if $\delta(a,q) + r < \delta(b,q) - r + c$ then $\forall x$ where $\delta(x,q) \leq r$ holds $\delta(a,x) < \delta(b,x) + c$.*

PROOF. Let us start with variant (a). By combination of assumptions $\delta(x,q) \leq r$ and $\delta(a,q) - r \geq \delta(b,q) + r + c$ we obtain $\delta(a,q) - \delta(x,q) \geq \delta(b,q) + \delta(x,q) + c$. From the triangle inequality we also have $\delta(a,x) \geq \delta(a,q) - \delta(x,q)$ and $\delta(b,q) + \delta(x,q) + c \geq \delta(b,x) + c$. Thus, by combination of these formulas we obtain $\delta(a,x) \geq \delta(a,q) - \delta(x,q) \geq \delta(b,q) + \delta(x,q) + c \geq \delta(b,x) + c$ which implies $\delta(a,x) \geq \delta(b,x) + c$. The proof of the variant (b) is similar. □

As a direct consequence of Lemma 1, we obtain a filtering rule for searching by ball-shaped queries. The filtering behavior of pGHP is similar as for the original GHP. In Figure 2, the two dotted curves depict the centers of query balls with fixed radius that have to visit both partitions (two such query balls are depicted). Note that pruning near the line connecting a and b is more effective (the border thickness is here close to $2r$), however this property is not specific to pGHP. Even in the original GHP the pruning near to the "connecting line" is more effective due to tighter lower- and upper-bound distances to the points in the query ball.

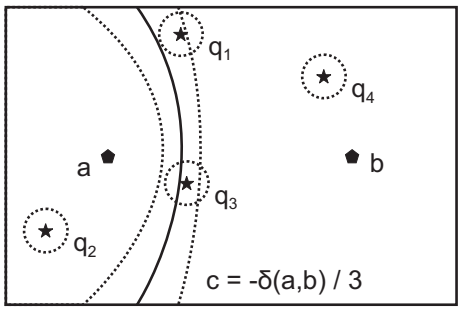


Figure 2: Filtering ability of pGHP

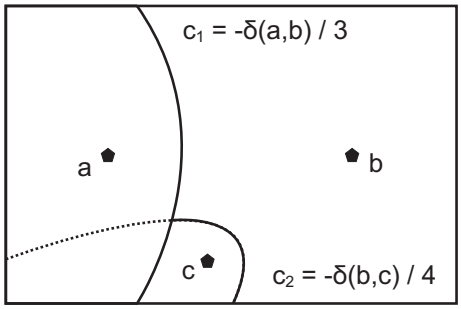


Figure 3: pGHP using multiple pivots

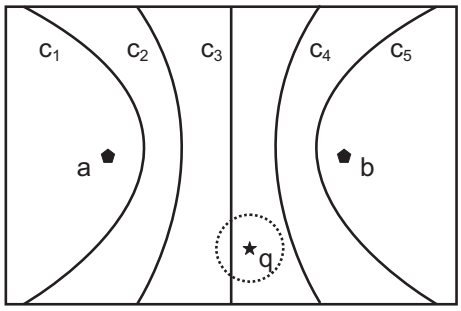


Figure 4: pGHP using multiple parameters

Unlike the original GHP, the pGHP utilization could be used for more flexible data partitioning.

1. We can dynamically adjust the parameter c in order to establish a balanced partitioning of data. The balancing is even more important if more than two pivots are used, e.g., in a future variant of GNAT based on pGHP. In Figure 3, the pGHP partitioning using pivots a, b, c was two-step, first the space was partitioned between a and b, while the partition b was further divided into partitions b and c.
2. We can employ more parameters c_i to define more partitions (as depicted in Figure 4). By the use of multiple parameters we can define multiple partitions, however, the interval of useful values c is bounded, as shown in the following lemma.

LEMMA 2. *Let a, b be pivots used for the pGHP with parameter c, then (a) if $c \leq -\delta(a,b)$ then partition A is always empty and (b) if $c > \delta(a,b)$ then partition B is always empty.*

PROOF. We just substitute a border value for c in the partitioning rule and we immediately obtain the statement. □

Besides indexing, a completely new family of multi-example query types based on pGHP could be defined within the framework proposed in [2]. An example of such a query is depicted, again, in Figure 3. The query region/partition c could be interpreted such that we search for objects similar to c (the example) and not very similar to a, b (anti-examples). Note that efficient processing of the new query is easy when using MAMs based on ball-partitioning (e.g., M-tree). Instead of having a query ball and a pGHP-defined partition in index (e.g., a sort of pGHP-based GNAT), we just interchange the roles of query and index regions, using the same pGHP filtering.

Acknowledgments

This research has been supported in part by Czech Science Foundation project Nr. 201/09/0683 and by the grant SVV-2010-261312.

3. CONCLUSIONS

We have introduced pGHP, a parameterized version of GHP that can be utilized for definition of flexible hyperplane-based metric regions. The pGHP can be used to modify the existing MAMs based on the GHP. We can also define specific query regions using more pivots in the role of examples and anti-examples. All the presented ideas concerning indexing and querying are the topic of our future work.

4. REFERENCES

[1] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in Metric Spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[2] P. Ciaccia, M. Patella, and P. Zezula. Processing complex similarity queries with distance-based access methods. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT 98)*, pages 9–23, 1998.

[3] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach (Advances in Database Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

Efficient and Effective Similarity-based Video Retrieval*

Ilaria Bartolini
DEIS, University of Bologna, Italy
i.bartolini@unibo.it

Corrado Romani
DEIS, University of Bologna, Italy
corrado.romani@unibo.it

ABSTRACT

The retrieval of videos of interest from large video collections is a main open problem which calls for the definition of new video content characterization techniques in term of both visual descriptors and semantic annotations. In this paper, we present an efficient and effective video retrieval system which profitably exploits the functionalities offered by a semantic-based automatic video annotator using video shots similarity to suggest relevant labels for the videos to be annotated. Similarity queries based on semantic labels and/or visual features are implemented and experimentally compared on real data in order to measure the retrieval contribution of each type of video content information.

1. INTRODUCTION

Efficient and effective video retrieval is one of the most challenging tasks of Information Retrieval [6]. Some of the systems that have been proposed in the last few years rely only on visual features for indexing data (e.g., [9]), thus suffering the drawback coming from the semantic gap existing between the user subjectivity notion of similarity and the one implemented by the system. Video indexing based on semantic annotations [7] seems to be a better option for the user than visual features, although they are still used as post-filtering of the semantic result. Finally, some systems focus on a multi-modal retrieval paradigm [6] allowing both semantic concept- and feature-based retrieval.

In this paper we present an effective and efficient video retrieval system which conjunctively exploits the accuracy of automatically provided semantic-based hierarchical video annotations and the similarity between video scenes in terms of visual features in order to satisfy user expectations. The annotation process uses labels of pre-annotated key frames, following the key idea to suggest, for a given key frame, those tags which are assigned to key frames that are *similar* to it. In this way we are able to attach tags to video shots that we then propagate at the whole video level, so as to obtain semantic indices for a hierarchical (two-level) browsing platform. In addition to the above described query paradigms, we also support a combined query modality useful to enlarge the query results when semantic annotations are incomplete. Preliminary results on the TRECVID benchmark [10] provide evidence of the accuracy of the video annotation and the precision of the video retrieval.

*This work is partially supported by the CoOPERARE MIUR Project.

SISAP 2010, September 18-19, 2010, Istanbul, Turkey.

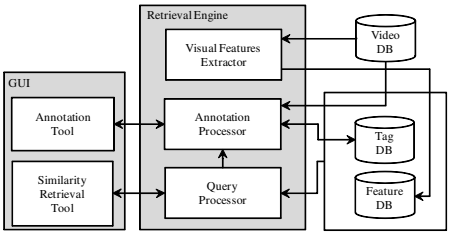


Figure 1: Retrieval architecture.

2. ARCHITECTURE AND PRINCIPLES

Figure 1 provides an overview of our video retrieval system. For each video in the *Video DB*, the *Visual Features Extractor* analyzes its shots and extracts a set of visual features from each key frame [3]. Each key frame is automatically segmented into a set of homogeneous regions which convey information about color and texture [2]. Extracted features are saved into the *Feature DB* and then indexed with an implementation of the M-tree metric index [4] to provide an efficient access and speed up the retrieval phase.

For each shot, the *Annotation Processor* assigns labels depending on its visual features. After processing all the shots, the annotator selects the most appropriate tags for the whole video and saves all the information into the *Tag DB*. An inverted file is maintained in order to guarantee an efficient tag-based retrieval. The tagging phase exploits the Imagination system [1] and uses a set of pre-annotated images as a knowledge base.

In details, given a key frame to be labelled, the annotator uses its visual features and, by exploiting the M-tree index, suggests those tags that are assigned to key frames in the knowledge base that are similar to the target one. In this way, a set of semantic concepts is attached to each representative key frame of a scene. Only terms recurring in the majority of key frames are selected as suitable concepts to describe the whole shot sequence. To avoid producing an overwhelming number of tags for each shot, only the most frequent tags retrieved for each key frame in the sequence are maintained. Shot tags are useful to browse sequences across different videos, but they could be too specific to in-

dex a whole video, especially if this contains a wide range of different visual content. A simple criterion to select video tags from the set of shot tags is to weigh every tag depending on its frequency and on the length of the shot it is associated to: concepts extracted from long shot sequences and/or that appear in several shots are probably more relevant, to describe the content of a whole video, than concepts occurring rarely or in short sequences.

At query time, the user submits her requests to the system; the *Query Processor* manages such requests in order to return the videos (respectively shots) of interest. Three main query paradigms are supported: *keyword*-based (*KS*), *feature*-based (*FS*) and *keyword&feaure*-based (*KFS*) searches, respectively. The *KS* paradigm is the easiest and most popular query modality used by traditional search engines, where the user enters a set of keywords as query semantic concepts. Videos/shots are selected by the query processor by applying a *co-occurrence* search on the tag DB. The search provides the set of videos (resp., shots) that share at least one tag with the input set. We rank the returned objects on the base of the co-occurrence value and return the top-k ones.

With the *FS* modality, the user is looking for those shots whose representative key frames are similar to an input query image. A *nearest-neighbors* (NN) search is performed on the Feature DB. Since a shot can be represented by more than a single key frame, we compute a k'-NN search, with ($k' \simeq 2k$) in order to derive a sufficient number of distinguishing shots. Candidate shots are ranked on the EMD distance we used to compare key frames [2] and the top-k are returned.

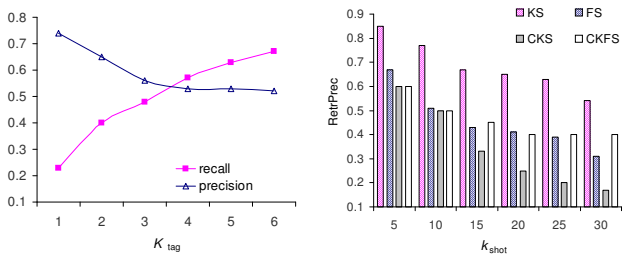
Finally, *KFS* queries combines *KS* and *FS* by returning shots in the intersection of both *KS* and *FS* results first, followed by shots in the *KS* list only and, finally, by shots in the *FS* result only. This query modality is particularly convenient when keyword-based search involves concepts which are poorly represented in knowledge base, i.e., shots annotated with that concept are less than k. In this case, by only applying *KS* we would not be able to return k shots. If *KS* is able alone to provide the desired number of objects, adding features in the query process is quite pointless since it just re-ranks the same result set.

3. EXPERIMENTAL VALIDATION

We implemented our retrieval system in Java JDK 6.0 and tested it on the TRECVID benchmarks [10]. We used the knowledge base of the *TRECVID-2007 High-Level Feature* task [10] which consists of 110 videos (50 hours total length), 21500 key frames, and 9000 shot cuts. Each shot is described by means of tags coming from 36 semantic concepts.

We first evaluate the accuracy of the annotator in term of classical *precision* (number of relevant retrieved tags over the returned labels) and *recall* (number of relevant retrieved tags over the total number of relevant tags for the shot) metrics over a set of provided testing videos which come with a ground truth for evaluation purposes. Then we tested the video retrieval effectiveness over a set of random selected queries (as described in the following) by means of the *retrieval precision* metric, i.e., the number of relevant shots returned over the number of returned shots.

Figure 2 (a) shows the average annotation precision and recall values, when varying the number of predicted tags. As one can observe, the annotator performs quite well, representing a good starting point upon which an effective video retrieval system can be built.



(a) (b)

Figure 2: Average accuracy of the annotator in terms of (a) precision and recall vs. number of predicted tags and (b) retrieval precision of the query processor vs. number of returned shots.

In Figure 2 (b) the average accuracy of the video retrieval vs. the number of returned shots is shown. The query workload consisted of 7 different queries: *outdoor scenes*, *scenes containing persons*, *faces*, *roads*, *sky views*, *vegetation*, and *waterscape/waterfront views*. We ran keyword-based (KS) similarity searches and evaluated the annotations of the top-k shots. Then we ran feature-based similarity search (FS) by computing the top-k shots which are most similar to the query image corresponding to each semantic query concept. KS performs very well (e.g., about 85% of precision for the first 5 shots and 65% when 25 shots are retrieved); the retrieval precision for FS is satisfactory, although quite worse than KS (as expected because of the semantic gap problem).

Finally, we tested the mixed query paradigm, by assuming a *poor* annotation scenario: the concept "Car" in our ground truth appears in 31 shots, but our annotator predicted it for 7 shots. We denote with CKS the "Car keyword-based search", while with CKFS the "Car keyword/feature-based search". It is immediate to derive from Figure 2 (b) the contribution of the visual features to CKS: CKFS is indeed able to enlarge the cardinality of the relevant results by providing 8 more correct matches, while maintaining acceptable level of precision values, especially when $k_{shot} > 10$.

4. REFERENCES

[1] I. Bartolini and P. Ciaccia. Imagination: Accurate Image Annotation Using Link-analysis Techniques. In *AMR 2007*, pages 32–44, 2007.

[2] I. Bartolini, P. Ciaccia, and M. Patella. Query Processing Issues in Region-Based Image Databases. In *Knowledge and Information Systems (KAIS)*, 2010. To appear.

[3] I. Bartolini, M. Patella, and C. Romani. SHIATSU: Semantic-Based Hierarchical Automatic Tagging of Videos by Segmentation using Cuts. In *AIEMPro 2010*. To appear.

[4] P. Ciaccia, M. Patella, and P. Zezula. M-tree: an Efficient Access Method for Similarity Search in Metric Spaces. In *VLDB 1997*, pages 426–435, 1997.

[5] C. Diou, G. Stephanopoulos, N. Dimitrou et al. VITALAS at TRECVID-2009. In *TRECVID 2009*, pages 16–17, 2009.

[6] P. Geetha, and V. Narayanan. A Survey of Content-based Video Retrieval. In *Journal of Computer Science*, 4(6), pages 474–486, 2008.

[7] A. G. Hauptmann, M. G. Christel, and R. Yan. Video Retrieval Based on Semantic Concepts. In *Proceedings of the IEEE*, 4(96), pages 602–622, 2008.

[8] C.-W. Ngo, Y.-G. Jiang, X.-Y. Wei et al. VIREO/DVMM at TRECVIDE 2009: High-Level Feature Extraction, Automatic Video Search and Content-Based Copy Detection. In *TRECVID 2009*, pages 16–17, 2009.

[9] T. N. Shanmugam, and P. Rajendran. An Enhanced Content-Based Video Retrieval System Based on Query-Clip. In *International Journal of Research and Reviews in Applied Sciences*, 1(3), pages 236–253, 2009.

[10] TRECVID Video Retrieval Evaluation: http://trecvid.nist.gov.

Author Index