Nieves Brisaboa
Oscar Pedreira
Pavel Zezula (Eds.)

LNCS 8199

Similarity Search and Applications

6th International Conference, SISAP 2013
A Coruña, Spain, October 2013
Proceedings

Springer

Nieves Brisaboa Oscar Pedreira
Pavel Zezula (Eds.)

Similarity Search and Applications

6th International Conference, SISAP 2013
A Coruña, Spain, October 2-4, 2013
Proceedings

Springer

Volume Editors

Nieves Brisaboa
Oscar Pedreira
Universidade da Coruña
Department of Computer Science, A Coruña, Spain
E-mail: {brisaboa, opedreira}@udc.es

Pavel Zezula
Masaryk University
Department of Computer Systems and Communications
Brno, Czech Republic
E-mail: zezula@fi.muni.cz

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-41061-1 e-ISBN 978-3-642-41062-8
DOI 10.1007/978-3-642-41062-8
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2013948103

CR Subject Classification (1998): H.3.1, I.5.3, E.1, H.3.3, H.2.4, H.2.8, F.2.2, G.1.2-3

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume contains the papers presented at the 6th International Conference on Similarity Search and Applications (SISAP 2013), held at A Coruña, Spain, during October 2–4, 2013.

The International Conference on Similarity Search and Applications (SISAP) is an annual forum for researchers and application developers in the area of similarity data management. It aims at the technological problems shared by many application domains, such as data mining, information retrieval, computer vision, pattern recognition, computational biology, geography, biometrics, machine learning, and many others that need similarity searching as a necessary supporting service.

Traditionally, SISAP conferences have put emphasis on the distance-based searching, but in general the conference concerns both the effectiveness and efficiency aspects of any similarity search approach.

In this edition, we had the ambition of widening SISAP's scope to cover even more aspects related to similarity. As a result, we have attracted a higher number of applications covering a wider range of approaches and application domains than in previous editions. In this way, we have achieved a more competitive conference, attractive to a larger part of the computer science community. In the future, we will keep that intent wishing SISAP to become a relevant conference on similarity search, bringing together a wide community of researchers and practitioners and welcoming contributions that range from theoretical aspects to innovative developments for which similarity search plays the central role.

The call for papers welcomed three types of contributions: (*i*) research papers (full or short papers) presenting previously unpublished research contributions, (*ii*) case studies and application papers (short papers) describing existing applications of similarity search in real scenarios, and (*iii*) demo papers describing real or prototype systems for which similarity search technology is a core component, presented at the conference with a system demonstration.

We received 44 submissions, from which 31 were full papers, 11 were short papers, and 2 were demo papers. The Program Committee (PC) comprised 29 researchers from 17 different countries. Each submission was assigned to at least three PC members. Reviews were discussed by the chairs and PC members when the reviews diverged and no sound decision had been reached. The final selection of papers was made by the PC chairs based on the reviews received by each submission. Finally, the conference program includes 19 full papers, 6 short papers, and 2 demo papers, which results in a 61,29% acceptance ratio for full papers and a 54,54% acceptance ration for short papers.

The conference program and the proceedings are organized into five parts. The first one comprises papers dealing with new scenarios or presenting new approaches to similarity search. A second part is devoted to papers proposing improvements to different methods and techniques for similarity search. The third part focuses on particular metrics and their effectiveness. The fourth part of the conference program includes papers devoted to solutions for similarity search in specific application domains, such as recommender systems, search engines, computational biology, and image and video retrieval. Finally, the last part comprises those papers dealing with efficient implementation and engineering solutions for similarity search in real settings.

The conference program also includes two invited talks from well-known researchers in the field. The first one, "Similarity in Web Search", by Ricardo Baeza-Yates, surveys different aspects of Web search in which similarity search plays an important role, considering the variety of objects that need to be compared, and the nature and features of the metrics involved in each case. The second one, "Large Scale Visual Object Retrieval", by Jiri Matas, presents the state of the art in visual retrieval of specific objects, and describes two new methods for large scale object retrieval.

As in previous editions, the proceedings are published by Springer-Verlag, in the *Lecture Notes in Computer Science* series. A selection of the best papers presented at the conference were recommended for publication in the journal *Information Systems*. The selection of best papers was made by the PC, based on the reviews received by each paper, and on the discussion during the conference.

SISAP conferences are organized by the SISAP initiative (`www.sisap.org`), which aims to become a forum to exchange real-world, challenging and innovative examples of applications, new indexing techniques, common test-beds and benchmarks, source code and up-to-date literature through its web page, serving the similarity search community.

We would like to acknowledge the generous collaboration and financial support from University of A Coruña, Spain (hosting instution), the Fields Institute for Research in Mathematical Sciences, Canada, and the Center for Research and Development in Information Technologies (CITIC) of University of A Coruña. We want to express our gratitude to the PC members for their effort and contribution to the conference. All the submission, reviewing, and proceedings generation processes were carried out through the EasyChair platform.

October 2013

Nieves Brisaboa
Oscar Pedreira
Pavel Zezula

Organization

Program Committee Chairs

Nieves R. Brisaboa	Universidade da Coruña, Spain
Pavel Zezula	Masaryk University, Czech Republic

Program Committee Members

Giuseppe Amato	Istituto di Scienza e Tecnologie dellInformazione (CNR), Italy
Laurent Amsaleg	Institut de Recherche en Informatique et Systèmes Aléatoires, France
Nieves Brisaboa	Universidade da Coruña, Spain
Benjamin Bustos	Universidad de Chile, Chile
Edgar Chavez	Universidad Nacional Autónoma de México, Mexico
Paolo Ciaccia	University of Bologna, Italy
Richard Connor	University of Strathclyde, UK
Andrea Esuli	Instituto di Scienza e Tecnologie dell'Informazione (CNR), Italy
Rosalba Giugno	University of Catania, Italy
Michael Houle	National Institute of Informatics, Japan
Alexis Joly	Inria, France
Björn Jónsson	Reykjavík University, Iceland
Daniel Keim	Universität Konstanz, Germany
Eamonn Keogh	University of California at Riverside, USA
Magnus Lie Hetland	Norwegian University of Science and Technology (NTNU), Norway
Yannis Manolopoulos	Aristotle University of Thessaloniki, Greece
Rui Mao	Shenzhen University, China
Luisa Micó	Universidad de Alicante, Spain
Henning Müller	University of Applied Sciences Western Switzerland, Switzerland
Gonzalo Navarro	Universidad de Chile, Chile
Arlindo Oliveira	Lisbon Technical University, Portugal
José Oncina	Universidad de Alicante, Spain
Apostolos Papadopoulos	Aristotle University of Thessaloniki, Greece
Marco Patella	University of Bologna, Italy
Oscar Pedreira	Universidade da Coruña, Spain
Vladimir Pestov	University of Ottawa, Canada

Matthias Renz	Ludwig-Maximilians-Universität München, Germany
Hanan Samet	University of Maryland, USA
Tomas Skopal	Charles University in Prague, Czech Republic
Pavel Zezula	Masaryk University, Czech Republic

Additional Reviewers

Bartolini, Ilaria
Buisson, Olivier
Falchi, Fabrizio
Hoksza, David
Krulis, Martin
Li, Hao
Lokoc, Jakub
Moss, Robert
Paredes, Rodrigo
Pedreira, Oscar
Reyes, Nora
Spretke, David
Stoffel, Andreas
Stoffel, Florian
Symeonidis, Panagiotis
Tellez, Eric Sadit
Wanner, Franz

Table of Contents

Invited Papers

Similarity in Web Search 1
Ricardo Baeza-Yates

Image Retrieval for Online Browsing in Large Image Collections 3
Andrej Mikulik, Ondřej Chum, and Jiří Matas

New Scenarios and Approaches

Rank Cover Trees for Nearest Neighbor Search 16
Michael E. Houle and Michael Nett

A New Concept of Sets to Handle Similarity in Databases: The SimSets 30
Ives R.V. Pola, Robson L.F. Cordeiro, Caetano Traina Jr., and Agma J.M. Traina

Similarity Search on Uncertain Spatio-temporal Data 43
Johannes Niedermayer, Andreas Züfle, Tobias Emrich, Matthias Renz, Nikos Mamoulis, Lei Chen, and Hans-Peter Kriegel

List of Clustered Permutations for Proximity Searching 50
Karina Figueroa and Rodrigo Paredes

Machine Learning for Image Classification and Clustering Using a Universal Distance Measure 59
Uzi Chester and Joel Ratsaby

Faster Algorithms for Tree Similarity Based on Compressed Enumeration of Bounded-Sized Ordered Subtrees 73
Kunihiro Wasa, Kouichi Hirata, Takeaki Uno, and Hiroki Arimura

Improving Similarity Search Methods and Techniques

Principal Directions-Based Pivot Placement 85
Fabrizio Angiulli and Fabio Fassetti

Pivot Selection Strategies for Permutation-Based Similarity Search 91
Giuseppe Amato, Andrea Esuli, and Fabrizio Falchi

Quantized Ranking for Permutation-Based Indexing 103
Hisham Mohamed and Stéphane Marchand-Maillet

Extreme Pivots for Faster Metric Indexes 115
Guillermo Ruiz, Francisco Santoyo, Edgar Chávez, Karina Figueroa, and Eric Sadit Tellez

Quicker Similarity Joins in Metric Spaces 127
Kimmo Fredriksson and Billy Braithwaite

Metrics and Evaluation

Evaluation of Different Metrics for Shape Based Image Retrieval Using a New Contour Points Descriptor 141
María Teresa García-Ordás, Enrique Alegre, Oscar García-Olalla, and Diego García-Ordás

Evaluation of LBP Variants Using Several Metrics and kNN Classifiers 151
Oscar García-Olalla, Enrique Alegre, María Teresa García-Ordás, and Laura Fernández-Robles

Evaluation of Jensen-Shannon Distance over Sparse Data 163
Richard Connor, Franco Alberto Cardillo, Robert Moss, and Fausto Rabitti

A Multi-way Divergence Metric for Vector Spaces 169
Robert Moss and Richard Connor

Optimal Distance Bounds for the Mahalanobis Distance 175
Tobias Emrich, Gregor Jossé, Hans-Peter Kriegel, Markus Mauder, Johannes Niedermayer, Matthias Renz, Matthias Schubert, and Andreas Züfle

Applications and Specific Domains

Text Categorization via Similarity Search: An Effcient and Effective Novel Algorithm 182
Hubert Haoyang Duan, Vladimir G. Pestov, and Varun Singla

Efficient Approximate Indexing in High-Dimensional Feature Spaces 194
Simone Santini

Semi-supervised Tag Extraction in a Web Recommender System 206
Vasily A. Leksin and Sergey I. Nikolenko

A Similarity Model for 3D Objects Based on Stable Sub-clouds 213
Markus Mauder, Peer Kröger, and Karl-Ludwig Schinner

Accurate and Efficient Search Prediction Using Fuzzy Matching and Outcome Feedback 219
Christopher Shaun Wagner, Sahra Sedigh, and Ali R. Hurson

Beyond Bag of Words for Concept Detection and Search of Cultural Heritage Archives 233
Costantino Grana, Giuseppe Serra, Marco Manfredi, and Rita Cucchiara

Large Scale Image Retrieval Using Vector of Locally Aggregated Descriptors 245
Giuseppe Amato, Paolo Bolettieri, Fabrizio Falchi, and Claudio Gennaro

Longest Common Subsequence in k Length Substrings 257
Gary Benson, Avivit Levy, and B. Riva Shalom

Implementation and Engineering Solutions

Database Similarity Join for Metric Spaces 266
Yasin N. Silva, Spencer S. Pearson, and Jason A. Cheney

Engineering Efficient and Effective Non-metric Space Library 280
Leonid Boytsov and Bilegsaikhan Naidan

Designing Similarity Indexes with Parallel Genetic Programming 294
Tomáš Bartoš and Tomáš Skopal

(Very) Fast (All) k-Nearest Neighbors in Metric and Non Metric Spaces without Indexing 300
Natalia Miranda, Edgar Chávez, María Fabiana Piccoli, and Nora Reyes

On Scalable Approximate Search with the Signature Quadratic Form Distance 312
Jakub Lokoč, Tomáš Grošup, and Tomáš Skopal

Demo Papers

PFMFind: A System for Discovery of Peptide Homology and Function 319
Aleksandar Stojmirović, Peter Andreae, Mike Boland, Thomas William Jordan, and Vladimir G. Pestov

Retrieving Similar Movements in Motion Capture Data 325
Jan Sedmidubsky and Jakub Valcik

Author Index 331

Similarity in Web Search

Ricardo Baeza-Yates

Yahoo! Labs Barcelona
Barcelona, Spain

Abstract. In this talk we survey how similarity plays an important role in many aspects of Web search, from crawling to indexing and from ranking to query recommendations. This implies similarity of different objects including text of web pages, web links, and web queries. Some similarities are measured directly while other similarities are inferred indirectly. Sometimes the similarity measure can be precomputed while other times needs to be calculated online.

1 Summary

Web search engines are crucial to find information among more than 660 million websites active at the middle of 2013[1]. In this ocean of data, users not only expect to find good results, also they expect the search engine to be fast and free.

A Web search engine has three major parts: the crawler that recollects data from the Web, the indexer that transforms that data to a compressed searchable representation (an index), and the query processor, which uses the index to return the most relevant web pages to a user query [4].

During the crawling phase, text similarity is used for de-duplication of web pages (e.g. finding mirrored websites [7]) and also for web text spam detection [10]. Given the volume of the web, text similarity has to be very efficient, so probabilistic techniques are used such as shingles [8].

Another important use of text similarity is to rank web pages by using standard information retrieval measures such as the cosine distance in a word-based vector model with weighting techniques such as BM25. Less known applications of text similarity are the lexical and semantic clustering of the Web [9], and the study of the genealogy of the Web, that is, how pages are created from other pages [3].

Similarity measures based on link attributes are mainly used for link spam detection [6] and also as part of web search ranking or web page recommendations.

Similarity measures for queries are very useful to generate query suggestions as well as query recommendations [2]. More over, similarity measures can also be used to extract semantic relations among queries [5]. As the query space is sparse, most of the query similarity measures are indirect [1].

[1] According to Netcraft, July 2013.

References

1. Baeza-Yates, R.: Graphs from search engine queries. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 1–8. Springer, Heidelberg (2007)
2. Baeza-Yates, R., Hurtado, C.A., Mendoza, M.: Improving search engines by query clustering. JASIST 58(12), 1793–1804 (2007)
3. Baeza-Yates, R., Pereira, A., Ziviani, N.: Genealogical trees on the Web: A search engine user perspective. In: WWW 2008: Proceedings of the 17th International Conference on World Wide Web, Beijing, China, pp. 367–376 (2008)
4. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval: The Concepts and Technology Behind Search, 2nd edn. Addison-Wesley (January 2011)
5. Baeza-Yates, R.A., Tiberi, A.: Extracting semantic relations from query logs. In: Berkhin, P., Caruana, R., Wu, X. (eds.) KDD, pp. 76–85. ACM, San Jose (2007)
6. Becchetti, L., Castillo, C., Donato, D., Leonardi, S., Baeza-Yates, R.: Link-based characterization and detection of Web Spam. In: Second International Workshop on Adversarial Information Retrieval on the Web (AIRWeb), Seattle, USA (August 2006)
7. Bharat, K., Broder, A.Z., Dean, J., Henzinger, M.R.: A comparison of techniques to find mirrored hosts on the WWW. Journal of the American Society of Information Science 51(12), 1114–1122 (2000)
8. Broder, A.: On the resemblance and containment of documents. In: SEQUENCES: Conf. on Compression and Complexity of Sequences, pp. 21–29. IEEE Computer Society, Salerno (1997)
9. Menczer, F.: Lexical and semantic clustering by web links. Journal of the American Society for Information Science and Technology 55(14), 1261–1269 (2004)
10. Ntoulas, A., Najork, M., Manasse, M., Fetterly, D.: Detecting spam web pages through content analysis. In: Proceedings of the World Wide Web Conference, Edinburgh, Scotland, pp. 83–92 (May 2006)

Image Retrieval for Online Browsing in Large Image Collections

Andrej Mikulik, Ondřej Chum, and Jiří Matas

Center of Machine Perception, Department of Cybernetics,
Faculty of Electrical Engineering, Czech Technical University in Prague

Abstract. Two new methods for large scale image retrieval are proposed, showing that the classical ranking of images based on similarity addresses only one of possible user requirements. The novel retrieval methods add zoom-in and zoom-out capabilities and answer the “What is this?” and “Where is this?” questions.

The functionality is obtained by modifying the scoring and ranking functions of a standard bag-of-words image retrieval pipeline. We show the importance of the DAAT scoring and query expansion for recall of zoomed images.

The proposed methods were tested on a standard large annotated image dataset together with images of Sagrada Familia and 100000 image confusers downloaded from Flickr. For completeness, we present in detail components of image retrieval pipelines in state-of-the-art systems. Finally, open problems related to zoom-in and zoom-out queries are discussed.

1 Introduction

A rapid increase in the size and ubiquity of shared image collections has motivated recent significant developments in image and specific-object retrieval. Most object-retrieval methods take into account the requirements for efficient content-based navigation and browsing of large-scale image collections.

Text search engines have provided the inspiration for the canonical approach to visual retrieval [30]. The user provides a query against which the retrieval engine ranks image relevance (or similarity). The performance of such an approach is typically assessed by a measure inherited from the text retrieval community: the average precision (AP). The state of the art and the standard components of the visual retrieval pipeline are reviewed in Section 2.

We show, however, that a similarity or relevance ranking of image-query results is not always suitable for browsing an image collection. This is demonstrated in the Fig. 1 rows denoted “nn”, which depict the output of a query in a large-scale image-retrieval system. All the results are similar to the original image in scale and viewpoint, providing little additional information. The phenomenon is an inherent problem of ranking by approaches using similarity. The problem becomes more pronounced as the size of the collection increases, since more images from similar viewpoints and of similar scales are present in the dataset. On the other hand, the rows of Fig. 1 denoted "zoom in" show regions of interest in the highest detected resolution. We advocate that *"the most detailed view"* or, in, short "zoom-in", is very probably the user intention after bounding-box selection.

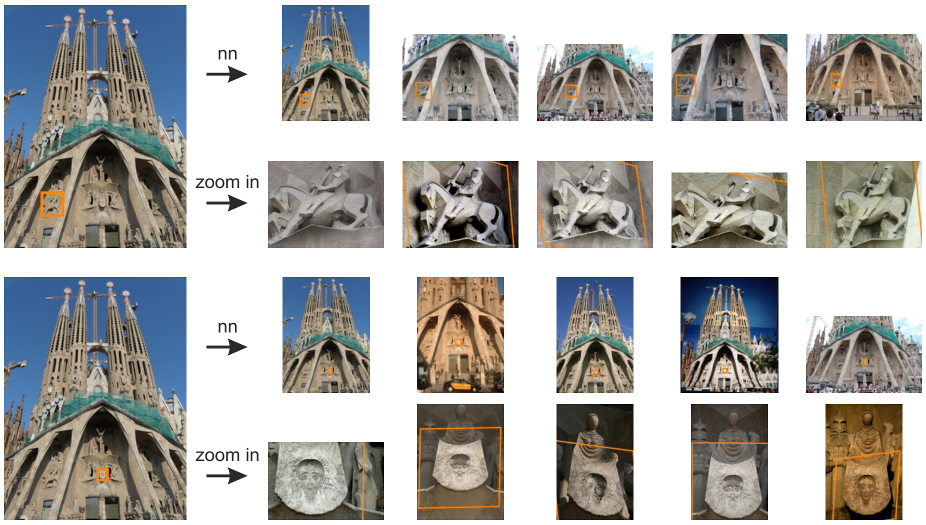


Fig. 1. Comparison of outputs of the standard and novel approaches. Two queries differing only by bounding-box were issued on the image in the leftmost column. The standard "most similar image" approach (nn, top rows) retrieves nearest neighbor matches, which provide no detailed images local to the bounding box and produce nearly identical results. The novel "most detailed view" approach or, *zoom-in*, maximizes the number of pixels inside the bounding box resulting in very different results (zoom in, bottom rows).

In the paper we address two tasks the user might be interested in: *"What is this?"* and *"Where is this?"*. The user expresses the first by selecting a bounding box from an image or simply moving a pointer over an image and forward-scrolling the mouse wheel. The expected result is a detailed image of the scene selected by the bounding box or of the local region centered around the pointer. The second expresses a desire for a broader contextual query.

In principle there are many tasks that might be of user interest: "What is to the left or right of this?"; "On which backgrounds can this object be seen?"; "Which objects can be seen on this background?"; "How does this object look like in the dark?". To demonstrate the concept we focus exclusively on the zoom-in and zoom-out tasks 2.

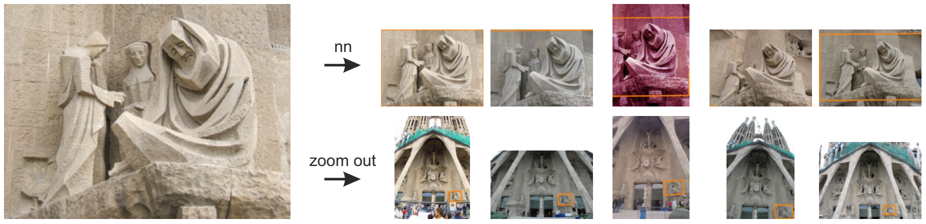


Fig. 2. Comparison of outputs of the standard and the proposed approach. The standard "most similar image" approach (nn, top rows) retrieves nearest neighbour matches, while the "context view" approach answers the question "Where is this?" by maximizing the scene content surrounding the bounding box, in this case, the whole query image (zoom out, bottom rows).

2 Standard Components and State-of-the-Art Methods in Large Scale Image Retrieval

In this section we review three popular approaches that each use vector representations for images. Additionally, we present image retrieval approaches derived from techniques used in text search as well as standard methods for increasing precision and recall after scoring in the index file.

2.1 The Bag of Words Image Representation

One of the most popular image representations is the *bag of words* (BoW). Images are represented as collections of local features. A local feature has its visual appearance represented by a visual word and its spatial extent defined by a point and an ellipse.

Features, typically affine covariant regions, are detected for each image in the dataset. The most frequently used detectors in image retrieval engines are the Harris-affine [19,29], Hessian-affine [19] and MSER [18], which have different detection characteristics, but collectively represent the state-of-the art. A comprehensive performance survey of features detectors is given by Mikolajczyk *et al.* [20], which confirms the high performance of the above listed detectors.

Detected interest regions are described by a feature descriptor. The SIFT descriptor [17], which describes an interest region by a point in a 128-dimensional space, is ubiquitous in state-of-the-art systems. Many modifications have been proposed in the literature, including two effective and popular variants: rootSIFT [2] and SURF [5].

Feature descriptors are vector quantized into visual words [30] creating a visual vocabulary. Many approaches have been studied in the literature, with modifications addressing different goals and constraints.

The canonical vocabulary construction method is the unsupervised k-means clustering. The parameter k denotes the number of visual words in the vocabulary. The choice of k varies: from $k \approx 10^3$, usually suitable for classification

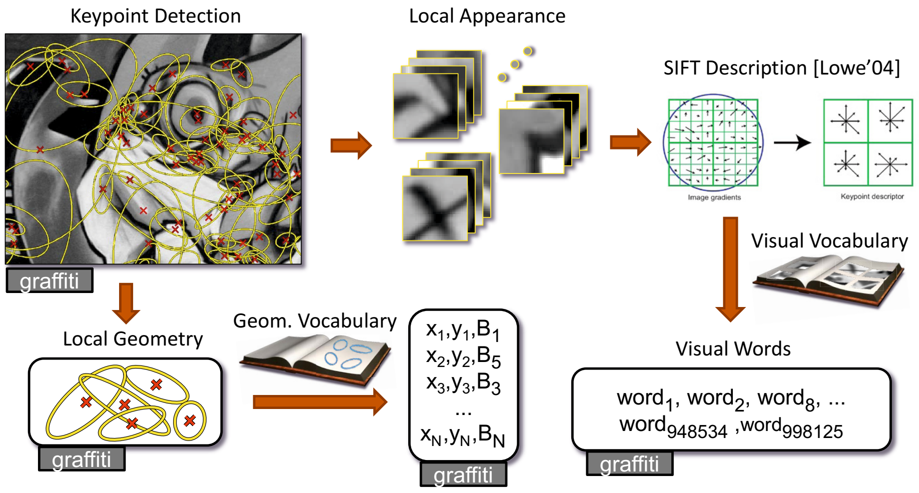


Fig. 3. Visualization of the bag of word image representation computation with geometry compression. Courtesy of Michal Perdˇoch.

tasks, up to $k \approx 10^7$ [21]. To efficiently construct large vocabularies, Nister *et al.* [23] proposed the use of a hierarchical vocabulary tree and Philbin *et al.* [26] use approximate nearest neighbour. Following the approach of Perdoch *et al.* [25], spatial information can be also compressed using unsupervised clustering without significant loss of precision. The process of image description is visualized in Figure 3.

2.2 Image Representation with VLAD

The vector of locally aggregated descriptors (VLAD) [15] is another successful image representation method. It combines the advantages of the bag of words and the Fisher kernel [12]. As in the BoW representation, local features are detected and described. The vocabulary is created with k-means, but, unlike the BoW method, only a small number of visual words k are used. Jegou *et al.* [15] show that good results are achieved for $k \in [16, 256]$ visual words. Visual words are constructed by finding k cluster centers as before, but the descriptor assigned to a cluster center is computed as a sum of signed differences between the cluster center and its nearest feature descriptors, resulting in a $k \times d$ dimensional vector (d is the dimension of the local descriptor, *e.g.* 128 for SIFT). Product quantization [14] is used to construct the final quantized descriptor creating a compact representation that fits into 20 bytes.

2.3 GIST Descriptor

A different approach to image representation is to create a global descriptor that captures the spatial layout and spatial relationships between regions or blobs of similar size, and the arrangement of basic geometric forms. One example is GIST, proposed by Oliva and Torralba [24]. A single GIST descriptor is used to represent an image, which results in a small memory footprint. The representation prevents partial matching of the image, it is sensitive to occlusion and there are no keypoints that can be used for spatial verification.

2.4 Image Retrieval

The nearest neighbor (NN) search for similar images is slow for large datasets, even if it uses sophisticated data structures avoiding exhaustively examination of the image database. Approximate NN search offers a big improvement.

Text search engines [1,4] face similar scalability problems for document retrieval, and the computer vision community has looked there for inspiration. In particular, image database indexing by the inverted file data structure leads to a dramatic speedup over the nearest neighbor search [30]. Inverted files map visual words to documents containing the words. The inverted file serves as in index into the database: upon a query, a subset of matching documents is returned, *i.e.*, those that contain the visual words of the query. The document ranking proceeds by calculating the similarity between the query vector and the matching document vectors. For sparse queries, the use of an inverted file ensures that only documents that contain query words are examined, which leads to a substantial speedup over the alternative of examining every document vector.

Efficient computation of the relevance of an image to a query is achieved by traversing the inverted file and reading the posting lists associated with the visual words of the query. The posting list (one row of the inverted file) associated with a visual word W is the list of image identifiers that contain visual word W. The standard tf-idf weighting scheme [3], also adopted from the document search community, is used to weight the document's relevance by de-emphasizing commonly occurring, less discriminative words.

Application of this approach is straightforward for sparse BoW vectors. For VLAD, similar speedup is achieved by combining the inverted file with asymmetric distance computation (IVFADC) proposed by Jegou *et al.* [14].

2.5 Spatial Verification and Query Expansion

As shown in [26,25], retrieval results are significantly improved by using the locations of features to verify their spatial consistency with the query region. This is achieved by a fast and robust hypothesize-and-test procedure that estimates an affine transformation between the query region and the target image. The RANSAC algorithm with local optimization [8] is widely used for spatial verification in state-of-the-art retrieval systems.

A caveat is that spatial verification is significantly more time consuming than BoW scoring. Thus it is performed only on the shortlist consisting of top scoring images. Furthermore, Chum *et al.* [9] show that if the model of the query (bag of words with feature geometries) is updated with newly spatially verified images by adding their visual words and geometries during the spatial verification, the probability of verifying other related images increases. Verified images in the shortlist are subsequently re-ranked.

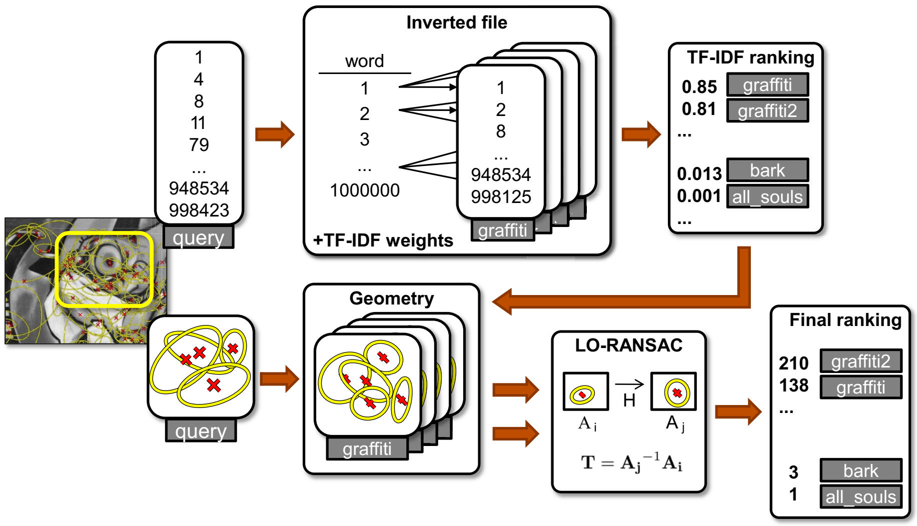


Fig. 4. Visualization of image retrieval with spatial verification for the bag of words representation. Courtesy of Michal Perďoch.

Chum *et al.* [10] proposed a query expansion (QE) method – another technique inspired by text retrieval [6,28] – to image retrieval and demonstrated impressive gains to recall. In QE, visual words from highly ranked images are composed in a new, expanded query. Unlike in text retrieval, features come with spatial information, typically keypoints, so geometric constraints and can be checked with spatial verification to ensure that the expanded query does not include visual words from a false positive image.

Chum *et al.* [9] added spatial context to queries by incorporating matching features that locally surround the initial query boundary into the query expansion. A latent model of the context of the query object is constructed by exploiting features surrounding the bounding-boxes of images verified by incremental spatial verification. A consistent context is learned and features belonging to the context can aid the expanded query, thus further improving recall. The process of image retrieval for BoW representation is summarized in Figure 4.

3 Overview of the Zooming Algorithm

The zooming algorithm, which implements the novel *"What is this?"* and *"Where is this?"* functionalities, is based on the standard bag of words image retrieval method. The distinction is in the choice of ranking function. Instead of ordering images according to similarity, it is designed to address new goals: maximizing detail or maximizing content.

To encourage a scale change, the ranking function requires knowledge of the geometric transformation between the query and the shortlisted images. The transformation is estimated by the RANSAC algorithm. The ranking function re-orders only verified images, *i.e.*, the images for which a geometric transformation was found, preferring zoomed-in or zoomed-out images.

To increase recall, scoring with the inverted file is weighted to account for scale change. To achieve this, compressed geometric information of the features is stored with their visual words and the *document at a time* (DAAT) scoring [31] is used to process the posting lists. Using DAAT, the geometry of the features is examined concurrently with computation of image scores, and the standard tf-idf score is re-weighted according to the scale change of features and user intention.

Query expansion plays an important role in the method, and the incremental spatial verification and context learning as proposed in [9] is used. In our experiments, good results were achieved when images selected for query expansion were chosen with the same ranking function as used for final ranking. Optionally, the query expansion step can be repeatedly issued until the requested zoom is found or the system fails to retrieve new, zoomed-in images. The method is summarized in Algorithm 1.

Algorithm 1. Overview of the zooming algorithm. Note that step 5 represents a trade-off between the query time and output quality.

```
Input: Bag-of-words of the query image
Output: Ranked list of images

1. Fetch posting lists for query visual words and score in DAAT order for
   each scale band separately.
2. Re-weight scores in scale bands to prefer desired change in scale and
   create a shortlist.
3. Spatially verify images in the shortlist, incrementally building an ex-
   panded query.
4. Rank images according to the desired goal (zoom-in/zoom-out)
5. Return the result or form the expanded query with context learning and
   goto 1
```

3.1 Ranking Functions

Many different tasks might be addressed with specific ranking functions. There are several options for zooming which can be useful for different tasks.

Zoom in. The simple option of ordering images according to the determinant of the geometric transformation between the query and the database image returns maximally zoomed images first. However, the top ranked images often cover only a small part of the scene selected by the bounding box. This ranking can be still useful if the images are going to be further processed, *i.e.*, compiled to a super-resolution image, used in a new expanded query, *etc.*.

We suggest that a user who browses the database expects to see the whole scene in the retrieved image. However, simply restricting the results to images that contain the whole bounding box often rejects significantly zoomed images with only a small fraction of the scene missing. Such images might be easily accepted by the user who usually does not want to be very precise while specifying the query bounding box.

A good trade-off between the zoom-in and a bounding box coverage was observed for the following ranking function:

$$z_{in} = \sqrt{\frac{A_r}{A_q}},$$

where A_r is the area inside the bounding box within the retrieved image and A_q is the area inside the query bounding box. The square root plays no role in the raking. It allows interpreting z_{in} as an estimate of the scaling of lengths (not the areas), which is consistent with zoom factor specification.

Zoom out. In this case, the naive “determinant of transformation” solution retrieves just images with similar scene content at lower resolution, providing no additional information.

To achieve the “*where is this*” or zoom-out goal, the user intuitively expects to see a large context of the query image. For this purpose, we propose the ranking function

$$z_{out} = \sqrt{\frac{A_r}{A_w}},$$

where A_r is the area inside the bounding box and A_w is the area of the whole retrieved image. In this case, we add the constraint that the whole bounding box must be visible in the result.

4 Experiments

A search engine was built for an expanded Oxford dataset (5063 images of Oxford landmarks) [26], which was augmented with 100000 confuser images and 15000

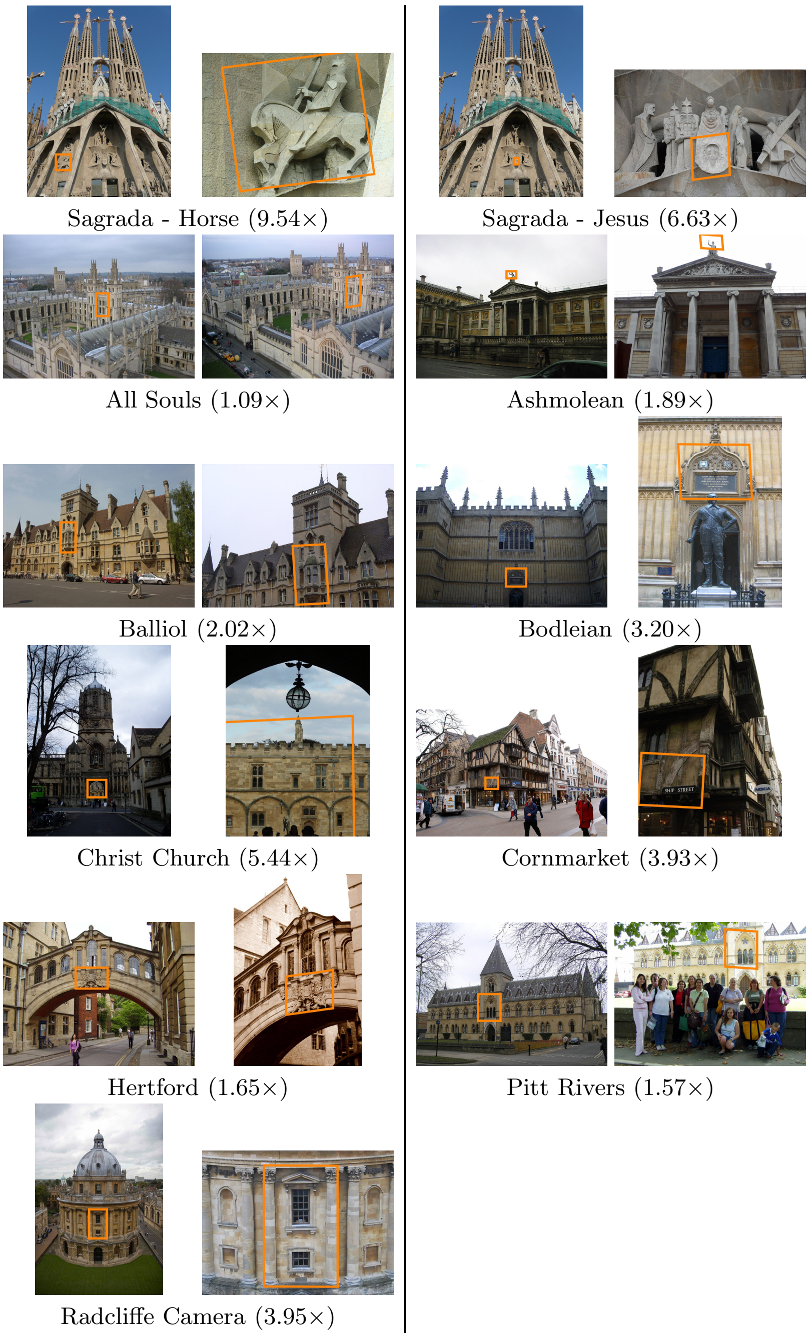


Fig. 5. Query images (on the left in each column) and the top results using the zoom-in method with DAAT scoring and query expansion. The effective zoom is in parentheses.

Table 1. Comparison of the standard method (nn) and zoom-in. We report the zoom of the first ranked image (top 1), and the average zoom of the top five images (top 5 average). Four methods were compared: 1. the baseline nearest neighbor search with query expansion (nn, QE), 2. Zoom-in only by shortlist re-ranking (rank), 3. DAAT scoring and re-rank (DAAT), 4. DAAT scoring, ranking function and query expansion (DAAT+QE). In all four cases, incremental spatial verification was used.

	top 1				top 5 average			
	nn	zoom-in			nn	zoom-in		
query	QE	rank	DAAT	DAAT+QE	QE	rank	DAAT	DAAT+QE
Sagrada - Horse	0.98	1.82	4.09	9.54	1.16	1.41	2.04	8.03
Sagrada - Jesus	0.86	2.75	2.75	6.63	1.22	1.22	1.87	6.00
All Souls	1.03	2.31	2.31	1.09	1.03	1.41	1.50	1.08
Ashmolean	1.43	1.43	1.43	1.89	1.28	1.28	0.77	1.45
Balliol	0.95	2.02	2.02	2.02	1.00	1.00	0.61	0.81
Bodleian	0.92	1.82	2.85	3.20	1.10	1.08	1.20	2.11
Christ Church	1.77	1.77	5.44	5.44	1.52	1.52	2.57	1.77
Cornmarket	1.57	3.93	3.93	3.93	1.39	1.97	1.97	1.97
Hertford	1.28	1.65	1.65	1.65	1.02	1.35	1.35	1.35
Pitt Rivers	1.30	1.36	1.57	1.57	1.30	1.22	1.10	1.10
Radcliffe Camera	1.29	3.95	3.95	3.95	1.23	2.03	2.04	2.35

landmark images. The Oxford dataset, as well as other standard datasets, is not very suitable for demonstrating the zoom capabilities since it does not contain significantly zoomed-in or zoomed-out images. For this reason we added 15000 images downloaded from flickr containing the tag *Sagrada Familia*. This favorite landmark is very well covered with photos from the distance up to the greatest details on the Sagrada Familia facade.

4.1 Design Choices

Following most of the recent work on image retrieval, multi-scale Hessian-affine features were used for feature detection. As we are interested in zooming, a global descriptor cannot be used, because it does not allow parts-based search of images.

Features were described by the 128-dimensional SIFT descriptor. The standard K-means algorithm with approximate nearest neighbor [22] is used to learn a vocabulary with one million visual words. The vocabulary is learned on the independent standard Paris dataset [27] (6412 images).

As in [25], feature geometries are compressed. Four bits are allocated for scale and 12 bits for shape compression. The compressed geometries are stored in the inverted file along with the visual words for fast access during DAAT scoring.

After scoring using the inverted file, a shortlist of the top 100 images is created. Incremental spatial verification is used and images are reordered with a chosen rank function. Optionally, the context of the query is learned and an expanded query is issued.

4.2 Evaluation Protocol

To our knowledge there is no standard dataset with an evaluation protocol suitable for testing zooming capabilities. To demonstrate the method, we chose 2 queries from Sagrada Familia and 9 queries from the Oxford dataset. The queries and the top results retrieved with the zoom-in method are shown in Figure 5. Note that even if the Oxford dataset is not well covered with detailed views of landmarks, the user can, for instance, use the zoom-in to view architectural detail (Sagrada), read street names (Cornmarket), boards (Bodleian) or virtually navigate through the scene (going through the archway at Christ Church).

Table 1 shows, for 11 selected queries, the zoom-in result in top ranked image and an average zoom in top 5 retrieved images. The baseline nearest neighbour (nn) search with context based query expansion (QE) is compared with three zoom-in methods. First includes only ranking function (rank), second utilizes DAAT scoring in inverted file (DAAT), and the last adds query expansion (DAAT+QE).

5 Conclusions

We have presented two new methods for large scale image retrieval demonstrating that the classical ranking of the images based on similarity is only one of many retrieval problems. In very large databases, the standard retrieval of the most similar images is unlikely to be useful as in many cases it returns just near duplicates.

The newly proposed retrieval methods add zooming capabilities and answer the “What is this?” and “Where is this?” questions. The functionality has been achieved by modifying two steps of the standard bag-of-words retrieval pipeline, namely the scoring and ranking functions.

We show the importance of the DAAT scoring and query expansion for recall of zoomed images. The proposed methods were tested on a standard large annotated image dataset together with images of Sagrada Familia and 100000 image confusers downloaded from Flickr.

Open Problems. Ordering images according to criteria other than similarity aggravates the problem of false positives. In a standard retrieval system, images are spatially verified and re-ordered according to the number of verified correspondences – the inliers of a geometric transformation obtained by the RANSAC algorithm. Irrelevant but highly similar (in the bag of words sense) images usually have only a small number of incidentally geometrically verified correspondences and are ranked after the true positive images. However, in the case of zooming, an incorrectly verified image often score as the zoomed version of the query. Such false positives are immediately recognized by the user. Moreover, the false positives are used in the subsequent query expansion which may lead to a complete failure of the system, *i.e.* to an irrelevant response to the query.

Retrieval with zooming is also sensitive to the presence of repetitive structures [16,11,32]. Zooming in on man-made objects very often reveals small repetitive patterns – textures on facades of building, bricks, fences, bars *etc.*, which can often cause failure of spatial verification and consequently of query expansion. Attention to burstiness [13], co-occurring features [7] and automatic failure recovery [9] alleviates the problem.

Acknowledgments. The research was supported by ERC-CZ-LL1303, the Czech Science Foundation Project GACR P103/12/G084 and Microsoft scholarship.

References

1. Aasheim, Y., Lidal, M., Risvik, K.M.: Multi-tier architecture for web search engines. In: Proceedings of First Latin American Web Congress (2003)
2. Arandjelovic, R., Zisserman, A.: Three things everyone should know to improve object retrieval. In: Proc. CVPR, pp. 2911–2918. IEEE (2012)
3. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. ACM Press, ISBN: 020139829 (1999)
4. Barroso, L.A., Dean, J., Holzle, U.: Web search for a planet: The google cluster architecture. IEEE Micro 23 (2003)
5. Bay, H., Tuytelaars, T., Van Gool, L.: SURF: Speeded up robust features. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) ECCV 2006, Part I. LNCS, vol. 3951, pp. 404–417. Springer, Heidelberg (2006)
6. Buckley, C., Salton, G., Allan, J., Singhal, A.: Automatic query expansion using smart: Trec 3, pp. 69–69. NIST Special Publication Sp (1995)
7. Chum, O., Matas, J.: Unsupervised discovery of co-occurrence in sparse high dimensional data. In: Proc. CVPR (2010)
8. Chum, O., Matas, J., Kittler, J.: Locally optimized RANSAC. In: Michaelis, B., Krell, G. (eds.) DAGM 2003. LNCS, vol. 2781, pp. 236–243. Springer, Heidelberg (2003)
9. Chum, O., Mikulik, A., Perdoch, M., Matas, J.: Total recall ii: Query expansion revisited. In: Proc. CVPR, pp. 889–896. IEEE Computer Society, Los Alamitos (2011) CD-ROM
10. Chum, O., Philbin, J., Sivic, J., Isard, M., Zisserman, A.: Total recall: Automatic query expansion with a generative feature model for object retrieval. In: Proc. ICCV (2007)
11. Doubek, P., Matas, J., Perdoch, M., Chum, O.: Image matching and retrieval by repetitive patterns. In: 2010 20th International Conference on Pattern Recognition (ICPR), pp. 3195–3198. IEEE (2010)
12. Jaakkola, T., Haussler, D.: Exploiting generative models in discriminative classifiers. In: Advances in Neural Information Processing Systems, pp. 487–493 (1999)
13. Jégou, H., Douze, M., Schmid, C.: On the burstiness of visual elements. In: Proc. CVPR (2009)
14. Jégou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE PAMI 33(1), 117–128 (2011)
15. Jégou, H., Douze, M., Schmid, C., Pérez, P.: Aggregating local descriptors into a compact image representation. In: Proc. CVPR (2010)
16. Leung, T., Malik, J.: Detecting, localizing and grouping repeated scene elements from an image. In: Buxton, B.F., Cipolla, R. (eds.) ECCV 1996. LNCS, vol. 1064, pp. 546–555. Springer, Heidelberg (1996)

17. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. In: Proc. ICCV, vol. 60(2), pp. 91–110 (2004)
18. Matas, J., Chum, O., Urban, M., Pajdla, T.: Robust wide baseline stereo from maximally stable extremal regions. In: Rosin, P.L., Marshall, D. (eds.) Proc. BMVC, vol. 1, pp. 384–393. BMVA, London (2002)
19. Mikolajczyk, K., Schmid, C.: An affine invariant interest point detector. In: Heyden, A., Sparr, G., Nielsen, M., Johansen, P. (eds.) ECCV 2002, Part I. LNCS, vol. 2350, pp. 128–142. Springer, Heidelberg (2002)
20. Mikolajczyk, K., Tuytelaars, T., Schmid, C., Zisserman, A., Matas, J., Schaffalitzky, F., Kadir, T., Van Gool, L.: A comparison of affine region detectors. IJCV 65(1/2), 43–72 (2005)
21. Mikulik, A., Perd̓och, M., Chum, O., Matas, J.: Learning vocabularies over a fine quantization. IJCV, 1–13 (2012)
22. Muja, M., Lowe, D.G.: Fast approximate nearest neighbors with automatic algorithm configuration. In: VISSAPP (2009)
23. Nister, D., Stewenius, H.: Scalable recognition with a vocabulary tree. In: Proc. CVPR (2006)
24. Oliva, A., Torralba, A.: Building the gist of a scene: The role of global image features in recognition. Visual Perception, Progress in Brain Research 155 (2006)
25. Perdoch, M., Chum, O., Matas, J.: Efficient representation of local geometry for large scale object retrieval. In: Proc. CVPR (2009)
26. Philbin, J., Chum, O., Isard, M., Sivic, J., Zisserman, A.: Object retrieval with large vocabularies and fast spatial matching. In: Proc. CVPR (2007)
27. Philbin, J., Chum, O., Isard, M., Sivic, J., Zisserman, A.: Lost in quantization: Improving particular object retrieval in largescale image databases. In: Proc. CVPR (2008)
28. Salton, G., Buckley, C.: Improving retrieval performance by relevance feedback. In: Readings in Information Retrieval, pp. 355–364 (1997)
29. Schaffalitzky, F., Zisserman, A.: Multi-view matching for unordered image sets, or how do I organize my holiday snaps? In: Heyden, A., Sparr, G., Nielsen, M., Johansen, P. (eds.) ECCV 2002, Part I. LNCS, vol. 2350, pp. 414–431. Springer, Heidelberg (2002)
30. Sivic, J., Zisserman, A.: Video google: A text retrieval approach to object matching in videos. In: Proc. ICCV, pp. 1470–1477 (2003)
31. Stewénius, H., Gunderson, S.H., Pilet, J.: Size matters: Exhaustive geometric verification for image retrieval accepted for ECCV 2012. In: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., Schmid, C. (eds.) ECCV 2012, Part II. LNCS, vol. 7573, pp. 674–687. Springer, Heidelberg (2012)
32. Torii, A., Sivic, J., Pajdla, T.: Okutomi M. Visual place recognition with repetitive structures. In: Proc. CVPR (2013)

Rank Cover Trees for Nearest Neighbor Search

Michael E. Houle[1] and Michael Nett[1,2]

[1] National Institute of Informatics, Tokyo 101-8430, Japan
{meh,nett}@nii.ac.jp
[2] The University of Tokyo, Tokyo 113-8656, Japan
michael_nett@mist.i.u-tokyo.ac.jp

Abstract. This paper introduces a k-NN search index, the *Rank Cover Tree* (RCT), whose pruning tests rely solely on the comparison of similarity values; other properties of the underlying space, such as the triangle inequality, are not employed. A formal theoretical analysis shows that with very high probability, the RCT returns a correct query result in time that depends competitively on a measure of the intrinsic dimensionality of the data set. Experiments show that the RCT is capable of meeting or exceeding the level of performance of state-of-the-art methods that make use of metric pruning or selection tests involving numerical constraints on distance values.

1 Introduction

Motivated at least in part by the impact of similarity search on problems in data mining, pattern recognition, and statistics, the design and analysis of scalable and effective similarity search structures has been the subject of intensive research for many decades. While early spatial index structures targeted low-dimension vector representations and L_p norms [16], more recently developed structures have focused on higher-dimensional vector representations and a wider variety of distance metrics [2, 4, 14]. Despite their advantages, spatial and metric search structures are limited by an effect referred to as the *curse of dimensionality*, in which distance values tend to concentrate strongly around their mean values as the dimension increases [1–3, 15].

The performance of search indices depends crucially on the way in which they use similarity information for the identification of objects relevant to a query. Most existing indices make use of numerical constraints, such as the triangle inequality or distance ranges, for pruning and selection [2, 5, 8]. A serious drawback of operations based on numerical constraints is that the number of objects actually examined can be highly variable. In an attempt to improve the scalability of applications that depend upon similarity search, researchers have investigated practical methods for speeding up the computation of neighborhood information at the expense of accuracy [5, 7, 8, 17].

In this paper, we propose a new similarity search structure, the *Rank Cover Tree*, whose internal operations avoid the use of numerical constraints involving similarity values. Instead, all internal selections of the RCT can be regarded as *ordinal*, in that objects are selected or pruned solely according to their rank with respect to the sorted order of distance from the query object. Rank thresholds precisely determine the number of objects to be selected, thereby avoiding a

```
Algorithm CT-Find-Nearest (Cover Tree T, query point q)

1. Initialize the root-level cover set, $V_h$, to contain all nodes at the root level of $T$.
2. For $j$ from $h-1$ down to 0, do:
   (a) Let $V_j^*$ be the set of all children of nodes of $V_{j+1}$.
   (b) Form cover set $V_j = \{v \in V_j^* \mid d(q,v) \le d(q,V_j^*) + 2^j\}$, where $d(q,V_j^*)$ is the
       minimum distance between $q$ and the points of $V_j^*$.
3. Return as the nearest neighbor the point $u \in V_0$ minimizing $d(q,u)$.
```

Fig. 1. Cover tree routine for finding the nearest neighbor of q

major source of variation in the overall query execution time. As ordinal pruning involves only direct pairwise comparisons of similarity values, the RCT is also an example of a *combinatorial* similarity search algorithm [6]. The main contributions of this paper are as follows:

- A similarity search index in which only ordinal pruning is used for node selection — no use is made of metric pruning or of other constraints involving distance values.
- Experimental evidence indicating that for practical k-NN search applications, our rank-based method is very competitive with approaches that make explicit use of similarity constraints.
- A formal theoretical analysis of performance showing that RCT k-NN queries produce correct results with very high probability. The performance bounds are expressed in terms of a measure of intrinsic dimensionality, independently of the full representational dimension of the data set. Due to space limitations, in this preliminary version of our report, some of the proofs in the analysis have been either sketched or omitted.

Very few efficient similarity search methods are known to use rank information as the sole basis for access or pruning. One approach, RanWalk [6], performs a random walk through a precomputed neighbor graph; however, its quadratic preprocessing time and space requirements are prohibitively large for many applications. The SASH similarity search structure [7] performs point location within a network of interconnected samples to efficiently find approximate query results. However, no theoretical guarantee of performance is known for this heuristic. To the best of our knowledge, the RCT is the first practical similarity search index that both depends solely on ordinal pruning, and admits a formal theoretical analysis of correctness and performance.

2 Cover Trees and the Expansion Rate

In [9], Karger and Ruhl introduced a measure of intrinsic dimensionality, the *expansion rate*. The complexity of many search algorithms strongly depends on the rate at which the number of visited elements grows as the search expands. Accordingly, Karger and Ruhl limited their attention to sets which satisfied the following *smooth-growth property*. Let $B_S(v,r)$ denote the set of points of S contained in the closed ball of radius r centered at $v \in S$. S is said

	CT	RCT ($h = 3$)	RCT ($h = 4$)	RCT ($h = 5$)	LSH
Space	n	n	n	n	$\approx n^{1+\rho}$
Build	$\delta^6 n \log n$	$c\delta^4 n^{5/3}$	$c\delta^5 n^{3/2}$	$c\delta^6 n^{7/5}$	$\approx n^{1+\rho} \log n$
Query	$\delta^{12} \log n$	$c\delta^4 kn^{2/3}$	$c\delta^5 kn^{1/2}$	$c\delta^6 kn^{2/5}$	$\approx n^{\rho} \log n$

	RCT (fixed $\Delta \geq 2, h \approx \log_\Delta n$)	RCT (fixed $h \geq 3$)
Space	n	n
Build	$c\delta^{1+\lfloor \log_\phi(\sqrt{5}\,h)\rfloor} n \log^2 n$	$c\delta^{1+\lfloor \log_\phi(\sqrt{5}\,h)\rfloor} n^{1+(2/h)}$
Query	$c\delta^{1+\lfloor \log_\phi(\sqrt{5}\,h)\rfloor} (k + \log n) \log n$	$c\delta^{1+\lfloor \log_\phi(\sqrt{5}\,h)\rfloor} kn^{2/h}$

Fig. 2. Asymptotic complexities of Rank Cover Tree (RCT), Cover Tree (CT), and LSH, stated in terms of $n = |S|$, neighbor set size k, and 2-expansion rate δ. For the RCT, we show the k-**NN** complexity bounds derived in Section 5 for several sample rates, both constant and sublinear. ϕ is the golden ratio $(1 + \sqrt{5})/2$. For the stated bounds, the RCT results are correct with probability at least $1 - \frac{1}{n^c}$. The query complexity stated for the CT algorithm is for single nearest-neighbor (1-**NN**) search, in accordance with the published description and analysis for this method [2]. Although a scheme exists for applying LSH to handle k-**NN** queries (see Section 6), LSH in fact performs $(1 + \varepsilon)$-approximate range search. Accordingly, the complexities stated for LSH are for range search; only the approximate dependence on n is shown, in terms of ρ, a positive-valued parameter that depends on the sensitivity of the family of hash functions employed. The dimensional dependence is hidden in the cost of evaluating distances and hash functions (not shown), as well as the value of ρ.

to have (b, δ)-expansion if for all $v \in S$ and $r > 0$, $|B_S(q, r)| \geq b$ implies $|B_S(q, 2r)| \leq \delta \cdot |B_S(q, r)|$. The *expansion rate* of S is the minimum value of δ such that the above condition holds. The value of $\log_2 \delta$ can be considered a measure of the intrinsic dimension, since doubling the radius of a Euclidean sphere in $\mathbb{R}^m$ increases its volume by a factor of 2^m. Upon sampling $\mathbb{R}^m$ by a uniformly distributed point set, the expanded sphere would contain proportionally as many points. However, as pointed out by the authors themselves, low-dimensional subsets in very high-dimensional spaces can have low expansion rates, whereas even for one-dimensional data the expansion rate can be linear in the size of S.

More recently, Beygelzimer, Kakade and Langford have proposed the *Cover Tree*, a similarity search structure with execution costs depending only on a constant number of factors of δ [2]. The organization of the Cover Tree satisfies the following invariants: for every node u at level $l - 1$ of the structure, its parent v at level l is such that $d(u, v) < 2^l$. Nearest-neighbor search progresses by identifying a *cover set* of nodes C_l at every tree level l whose descendants are guaranteed to include all nearest neighbors. Figure 1 shows the method by which the cover sets are generated, adapted from the original description in [2].

The asymptotic complexities of the Cover Tree are summarized in Figure 2. Experimental results for the Cover Tree show good practical performance for many real data sets [2], but the formal analysis still depends very heavily on δ.

3 Preliminaries

In this section, we introduce some of the basic concepts needed for the description of the RCT structure, and prove a technical lemma needed for the analysis.

3.1 Level Sets

The nodes in a Rank Cover Tree for S are organized in a random leveling $\mathcal{L} = (L_0, L_1, \dots)$ of subsets of S. Each node of the bottom level L_0 is associated with a unique element of S. For any integer $j \geq 0$ the membership of L_{j+1} is determined by independently selecting each node $v \in L_j$ for inclusion with probability $\frac{1}{\varDelta}$ for some real-valued constant $\varDelta > 1$. The smallest index h such that $L_h = \emptyset$ is the *height* of the random leveling. The probability that $v \in S$ belongs to the (non-empty) level set L_j is $\varDelta^{-j}$. The expected height h of $\mathcal{L}$ is $\log |S|$, and the probability of h exceeding its expectation by more than a multiplicative factor of $(c+1)$ is $|S|^{-c}$.

3.2 Rank Function

Let S be a data set contained in some domain $\mathcal{U}$. We assume the existence of an oracle which, given a query point $q \in \mathcal{U}$ and two objects in S, determines the object most similar to the query. The *rank function* $\rho_S : \mathcal{U} \times S \to \mathbb{N}$ yields $\rho_S(q, v) = i$ if and only if $(v_1, \dots, v_n)$ is the ordering provided by the oracle for q and $v = v_i$.

To facilitate the analysis of the RCT, we assume that the oracle rankings are induced by an underlying distance metric. For simplicity, we will assume the absence of tied distance values. Furthermore, the total ranking of S can also serve to rank any subset of S. For each level set $L_j \in \mathcal{L}$, we define the rank function $\rho_{L_j} : \mathcal{U} \times L_j \to \mathbb{N}$ as $\rho_{L_j}(q, v) = |\{u \in L_j \mid \rho(q, u) \leq \rho(q, v)\}|$, and henceforth we take ρ_j and ρ to refer to ρ_{L_j} and ρ_S, respectively.

3.3 Expansion Rate

As with the Cover Tree, the RCT is analyzed in terms of Karger and Ruhl's expansion rate. With respect to a query set $\mathcal{U}$, a random leveling $\mathcal{L}$ that, for all $q \in \mathcal{U}$ and $L_j \in \mathcal{L}$, satisfies $|B_{L_j}(q, ir)| \leq \delta_i \cdot |B_{L_j}(q, r)|$ for some real value $\delta_i > 0$ and some integer $i > 1$, is said to have an i-expansion of δ_i. We consider random levelings with 2- and 3-expansion rates δ_2 and δ_3, respectively. For simplicity, in this version of the paper we will place no constraints on the minimum ball size, although such constraints can easily be accommodated in the analysis if desired. Henceforth, we shall take $B_j(q, r)$ to refer to $B_{L_j}(q, r)$. Note also that the expansion rates of random levelings may be higher than those based only on the data set itself (level 0).

One of the difficulties in attempting to analyze similarity search performance in terms of ranks is the fact that rank information, unlike distance, does not satisfy the triangle inequality in general. To this end, Goyal, Lifshits and Schütze [6] introduced the *disorder inequality*, which can be seen as a relaxed, combinatorial generalization of the triangle inequality. A point set S has real-valued disorder constant D if all triples of points $x, y, z \in S$ satisfy $\rho(x, y) \leq D \cdot$

$(\rho(z,x)+\rho(z,y))$, and D is minimal with this property. We can derive a similar relationship in terms of expansion rates of a set S.

Lemma 1 (Rank Triangle Inequality). *Let δ_2 and δ_3 be the* 2- *and* 3-*expansion rates for $\mathcal{U}$ and the random leveling $\mathcal{L}$ of S. Then for any level set $L_l \in \mathcal{L}$, and any query object $q \in \mathcal{U}$ and elements $u, v \in S$,*

$$\rho_l(q,v) \leq \max\left\{\delta_2 \cdot \rho_l(q,u), \min\{\delta_2^2, \delta_3\} \cdot \rho_l(u,v)\right\}.$$

Proof. From the triangle inequality, we know that $d(q,v) \leq d(q,u) + d(u,v)$. There are two cases to consider, depending on the relationship between $d(q,u)$ and $d(q,v)$. First, let us suppose that $d(q,u) \geq d(u,v)$. In this case, we have $d(q,v) \leq 2d(q,u)$. Since $B_l(q,d(q,v)) \subseteq B_l(q,2d(q,u))$, and since $\rho_l(q,v) = |B_l(q,d(q,v))|$, we see that $\rho_l(q,v) \leq |B_l(q,2d(q,u))| \leq \delta_2|B_l(q,d(q,u))| \leq \delta_2\,\rho_l(q,u)$.

Now assume that $d(q,u) \leq d(u,v)$. Let w be any point contained in the ball $B_l(q,d(q,v))$. Since

$$\begin{aligned} d(u,w) &\leq d(u,q) + d(q,w) \leq d(q,u) + d(q,v) \\ &\leq d(q,u) + d(q,u) + d(u,v) \leq 3d(u,v), \end{aligned}$$

the ball $B_l(u,3d(u,v))$ entirely encloses $B_l(q,d(q,v))$. Therefore, we obtain the bound $\rho_l(q,v) \leq |B_l(u,3d(u,v)| \leq \delta_3|B_l(u,d(u,v))| \leq \delta_3\,\rho_l(u,v)$. Alternatively, $\rho_l(q,v) \leq |B_l(u,4d(u,v)| \leq \delta_2^2|B_l(u,d(u,v))| \leq \delta_2^2\,\rho_l(u,v)$. Combining the two bounds on $\rho_l(q,v)$, the result follows. □

Henceforth, as a simplification, we shall take δ to refer to δ_2, and use only the weaker claim that $\rho_l(q,v) \leq \max\left\{\delta \cdot \rho_l(q,u), \delta^2 \cdot \rho_l(u,v)\right\}$. In [10], Lifshits and Zhang showed that disorder inequalities and expansion rates are related, in that $D \leq \delta^2$. Note that our bound is tighter than that presented in [10]; this improvement will be crucial to the analysis of the RCT.

4 Rank Cover Tree

The proposed Rank Cover Tree blends some of the design features of the SASH similarity search structure and the Cover Tree. Like the SASH (and unlike the Cover Tree), we shall see that its use of ordinal pruning allows for tight control on the execution costs associated with approximate search queries. By restricting the number of neighboring nodes to be visited at each level of the structure, the user can reduce the average execution time at the expense of query accuracy. The algorithmic descriptions of RCT construction and query processing are outlined in Figures 3 and 4 respectively.

The underlying structure of the RCT is that of a tree T imposed on a random leveling $\mathcal{L}$ of the data set S. Given any $0 \leq l < h$, the subtree $T_l \subset T$ spanning only the level sets $L_l, L_{l+1}, \ldots, L_{h-1} \in \mathcal{L}$ is also a Rank Cover Tree for the set L_l; T_l will be referred to as the *partial Rank Cover Tree* of T for level l. Now, for any $u \in L_l$ and any choice of $l < j < h$, we define $a_j(u) \in L_j$ to be the unique ancestor of u in T_l at level j.

```
Algorithm RCT-Build (leveling $\mathcal{L}$, coverage parameter $\omega$)

1. Level assignment.
   For each item $u \in S$, determine level $\lambda(u)$. Introduce $u$ into each of the level sets
   $L_0, L_1, \ldots, L_{\lambda(u)}$.
2. Let $h$ be the smallest level index such that $L_h = \emptyset$. Produce partial RCT $T_{h-1}$ by
   connecting an artificial node notionally at level $h$ to each of the items of $L_{h-1}$.
3. Insertion loop.
   For $j = h-2$ down to 0, and for every item $u \in L_j$, insert $u$ into $T_j$ as follows:
   (a) If a copy of $u$ also belongs to level set $L_{j+1}$, then set this copy to be the parent
       of $u$.
   (b) Otherwise, set the parent of $u$ to be the element
       $v = \textbf{RCT-Find-}k\textbf{-Nearest}(T_{j+1}, u, 1, \omega)$.
4. Return tree $T_0$ as the RCT for $\mathcal{L}$.
```

Fig. 3. Offline construction routine for the Rank Cover Tree

RCT search proceeds from the root of the tree, by identifying at each level j a set of nodes V_j (the *cover set*) whose subtrees will be explored at the next iteration. For an item u to appear in the query result, its ancestor at level j must appear in the cover set associated with level j. V_j is chosen so that, with high probability, each true k-nearest neighbor u satisfies the following conditions: the ancestor $u_j = a_j(u)$ of u is contained in V_j, and for any query point q, the rank $\rho_j(q, u_j)$ of u_j with respect to L_j is at most a level-dependent *coverage quota* $k_j = \omega \max\left\{\frac{k}{\Delta^j}, 1\right\}$. The real-valued parameter ω is the *coverage parameter*. It influences the extent to which the number of requested neighbors k impacts upon the accuracy and execution performance of RCT construction and search, while also establishing a minimum amount of coverage independent of k. The effects of this parameter on RCT performance is the subject of the analysis in Section 5.

Offline construction of the RCT is performed by level-ordered insertion of the items of the random leveling, with the insertion of node $v \in L_j$ performed by first searching for its nearest neighbor $w \in L_{j+1}$ using the partial Rank Cover Tree T_{j+1}, and then linking v to w. The construction steps are summarized in Figure 3.

A Rank Cover Tree T will be said to be *well-formed* if for all nodes $u \in T$ with parent $v \neq u$, the parent v is the nearest neighbor of u from among the nodes at that level — that is, if $\rho_{\lambda(u)+1}(u, v) = 1$. In the analysis to follow, we determine conditions upon the choice of the coverage parameter ω for which the construction algorithm produces a well-formed RCT with high probability. We also derive bounds on the error probability for k-**NN** queries on a well-formed RCT.

5 Analysis

In this section, we prove that an RCT can be constructed using $\mathbf{O}(n)$ space and $\mathbf{O}(\delta^{1+\lfloor \log_\phi(\sqrt{5}\,h)\rfloor} n^{1+\frac{2}{h}})$ time, and correctly answer k-nearest neighbor queries in $\mathbf{O}(\delta^{1+\lfloor \log_\phi(\sqrt{5}\,h)\rfloor} k n^{\frac{2}{h}})$ time, all with high probability. Here, n is the size of the

```
*Algorithm* **RCT-Find-$k$-Nearest** (*RCT $T$, query point $q$, neighborhood size $k$, coverage parameter $\omega$*)

1. Set $V_{h-1} = L_{h-1}$.
2. For $j$ from $h-2$ down to 0 do:
    (a) Consider the set of children of $V_{j+1}$: $V_j^* = \bigcup_{v \in V_{j+1}} C(v)$.
    (b) Let $k_j = \omega \max\left\{\frac{k}{\Delta^j}, 1\right\}$ be the quota of children to be retained at level $j$.
        i. If $\left|V_j^*\right| \le k_j$, then choose $V_j \leftarrow V_j^*$.
        ii. Otherwise, let $V_j$ be the set of items of $V_j^*$ attaining the $\lfloor k_j \rfloor$ smallest
            distances from $q$ — that is, satisfying $|V_j| = \lfloor k_j \rfloor$ and $d(q,v) < d(q,w)$ for
            all $v \in V_j$ and $w \in V_j^* \setminus V_j$.
3. Return the $k$ elements of $V_0$ closest to $q$ according to $d$.
```

Fig. 4. k-nearest neighbor search for the Rank Cover Tree

data set S, $h \ge 3$ is the height of the random leveling $\mathcal{L}$, $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio, and δ is the maximum over the expansion rates of S and each level of $\mathcal{L}$.

We begin the RCT analysis with two technical lemmas, one of which relates the ranks of a query-neighbor pair with respect to two different level sets. The other bounds the average degree of nodes in an RCT.

Lemma 2. *Let v be any item of level set L_j, $0 < j < h$. Let $\alpha > e$ and $\beta > 0$ be real-valued constants. For any query item $q \in \mathcal{U}$, with respect to any level set L_l with $0 \le l < j$, we have*

$$\mathsf{Pr}\left[\rho_j(q,v) > \max\left\{\frac{\alpha}{\Delta^{j-l}}\rho_l(q,v), \beta\right\}\right] \le \max\left\{\left(\frac{e}{\alpha}\right)^{\frac{\alpha \cdot \rho_l(q,v)}{\Delta^{j-l}}}, \left(\frac{e \cdot \rho_l(q,v)}{\beta \cdot \Delta^{j-l}}\right)^{\beta}\right\}.$$

Proof. Let $U = \{u \in L_l \mid \rho_l(q,u) \le \rho_l(q,v)\}$ be the set of $\rho_l(q,v)$-nearest neighbors of q in L_l. With respect to q, the rank of v restricted to L_j is thus $\rho_j(q,v) = |U \cap L_j|$. Since L_j is formed via independent selection of the members of L_l with probability $1/\Delta^{j-l}$, the expected size of $U \cap L_j$ is $\mu = \rho_l(q,v)/\Delta^{j-l}$. We analyze two cases according to how the maximum of $\alpha\mu$ and β is determined.

Suppose that $\alpha\mu \ge \beta$. Applying the standard Chernoff bound technique [11] for the upper tail probability of $\rho_j(q,v)$ yields

$$\mathsf{Pr}[\rho_j(q,v) > \alpha\mu] < \frac{1}{e^{\mu}}\left(\frac{e}{\alpha}\right)^{\alpha\mu} < \left(\frac{e}{\alpha}\right)^{\alpha\mu}.$$

If on the other hand $\alpha\mu \le \beta$, by using the Chernoff bound technique again, we obtain

$$\mathsf{Pr}[\rho_j(q,v) > \beta] = \mathsf{Pr}\left[\rho_j(q,u) > \frac{\beta}{\mu} \cdot \mu\right] \le \left(\frac{e\mu}{\beta}\right)^{\beta}.$$

□

Lemma 3. *Let T be a well-formed Rank Cover Tree, and let v be a node of T at level j, where $h > j > 0$. Then the expected number of children of v is at most $\delta\Delta$.*

Proof. Omitted in this version.

Let $u \in S$ be a k-nearest neighbor of a query point $q \in \mathcal{U}$. Consider the unique ancestors $u_1, u_2, \ldots, u_h$ of u on each level. Since the RCT is a tree, the search finds u if and only if the coverage parameter ω is chosen such that the $k_j \geq \rho_j(q, u_j)$ for each level $0 \leq j < h$ (see Figure 4). The following lemma provides a useful bound on the ranks of these ancestors with respect to their levels.

Lemma 4. *Let T be a well-formed Rank Cover Tree of height $h > 0$ on data set S. For any given query item $q \in \mathcal{U}$ and size $k \geq 1$, let u be a k-nearest neighbor of q with respect to $L_0 = S$, and let $u_j = a_j(u)$ be the unique ancestor of u at level j. Furthermore, let $u_j = a_j(u)$ be the unique ancestor of u at level j, for $0 \leq j < h$, and let $u_{-1} = q$. The rank of u_j with respect to q is at most*

$$\rho_j(q, u_j) \leq \chi_j \cdot \max_{0 \leq i \leq j} \rho_j(u_{i-1}, u_i),$$

where $\chi_j \leq \delta^{\lfloor \log_\phi(\sqrt{5}\,(j+1)) \rfloor}$, and $\phi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.

Proof. (Sketch.) Consider the sequence $u_{-1}, \ldots, u_j$, where $u_{-1} = q$. By applying the rank triangle inequality (Lemma 1) with respect to $q = u_{-1}$, u_i, and u_j, for any choice of $0 \leq i < j$, we obtain the bound

$$\rho_j(q, u_j) \leq \min_{0 \leq i < j} \max \left\{ \delta \cdot \rho_j(u_{-1}, u_i), \delta^2 \cdot \rho_j(u_i, u_j). \right\}$$

Repeated application of the rank triangle inequality over the resulting subsequences ultimately yields a bound of the form

$$\rho_j(q, u_j) \leq \max_{0 \leq i \leq j} \{ \delta^{a_i} \cdot \rho_j(u_{i-1}, u_i) \},$$

where a_i is the maximum number of factors of δ accumulated by the term $\rho_j(u_{i-1}, u_i)$ during the expansion. Let $f(m)$ be the minimum worst-case number of factors of δ necessarily incurred in deriving a bound of the form stated above, for a sequence of $m+1$ points (or m gaps). The bound would then become

$$\rho_j(q, u_j) \leq \delta^{f(j+1)} \cdot \max_{0 \leq i \leq j} \{ \rho_j(u_{i-1}, u_i) \},$$

and therefore it suffices to show that $f(m) \leq \lfloor \log_\phi(\sqrt{5}\, m) \rfloor$. Clearly, the bound holds for $m = 1$.

Suppose for some $s \in \mathbb{N}$ that m is the s-th Fibonacci number F_s, which satisfies $F_1 = 1$, $F_2 = 1$, and for any integer $s > 2$,

$$F_s = \frac{1}{\sqrt{5}} \left(\phi^s - \left(-\frac{1}{\phi} \right)^s \right) = F_{s-1} + F_{s-2}.$$

Observe that $f(F_1) = f(F_2) = f(1) = 0$ and $f(F_3) = f(2) = 2$. Over a subsequence (u_a, u_{a+F_b}) with $b > 2$, we may apply the rank triangle inequality at u_a, $u_{a+F_{b-1}}$, and u_{a+F_b}, yielding the recurrence

$$f(F_s) \leq \max \{ 1 + f(F_{s-1}), 2 + f(F_{s-2}) \}.$$

This recurrence has solution $f(F_s) \leq s - 1$ for all $s \in \mathbb{N}$.

Even if m is not a Fibonacci number, for $m \geq 2$ one can always find a unique integer $s \geq 3$ such that $F_{s-1} + 1 \leq m \leq F_s$. Since f is a monotonically non-decreasing function, $f(m) \leq f(F_s) \leq s - 1$. The proof can be completed by showing that $\lfloor \log_{\phi}(\sqrt{5}\, m) \rfloor \geq s - 1$; the details are omitted. □

On its own, Lemma 4 is not sufficient to provide a deterministic performance bound, since neither u nor its ancestors are known until the search has reached the leaf level of the RCT. However, by combining Lemmas 2 and 4, we can derive a bound on the probability of an ancestor u_j of a true k-nearest neighbor u not being found at level j. This error probability decreases exponentially with ω.

Lemma 5. *Let T be a well-formed Rank Cover Tree of height $h > 0$ on data set S. For any given query item $q \in \mathcal{U}$ and size $k \geq 1$, let u be an element of the k-nearest-neighbor set U of q with respect to $L_0 = S$. Consider the cover sets V_j $(0 \leq j < h)$ generated as a result of a call to* **RCT-Find-*k*-Nearest** *(T, q, k, ω). If the coverage parameter satisfies $\omega > e\chi_j$, the probability that ancestor $u_j = a_j(u)$ fails to appear in cover set V_j whenever ancestor $u_{j+1} = a_{j+1}(u)$ appears in V_{j+1} is at most*

$$\mathsf{Pr}[u_j \notin V_j \mid u_{j+1} \in V_{j+1}] \leq \left(\frac{e\chi_j}{\omega}\right)^{\frac{\omega}{\chi_j}} .$$

Proof. Omitted in this version.

As with the previous lemma, which bounds the probability of an ancestor of a neighbor failing to belong to the cover set at a given level, we can bound the overall probability of the failure of the search and construction operations by summing these individual error rates over all ancestors involved in the operation.

Theorem 1. *Let T be a well-formed Rank Cover Tree of height $h = \log_{\Delta} n$ on data set S. For any given query item $q \in \mathcal{U}$ and size $1 \leq k \leq n$, consider the k-nearest neighbor set U of q with respect to S. Given some constant $c > 0$, if the coverage parameter ω is chosen such that $\omega \geq \chi_{h-1}\left(ch + \max\{2h, e\Delta\}\right)$, then with probability at least $1 - \frac{1}{n^c}$, a call to* **RCT-Find-*k*-Nearest** *(T, q, k, ω) correctly returns the entire set U, with the expected number of calls to the distance oracle being in $\mathbf{O}(\omega\Delta\delta(k + h))$.*

Proof. Omitted in this version.

Note that Theorem 1 bounds the number of calls to the oracle made during the processing of a query. The asymptotic complexity bounds also apply to the additional cost incurred at runtime, such as that of maintaining a set of tentative nearest neighbors.

Theorem 2. *Let $S \subseteq \mathcal{U}$ be a finite set, and let $\mathcal{L}$ be a random leveling for S of height $h = \log_{\Delta} n > 1$. Let the 2-expansion rates of S and any level set $L \in \mathcal{L}$ be no greater than δ. Consider the RCT produced via a call to* **RCT-Build***$(h, \mathcal{L}, \omega)$. If the coverage parameter is chosen such that*

$$\omega \geq \chi_{h-1}\left(ch + \max\{2h, e\Delta\}\right),$$

then with probability at least $1 - \frac{1}{n^c}$*, the tree is well-formed, and the execution time required is in* $\mathbf{O}(\delta \Delta \omega n h)$*.*

Proof. Analogous to Theorem 1.

The RCT complexity bounds can be further simplified if one assumes either that the sampling rate Δ is constant, or the number of levels h is constant ($\Delta = n^{\frac{1}{h}}$). These cases are shown in Figure 2. It should be noted that when the number of levels is fixed to $h = 3$ or $h = 4$, the dependence of the RCT search time bound on δ is δ^4 or δ^5, respectively. For small fixed choices of h, the dependence is much less than that of the Cover Tree (at δ^{12}), while still maintaining sublinear dependence on n.

6 Evaluation

We investigated and compared the performance of different approaches to exact and approximate k-nearest neighbor search in general metric spaces. In particular, we compared the fixed-height variant of the RCT (for heights 3, 4, 8 and 16) against the Cover Tree, the SASH, and (for those instances in which the L_2-norm is applicable) the popular E^2LSH [1] implementation of LSH. The accuracy of LSH may be influenced by the choice of a parameter governing failure probability. For the Cover Tree, we used an implementation provided by the authors [2] that is capable of handling k-**NN** queries. In addition, we compared the performance of RCT against two libraries for approximate k-NN search based on the KD-Tree: the first library, ANN [12], provides implementations of the KD-Tree and BD-Tree (box decomposition tree). The BD-Tree was run as recommended by the authors, using the ‘sliding midpoint’ and ‘simple shrinking’ rules. The accuracies of the KD-Tree and BD-Tree may be influenced by parameters governing admissible distance error. The second library, FLANN [13], uses an ensemble of KD-Tree indices and clustering-based KMeans trees, along with automatic parameter tuning of these indices for optimized performance. The experimentation was performed with the FLANN default parameter settings. These methods were also tested against a sequential search (linear scan) over random samples of the data, of varying sizes.

We chose a variety of publicly available data sets in order to demonstrate the behavior of the investigated methods across different data types, set sizes, dimensions and similarity measures. For each individual data set, we selected 100 objects uniformly at random as query locations. We measured the average time required to find the 100-nearest neighbors for each of these queries. For those methods making use of randomization, we averaged the results across 10 successive builds of the index structure. Except when otherwise stated, the distance measure employed was the Euclidean distance. For those data sets for which some other distance measure is more appropriate, the E^2LSH, KD-Tree, BD-Tree and FLANN were not evaluated (as the available implementations require the use of the L_2 or L_p norms).

[1] `www.mit.edu/andoni/LSH/`

Table 1. Construction times (in seconds) for the various methods tested. For the RCT experiments, the trees were constructed using a coverage parameter value of $\omega = 64$. The measurements presented for E^{2}LSH were obtained by setting the success probability ρ to 90%.

			ANN Library						RCT	
Data Set	Size	Dim.	BDTree	KDTree	CT	LSH	FLANN	SASH	$h = 3$	$h = 4$
ALOI	109500	641	164	2.6	12	6787	140	58	399	340
Chess	28056	6	—	—	0.1	—	—	1	2.3	2.6
CoverType	581012	55	53	4.7	5.6	3097	93	82	884	498
Gisette	7000	5000	11	3.7	243	3106	36	16	25	42
MNIST	70000	784	110	33	145	8236	94	43	219	201
Poker	1022771	10	—	—	5.9	—	—	112	1044	641
Reuters	554651	320647	—	—	97435	—	—	415	4393	2736

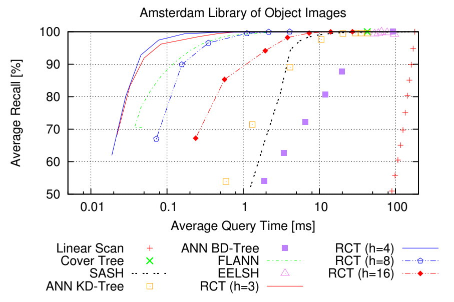
We measure the accuracy of the individual methods in terms of the achieved recall rates. The recall is defined as the proportion of true nearest neighbors returned by an index structure. Note that we do not penalize missing a true k-nearest neighbor, if instead another point with identical query-distance is returned.

6.1 Results

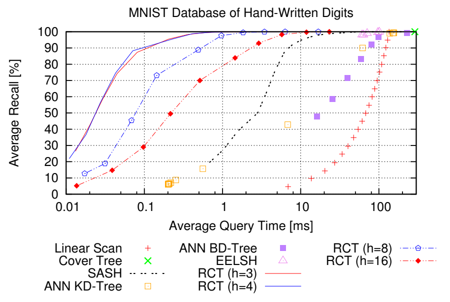
The experimental evaluation assessed the query performance of the various methods in terms of the tradeoff between accuracy and execution time; the results are presented in Figures 5a-6c, with accuracy plotted in terms of recall rates. The construction costs for the various indices are reported in Table 1.

In all instances tested, the fixed-height variants of the RCT for heights 3 and 4 performed very well, in that speed-ups over sequential search of more than 10 times (and in some cases, 100 times or more) were obtained for recall rates in excess of 90%. The performances of the RCT and the SASH were generally competitive with those of the other methods for the lower-dimensional data sets; for the higher-dimensional sets, the RCT and SASH dominated. Their main competitor was the ensemble method FLANN, which tended to outperform RCT for those data sets for which the Euclidean distance was appropriate. However, it must be noted that these data sets were of relatively small representational dimension. In order to show the relative performance of FLANN and RCT (with $h = 3$) on data sets of extremely high (but sparse) dimensionality, we compared their performance on projections of a document data set along randomly-selected dimensions between 512 and 4096 (see Figure 6d). In order that FLANN could be applied, the resulting vectors were normalized, and the L_2 distance was employed as the similarity measure for both methods. Although the accuracies achieved by FLANN were high, the results show that FLANN query times become impractically large as the dimensionality rises, due to their computation of dense mean vectors. On the other hand, as can be seen in Figure 6c, the RCT is capable of good performance even for the full 320,647-dimensional data set.

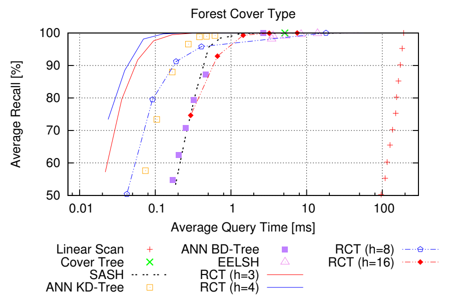
Although LSH is known to provide fast approximations of high quality for range-based queries, it performed rather poorly on all but one of the sets tested



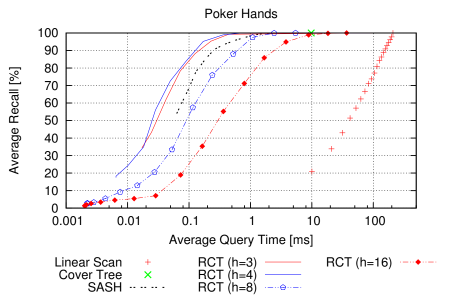
(a) The *Amsterdam Library of Object Images* contains **110,250** vectors, each containing **641** color and texture histogram features.



(b) The *MNIST Database of Hand-written Digits* accommodates **70,000** recordings of hand-written digits by 500 writers. The set contains **784** features.



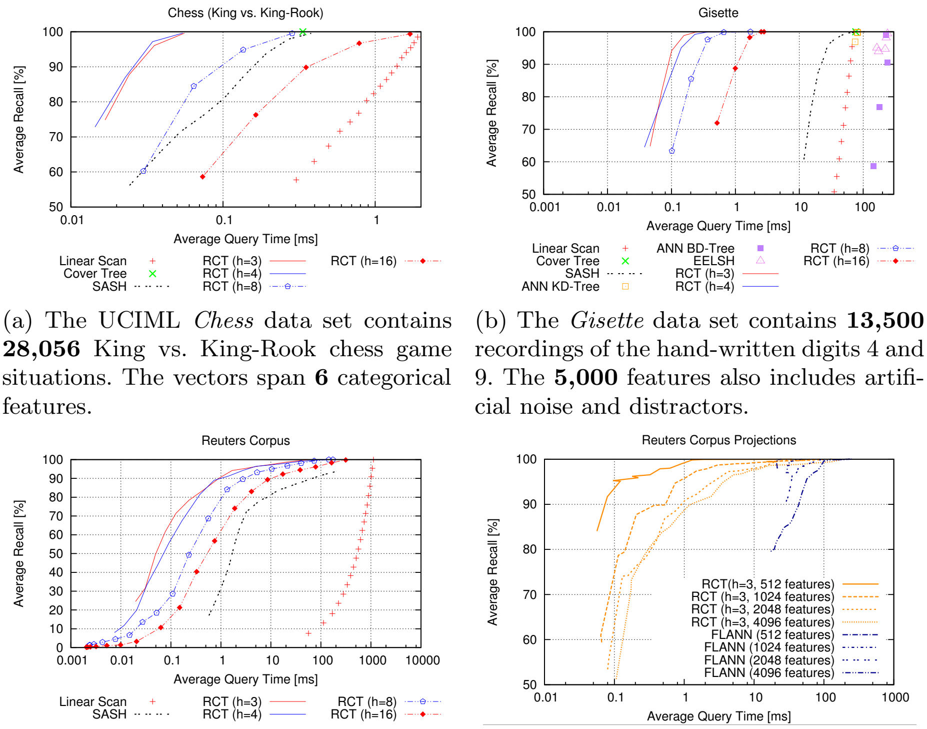
(c) The *Forest CoverType* consists of **581,012** instances of topological information on forest cells of $900m^2$ each. The **53** attributes include elevation, slope, soil and vegetation cover types.



(d) The *Poker Hand* data set consists of **1,025,010** poker hands. A hand is represented by **10** categorical attributes describing the suit and rank of each individual card in the hand.

Fig. 5. Comparison of query performances of the compared methods on the *ALOI*, *MNIST*, *Forest CoverType* and *Poker Hand* data sets

(those for which the Euclidean distance was used as a similarity measure). The time-accuracy trade-off controlled by the failure probability parameter of the LSH was in many cases so erratic that performance curves could not be generated; instead, the measurements are displayed as a scatter plot. The performance of the Cover Tree was more competitive than E^2LSH, substantially improving upon sequential search for the Forest Cover Type (Figure 5c) and the Poker Hand (Figure 5d) data sets. However, it was generally outperformed by the RCT, even for very high recall rates. Trends observable among the results for the KD-Tree show that it is somewhat competitive on data sets of low representational dimensionality. Its performance, however, degrades to that of the sequential scan on data sets of moderate to high representational dimension. On only one of the investigated instances did the KD-Tree outperform the RCT variants of heights 3 and 4 (the 53-dimensional Forest Cover Type set, shown in Figure 5c).



(a) The UCIML *Chess* data set contains **28,056** King vs. King-Rook chess game situations. The vectors span **6** categorical features.

(b) The *Gisette* data set contains **13,500** recordings of the hand-written digits 4 and 9. The **5,000** features also includes artificial noise and distractors.

(c) A selection of **554,651** documents drawn from the *Reuters Corpus Vol. 2* newspaper article set. The (sparse) vectors spanned **320,647** keyword dimensions.

(d) Comparison of FLANN and RCT on projections of the RCV2 set, consisting of 512, 1024, 2048 and 4096 dimensions, selected randomly.

Fig. 6. Comparison of query performances of the compared methods on the *Chess*, *Gisette*, and *Reuters* data sets, as well as (dense) projections of the *Reuters* data set

Finally, we note that the consistently good RCT performance for $h = 3$ and $h = 4$ compared with higher choices of h reflects the importance of reducing the dependence of the execution cost on the (intrinsic) dimensionality of the data. This trend in the performance of the fixed-height variants of the RCT has been validated on categorical data as well as numerical data, and for text data as well as image data and other data types.

Acknowledgments. Michael Houle acknowledges the financial support of JSPS Kakenhi Kiban (A) Research Grant 25240036, the JST ERATO Kawarabayashi Large Graph Project, and the JST ERATO Minato Discrete Structure Manipulation System Project. We are also grateful to Ilia Zvedeniouk for providing us with patches that allowed us to include the ANN library in our experimental evaluation.

References

1. Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is nearest neighbor meaningful? In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 217–235. Springer, Heidelberg (1998)
2. Beygelzimer, A., Kakade, S., Langford, J.: Cover trees for nearest neighbor. In: ICML 2006: Proc. 23rd Intern. Conf. on Machine Learning, pp. 97–104 (2006)
3. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. 33(3), 273–321 (2001)
4. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: Proc. 23rd Intern. Conf. on Very Large Data Bases, VLDB 1997, pp. 426–435 (1997)
5. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: VLDB 1999: Proc. 25th Intern. Conf. on Very Large Data Bases, pp. 518–529 (1999)
6. Goyal, N., Lifshits, Y., Schütze, H.: Disorder inequality: a combinatorial approach to nearest neighbor search. In: WSDM 2008: Proc. Intern. Conf. on Web Search and Web Data Mining, pp. 25–32 (2008)
7. Houle, M.E., Sakuma, J.: Fast approximate similarity search in extremely high-dimensional data sets. In: ICDE 2005: Proc. 21st Intern. Conf. on Data Engineering, pp. 619–630 (2005)
8. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: STOC 1998: Proc. 30th ACM Symp. on Theory of Computing, pp. 604–613 (1998)
9. Karger, D.R., Ruhl, M.: Finding nearest neighbors in growth-restricted metrics. In: STOC 2002: Proc. 34th ACM Symp. on Theory of Computing, pp. 741–750 (2002)
10. Lifshits, Y., Zhang, S.: Combinatorial algorithms for nearest neighbors, near-duplicates and small-world design. In: SODA, pp. 318–326 (2009)
11. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, New York (1995)
12. Mount, D.M., Arya, S.: ANN: A library for approximate nearest neighbor searching (2010)
13. Muja, M., Lowe, D.G.: Fast approximate nearest neighbors with automatic algorithm configuration. In: International Conference on Computer Vision Theory and Application (VISAPP 2009), pp. 331–340. INSTICC Press (2009)
14. Navarro, G.: Searching in metric spaces by spatial approximation. In: SPIRE 1999: Proceedings of the String Processing and Information Retrieval Symposium & International Workshop on Groupware, p. 141. IEEE Computer Society, Washington, DC (1999)
15. Pestov, V.: On the geometry of similarity search: dimensionality curse and concentration of measure. Inf. Process. Lett. 73(1-2), 47–51 (2000)
16. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann, San Francisco (2006)
17. Ye, N.: The Handbook of Data Mining. Lawrence Erlbaum (2003)

A New Concept of Sets to Handle Similarity in Databases: The SimSets

Ives R.V. Pola, Robson L.F. Cordeiro, Caetano Traina Jr., and Agma J.M. Traina

Computer Science Department - ICMC
University of São Paulo at São Carlos, Brazil
{ives,robson,caetano,agma}@icmc.usp.br

Abstract. Traditional DBMS are heavily dependent on the concept that a set never includes the same element twice. On the other hand, modern applications require dealing with complex data, such as images, videos and genetic sequences, in which exact match of two elements seldom occurs and, generally, is meaningless. Thus, it makes sense that sets of complex data should not include two elements that are "too similar". How to create a concept equivalent to "sets" for complex data? And how to design novel algorithms that allow it to be naturally embedded in existing DBMS? These are the issues that we tackle in this paper, through the concept of "similarity sets", or **SimSets** for short. Several scenarios may benefit from our SimSets. A typical example appears in sensor networks, in which SimSets can identify sensors recurrently reporting similar measurements, aimed at turning some of them off for energy saving. Specifically, our main contributions are: (i) highlighting the central properties of SimSets; (ii) proposing the basic algorithms required to create them from metric datasets, which were carefully designed to be naturally embedded into existing DBMS, and; (iii) evaluating their use on real world applications to show that our SimSets can improve the data storage and retrieval, besides the analysis processes. We report experiments on real data from networks of sensors existing within meteorological stations, providing a better conceptual underpinning for similarity search operations.

1 Introduction

Traditional Database Management Systems (DBMS) are heavily dependent on the concept that a set never includes the same element twice. In fact, the Set Theory is the main mathematical foundation for most existing data models, including the Relational Model [3]. On the other hand, applications are nowadays demanding the DBMS to handle increasingly complex data, such as images, audio, long texts, multidimensional time series, large graphs and genetic sequences, in which exact match on pairs of elements seldom occurs or makes sense. Therefore, the usual concept of "sets" blurs for these data, suggesting that a new, equivalent concept must take place. Thus, the questions we pose are: What is a concept equivalent to "sets" for complex data? Furthermore, how to design novel algorithms that allow it to be naturally embedded in existing DBMS?

The problem stems from the need to compare complex data elements by similarity, as opposed to comparing them by equality, which is the standard strategy for traditional data (i.e., data in numeric or in short character string domains). Then, data is represented in a metric space $M =< \mathbb{S}, d >$ where $\mathbb{S}$ is the data domain and $d : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}^{+}$ is the distance function, that holds the following properties for any $s_1, s_2, s_3 \in \mathbb{S}$: Identity: $d(s_1, s_2) = 0 \rightarrow s_1 = s_2$; Non-negativity: $0 < d(s_1, s_2) < \infty$, $s_1 \neq s_2$; Symmetry: $d(s_1, s_2) = d(s_2, s_1)$ and Triangular inequality: $d(s_1, s_2) \leq d(s_1, s_3) + d(s_3, s_2)$. As a matter of fact, similarity evaluation is what people naturally do for complex data, as it is generally senseless to use identity to compare them [6]. A typical example is the decision making process that physicians do when analyzing medical images. Exact match provides very little help here, since it is highly unlikely that identical images exist in the data, even within images of exams from the same patient. On the other hand, it is often important to retrieve related past cases to take advantage of their analysis and treatment, by searching for *similar* images from previous patients' exams. Thus, similarity search is almost mandatory, for this example and also for many others, such as in data mining [11], duplicate document detection [5] and complex data analysis in general.

In such a context, this paper introduces a concept equivalent to "sets" for complex data: the "similarity sets", or SimSets for short. Briefly, a SimSet is one set of complex elements where there are no two elements "too similar" to each other, following the definition that a set of scalar elements does not contain the same element twice. Although intuitive, this concept brings some interesting issues, such as:

- Among many "too similar" elements, how to select the one that pertains to a SimSet?
- What properties hold for SimSets?
- How to handle SimSets on real data?

In this paper we formally define the concept of SimSets and develop basic resources to handle SimSets in a database environment. Specifically, our main contributions are:

1. **Concept and properties:** We introduce the new concept of SimSets, its terminology and its most important properties;
2. **Algorithm:** We propose the algorithm *Distinct* to extract SimSets from any metric dataset, carefully designing it to be naturally embedded in existing DBMS.
3. **Evaluation on real data:** We evaluate SimSets on real world applications, showing that SimSets can improve data storage and retrieval, besides analysis processes.

Several scenarios may benefit from SimSets. For example, Information Retrieval Systems would be more effective in querying sets of documents if they avoid reporting similar texts. In a Movie Library, video sequences may be identified by selecting frames significantly distinct from others, even within the same video. In Social Networks, one may better rank images by discarding those that are similar to others already posted. Sensor networks can turn off stations recurrently reporting similar measurements to save power.

To validate our proposal we decided to investigate this latter scenario. We analyzed sensor networks using SimSets to identify sensors that recurrently report similar measurements, aimed at turning some of them off for energy saving. Specifically, we report experiments on simulated data for sensor networks of ground-based weather stations from Brazil. Our findings point out which areas would require more or less sensors based on the similarity from the data estimated by a climate forecasting model.

The rest of the paper follows a traditional organization: background and related work (Sections 2 and 3), proposed concepts and algorithms (Sections 4 and 5), experiments (Section 6) and conclusions (Section 7). Table 1 lists the symbols of our notation.

Table 1. Table of symbols

Symbol	Description
$\mathbb{R}, \mathbb{S}, \mathbb{T}$	Data domain
$\mathcal{R}, \mathcal{S}, \mathcal{T}$	Database relations
$\mathsf{R}, \mathsf{S}, \mathsf{T}$	Sets of elements
r_i, s_i, t_i	Elements in a set ($r_i \in \mathsf{R}$, $s_i \in \mathsf{S}$, $t_i \in \mathsf{T}$)
σ	Relational selection operator
$\ddot{\sigma}$	Similarity selection operator
$\hat{\sigma}$	Similarity extraction operator

Symbol	Description
ξ	Similarity threshold
d	Distance function (metric)
$\hat{=}^{\xi}$	Sufficiently similar operator
$\hat{R}^{\xi}, \hat{S}^{\xi}, \hat{T}^{\xi}$	ξ-similarity-sets
$\hat{G}^{\xi}$	A ξ-Similarity Graph
$P\hat{S}^{\xi}(\mathsf{S})$	The ξ-Similarity Cover of a set S
λ	Extraction policy

2 Background

The problem of extracting *SimSets* from a set S involves representing all similarities occurring among pairs of elements in S as a graph. Extracting *SimSets* from a set S is therefore closely related to the concept of independent dominant sets, from the Graph Theory [4]. For a graph $G = \{V, E\}$ composed of a set of vertices V and set of edges E, one independent dominating set is another graph $\hat{G} = \{\hat{V}, \hat{E}\}, \hat{V} \subset V$ and $\hat{E} = \emptyset$ such that, for any vertex v, $v \in V$ and $v \notin \hat{V}$, the vertex v has at least one edge $e \in E$ connecting it to at least a vertex in $\hat{V}$. Finding independent dominant sets is a NP-hard optimization problem and may take a polinomial time to solve it [4]. To complete the formulation of our SimSets, in this paper we propose the algorithm *Distinct*, an approximate solution to quickly and accurately find both maximum and minimum independent dominant sets.

3 Related Work

There are several work in the literature aimed at developing or improving similarity operators. Silva et. al. [10] investigate how the similarity-based operators interact with each other. The work describes architectural changes required to implement several similarity operators in a DBMS, and it also shows how these operators interact both with each other and with regular operators. The work of Budikova et. al. [2] proposes a query language that allows to formulate content-based queries supporting complex operations, such as similarity joins.

The work of Xiao [12] presents an algorithm to handle duplicates in string data, aimed at tackling diverse problems, such as web page duplicate detection and document versioning, considering that each object is composed of a list of tokens that can be ordered. Zhu et. al. [13] investigated the cosine similarity problem. As many studies require a minimum threshold to perform the computation, this work proposes to use the cosine similarity to retrieve the top-k most correlated object pairs. They first identify an upper bound property, by exploiting a diagonal traversal strategy, and then retrieve the required pairs. Some related works also focus on solving particular problems, such as sensor placement [8] and classification [7].

The selection of representative objects that are not too similar to each other has already been addressed in metric spaces as a selection of pivots. The method Sparse

Spatial Selection – SSS [9] is a greedy algorithm able to choose a set of objects (i.e., pivots) that are not too close to each other, based on a distance threshold α. Although the set of pivots refers to a SimSet, SSS targets the use of a few pivots to improve the efficiency of similarity searches in metric spaces, being not appropriate to cases where the number of pivots can be large, such as in our problem formulation of SimSets. Unfortunately, SSS has a quadratic time complexity on the number of pivots selected, and the largest such number seen in its original experiments is only 258.

In spite of the several qualities found in the existing techniques to handle similarity operations, none of them throughly consider the existence of "too similar" elements in the dataset, which is necessary for the creation of a concept equivalent to "sets" for complex data. In this paper we tackle that problem by formally defining the concept of *SimSets* and developing fundamental resources to handle *SimSets* in a database environment.

4 The Similarity-Set Concept

Comparing two elements by equality has very little interest when performing queries over complex data. Thus, exact comparison must give ground to similarity comparison, where pairs of elements close enough should be accounted as representing "the same" real world object for the purposes of the application at hand. The relational model depends upon the concept that two values in a given domain compared by the equality comparison operator "=" results true if and only if the two values represent the same object. When an application requires that two very similar (but not exactly equal) values s_i and s_j refer to the same object, the equality operator should be replaced by a "sufficiently similar" comparison operator, as defined as follows.

Definition 1 (Sufficiently ξ-similar). *Given a metric space $M = \langle \mathbb{S}, d \rangle$ and two elements $s_i, s_j \in \mathbb{S}$, we say that s_i and s_j are sufficiently similar up to ξ if $d(s_i, s_j) \leq \xi$, where ξ stands for the minimum distance that two elements must be separated to be accounted as distinct. Whenever that condition applies, it is denoted as $s_i \hat{=}^{\xi} s_j$ and we say that s_i is ξ-similar to s_j*[1]*, where ξ is also called the "similarity threshold".*

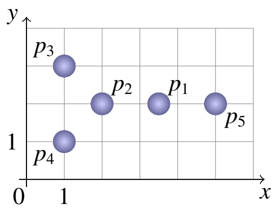
Taking into account the properties of a metric space, it follows that the $\hat{=}^{\xi}$ comparison operator is reflexive ($s_i \hat{=}^{\xi} s_i$) and symmetric ($s_i \hat{=}^{\xi} s_j \Rightarrow s_j \hat{=}^{\xi} s_i$), but it is not transitive. In fact, it is easy to prove that the $\hat{=}^{\xi}$ comparison operator is transitive if and only if $\xi = 0$, and in this case "$\hat{=}^{0}$" is equivalent to "=".

The idea that a set never has the same element twice must be acknowledged when pairs of ξ-similar elements are accounted as the same, so they should not occur twice in the set. To represent this idea we propose the concept of "similarity-set" as follows.

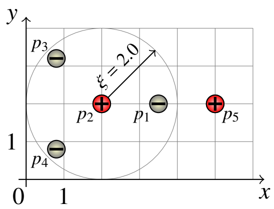
Definition 2 (ξ-Similarity-Set). *A ξ-similarity-set $\hat{S}^{\xi}$ is a set of elements from a metric domain $M = \langle \mathbb{S}, d \rangle$, $\hat{S}^{\xi} \subset \mathbb{S}$, such that there is no pair of elements $s_i, s_j \in \hat{S}^{\xi}$ where $s_i \hat{=}^{\xi} s_j$. We say that $\hat{S}^{\xi}$ is a ξ-similarity-set (or a "ξ-simset" for short) that has ξ as its similarity threshold.*

[1] The symbol "ˆ" superimposed to an operator indicates that the operator is similarity-based.

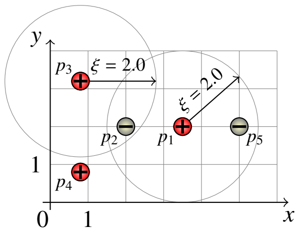
Note that varying ξ generates different similarity-sets. This is why the symbol ξ is attached to the ξ-simset $\hat{S}^{\xi}$. The value of ξ depends on the application, on the metric d and on the features extracted from each element s_i to allow comparisons. Given a bag of elements (that is, a collection where elements can repeat), extracting a set from it is straightforward. But, extracting a ξ-simset $\hat{S}^{\xi}$ from a set S is not. The increased complexity derives from the fact that there may exist many distinct ξ-simsets within a set S. For example, let us consider a set of points in a 2-dimensional space using the Euclidean distance, as shown in Figure 1(a). A set of points associated with distance $\xi = 2$ produces many distinct 2-simsets. Examples are $\{p_2, p_5\}$ from Figure 1(b), $\{p_1, p_3, p_4\}$ from Figure 1(c) and $\{p_3, p_4, p_5\}$ from Figure 1(d), since the distance between p_2 and p_5 is bigger than to 2, and it also happens for p_1, p_3 and p_4, and for p_3, p_4 and p_5.



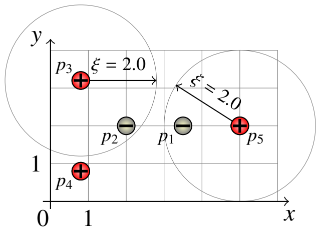
(a) 2-dimensional toy dataset



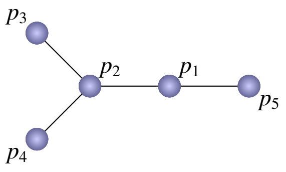
(b) Example of a 2-simset



(c) Example of a 2-simset



(d) Example of a 2-simset



(e) The 2-similarity graph

Fig. 1. (a) A 2-dimensional toy dataset; (b),(c),(d) Examples of 2-simsets produced from our toy dataset using the Euclidean distance; (e) Corresponding ξ-similarity graph for $\xi = 2$.

Our proposal to extract a ξ-simset $\hat{S}^{\xi}$ from a set S involves representing in a graph all ξ-similarities occurring among pairs of elements in S as we define as follows.

Definition 3 (ξ-Similarity Graph). *Let S be a dataset in a metric space and ξ a similarity threshold. Then, the corresponding ξ-similarity graph $\hat{G}^{\xi}(\mathsf{S}) = \{V, E\}$ is a graph where each node $v_i \in V$ corresponds to an element $s_i \in \mathsf{S}$, and there is an edge $\left\langle e_i, e_j \right\rangle \in E$ if and only if $s_i \hat{=}^{\xi} s_j$. We name $\hat{G}^{\xi}(\mathsf{S})$ as the ξ-simgraph of S.*

Figure 1(e) illustrates the ξ-similarity graph for the example dataset in Figure 1(a), considering $\xi = 2$. It is important to note that, given the ξ-simgraph of S, the following rule allows checking if S is a ξ-simset or not: "*The ξ-Similarity Graph $\hat{G}^{\xi}(\mathsf{S})$ has the set of edges $E = \emptyset$ if and only if S is a ξ-simset.*"

If a set S is not a ξ-simset, it is always possible to select some of its elements to extract a $\hat{S}^{\xi}$ from S. As already said, there are several ways to perform this task. Therefore, rules must be defined to efficiently drive the selection process. We name a rule employed to select elements from S to extract $\hat{S}^{\xi}$ as a "policy". In this paper we consider two policies, as follows.

[*min*] – This policy aims at selecting elements to extract a ξ-simset $\hat{S}^{\xi}$ with the minimum possible cardinality from S, such that $\forall s_i \in \mathsf{S} : \exists s_j \in \hat{S}^{\xi} : s_i \hat{=}^{\xi} s_j$;
[*Max*] – This policy aims at selecting elements to extract a ξ-simset $\hat{S}^{\xi}$ with the maximum possible cardinality from S, such that $\forall s_i \in \mathsf{S} : \exists s_j \in \hat{S}^{\xi} : s_i \hat{=}^{\xi} s_j$.

In the context of the Relational Model, the statement that an attribute is able to store values from a ξ-simset can be expressed as a constraint over the attribute. It must indicate the similarity threshold, and one of the policies for selecting ξ-simsets. For example, let us suppose that SQL is extended to allow representing similarity-based constraints. A possibility for a table constraint syntax is shown as follows:

```
CONSTRAINT <name> CHECK
 (SIMILARITY <attribute> UP TO <ξ> USING <distance_function>
   <selecting_policy>)
```

In this way, the creation of a table with a primary key `PK` of type `CHAR(10)` and an attribute `Img` of type `STILLIMAGE` that is a ξ-simset under the LP_2 distance function would be expressed as follows:

```
CREATE TABLE Tab AS (
PK CHAR(10) PRIMARY KEY,
Img STILLIMAGE,
CONSTRAINT SimSet_Img CHECK (SIMILARITY Img UP TO 2 USING LP2 MAX) );
```

In this example, we assume that `STILLIMAGE` is a data type over a metric domain using the Lp_2 (euclidean) distance function, and a similarity radius of 2 units define the similarity threshold under that distance function. The 2-simset is selected by the `MAX` policy. Note that a 1-SimSet is easy to obtain: every unit set is a *SimSet*. However, we are interested in *SimSets* that express the information stored in a set of elements. Thus we are interested in *SimSets* "extracted" from sets, as we will define following.

It is straightforward to maintain the consistency of a ξ-SimSet when it is created starting from the empty set and new elements are incrementally inserted. However, when the constraint is stated over an already existing set, it is required the ξ-simset to be extracted following either the [*min*] or the [*Max*] selecting policies. The extraction must execute the *Distinct* operation, which is defined as follows.

Definition 4 (Similarity-set extraction). *A ξ-simset $\hat{S}^{\xi}$ is extracted from a set* $\mathsf{S} \subset \mathbb{S}$ *existing in a metric domain $M = \langle \mathbb{S}, d \rangle$ by a mapping Distinct* : $\mathsf{S} \times \xi \times \lambda \mapsto \hat{S}^{\xi}$*, where the result has no pair of elements $s_i, s_j \in \hat{S}^{\xi}$, such that $s_i \hat{=}^{\xi} s_j$, and for every element $s_k \in \mathsf{S}$ there is at least one element $s_l \in \hat{S}^{\xi}$, such that $s_k \hat{=}^{\xi} s_l$. Policy λ can be either* [*min*] *or* [*Max*]*, and we say that $\hat{S}^{\xi}$ is an extraction of* S *under policy λ for the similarity threshold ξ.*

Following this definition, the ξ-similarity graph $\hat{G}^{\xi}(\hat{S})$ is a maximum/minimum independent dominant set of graph $\hat{G}^{\xi}(\mathsf{S})$. Notice that although a ξ-simset $\hat{S}^{\xi} \subseteq \mathsf{S} \subset \mathbb{S}$ exists independent from S, an extracted ξ-simset is associated to S. Also, note that the function *Distinct* generates possibly non-univocal queries, that is, two extractions with the same parameters can produce distinct results even if the database does not change, due to the NP-hard problem complexity involved when finding independent dominant sets, as discussed in Section 2. We name the set of all possible extractions of ξ-simsets from S as its ξ-Similarity cover, which is defined as follows.

Definition 5 (Similarity-cover). *The power set $P\hat{S}^{\xi}(\mathsf{S})$, named as the ξ-Similarity Cover of a set $\mathsf{S} \subset \mathbb{S}$ existing in a metric domain $M = \langle \mathbb{S}, d \rangle$, is the set of all ξ-similarity-sets that can be extracted from* S. *The power set $\lambda P\hat{S}^{\xi}(\mathsf{S}) \subseteq P\hat{S}^{\xi}(\mathsf{S})$ includes all ξ-simsets that can be extracted from* S *following policy λ.*

For example, the power set $[min]P\hat{S}^{2}(\mathsf{S})$ for the example dataset S, shown in Figure 1, under policy $[min]$ and using the similarity threshold $\xi = 2$ is $[min]P\hat{S}^{2}(\mathsf{S}) = \{\{p_2, p_5\}\}$, as there is only one possible ξ-simset extraction from S with the minimum number of elements 2. However, $[Max]P\hat{S}^{2}(\mathsf{S}) = \{\{p_1, p_3, p_4\}, \{p_3, p_4, p_5\}\}$, since both ξ-simsets have the same, largest cardinality 3.

To include support for similarity sets in a DBMS, the ξ-simset extraction can be seen as a selection operation: it selects every tuple from a relation such that the value of a given attribute is in the corresponding ξ-simset. However, the algebraic selection operator, represented by σ in the Relational Model, selects each tuple based solely on the attribute values existing in that tuple. Distinctly, the extraction operator depends on values not only from the current tuple, but from the entire active attribute domain. Thus, we assume that "extract" is a new unary operator, and we use the following notation to represent the ξ-simset extraction operator in the Relational Algebra, according to Definition 4: $\hat{\sigma}_{(\mathsf{A},d,\xi,\lambda)}\mathcal{T}$, where $\mathcal{T}$ is a database relation, A is the attribute (or set of attributes) in $\mathcal{T}$ that is employed to perform the extraction, d is the distance function of the metric space for the domain of A, ξ is the similarity threshold and λ is the extraction policy, which can assume either the min or the Max policies.

5 The *Distinct* Algorithm

The problem of extracting a ξ-simset $\hat{S}^{\xi}$ from a set S involves representing in the ξ-similarity graph $\hat{G}^{\xi}(\mathsf{S})$ the set of all ξ-similarities occurring among pairs of elements from S. It can be inferred from Definitions 2, 3 and 4 that extracting SimSets is therefore closely related to the concept of independent dominant sets described in Section 2: *The vertices of an independent dominant set extracted from graph $\hat{G}^{\xi}(\mathsf{S})$ define one ξ-simset $\hat{S}^{\xi}$ for* S. Finding independent dominant sets is a NP-hard optimization problem and may take a polinomial time to solve it. To complete the formulation of our SimSets, in this section we propose the algorithm *Distinct*, an approximate solution to quickly and accurately find maximum and minimum independent dominant sets, thus giving support to the Similarity-set extraction using both the [Max] and [min] policies.

The algorithm *Distinct*$(\mathsf{S}, \xi, \lambda)$ generates approximate independent dominant sets guaranteeing the properties of SimSets. Given the maximum radius ξ among elements

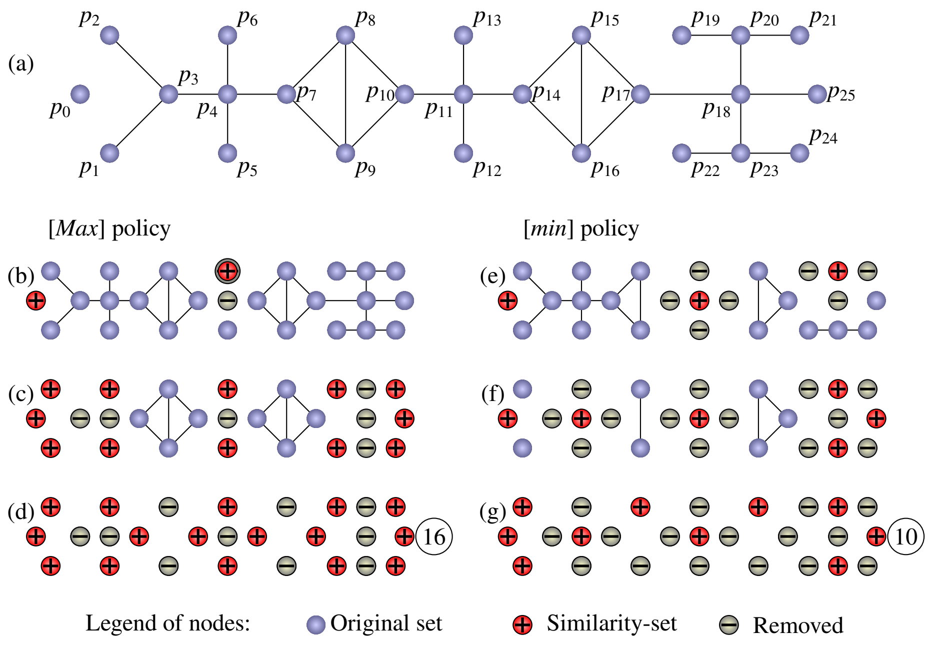


Fig. 2. Steps performed by algorithm $\hat{S}^{\xi} = Distinct(\mathsf{S}, \xi, \lambda)$ to obtain a ξ-simset from $\hat{G}^{\xi}$, using policy λ. (a): The original dataset. (b), (c) and (d): Steps to obtain the ξ-simset using the [*Max*] policy. (e), (f) and (g): Steps to obtain the ξ-simset using the [*min*] policy.

that are considered "too similar", it can always extract a ξ-simset from a dataset S following either the [*min*] or the [*Max*] policy. In a nutshell, the *Distinct* algorithm is twofold:

1. Generate a ξ-similarity graph $\hat{G}^{\xi}$ from a metric dataset S: $\hat{G}^{\xi} = \text{SimGraph}(\mathsf{S}, \xi)$;
2. Obtain the ξ-similarity-set $\hat{S}^{\xi}$ from graph $\hat{G}^{\xi}$, following the $\lambda \in \{[min], [Max]\}$ policies: $\hat{S}^{\xi} = Distinct(\mathsf{S}, \xi, \lambda)$.

The first phase generates a ξ-similarity graph $\hat{G}^{\xi}$ from the metric dataset S. It can be performed executing an auto range-join with the range radius as the ξ value, using any of the several range join algorithms available in the literature. Each element $s_i \in \mathsf{S}$ becomes a node in the graph, and the auto range-join creates an edge linking every pair (s_i, s_j) whenever $d(s_i, s_j) \leq \xi$. The second phase must extract a similarity-set *SimSet* using graph $\hat{G}^{\xi}$, based on either the [*min*] or the [*Max*] policy. We propose the Distinct algorithm to tackle that problem, which is described in Algorithm 1. In order to get an intuition on how it works, we use the graph shown in Figure 2(a).

According to Algorithm 1, isolated nodes such as node p_0 in Figure 2(a) correspond to elements in the original dataset having no neighbors within the ξ-similarity threshold, so they are sent to the ξ-simset and dropped from the graph, in Line 4. The algorithm is based on the node degrees, which are evaluated in lines 6 to 8. During the execution, as nodes are being dropped from the graph, the variables Totals, $\mathsf{Min0}$ and lNodes

Algorithm 1. $\hat{S}^{\xi}$ = *Distinct*(S, ξ, λ)

```
Input: Dataset S, Radius $\xi$, Policy $\lambda$
Output: $\xi$-simset SimSet
1  Array Totals [ |S| ];
2  Set INodes, JNodes, SimSet;
3  SimSet ← $\emptyset$; GraphS ← $SimGraph(S, \xi)$;
4  foreach node $s_i$ in GraphS with zero degree do
5      remove $s_i$ from GraphS and insert it into SimSet;
6  Totals [i]← degree of node $s_i$ in graph GraphS;
7  Min0 ←smallest value in Totals;
8  INodes ← nodes having degree Min0;
9  while edges exist in GraphS do
10     if $\lambda$ is Max then
11         randomly select a node $s_i$ from INodes;
12         remove every node linked to $s_i$ from GraphS;
13         remove $s_i$ from GraphS and insert it into SimSet;
14     if $\lambda$ is min then
15         JNodes ← set of nodes linked to nodes in INodes;
16         foreach $s_i$ in JNodes do
17             $C1[i]$ =number of nodes with degree Min0 linked to $s_i$;
18         foreach $s_i$ in JNodes do
19             $C2[i] = -C1[i]$;
20             foreach $s_j$ in JNodes connected to $s_i$ do
21                 $C2[i] = C2[i] + C1[j]$;
22         Min1 ←smallest value in $C2$;
23         randomly select a node $s_i$ from JNodes having $C2[i]$ = Min1;
24         remove every node linked to $s_i$ from GraphS;
25         remove $s_i$ from GraphS and insert it into SimSet;
26     update Totals;
27     foreach Node $s_i$ in GraphS with zero degree do
28         remove $s_i$ from GraphS and insert it into SimSet;
29     update Min0 and INodes;
30 end
```

are maintained updated, in Lines 26 and 29. Nodes that become isolated during the execution are moved from the graph into the ξ-simset, in Line 27. Notice that evaluating the node degrees requires a full scan over the dataset, but thereafter, as those variables are locally maintained, no scan is required anymore.

Let us first exemplify how the $\lambda = [Max]$ policy is evaluated over our example graph from Figure 2. In the first iteration of Line 9, nodes $\{p_1, p_2, p_5, p_6, p_{12}, p_{13}, p_{19}, p_{21}, p_{22}, p_{24}, p_{25}\}$ have the smallest count of edges (degree). So, one of them is removed from the graph and inserted into the result in Line 10. Figure 2(b) depicts the case when one of such nodes (in this example node p_{13}) is inserted in the result, and all the nodes linked to it are removed from the graph (in this example node p_{11} is removed). Figure 2(c) shows the graph after all of those nodes have been inserted into the result and the nodes linked to them removed, in successive interactions of Line 9. Now, as nodes $\{p_7, p_{10}, p_{14}, p_{17}\}$ have two edges each, they are inserted into the result in the next iterations of Line 9, as shown in Figure 2(d). As no other node remains in the graph, the result set is the final ξ-simset, which in this example has 16 nodes. In fact, no SimSet with more than 16 nodes can exist.

Let us now exemplify how the algorithm works for policy $\lambda = [min]$, processed in Line 14. Once more we employ the graph in Figure 2(a) to be our running example. In the first iteration of Line 9, the nodes having the smallest count of edges are the same ones selected for the λ =[*Max*] policy. However, now these nodes are employed in Line 15 to select those nodes directly linked to them, which results in the set $\{p_3, p_4, p_{11}, p_{18}, p_{20}, p_{23}\}$. In Line 16, it is counted how many neighbors each of the selected nodes have in INodes, resulting in the counting vector $C1 = \{p_3 : 2, p_4 : 2, p_{11} : 2, p_{18} : 1, p_{20} : 2, p_{23} : 2\}$. Following, these counts are discounted from the total number of nodes with minimum degree linked to each of their neighbors in Line 18, resulting in the counting vector $C2 = \{p_3 : 0, p_4 : 0, p_{11} : -2, p_{18} : 3, p_{20} : -1, p_{23} : -1\}$. As p_{11} is the single node having the minimum count (-2) in $C2$, it is moved into the result, also removing all of its neighbors from the graph, what is performed in Lines 22 to 25.

In the next iteration of Line 9, the same values result in the counting vectors $C1$ and $C2$, this time without node p_{11}. Thus, now both nodes p_{20} and p_{23} tie with the minimum count -1, so either of them can be inserted into the result. Figure 2(e) shows the graph just after this second iteration. Proceeding, nodes p_5 and p_{23} are moved to the result, as shown in Figure 2(f). Finally, nodes p_9 and p_{15} are successively moved, so the *SimSet* that corresponds to the final graph shown in Figure 2(g) is obtained, which in this example has 10 nodes. In fact, no *SimSet* with fewer than 10 nodes can be extracted.

Whenever a node is moved from the graph into the result, the graph and the corresponding degree of the remaining nodes are updated in Line 26. Disconnected nodes are also moved to the resulting *SimSet* in Line 27. Algorithm 1 was canonically stated for the sake of simplicity, but it can be easily improved. First of all, Line 26 can be performed locally to Lines 12 and 24, precluding the need to re-process large graphs. Second, when the minimum degree is pursued for the $\lambda = [min]$ policy, all nodes in the INodes set can be removed at once (provided whenever a node is removed from the graph it is also removed from the remaining INodes), thus reducing the overhead of Line 27. In order to improve the access to INodes and JNodes, it is recommended that both be kept in bitmap indexes.

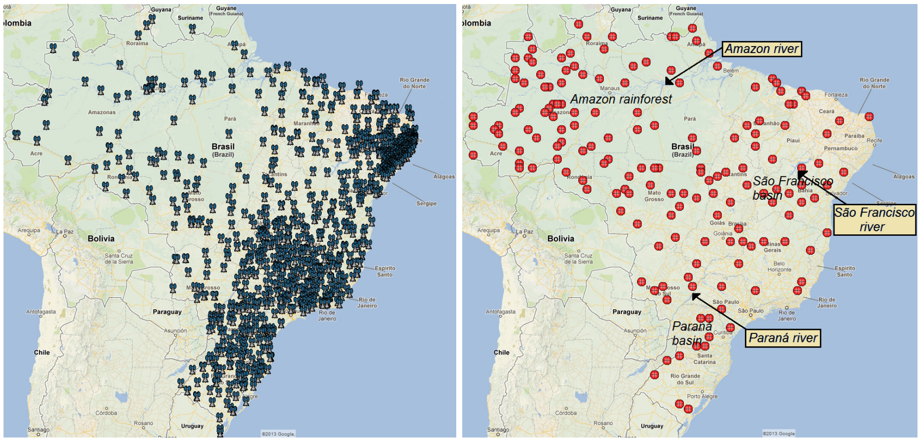
It is important to highlight that this algorithm does not aim at finding "the" maximum independent dominant sets of a graph (or its corresponding minimum counterpart), but a very good approximation of them. We evaluated an implementation of *Distinct* in C++ that really chooses random nodes at steps 11 and 23, and we executed it one thousand times with the same setting over several, varying sized graphs, and for every set of executions it always returned answers with the same number of nodes. This result gives us a strong indication that, although it does not guarantee that the theoretical solution is always found, the algorithm in practice finds it almost always.

6 Experiments

We performed experiments to validate our proposal using an implementation of the $\hat{S}^{\xi} = Distinct(S, \xi, \lambda)$ algorithm in C++, exploring meteorological data.

We use as raw data the results of climate forecasting models. A climate model is a mathematical formulation of atmospheric behavior based on Navier-Stokes equations over a detailed representation of the land. The equations are iteratively evaluated over the atmosphere segmented as cubes of fixed sizes, executing the simulation in time steps that span several years in few hours steps. A simulation run usually takes several weeks of processing in large supercomputers of the climate research centers. For the experiment reported here, we used the Eta-CPTEC climate model [1]. The model analyzed the entire South America creating a 0.4×0.4 degrees (Latitude $\times$ Longitude) grid.

We took the first simulated value of each geographical coordinate for three atmospheric measures: maximum temperature, minimum temperature and precipitation, for each month of year 2012. Thus, there are 9507 geographic points, and each point is represented by an array of $3 * 12 = 36$ climate measurements, plus its latitude and longitude. To evaluate the similarity, we used the weighted Euclidean distance where the geographical coordinates weights 25% (12.5% each) and the climate measures equally share the other 75%. Our objective is to answer the following question: "*If we want to validate the climate model (and improve our ability to make accurate climate predictions), where should we put a limited number of meteorological stations across the Brazilian territory so that we can take the measurements that are the most distinct from the full set of grid points?*" The similarity metric was defined to measure the distinctness of each point, and we included the geographical coordinates to force the distribution of sensor stations over the full territory.



(a) Real station network. (b) Resulting *SimSet* [*min*] policy.

Fig. 3. (a) Distribution of the existing network of meteorological stations in Brazil. (b) Distribution of stations selected from the Climate dataset using Distinct with min policy.

For comparison purposes, Figure 3(a) shows the distribution of the real meteorological network of sensor stations currently existing in Brazil [2]. As it can be seen, most of the stations are concentrated on regions of high population density (south and east regions). *SimSets* may benefit from this scenario, pointing out which stations could be turned off and suggesting the best locations to install new ones.

Figure 3(b) shows a *SimSet* obtained by the *Distinct*$(\mathsf{S}, \xi, \lambda)$ algorithm, using $\xi = 1.0$ and $\lambda = min$, which suggests the best places to install sensors over the Brazilian territory. The Distinct algorithm took approximately 10 minutes to execute in a machine with Intel Pentium I7 2.67GHz processor and 8Gb RAM. This setting produces a smaller number of sensors, compared to the number of existing ones, and we set the experiment in this way so that it is easy to visualize and interpret the results in the low resolution we are able to plot in this paper. *SimSets* with higher cardinalities can be generated by changing those parameters. In fact, we generated several different *SimSets* extensively modifying the parameters. However, although the number of suggested stations and their exact location varies between runs with distinct settings, the overall distribution closely follows the one shown in Figure 3(b) and the corresponding analyses always remain equivalent.

Analyzing the sensors selected, we see that forests (Amazon rainforest on the northwest and the Highlands in the central region) require more sensors than the desert or semi-arid areas (north and northeast). This finding reveals that the best places to build stations with regard to climate monitoring were not followed by the existing network, which was historically driven from the economics and the easiness of access point of view. In fact, interestingly, using the model, our *SimSets* kind of "refuse" to put sensors close to densely populated areas (cities of São Paulo, Rio de Janeiro and Recife). We also see in Figure 3(b) that the rivers indeed require close monitoring, like those close to the Amazon River and along the São Francisco River at the east region. Moreover, the headwaters of the drainage basins were selected as requiring more stations than the main water stream (as it can be seen at the Amazon, São Francisco and Paraná basins). Finally, note that all of those analyses are consistent with the theory and are confirmed by meteorologists.

7 Conclusion

In this paper we introduced the novel concept of "similarity-sets", or *SimSets* for short. A "set" is a data structure storing a collection of objects that has no duplicates. A *SimSet* $\hat{S}^{\xi}$ is a data structure without any pair of elements $s_i, s_j \in \hat{S}^{\xi}$ such that $d(s_i, s_j) \leq \xi$, where $d(s_i, s_j)$ measures the (dis)similarity between both elements and ξ is a similarity threshold. Thus, the ξ-simset concept extends the idea of "sets" in a way that SimSets with similarity threshold ξ do not include two elements more similar than ξ, that is, in a ξ-simset $\hat{S}^{\xi}$ there are no two elements $s_i, s_j \in \hat{S}^{\xi}$ such that $s_i \hat{=}^{\xi} s_j$. Similarity-sets are in fact a generalization of sets: a set is a ξ-simset where $\xi = 0$, because the $\hat{=}^{\xi}$ comparison operator is equivalent to "=" for $\xi = 0$. Therefore, the concept of SimSets is adequate to perform similarity queries and gives the underpinning to include them into the Relational Model, and in most DBMS.

[2] Source: `http://www.agritempo.gov.br/estacoes.html`

The operator to extract a ξ-simset from a metric dataset S was presented as the $\hat{S}^{\xi} = \mathit{Distinct}(\mathsf{S}, \xi, \lambda)$ algorithm, that is able to generate ξ-simsets following either the [*min*] or the [*Max*] policies and guaranteeing the $\hat{S}^{\xi}$ extraction properties to validate our proposal. We analyzed sensor networks using *SimSets* to identify sensors that recurrently report similar measurements, aimed at turning some of them off for energy saving. Specifically, we reported experiments on real data from sensor networks of ground-based weather stations.

The concepts of ξ-simsets presented here fulfills the fundamental requirements to use them to query real world applications. However, more elaborated theoretical studies will be developed in future work, including the definition and property statement of operations with ξ-simsets as well as the definition of set theoretical operators that take into account maintaining the relationship among the original set S and its extracted ξ-simset $\hat{S}^{\xi}$.

References

1. Black, T.L.: The new NMC mesoscale eta model: Description and forecast examples. Weather and Forecasting 9, 265–284 (1994)
2. Budikova, P., Batko, M., Zezula, P.: Query language for complex similarity queries. In: Morzy, T., Härder, T., Wrembel, R. (eds.) ADBIS 2012. LNCS, vol. 7503, pp. 85–98. Springer, Heidelberg (2012)
3. Garcia-Molina, H., Ullman, J.D., Widom, J.: Database Systems: The Complete Book, 2nd edn. Prentice Hall Press, Upper Saddle River (2008)
4. Godsil, C., Royle, G.: Algebraic Graph Theory. Springer (April 2001)
5. Henzinger, M.: Finding near-duplicate web pages: a large-scale evaluation of algorithms. In: ACM SIGIR, pp. 284–291 (2006)
6. Hjaltason, G.R., Samet, H.: Index-driven similarity search in metric spaces (survey article). ACM Transactions on Database Systems 28(4), 517–580 (2003)
7. Kuncheva, L.I., Jain, L.C.: Nearest neighbor classifier: Simultaneous editing and feature selection (1999)
8. Lin, F.-S., Chiu, P.L.: A near-optimal sensor placement algorithm to achieve complete coverage-discrimination in sensor networks. IEEE Communications Letters 9(1), 43–45 (2005)
9. Pedreira, O., Brisaboa, N.R.: Spatial selection of sparse pivots for similarity search in metric spaces. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 434–445. Springer, Heidelberg (2007)
10. Silva, Y.N., Aly, A.M., Aref, W.G., Larson, P.-A.: Simdb: a similarity-aware database system. ACM SIGMOD, 1243–1246 (2010)
11. Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Yu, P.S., Zhou, Z.-H., Steinbach, M., Hand, D.J., Steinberg, D.: Top 10 algorithms in data mining. Knowledge and Information Systems 14(1), 1–37 (2007)
12. Xiao, C., Wang, W., Lin, X., Yu, J.X., Wang, G.: Efficient similarity joins for near-duplicate detection. ACM Transactions on Database Systems, 36,15:1–15:41 (2011)
13. Zhu, S., Wu, J., Xiong, H., Xia, G.: Scaling up top-k cosine similarity search. Data and Knowledge Engineering 70(1), 60–83 (2011)

Similarity Search on Uncertain Spatio-temporal Data

Johannes Niedermayer[1], Andreas Züfle[1], Tobias Emrich[1], Matthias Renz[1], Nikos Mamoulis[2], Lei Chen[3], and Hans-Peter Kriegel[1]

[1] Institute for Informatics, Ludwig-Maximilians-Universität München
{niedermayer,zuefle,emrich,renz,kriegel}@dbs.ifi.lmu.de
[2] University of Hong Kong
nikos@cs.hku.hk
[3] Hong Kong University of Science and Technology
leichen@cse.ust.hk

Abstract. In this work, we address the problem of similarity search in a database of uncertain spatio-temporal objects. Each object is defined by a set of observations ((time,location)-tuples) and a Markov chain which describes the objects uncertain motion in space and time. To model similarity - which is an important building block for many applications such as identifying frequent motion patterns or trajectory clustering - we employ the well-known Longest Common Subsequence (LCSS) measure, which becomes a distribution on uncertain spatio-temporal data (ULCSS). We show how the aligned version (without time shifting) of the ULCSS can be exactly computed in PTIME, which is also verified by extensive experiments.

1 Introduction

Similarity search on trajectory data has an increasing number of applications, especially after the widespread availability of location data, such as GPS tracks. Exemplarily, data analysis tasks such as identifying frequent motion patterns or trajectory clustering require finding objects that moved in a similar way or followed a certain motion pattern. A number of similarity measures have been proposed for trajectory data. One of these is the Longest Common Subsequence (LCSS). The LCSS between two trajectories (i.e., moving objects) can be interpreted as the maximum amount of time the two objects were located at the same position. However, most of the previous work on similarity search in trajectory databases assumes the data to be certain or deterministic, which is not the case in many real applications. For example, even though we can get snapshots of the positions of a mobile object through RFID technology, the trajectory data is incomplete and uncertain: because the locations of the object between two consecutive RFID readers are unknown, they have to be derived from the observations which introduces uncertainty. Therefore, it is essential to develop new techniques to find similar trajectories on uncertain data. In this paper we study how the LCSS can be extended to apply to uncertain trajectories (ULCSS). Since in our scenario the exact motion of an object is unknown, we can model the ULCSS as a distribution of all possible LCSS results. The LCSS over uncertain data has many applications. For example, it can be used to evaluate the spread of flu or other diseases. Suppose that an object was diagnozed with a serious communicable disease (*source object*). To curtail such diseases, a facebook app could identify all individuals possibly been infected by the source object. Let us assume that enough virus cells are transmitted between two individuals if the two persons share the same location for at least k points in time. To identify individuals that might have been infected, the app could run an ULCSS query to find all objects having a large enough probability of being at the same location as the source object for at least k points in time, warning the affected persons.

2 Problem Definition

A spatio-temporal database $\mathcal{D}$ stores triples (*oid*, *location*, *time*), where *oid* $\in \{o_1, ..., o_{|\mathcal{D}|}\}$ is a unique object identifier, *location* $\in \mathcal{S}$ is a spatial position and *time* $\in \mathcal{T}$ is a point in time. Semantically, each such triple corresponds to an *observation* that object o_i has been seen at some location at some time. In $\mathcal{D}$, an object can be described by a function $tr_{o_i} : \mathcal{T} \rightarrow \mathcal{S}$ that maps each point in time to a location in space; this function is called *trajectory*. In this work, we assume a discrete time domain $\mathcal{T} = \{0, \ldots, n\}$. Thus, a trajectory becomes a sequence, i.e., a function on a discrete and ordinal scaled domain. Furthermore, we assume a discrete state space of possible locations (*states*) $\mathcal{S} = \{s_1, ..., s_{|\mathcal{S}|}\} \subset \mathbb{R}^d$.

Uncertain Trajectory Model. Since we consider uncertainty, a trajectory may not be modeled by a simple single path but rather by a (possibly large) set of paths, i.e., a set of possible worlds. In particular, let $\mathcal{D} = \{o_1, ..., o_{|\mathcal{D}|}\}$ be a database containing $|\mathcal{D}|$ uncertain moving objects. For each object $o \in \mathcal{D}$ we store a set of observations $\Theta^o = \{< t_1^o, \theta_1^o >, < t_2^o, \theta_2^o >, \ldots, < t_{|\Theta^o|}^o, \theta_{|\Theta^o|}^o >\}$ where $t_i^o \in \mathcal{T}$ denotes the time and $\theta_i^o \in \mathcal{S}$ the location of observation Θ_i^o. W.l.o.g. let $t_1^o < t_2^o < \ldots < t_{|\Theta^o|}^o$. According to [EKM+12], we can interpret the location of an uncertain spatio-temporal object $o \in \mathcal{D}$ at time t as a realization of a random variable $o(t)$. Given a time interval $[t_0, t_1]$, the set of corresponding uncertain locations becomes an uncertain trajectory. A formal definition of uncertain trajectories can be found in [EKM+12].

This technique allows us to assess the probability of a possible trajectory (i.e., a realization of all random variables). In this work we follow the approaches from [EKM+12] and employ the first-order Markov Chain model as a specific instance of a stochastic process. Markov Chains have been employed for modelling human movement in [MJS11]; furthermore, according to [EKM+12] they enable query processing of a class of queries in polynomial time while following possible worlds semantics. A Markov chain T is a matrix containing the conditional *transition probabilities* $T_{ij}^o(t) := P(o(t+1) = s_j | o(t) = s_i)$ of o from state s_i to state s_j at a given time t. Let $\boldsymbol{s}^o(t) = (p_1, \ldots, p_{|\mathcal{S}|})^T$ be the distribution vector of an object o at time t, where $\boldsymbol{s}_i^o(t) = P(o(t) = s_i)$. The distribution vector $\boldsymbol{s}^o(t+1)$ can be inferred from $\boldsymbol{s}^o(t)$ as follows: $\boldsymbol{s}^o(t+1) = T^o(t)^T \cdot \boldsymbol{s}^o(t)$

Similarity between Uncertain Trajectories. Given two database objects o_1 and o_2, our goal is to assess the similarity between these two uncertain objects by employing the LCSS [VGK02]. Let A and B be two trajectories of moving objects with size n and m respectively, where $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$. Let $Head(A) :=$

$(a_1, \ldots, a_{n-1})$. Given an integer δ and a real number ϵ, the Longest Common Subsequence is defined as follows: $LCSS_{\delta,\epsilon}(A, B) :=$

$$\begin{cases} 0 \text{ if } A = \emptyset \text{ or } B = \emptyset, \\ 1 + LCSS_{\delta,\epsilon}(Head(A), Head(B)) \quad \text{if } dist(a_n - b_m) < \epsilon \text{ and } |n - m| \leq \delta \\ max(LCSS_{\delta,\epsilon}(Head(A), B), LCSS_{\delta,\epsilon}(A, Head(B))) \quad otherwise \end{cases}$$

Parameter ϵ constraints the spatial distance between two locations in order to match in space; in many applications, objects rarely visit the same locations, but being "close enough" is equivalent to meeting. Parameter δ controls how far in time we can expand in order to match a given point from one trajectory to a point in another trajectory. In this work, we aim at working towards efficiently computing $LCSS_{\delta,\epsilon}(o_1, o_2)$ for two uncertain trajectories o_1 and o_2. According to possible world semantics, the result of $LCSS_{\delta,\epsilon}(o_1, o_2)$ is not a single scalar, but rather a probability density function on $\mathbb{N}$, mapping each possible outcome $k \leq min(length(o_1), length(o_2))$ to a probability $P(LCSS_{\delta,\epsilon}(o_1, o_2) = k)$.

Definition 1 (ULCSS). *Let o_1 and o_2 be two uncertain trajectories. The Uncertain Longest Common Subsequence (ULCSS) between o_1 and o_2 is a random variable, defined by the following probability density function:*

$$ULCSS_{\delta,\epsilon}(o_1, o_2) : \mathcal{D} \times \mathcal{D} \to (\mathbb{N} \to [0, 1] \in \mathbb{R})$$

$$ULCSS_{\delta,\epsilon}(o_1, o_2) := pdf(x \in \mathbb{N}) = P(LCSS_{\delta,\epsilon}(o_1, o_2) = x)$$

However, calculating the exact distribution and the expected value of the length of the LCSS between two random sequences can currently only be achieved by employing exponential algorithms [FL08]. Therefore, we study the special case of ULCSS, where $\delta = 0$; for this case, we propose a PTIME algorithm for its exact computation:

Definition 2 (UALCSS). *Let o_1, o_2 be two uncertain trajectories. The Uncertain Aligned Longest Common Subsequence is defined by $UALCSS_\epsilon(o_1, o_2) = ULCSS_{0,\epsilon}$ (o_1, o_2).*

3 Related Work

The stochastic model used in this paper is taken from [EKM+12], where window queries in an uncertain setting have been addressed. Existing approaches for measuring trajectory similarity mainly adapt to trajectories in a certain setting. [VGK02] relaxed this assumption by taking the effect of noise into account. Uncertain trajectory similarity has been investigated in the context of trajectory clustering by [PKK+09]. However this work addresses *position* (e.g. GPS) uncertainty instead of *motion* uncertainty, where neither position nor motion are known at a given point in time, but might have been known at some time in the past. Recently, [LG12] addressed the problem of computing the windowed LCSS on strings. However, the different characters in a word are drawn independently in this context, whereas the state at time t depends on the previous state in our problem setting. Statistical work such as [AW85] provides statistical approximations of the length of the (unaligned) longest common subsequence on the Markov model, e.g. under the assumption that the underlying Markov chains are aperiodic and irreducible and if the length of the underlying sequence is large. Another common application area of Markov models is in the area of bioinformatics where gene sequences have to be matched (e.g.[HK96]). When employing hidden Markov models, viterbi-like algorithms and extensions that can handle insertions and deletions (c.p. e.g. [AV98]) are usually employed for computing the maximum likelihood, and not a distribution. Besides their advantage of estimating the *edit distance* between sequences, these approaches can only be used to match sequences to Markov chains but not two Markov chains. There further exists an exponential approach for calculating the exact distribution and the expected value of the length of the LCSS between two random sequences [FL08]. This algorithm neither considers observations, nor the amount of time shifting δ. Furthermore, the size of its transition matrix depends on the length of the time interval for which the LCSS has to be computed.

4 UALCSS Computation

Overview. While it remains unsolved how to compute the exact ULCSS in the general case, in this section we show how to exactly compute the UALCSS (ULCSS for the special case of $\delta = 0$) between two uncertain spatio-temporal objects, which represents the pdf over all possible lengths of the LCSS between the two evaluated objects with a polynomial time algorithm. The UALCSS is relevant to many spatial applications, like the infection application mentioned in the introduction; virus particles in a droplet infection can only be spread through space, but not through time.

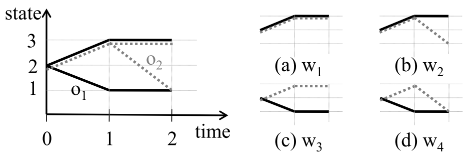


Fig. 1. Possible worlds $\{w_{1-4}\}$ of two uncertain objects

Figure 1 (left) shows the uncertain trajectory of objects o_1 (represented by the solid line) and o_2 (represented by the dotted line). From these, we derive four possible worlds as illustrated in Figure 1 (right). We can see that in w_1 the (certain) LCSS equals 3, in worlds w_2 and w_4 LCSS=2, while in w_3 LCSS=1. If we assume, in this example, that for each object each alternative trajectory has a probability of 0.5, we get a probability vector [0,0.25,0.5,0.25] for the UALCSS, where the kth element in the list denotes k hits between two paths. Clearly, such an approach of enumerating all possible worlds, and aggregating their probabilities is not a viable option, since in general, the number of possible trajectories of an uncertain trajectory is exponential in the length of the uncertain trajectory.

Algorithm. Algorithm 1 is a pseudocode of the UALCSS algorithm. For computing UALCSS, we have to take certain dependencies of the two objects into account, i.e., the relative position of o_1 to o_2 at time $t = 0$ (w.l.o.g. we assume that the first observations of o_1 and o_2 are at time $t = t_1^{o_1} = t_1^{o_2} = 0$) will affect the length of the UALCSS at a later time $t > 0$. For this reason, we have to take the conditional probabilities of object o_1 being in state s_i when o_2 is in state s_j into account: At the initial time $t = 0$ we assume both object locations to be independent, and therefore we can write $P(o_1(0) = s_i \wedge o_2(0) = s_j) = P(o_1(0) = s_i) \cdot P(o_2(0) = s_j)$. Let $M(t)$ be a probability matrix with $M_{ij}(t) = P(o_1(t) = s_i \wedge o_2(t) = s_j)$, denoting that the corresponding objects are within state s_i and s_j at time t. The matrix $M^0(0)$ can be easily computed as follows (the superscript 0 denotes the numbers of hits gained so far): $M^0(0) = \boldsymbol{s}^{o_2}(0) \cdot \boldsymbol{s}^{o_1}(0)^T$. This is the case because we have $M^0_{ij}(0) = \boldsymbol{s}^{o_1}(0)_i \cdot \boldsymbol{s}^{o_2}(0)_j = P(o_1(0) = s_i) \cdot P(o_2(0) = s_j)$. The elements $M^0_{ii}(0)$ denote the probabilities that both uncertain objects are located within the same state i at time 0, increasing the longest common subsequence by 1, such that these *possible worlds* have to be marked. This can be simply achieved by moving them into a second matrix $M^1(0)$, where $M^1_{ii}(0) = M^0_{ii}(0)$ and $M^1_{ij}(0) = 0$ for $i \neq j$. Besides, the shifted elements have to be deleted from $M^0(0)$ by performing $M^0_{ii}(0) = 0$. Now both matrices contain possible worlds, split by their number of hits.

Algorithm 1. UALCSS(o_1, o_2, t_{max})

```
 1: $M^0 = \boldsymbol{s}^{o_2}(0) \cdot \boldsymbol{s}^{o_1}(0)^T$
 2: $M^1_{i \neq j} = 0$
 3: $M^1_{ii} = M^0_{ii}$
 4: $M^0_{ii} = 0$
 5: for $t = 1; t \leq t_{max}; t++$ do
 6:     for $k = t; k \geq 0; k--$ do
 7:         $M^k = T^{o_1}(t-1)^T \cdot M^k \cdot T^{o_2}(t-1)$
 8:         $M^{k+1}_{ii} = M^{k+1}_{ii} + M^k_{ii}$
 9:         $M^k_{ii} = 0$
10:     end for
11:     if $\exists t^{o_1}_i : t^{o_1}_i = t \vee \exists t^{o_2}_j : t^{o_2}_j = t$ then
12:         reweight($\{M^k\}, \theta^{o_1}_i, \theta^{o_2}_j$)
13:     end if
14: end for
15: $p = Array[t_{max} + 1]$
16: for $t = 0; t \leq t_{max}; t++$ do
17:     $p_k = |M^k|_{L_1}$
18: end for
19: return $p$
```

After initialization, this method can be applied in a similar manner to compute the equivalence classes of possible worlds within each time $t \neq 0$, which is achieved by updating all state matrices $M^k(t-1)$. As a first step, the states of o_1 and o_2 in $M^k(t-1)$ have to be transitioned. Given a state vector $\boldsymbol{s}^{o_i}(t-1)$, this transition is usually performed by multiplying $\boldsymbol{s}^{o_i}(t-1)$ with its corresponding, pre-determined, transition matrix $T^{o_i}(t-1)$, i.e., $\boldsymbol{s}^{o_i}(t) = T^{o_i}(t-1)^T \cdot \boldsymbol{s}^{o_i}(t-1)$. However, in our scenario, we do not have a single state vector, but a state matrix $M^k(t-1)$, containing conditional probabilities of both objects. The elements in this matrix have to be transitioned according to both transition matrices $T^{o_1}(t-1)$ and $T^{o_2}(t-1)$. It can be proven that $M^k(t) = T^{o_1}(t-1)^T \cdot M^k(t-1) \cdot T^{o_2}(t-1)$.

After performing the transition, the matrix element $M^k_{ij}(t)$ again contains the conditional probabilities at time t that o_1 is in state j while o_2 is in state i. After transitioning, the hits are extracted from the matrix, by shifting the diagonal elements to $M^{k+1}_{ii}(t)$ and removing them from $M^k_{ii}(t)$. Therefore each of the t transitions leads to at most one additional matrix; thus, the total space complexity of this algorithm is at most $O(|S|^2 \cdot t)$ and the runtime complexity is $O(\Delta t^2 \cdot |S|^3)$, with Δt beeing the number of considered timesteps. In practice, these costs are much lower since vectors $\boldsymbol{s}^{o_i}(t)$ and $T^{o_i}(t)$ are both sparse and we can save space and computations by employing sparse matrix operations on compressed representations. After having completed t transitions, we can derive the probability distribution for the relative frequency of worlds that had a given number k of hits:

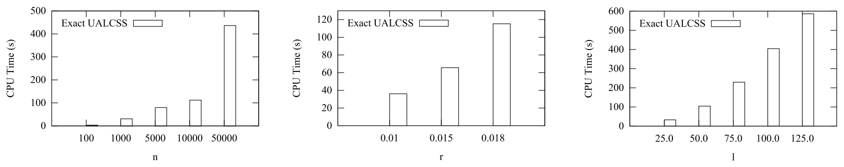


Fig. 2. Experimental Results

$P(|\{x \in \mathcal{T}|o_1(x) = o_2(x)\}| = k) = \sum_{\forall i,j} M^k_{ij}(t)$ Incorporating further observations (function *reweight()* in Algorithm 1) can be achieved as follows. Let us first assume that o_1 was observed at state s. Then all columns $j \neq s$ in M^k have to be set to 0 and all matrices have to be reweighted such that $\sum_{x \in t} M^k = 1$. Accordingly, if o_2 has been observed at a given state, the corresponding rows have to be set to zero. Furthermore, the algorithm can be easily adapted for $\epsilon > 0$. In this case, not only diagonal elements from M^k have to be shifted, but also further matrix elements that correspond to locations with a distance $\leq \epsilon$ to a given location of an uncertain object.

5 Experiments

The experiments are based on a discrete state space in the two-dimensional Euclidean space, consisting of n states. Each of these states is drawn uniformly from the $[0, 1]^2$ space. Afterwards, a graph was created from these states by connecting points with an Euclidean distance smaller than $r = \sqrt{\frac{b}{n*\pi}}$, with b being the average number of neighbours of a state, i.e. the branching factor. The graph's edge weights, i.e. the transition probabilities were assigned indirectly proportionally to the distance of a state to its neighbour, assuming that it is more probable that during a transition an object moves to a closer state than to a state further away. Based on the resulting transition matrix, a random trajectory was drawn to construct (certain) observations of an uncertain object, and every i-th point from this trajectory was used as an observation of the uncertain object. In the evaluation, we varied the number of states n from 100 to 50K (default 10000), the length l from 25 to 125 (default 50), and the range r from 0.01 to 0.075 (default $\sqrt{\frac{b}{n*\pi}}$). The interval between two observations is 10 timestamps. In the first experiment we aimed at varying the worlds *size* (n), keeping the graph's branching factor constant, while increasing r (the next experiment) can be interpreted as increasing the *resolution* of a world, i.e. the branching factor. As shown in Figure 2 (left), with increasing n, matrix operations become more costly such that the performance of the UALCSS drops. Varying the range of connectivity r (Figure 2 (center)) clearly shows a negative impact on the performance of the UALCSS algorithm. With a higher connectivity, the filling degree of transition matrices increases, such that more states can be reached in a shorter amount of time. Increasing the length of the time interval for which the UALCSS has to be computed (Figure 2 (right)) increases the number of iterations such that more matrix multiplications have to be performed. Note that the number of matrix multiplications for this algorithm is $O(t^2)$.

To compute the general ULCSS, we plan to investigate sampling techniques. The main problem for sampling approaches is to incorporate observations.

References

[AV98] Amengual, J.C., Vidal, E.: Efficient Error-Correcting Viterbi Parsing. IEEE Transactions on Pattern Analysis and Machine Intelligence 20, 1109–1116 (1998)

[AW85] Arratia, R., Waterman, M.S.: An Erdös-Rényi Law with Shifts (1985)

[EKM$^+$12] Emrich, T., Kriegel, H.-P., Mamoulis, N., Renz, M., Züfle, A.: Querying Uncertain Spatio-Temporal Data. In: Proc. ICDE (2012)

[FL08] Fu, J.C., Wendy Lou, W.Y.: Distribution of the length of the longest commmon subsequence of two multi-state biological sequences. Journal of Statistical Planning and Inference 138, 3605–3615 (2008)

[HK96] Hughey, R., Krogh, A.: Hidden Markov models for sequence analysis: extension and analysis of the basic method (1996)

[LG12] Li, Z., Ge, T.: Online windowed subsequence matching over probabilistic sequences. In: Proc. SIGMOD, pp. 277–288 (2012)

[MJS11] Moghadam, A., Jebara, T., Schulzrinne, H.: A markov routing algorithm for mobile DTNs based on spatio-temporal modeling of human movement data. In: Proc. WSIM (2011)

[PKK$^+$09] Pelekis, N., Kopanakis, I., Kotsifakos, E.E., Frentzos, E., Theodoridis, Y.: Clustering Trajectories of Moving Objects in an Uncertain World. In: Proc. ICDM, pp. 417–427 (2009)

[VGK02] Vlachos, M., Gunopulos, D., Kollios, G.: Discovering Similar Multidimensional Trajectories. In: Proc. ICDE, pp. 673–684 (2002)

List of Clustered Permutations for Proximity Searching[⋆]

Karina Figueroa[1] and Rodrigo Paredes[2]

[1] Facultad de Ciencias Físico-Matemáticas,Universidad Michoacana, Mexico
[2] Departamento de Ciencias de la Computación, Universidad de Talca, Chile
karina@fismat.umich.mx, raparede@utalca.cl

Abstract. The *permutation based algorithm* has been proved unbeatable in high dimensional spaces, requiring $O(|\mathbb{P}|)$ distance evaluations when solving similarity queries (where $\mathbb{P}$ is the set of permutants); but needs n evaluations of the permutant distance to compute the order to review the metric dataset, requires $O(n|\mathbb{P}|)$ space, and does not take much benefit from low dimensionality. There have been several proposals to avoid the n computations of the permutant distance, however all of them lost precision. Inspired in the *list of cluster*, in this paper we group the permutations and establish a criterion to discard whole clusters according the permutation of their centers. As a consequence of our proposal, we now reduce not only the space of the index and the number of distance evaluations but also the cpu time required when comparing the permutations themselves. Also, we can use the permutations in low dimensions.

1 Introduction

Several modern applications —for instance, pattern recognition or multimedia retrieval— require similarity retrieval systems to find relevant objects when solving a query. In these applications the pattern is the same, the search problem is often stated in terms of expensive comparison between two objects in a huge database.

The problem can be mapped into a metric space $(\mathbb{X}, d)$, where a metric d compares objects out of a universe $\mathbb{X}$ and reveals how close is an object with respect to other. This metric must satisfy the follow properties: positiveness $d(x, y) \geq 0$, symmetry $d(x, y) = d(y, x)$ and triangle inequality $d(x, y) \leq d(x, z) + d(z, y)$. Given a dataset $\mathbb{U} \subset \mathbb{X}$, this kind of queries can be classify basically in two: range and k-nearest neighbor queries. The first one consists in retrieving those objects out of $\mathbb{U}$ within a radius to a given query, that is, $R(q, r) = \{d(u, q) \leq r, \forall\, u \in \mathbb{U}\}$; the second one is to retrieve the k elements of $\mathbb{U}$ that are closest to q.

In general metric spaces, the (black-box) distance function is the only way to distinguish between objects, and usually, the distance function is expensive to

[⋆] This work is partially funded by National Council of Science and Technology (CONACyT) of México, Universidad Michoacana de San Nicolás de Hidalgo, México, and Fondecyt grant 1131044, Chile.

compute (e.g., consider comparing two multimedia objects). Hence the complexity is absorbed by the distances evaluated.

Since this kind of datasets lack of total order, to avoid a full linear scan, a dataset preprocessing, consisting in building an index structure that allows to get the answer with less effort, is common. In this respect, the *list of cluster* [5] is one of the most efficient algorithms in high dimensional spaces, however it takes $O(n^2)$ distances computation to make the index.

In other hand, the *permutation based algorithm* [3] has been proved unbeatable in practice but only works well in high dimensions, as the authors claim. To use this index, we have to compute the permutation of the query and compare it with all the dataset permutations so as to compute the order to review the permutations. This takes at least $O(|\mathbb{P}|)$ distances computations ($|\mathbb{P}|$ is the size of the permutations) and $O(n)$ evaluations of the permutation distance. There have been several proposal to avoid the sequential scanning in the permutation based algorithm, however all of them lost precision with respect to the original technique [7,10].

In this article we combine the ideas of the list of cluster and the permutation based index and present a new metric index that improves on both of them. The rest of this paper is organized as follows. In Section 2 we introduce some basic concepts. Next, in Section 3 we describe the *List of Clustered Permutations* and in Section 4 we show the experimental evaluation of our technique. Finally, we draw our conclusions and future work directions in Section 5.

2 Previous Work

One can approach the similarity search problem in either an exact or approximated way. In the first case, we want to retrieve all the objects satisfying the similarity query. The main families of this kind of algorithms are the pivot based indices and the ones based on compact partitions [6,1]. In the second, the idea is to retrieve most of the relevant elements that fulfill the similarity query. In this case, we accept to miss some relevant elements for the sake of speed up the query solving. There are already some non-exact approaches [4,3,12,9].

In this paper, we combine the list of clusters with the permutation based algorithm. Hence, in the next sections we describe both indices.

2.1 List of Clusters

There are many indices in metric spaces [6,1]. One of the most economical and rather efficient is the *list of clusters* (LC) [5], because it uses $O(n)$ space and has an excellent performance in high dimension. Regretably, its construction requires $O(n^2)$ distance evaluations, which is very expensive. The LC is built as follows:

Firstly, a center c is selected from the database and a bucket size b is given. c chooses its b-closest elements out of the database and build the set I, which is the answer of a b-nearest neighbour query. Let cr_c be the distance from c to its farthest neighbor in I. The tuple (c,I,cr_c) is called a cluster. (Notice that the parameter b could be replaced by specifying the global covering radius, but this alternative has worse performance [5].) This process is repeated recursively with the rest of the non-clustered objects.

To solve queries, the query object is compared with all the cluster centers. So, for each cluster, if the distance from its center to the query is larger than the its covering radius plus the query radius we can discard its whole bucket, otherwise we review it exhaustively.

2.2 Permutation Based Algorithm

In [3], the authors introduce the permutation based algorithm (PBA), a novel technique that shows a different way to sort the space. During the preprocessing time, a subset of objects $\mathbb{P} = \{p_1, p_2, \ldots, p_{|\mathbb{P}|}\} \subset \mathbb{U}$ is selected out of the database, which are called the permutants. Each $u \in \mathbb{U}$, computes its distance to all the permutants (that is, compute $d(u,p)$ for all $p \in \mathbb{P}$) and sort them increasingly by proximity. Then, for each object $u \in \mathbb{U}$, we store just the order of the permutants (not the distances) in the index.

If we define Π_u as the permutation of $(1 \ldots |\mathbb{P}|)$ for object u, so $\Pi_u(i)$ is the i-th cell in the u's permutation and $p_{\Pi_u(i)}$ denotes the i-th permutant. For instance, if $\Pi_u = (5,1,2,4,3)$ then $p_{\Pi_u(3)} = p_2$. Within the permutation, for all $1 \leq i < |\mathbb{P}|$ it holds either $d(p_{\Pi_u(i)}, u) < d(p_{\Pi_u(i+1)}, u)$ or, if there is a tie $(d(p_{\Pi_u(i)}, u) = d(p_{\Pi_u(i+1)}, u))$, then the permutant with the lowest index appears first in Π_u. We call the i-th permutant $\Pi_u(i)$, the *inverse permutation* Π_u^{-1}, and the position of i-th permutant $\Pi_u^{-1}(p_i)$. The set of all the permutations stored in the index needs $O(n|\mathbb{P}|)$ memory cells.

At query time, we compute the distance from the query q to all the permutants in $\mathbb{P}$ and calculate the query permutation Π_q. Next, Π_q is compared with all the permutations stored in the index, that is $O(n)$ permutation distances. In [11], authors introduce how to index the *permutations' space* as a new metric space, however they do not mix both kind of distances and they use a bigger index. Authors in [3] claim that the order induced by Π_q is extremely promising and a reviewing a small fraction of the dataset is enough to get a good answer.

The permutation distance is calculated as follows: let Π_u and Π_q permutations of $(1 \ldots |\mathbb{P}|)$. We compute how different is a permutation from the other using Spearman Rho S_ρ metric. In [8], S_ρ is defined as:

$$S_\rho(\Pi_u, \Pi_q) = \sqrt{\sum_{1 \leq i \leq |\mathbb{P}|} \left(\Pi_u^{-1}(i) - \Pi_q^{-1}(i)\right)^2} \qquad (1)$$

Since S_ρ is monotonic we omit the square root as it preserves the ordering.

The main disadvantage of the PBA is that its memory requirement could be prohibitive in some scenarios, especially where n is huge. Also, like other indices, the dimension of the space has an impact on the index performance; in particular, it has an effect on how long is the fraction to consider when solving the approximated query.

3 List of Clustered Permutations

The simplest way to reduce the time consumed when building a list of clusters is to avoid distance computations. For this sake, we have two possibilities: a bigger bucket size, or using another, cheaper, way to construct the structure. Follow the second possibility, we propose to combine the PBA with the LC. We choose a set of permutants, where each one within this set has a double role, as permutant and as a cluster center; and only the cluster centers store their permutation. We call this structure the *List of Clustered Permutations (LCP).*

When solving a proximity query q with the standard PBA, we need to spend $|\mathbb{P}|$ distance evaluations to compute the query permutation Π_q, plus n evaluations of the permutation distance to compute the order induced by Π_q, and $O(fn)$ distance evaluations to compare q with the fraction f of the dataset objects that are the most promising to be relevant for the query. With the LCP, we spend only $|\mathbb{P}|$ ($< n$) evaluations of the permutation distance to compare Π_q with the permutation of each cluster center, and distances evaluations needed to review non-discarded clusters. In our experiments, we verify that this is an improvement over the traditional LC.

3.1 Building

Firstly, we randomly select a set $\mathbb{P}$ of centers and we compare every object within the database with this set. This way, we compute permutations for all the objects in the dataset. Then, we choose the first center and group its $b = \frac{n}{|\mathbb{P}|} - 1$ most similar objects according to the *permutation distance* (excluding all the cluster centers, so that no center can be inside the bucket of another one). We continue the process iteratively with the rest of elements in the dataset until every element is clustered. Every center keeps its covering radius cr_c (that is, the distance to the farthest object in the bucket), its bucket and its permutation (hence, we discard the permutations of all the objects within a bucket).

The space used is $n + |\mathbb{P}|^2$ cells, and the construction time is $O(n|\mathbb{P}|)$ evaluations of both the space distance and the permutation distance. Note that we can pack the whole LCP index using just $(n + |\mathbb{P}|^2) \log_2 |\mathbb{P}|$ bits.

3.2 Querying

The standard LC discards clusters with the covering radius rule. Let $d(q, c)$ be the distance between the query and the center, r the query radius, and cr_c the covering radius of center c. So, if $d(q, c) > r + cr_c$ the cluster is discarded.

Since we have permutations, we introduce a heuristic method to discard a cluster, modifying the criteria explained in [5]. Our preliminary experimental results shown that if we have an object (for instance, a cluster center), and its permutation has (just) one permutant that moved far away with respect to its position inside query permutation, then this object is not relevant, so we can discard it (and also its bucket). For instance, if the permutation of the query is (1,2,3,4) and the permutation of the center is (4,1,2,3), even though most of both permutations are similar, the position shifting of permutant 4 suggests that the object can be discarded.

Of course, we need to establish a criterion to measure our finding. Basically, we need to know how much could a permutant move away inside the permutation of an object. So, using the query permutation and the range query radius, we estimate how far a permutant can shift. To do that, for a pair of permutants p_i, p_j, where p_i is closer to the query than p_j, and $d(p_j, q) - d(p_i, q) \leq r$, our method does not discard an object whose permutation has an inversion of these permutants; this is, it does not discard an object that is closer to p_j than to p_i. But, if the distance difference is larger, even though permutant inversion is possible there as a big chance that the object were irrelevant so the object can be discarded. Therefore, we take note of how many slots the permutant moves; this is computed in Algorithm 1.

Algorithm 1. ComputingShift(Q, r)

```
1: Let $Q$ the set of pairs (permutant, distance) to $q$, sorted by distance
2: $permShift \leftarrow 0$
3: for $i \leftarrow 0$ to $|\mathbb{P}| - 2$ do
4:     $cont \leftarrow 0, j \leftarrow i + 1$
5:     while $j < |\mathbb{P}|$ AND $Q[j].dist - Q[i].dist \leq r$ do
6:         $cont \leftarrow cont + 1, j \leftarrow j + 1$
7:     end while
8:     $permShift \leftarrow \max\{permShift, cont\}$
9: end for
10: return $permShift$
```

In the query procedure, we discard a cluster center (and its bucket) when a permutant shifts more than tolerated.

4 Experiments

In this section we evaluate and compare the performance of our technique in different metric spaces, such as synthetic vectors on the unitary cube and a real life database. The experiments were run on an Intel Xeon workstation with 2.4 GHz CPU and 32 GB of RAM with Ubuntu server, running kernel 2.6.32-22.

4.1 Synthetic Databases

In these experiments we used a synthetic database with vectors uniformly distributed on the unitary cube. We use 100,000 points in different dimensions under Euclidean distance. As we can precisely control the dimensionality of the space, we use these experiments to show how much the predictive power of our technique varies with the dimensionality.

Since ours is an approximated method, we relax the discarding criteria by accepting bigger shifts and tabulate the results. They are shown in Figures 1, 2, and 3. In this plots, the labels bx000 indicates the size b of the LCP buckets. Since $b = \frac{n}{|\mathbb{P}|} - 1$, bx000 also fixes a value for $|\mathbb{P}|$.

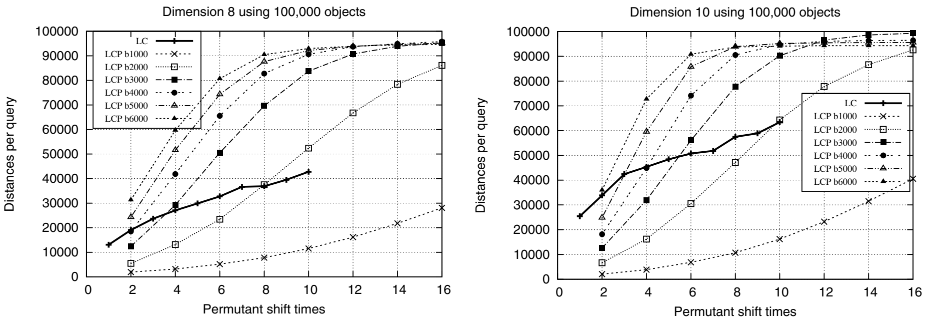


Fig. 1. Unitary cube using 100,000 vectors. (Left) Dimension 8, (right) dimension 10.

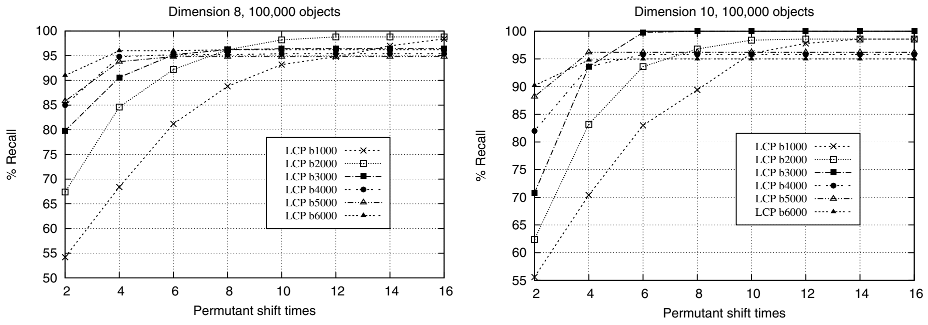


Fig. 2. The plots show the percentage of recall using the distances show in Figure 1

In Figures 1 and 3, the solid line is the original LC. For this line, the axis x represent the bucket size per thousand (from 1,000 to 10,000). As expected, Figure 1 shows that the smaller the bucket size the better the query results, since it is easier to discard a cluster with any of both criteria (this applies both for LC and LCP). On the other hand, Figure 2 illustrates that as long as the shifting criterion is relaxed, the recall of the method improves; but, it also increases the number of distance evaluations needed to solve the query. In several cases, accepting eight times in the permutant shift is enough to obtain an acceptable recall, saving distance computations and cpu time. Finally, the time computed for our method is lower than the standard LC, as evidenced in Figure 3.

In order to illustrate the performance of our method, in dimension 10, using buckets of 1,000 objects and accepting eight times in the permutation shifting,

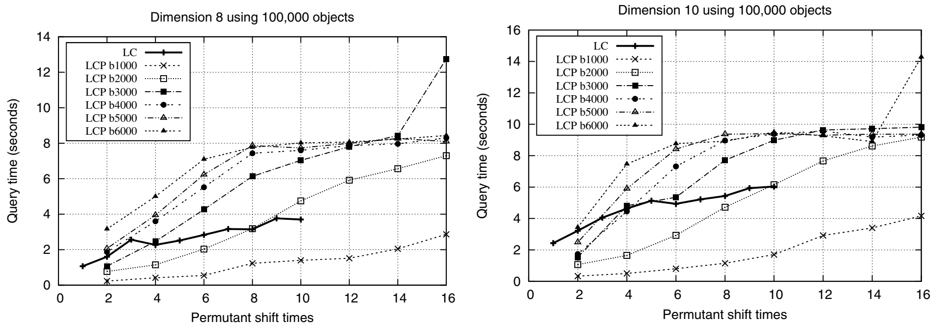


Fig. 3. Time consumed for experiments showed in Figures 1 and 2

our method requires 44% of distance evaluations of LC, obtains a 88% of recall and uses 48% of LC cpu time.

Note that the LCP index uses very little space: one identifier for each non-center object and only $|\mathbb{P}|^2$ cells for the permutations of centers. In this case, when using buckets of 1,000 objects (so $|\mathbb{P}| = 100$), this translates approximately to 7.7 bits per object.

Figure 4 compares the LTC with standar PBA. In order to perform a fair comparison, we allow 8 bits for each permutation, that is, four permutants coded in two bits. As can be seen, LTC with buckets of 1,000 objects outperforms by far the recall of standard PBA.

4.2 Real Databases

In this section we show the performance of our heuristic in a real-world space of images.

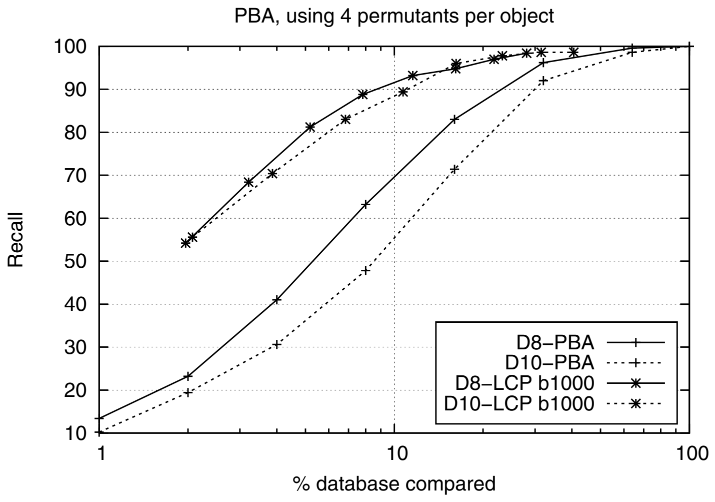


Fig. 4. Comparison between LCP and standard PBA

Cophir Database. In this section we show the experiments made on a large database. The CoPhIR is Content-based Photo Image Retrieval, with 1,000,000 of images [2] and buckets de 2,000 objects per cluster. For each image, the standard MPEG-7 image feature have been extracted. So, each image is a vector of 208 components.

In Figure 5, the label *List of Cluster* is the original technique retrieving the exact nearest neighbors. It shows that the LC requires to review almost 30% of the images. The label *Recall* is our proposal (it reviews from 1 to 7 % of the database) and the label *Distances* is the distance evaluation used to retrieval that recall. In this space, LCP performance is rather good. For instance, accepting forty times of shifting, we get the best retrieval (94%), reviewing just 7% of the database, in compare with LC that requires almost the 30% of the database.

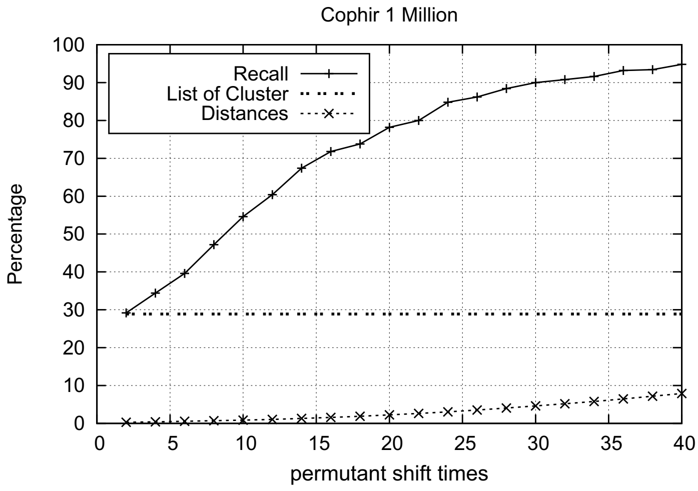


Fig. 5. 1 million of images. The solid line is the recall and the dashed line is the percentage of distance evaluations. The dotted line is the original LC.

5 Contributions and Future Work

Similarity searching is a very important operation in multimedia databases nowadays. It involves finding objects in a dataset similar to a query object q, based on some distance measure d. To do so, it is common to compute an index structure in order to solve similarity queries efficiently. One of the most successful indices is the permutation based algorithm. In this paper, we present a novel way to index permutations so that we can save space when computing the distance between permutations. The advantage of our proposal is that it is now possible to use the permutations in low dimensions and also we propose a parameter to avoid to sequential scanning in the permutation based algorithm.

As a future work we consider two lines, namely:

1. For the sake of maintain small clusters, we can divide the LCP construction in three phases. In the first, we choose the permutants and compute the permutations for all the objects. In the second, we compute the clusters for the permutants, and finally, we compute the other clusters. This way, we expect to compute the LCP using very few distance computations, but the amount of work computing the permutation distances should increase.
2. Since our method uses very little memory, we want to explore the possibility of using short permutations for objects inside the clusters. This is supported by the facts that the beginning of the permutation is the most important data portion to process and that we can trade space in order to improve the recall results.

References

1. Böhm, C., Berchtold, S., Keim, D.A.: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Computing Surveys 33(3), 322–373 (2001)
2. Bolettieri, P., Esuli, A., Falchi, F., Lucchese, C., Perego, R., Piccioli, T., Rabitti, F.: CoPhIR: A test collection for content-based image retrieval. CoRR abs/0905.4627v2 (2009), http://cophir.isti.cnr.it
3. Chávez, E., Figueroa, K., Navarro, G.: Proximity searching in high dimensional spaces with a proximity preserving order. In: Gelbukh, A., de Albornoz, Á., Terashima-Marín, H. (eds.) MICAI 2005. LNCS(LNAI), vol. 3789, pp. 405–414. Springer, Heidelberg (2005)
4. Chávez, E., Navarro, G.: Probabilistic proximity search: Fighting the curse of dimensionality in metric spaces. Information Processing Letters 85(1), 39–46 (2003)
5. Chávez, E., Navarro, G.: A compact space decomposition for effective metric indexing. Pattern Recognition Letters 26(9), 1363–1376 (2005)
6. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.: Proximity searching in metric spaces. ACM Computing Surveys 33(3), 273–321 (2001)
7. Esuli, A.: Mipai: using the pp-index to build an efficient and scalable similarity search system. In: Proc. 2nd Intl. Workshop on Similary Searching and Applications (SISAP 2009), pp. 146–148. IEEE Computer Society (2009)
8. Fagin, R., Kumar, R., Sivakumar, D.: Comparing top k lists. SIAM J. Discrete Math. 17(1), 134–160 (2003)
9. Figueroa, K., Chávez, E., Navarro, G., Paredes, R.: Speeding up spatial approximation search in metric spaces. ACM Journal of Experimental Algorithmics (JEA) 14, article 3.6, 21 pages (2009), doi: http://doi.acm.org/10.1145/1498698.1564506
10. Figueroa Mora, K., Paredes, R., Rangel, R.: Efficient group of permutants for proximity searching. In: Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A., Ben-Youssef Brants, C., Hancock, E.R. (eds.) MCPR 2011. LNCS, vol. 6718, pp. 42–49. Springer, Heidelberg (2011)
11. Figueroa, K., Fredriksson, K.: Speeding up permutation based indexing with indexing. In: Proceedings of the 2009 Second International Workshop on Similarity Search and Applications, SISAP 2009, pp. 107–114. IEEE Computer Society, Washington, DC (2009), http://dx.doi.org/10.1109/SISAP.2009.12
12. Patella, M., Ciaccia, P.: Approximate similarity search: A multi-faceted problem. Journal of Discrete Algorithms 7(1), 36–48 (2009)
13. Skala, M.: Counting distance permutations. J. of Discrete Algorithms 7(1), 49–61 (2009), http://dx.doi.org/10.1016/j.jda.2008.09.011

Machine Learning for Image Classification and Clustering Using a Universal Distance Measure

Uzi Chester and Joel Ratsaby*

Electrical and Electronics Engineering Department,
Ariel University of Samaria, ARIEL 40700
ratsaby@ariel.ac.il
http://www.ariel.ac.il/sites/ratsaby/

Abstract. We present a new method for image feature-extraction which is based on representing an image by a finite-dimensional vector of distances that measure how different the image is from a set of image prototypes. We use the recently introduced Universal Image Distance (UID) [1] to compare the similarity between an image and a prototype image. The advantage in using the UID is the fact that no domain knowledge nor any image analysis need to be done. Each image is represented by a finite dimensional feature vector whose components are the UID values between the image and a finite set of image prototypes from each of the feature categories. The method is automatic since once the user selects the prototype images, the feature vectors are automatically calculated without the need to do any image analysis. The prototype images can be of different size, in particular, different than the image size. Based on a collection of such cases any supervised or unsupervised learning algorithm can be used to train and produce an image classifier or image cluster analysis. In this paper we present the image feature-extraction method and use it on several supervised and unsupervised learning experiments for satellite image data. The feature-extraction method is scalable and is easily implementable on multi-core computing resources.

1 Introduction

Image classification research aims at finding representations of images that can be automatically used to categorize images into a finite set of classes. Typically, algorithms that classify images require some form of pre-processing of an image prior to classification. This process may involve extracting relevant features and segmenting images into sub-components based on some prior knowledge about their context [2,3].

In [1] we introduced a new distance function, called Universal Image Distance (UID), for measuring the distance between two images. The UID first transforms each of the two images into a string of characters from a finite alphabet and then uses the string distance of [4] to give the distance value between the images. According to [4] the distance between two strings x and y is a normalized

* Corresponding author.

difference between the complexity of the concatenation xy of the strings and the minimal complexity of each of x and y. By complexity of a string x we mean the Lempel-Ziv complexity [5].

In the current paper we use the UID to create a finite-dimensional representation of an image. The i^{th} component of this vector is a feature that measures how different the image is from the i^{th} feature catgory. One of the advantages of the UID is that it can compare the distance between two images of different sizes and thus the prototypes which are representative of the different feature categories may be relatively small. For instance, the prototypes of an *urban* category can be small images of size 45×70 pixels of various parts of cities.

In this paper we introduce a process to convert the image into a labeled case (feature vector). Doing this systematically for a set of images each labeled by its class yields a data set which can be used for training any supervised and unsupervised learning algorithms. After describing our method in details we report on the accuracy results of several classification-learning algorithms on such data. As an example, we apply out method to satellite image classification and clustering.

We note that our process for converting an image into a finite dimensional feature vector is very straightforward and does not involve any domain knowledge about the images. In contrast to other image classification algorithms that extract features based on sophisticated mathematical analysis, such as, analyzing the texture, the special properties of an image, doing edge-detection, or any of the many other methods employed in the immense research-literature on image processing, our approach is very basic and universal. It is based on the complexity of the 'raw' string-representation of an image. Our method extracts features automatically just by computing distances from a set of prototypes. It is therefore scalable and can be implemented using parallel processing techniques, such as on system-on-chip and FPGA hardware implementation [6,7,8].

Our method extracts image features that are unbiased in the sense that they do not employ any heuristics in contrast to other common image-processing techniques [2]. The features that we extract are based on information implicit in the image and obtained via a complexity-based UID distance which is an information-theoretic measure. In our method, the feature vector representation of an image is based on the distance of the image from some fixed set of representative class-prototypes that are initially and only once picked by a human user running the learning algorithm.

Let us now summarize the organization of the paper: in section 2 we review the definitions of LZ-complexity and a few string distances. In section 3 we define the UID distance. In section 4 we describe the algorithm for selecting class prototypes. In section 5 we describe the algorithm that generates a feature-vector representation of an image. In section 6 we discuss the classification learning method and in section we conclude by reporting on the classification accuracy results.

2 LZ-Complexity and String Distances

The UID distance function [1] is based on the LZ- complexity of a string. The definition of this complexity follows [4,5]: let S,Q and R be strings of characters that are defined over the alphabet $\mathcal{A}$. Denote by $l(S)$ the length of S, and $S(i)$ denotes the i^{th} element of S. We denote by $S(i,j)$ the substring of S which consists of characters of S between position i and j (inclusive). An extension $R = SQ$ of S is reproducible from S (denoted as $S \rightarrow R$) if there exists an integer $p \leq l(S)$ such that $Q(k) = R(p+k-1)$ for $k = 1, \ldots, l(Q)$. For example, $aacgt \rightarrow aacgtcgtcg$ with $p = 3$ and $aacgt \rightarrow aacgtac$ with $p = 2$. R is obtained from S (the seed) by first copying all of S and then copying in a sequential manner $l(Q)$ elements starting at the p^{th} location of S in order to obtain the Q part of R.

A string S is *producible* from its prefix $S(1,j)$ (denoted $S(1,j) \Rightarrow R$), if $S(1,j) \rightarrow S(1,l(S)-1)$. For example, $aacgt \rightarrow aacgtac$ and $aacgt \rightarrow aacgtacc$ both with pointers $p = 2$. The production adds an extra 'different' character at the end of the copying process which is not permitted in a reproduction.

Any string S can be built using a *production process* where at its i^{th} step we have the production $S(1,h_{i-1}) \Rightarrow S(1,h_i)$ where h_i is the location of a character at the i^{th} step. (Note that $S(1,0) \Rightarrow S(1,1)$).

An m-step production process of S results in parsing of S in which $H(S) = S(1,h_1) \cdot S(h_1+1,h_2) \cdots S(h_{m-1}+1,h_m)$ is called the *history* of S and $H_i(S) = S(h_{i-1}+1,h_i)$ is called the i^{th} component of $H(S)$. For example for $S = aacgtacc$ we have $H(S) = a \cdot ac \cdot g \cdot t \cdot acc$ as the history of S.

If $S(1,h_i)$ is not reproducible from $S(1,h_{i-1})$ then the component $H_i(S)$ is called *exhaustive* meaning that the copying process cannot be continued and the component should be halted with a single character *innovation*. Every string S has a unique exhaustive history [5].

Let us denote by $c_H(S)$ the number of components in a history of S. The LZ complexity of S is $c(S) = \min\{c_H(S)\}$ where the minimum is over all histories of S. It can be shown that $c(S) = c_E(S)$ where $c_E(S)$ is the number of components in the exhaustive history of S.

A distance for strings based on the LZ-complexity was introduced in [4] and is defined as follows: given two strings X and Y, denote by XY their concatenation then define

$$d(X,Y) := \max\{c(XY) - c(X), c(YX) - c(Y)\}.$$

In [1] we have found that the following normalized distance

$$\mathtt{d}(X,Y) := \frac{c(XY) - \min\{c(X), c(Y)\}}{\max\{c(X), c(Y)\}}. \tag{1}$$

performs well in classification and clustering of images.

We note in passing that (1) resembles the normalized compression distance of [9] except that here we do not use a compressor but instead resort to the LZ-complexity c of a string. Note that $\mathtt{d}$ is not a metric since it does not satisfy the triangle inequality and a distance of 0 implies that the two strings are close but not necessarily identical. We refer to $\mathtt{d}$ as a universal distance because it is not dependent on some specific representation of a string nor on heuristics common to other string distances such as edit-distances [10]. D only depends on the LZ-complexity of each of the two strings and their concatenation and this is a pure information-quantity which depends on the string's context but not its representation.

3 Universal Image Distance

Based on $\mathtt{d}$ we now define a distance between images. The idea is to convert each of two images I and J into strings $X^{(I)}$ and $X^{(J)}$ of characters from a finite alphabet of symbols. Once in string format, we use $\mathtt{d}(X^{(I)}, X^{(J)})$ as the distance between I and J. The details of this process are described in Algorithm 1 below.

Algorithm 1. UID distance measure

```
1. Input: two color images $I$, $J$ in jpeg format (RGB representation)
2. Transform the RGB matrices into gray-scale by forming a weighted sum of the
   R, G, and B components according to the following formula: $grayscaleValue :=$
   $0.2989R + 0.5870G + 0.1140B$, (used in Matlab©). Each pixel is now a single
   numeric value in the range of 0 to 255 . We refer to this set of values as the
   alphabet and denote it by $\mathcal{A}$.
3. Scan each of the grayscale images from top left to bottom right and form a string
   of symbols from $\mathcal{A}$. Denote the two strings by $X^{(I)}$ and $X^{(J)}$.
4. Compute the LZ-complexities: $c\left(X^{(I)}\right)$, $c\left(X^{(J)}\right)$ and the complexity of their con-
   catenation $c\left(X^{(I)}X^{(J)}\right)$
5. Output: $UID(I, J) := \mathtt{d}\left(X^{(I)}, X^{(J)}\right)$.
```

Remark 1. The transformation into gray-scale is a matter of representational convenience. To deal with color images without this transformation one can create a 3D alphabet whereby each 'letter' in this alphabet corresponds to an RGB triple with each component in the range 0 to 255. This way the image color information remains in the string representation.

4 Prototype Selection

In this section we describe the algorithm for selecting image prototypes from each of the feature categories . This process runs only once before the stage of converting the images into finite dimensional vectors, that is, it does not run once per image but once for all images. For an image I we denote by $P \subset I$ a sub-image P of I where P can be any rectangular-image obtained by placing a window over the image I where the window is totally enclosed by I. The size of the window depends on how much information a single prototype will convey about the associated feature-category.

In the following algorithm we use clustering as a simple means of validation that the prototypes selected maintain the inherent differences between the feature-categories (the clustering algorithm is not given the feature-category information but only the inter-prototype distance information).

Algorithm 2. Prototypes selection

```
1. Input: $M$ image feature categories, and a corpus $\mathcal{C}_N$ of $N$ unlabeled colored images $\{I_j\}_{j=1}^{N}$ .
2. for ($i := 1$ to $M$) do
   (a) Based on any of the images $I_j$ in $\mathcal{C}_N$, let the user select $L_i$ prototype images $\left\{P_k^{(i)}\right\}_{k=1}^{L_i}$ and set them as feature category $i$. Each prototype is contained by some image, $P_k^{(i)} \subset I_j$, and the size of $P_k^{(i)}$ can vary, in particular it can be much smaller than the size of the images $I_j$, $1 \le j \le N$.
   (b) end for;
3. Enumerate all the prototypes into a single unlabeled set $\{P_k\}_{k=1}^{L}$, where $L = \sum_{i=1}^{M} L_i$ and calculate the distance matrix $H = \left[UID\left(X^{(P_k)}, X^{(P_l)}\right)\right]_{k=1,l=1}^{L}$ where the $(k,l)$ component of $H$ is the UID distance between the unlabeled prototypes $P_k$ and $P_l$.
4. Run hierarchical clustering on $H$ and obtain the associated dendrogram (note: $H$ does not contain any 'labeled' information about feature-categories, as it is based on the unlabeled set).
5. If there are $M$ clusters with the $i^{th}$ cluster consisting of the prototypes $\left\{P_k^{(i)}\right\}_{k=1}^{L_i}$ then terminate and go to step 7.
6. Else go to step 2.
7. Output: the set of labeled prototypes $\mathcal{P}_L := \left\{\left\{P_k^{(i)}\right\}_{k=1}^{L_i}\right\}_{i=1}^{M}$ where $L$ is the number of prototypes.
```

From the theory of learning pattern recognition, it is known that the dimensionality M of a feature-vector is usually taken to be small compared to the data size N. A large L will obtain better feature representation accuracy of the image, but it will increase the time for running Algorithm 3 (described below).

Algorithm 2 convergence is based on the user's ability to select good prototype images. We note that from our experiments this is easily achieved primarily because the UID permits to select prototypes $P_k^{(i)}$ which are considerably *smaller* than the size of the full images I_j. For instance, in our experiments we used 45×70

pixels prototype size for all feature categories. This fact makes it easy for a user to quickly choose typical representative prototypes from every feature-category. This way it is easy to find informative prototypes, that is, prototypes that are distant when they are from different feature-categories and close when they are from the same feature category. Thus Algorithm 2 typically converges rapidly.

As an example, Figure 1 displays 12 prototypes selected by a user from a corpus of satellite images. The user labeled prototypes $1,\ldots,3$ as representative of the feature category *urban,* prototypes $4,\ldots,6$ as representatives of class *sea,* prototypes $7,\ldots,9$ as representative of feature *roads* and prototypes $10,\ldots,12$ as representative of feature *arid.* The user easily found these representative prototypes as it is easy to fit in a single picture of size 45×70 pixels a typical image. The dendrogram produced in step 4 of Algorithm 2 for these set of 12 prototypes is displayed in Figure 2. It is seen that the following four clusters were found $\{10,12,11\},\{1,2,3\},\{7,8,9\},\{4,6,5\}$ which indicates that the prototypes selected in Algorithm 2 are good.

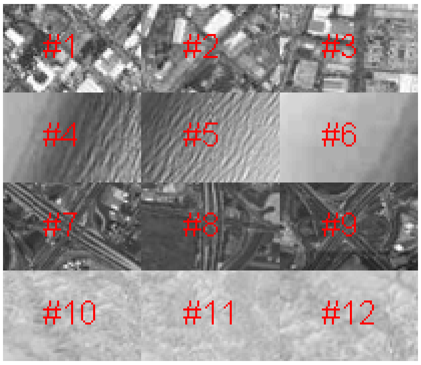


Fig. 1. Labeled prototypes of feature-categories *urban*, *sea* , *roads*, and *arid* (each feature has three prototypes, starting from top left and moving right in sequence)

5 Image Feature-Representation

In the previous section we described Algorithm 2 by which the prototypes are manually selected. This algorithm is now used to create a feature-vector representation of an image. It is described as Algorithm 3 below (in [1] we used a similar algorithm UIC to soft-classify an image whilst here we use it to only produce a feature vector representation of an image which later serves as a single labeled case for training any supervised learning algorithm or a single unlabeled case for training an unsupervised algorithm).

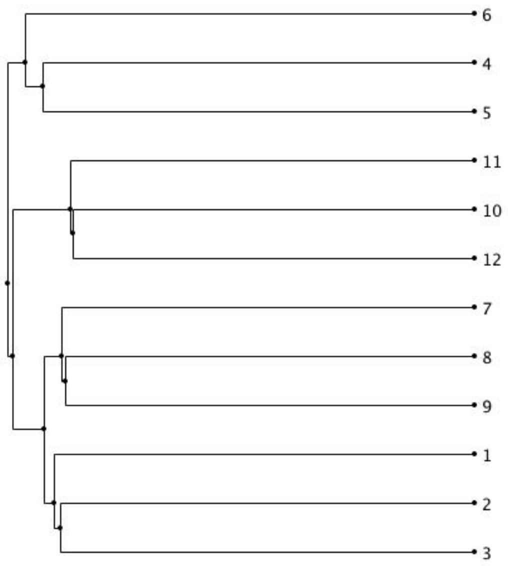


Fig. 2. Dendrogram of prototypes of Figure 1

Algorithm 3. Feature-vector generation

```
1. **Input**: an image $I$ to be represented on the following feature categories $1 \le i \le M$,
   and given a set $\mathcal{P}_L := \left\{ \left\{ P_k^{(i)} \right\}_{k=1}^{L_i} \right\}_{i=1}^{M}$ of labeled prototype images (obtained from
   Algorithm 2).
2. **Initialize** the count variables $c_i := 0$, $1 \le i \le M$
3. Let $W$ be a rectangle of size equal to the maximum prototype size. (See remark
   below)
4. Scan a window $W$ across $I$ from top-left to bottom-right in a non-overlapping way,
   and let the sequence of obtained sub-images of $I$ be denoted as $\{I_j\}_{j=1}^{m}$.
5. **for** $(j := 1$ to $m)$ **do**
   (a) **for** $(i := 1$ to $M)$ **do**
       i. $temp := 0$
       ii. **for** $(k := 1$ to $L_i)$ **do**
           A. $temp := temp + \left(UID(I_j, P_k^{(i)})\right)^2$
           B. **end for;**
       iii. $r_i := \sqrt{temp}$
       iv. **end for;**
   (b) Let $i^*(j) := \text{argmin}_{1 \le i \le M} r_i$, this is the decided feature category for sub-image
       $I_j$.
   (c) **Increment** the count, $c_{i^*(j)} := c_{i^*(j)} + 1$
   (d) **end for;**
6. **Normalize** the counts, $v_i := \frac{c_i}{\sum_{l=1}^{M} c_l}$, $1 \le i \le M$
7. **Output**: the normalized vector $v(I) = [v_1, \ldots v_M]$ as the feature-vector represen-
   tation for image $I$
```

Remark 2. In Step 3, we choose the size of W to be the maximal size of a prototype but this is not crucial since D can measure the distance between two images of different sizes. From our experiments, the size of W needs to be large enough such that the amount of image information in W is not smaller than that captured in any of the prototypes.

6 Supervised and Unsupervised Learning on Images

Given a corpus $\mathcal{C}$ of images and a set $\mathcal{P}_L$ of labeled prototypes we use Algorithm 3 to generate the feature-vectors $v(I)$ corresponding to each image I in $\mathcal{C}$. At this point we have a database $\mathcal{D}$ of size equal to $|\mathcal{C}|$ which consists of feature vectors of all the images in $\mathcal{C}$. This database can be used for *unsupervised* learning, for instance, discover interesting clusters of images. It can also be used for *supervised* learning provided that each of the cases can be labeled according to a value of some target class variable which in general may be different from the feature categories. Let us denote by T the class target variable and the database $\mathcal{D}_T$ which consists of the feature vectors of $\mathcal{D}$ with the corresponding target class values. The following algorithm describes the process of learning classification of images.

Algorithm 4. Image classification learning

```
1. Input: (1) a target class variable $T$ taking values in a finite set $\mathcal{T}$ of class categories,
   (2) a database $\mathcal{D}_T$ which is based on the $M$-dimensional feature-vectors database
   $\mathcal{D}$ labeled with values in $\mathcal{T}$ (3) any supervised learning algorithm $\mathcal{A}$
2. Partition $\mathcal{D}_T$ using $n$-fold cross validation into Training and Testing sets of cases
3. Train and test algorithm $\mathcal{A}$ and produce a classifier $C$ which maps the feature
   space $[0,1]^M$ into $\mathcal{T}$
4. Define Image classifier as follows: given any image $I$ the classification is $F(I) :=$
   $C(v(I))$, where $v(I)$ is the $M$-dimensional feature vector of $I$
5. Output: classifier $F$
```

7 Computational Time

Given an image I let us denote by $\tau(I)$ the total time that it takes Algorithm 3 to generate the case (vector-representation) $v(I)$ of I that can be used as a training case or as an input to the classifier F in order to classify the image I.

As mentioned above, Algorithm 2 involves a one-time manual selection of prototypes and the speed is dictated by the user (not the computer). Algorithms 3 is where the computational time is relevant. Step 5 of Algorithm 3 is the time-critical section which governs the overall computational time of the algorithm. This step iterates over all subimages I_j, $1 \leq j \leq m$, of the input image I, and for each subimage it computes the values r_i, $1 \leq i \leq M$, one for each feature-category. In order to compute r_i it computes the UID distance between I_j and prototype $P_k^{(i)}$, $1 \leq k \leq L_i$. To compute $UID(I, J)$ requires building the exhaustive history of both strings $X^{(I)}$, $X^{(J)}$ and of their concatenation $X^{(I)}X^{(J)}$. So the time to compute $UID(I, J)$ is $O(c(X^{(I)}X^{(J)}))$ where $c(X^{(I)}X^{(J)})$ is the length of the exhaustive history of their concatenation. Denoting by $\tau(I, J)$ the time to compute $UID(I, J)$ then it is clear that $\tau(I, J)$ depends on the images I, J and not just on their sizes. That is, $\tau(I, J)$ depends on the LZ-complexity of the two images and on their similarity–the more similar the two, the less complex the concatenation string $X^{(I)}X^{(J)}$ and the smaller $\tau(I, J)$ is.

The time to compute the decided feature-category for subimage I_j of I is big-O the time that it takes to perform the j^{th} iteration of the outer for-loop of step 5. We refer to this as *subimage-time* and denote it by $\tau_j(I)$. We have

$$\tau_j(I) := O\left(\sum_{i=1}^{M}\sum_{k=1}^{L_i} \tau(I_j, P_k^{(i)})\right)$$

where M is the number of categories, and L_i is the number of prototypes for category i.

Hence if we run on a single processor (single core) the case-generation time $\tau(I)$ of an image I is

$$\tau(I) = \sum_{j=1}^{m} \tau_j(I) \tag{2}$$

where m is the number of subimages in a single image I. It is clear from this formula that parallel computations (in particular, stream processing where the same function is applied to different data) can be very advantegeous for reducing the case-generation time $\tau(I)$.

For instance, on a processor with n cores, where $n \geq m$, each of the cores can compute in parallel a different subimage. This yields a total time

$$\tau(I) = O(\max_{1\leq j\leq m} \tau_j(I)).$$

If the number of cores n satisfies $m > n \geq M$ then we can let each core compute a different category-sum. This takes a single *sub-image-category* time

$$\tau_j^{(i)}(I) := O\left(\sum_{k=1}^{L_i} \tau\left(I_j, P_k^{(i)}\right)\right)$$

and in this case the total time is

$$\tau(I) = \sum_{j=1}^{m} \max_{1\leq i\leq M} \tau_j^{(i)}(I). \tag{3}$$

If the number of cores $n \geq m \cdot \sum_{i=1}^{M} L_i$ then the total time to generate $v(I)$ from I is

$$\tau(I) = O\left(\max_{\substack{1\leq j\leq m\\ 1\leq i\leq M\\ 1\leq k\leq L_i}} \tau\left(I_j, P_k^{(i)}\right)\right). \tag{4}$$

8 Experimental Setup and Results

We created a corpus $\mathcal{C}$ of 60 images of size 670×1364 pixels from GoogleEarth©of various types of areas. Figure 3 displays a few scaled-down examples of such images. From these images we let a user define four feature-categories: *sea*, *urban*, *arid*, *roads* and choose three relatively-small image-prototype of size 45×70 pixels from each feature-category, that is, we ran Algorithm 2 with $M = 4$ and $L_i = 3$ for all $1 \leq i \leq M$. We then ran Algorithm 3 to generate the feature-vectors for each image in the corpus and obtained a database $\mathcal{D}$.

We then let the user label the images by a target variable *Humidity* with possible values 0 or 1. An image is labeled 0 if the area is of low humidity and labeled 1 if it is of higher humidity. We note that an image of a low humidity region may be in an arid (dry) area or also in the higher-elevation areas which are not necessarily arid. Since elevation information is not available in the feature-categories that the user has chosen then the classification problem is hard since the learning algorithm needs to discover the dependency between humid regions and areas characterized only by the above four feature categories.

With this labeling information at hand we produced the labeled database $\mathcal{D}_{Humidity}$. We used Algorithm 4 to learn an image classifier with target *Humidity.* As the learning algorithm $\mathcal{A}$ we used the following standard supervised algorithms: $J48$, $CART$, which learn decision trees, *NaiveBayes* and *Multi-Layer Perceptrons* (backpropagation) all of which are available in the WEKA©toolkit.

We performed 10-fold cross validation and compared their accuracies to a baseline classifier (denoted as ZeroR) which has a single decision that corresponds to the class value with the highest *prior* empirical probability. As seen in Table 1 (generated by WEKA©) $J48$, $CART$, NaiveBayes and Backpropagation performed with an accuracy of 86.5%, 81.5%, 89.25%, and 87.25%, respectively, compared to 50% achieved by the baseline $ZeroR$ classifier. The comparison concludes that all three learning algorithms are significantly better than the baseline classifier, based on a T-test with a significance level of 0.05.

Next, we performed clustering on the unlabeled database $\mathcal{D}$. Using the k-means algorithm, we obtained 3 significant clusters, shown in Table 2.

One can read the information directly from Table 2 and see that the first cluster captures images of *highly urban* areas that are next to concentration of roads, highways and interchanges. The second cluster contains *less populated* (urban) areas in arid locations (absolutely no sea feature seen) with very low concentration of roads. The third cluster captures the *coastal areas* and here we can see that there can be a mixture of urban (but less populated than images of the first cluster) with roads and extremely low percentage of arid land.

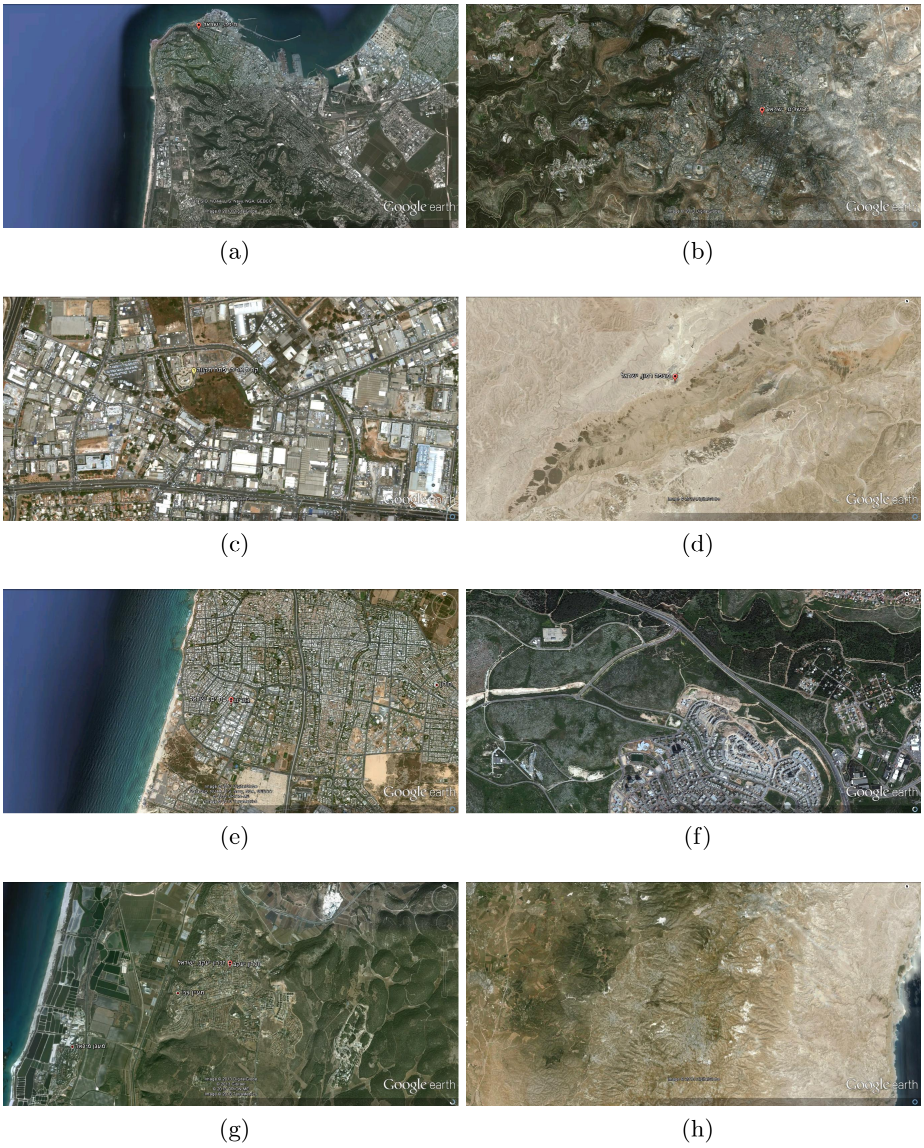


Fig. 3. Examples of images in the corpus

Table 1. Percent correct results for classifying *Humidity*

Dataset $\mathcal{D}_{Humidity}$	(1)	(2)	(3)	(4)	(5)
Classify Image into Humidity:	50.00	86.50 ○	81.50 ○	89.25 ○	87.25 ○

○, ● statistically significant improvement or degradation

(1) rules.ZeroR '' 48055541465867954
(2) trees.J48 '-C 0.25 -M 2' -217733168393644444
(3) trees.SimpleCart '-S 1 -M 2.0 -N 5 -C 1.0' 4154189200352566053
(4) bayes.NaiveBayes '' 5995231201785697655
(5) functions.MultilayerPerceptron '-L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a' -5990607817048210779

Table 2. k-means clusters found on unsupervised database $\mathcal{D}$

Feature	Full data	Cluster#1	Cluster#2	Cluster#3
urban	0.3682	0.6219	0.1507	0.2407
sea	0.049	0.0085	0	0.1012
road	0.4074	0.2873	0.0164	0.655
arid	0.1754	0.0824	0.8329	0.003

The fact that such interesting knowledge can be extracted from raw images using our feature-extraction method is very significant since as mentioned above our method is fully automatic and requires no image or mathematical analysis or any sophisticated preprocessing stages that are common in image pattern analysis.

Let us report on some computational time statistics. The hardware we used is a 2.8Ghz AMD Phenom©II X6 1055T Processor with number of cores $n = 6$. The operating system is Ubuntu Linux 2.6.38-11-generic. The corpus consists of images of size 670×1364 pixels, with a sub-image and prototype sizes of 45×70 pixels, the number of subimages per image is $m = 266$. For this test we chose the number of feature-categories $M = 3$ with a total number of prototypes $\sum_{i=1}^{M} L_i = 11$. We implemented the algorithms in Matlab©in standard code style, that is, with no time-efficiency optimization, except in step 5.a. where the for statement is implemented using a `parfor` statement of Matlab. Note that in this case $m > n > M$ hence the total time to compute one image I is as stated in (3).

We obtained the set of average computational times

$$Timing\_Data = \left\{ \overline{\tau}_j^{(i)} : 1 \le j \le m, 1 \le i \le M \right\}$$

where

$$\overline{\tau}_j^{(i)} := \frac{1}{L_i} \sum_{k=1}^{L_i} \tau \left(I_j, P_k^{(i)} \right).$$

Figure 4 shows the histogram for this *Timing_Data*, where the horizontal axis is time in units of seconds. The mean time is 0.851 sec. and the standard deviation is 0.264 sec. Some of the state-of-the-art Graphics Processor Unit (GPU) accelerators have thousands of execution cores (see for instance, NVIDIA Tesla© K20 which has 2, 496 cores) and are offered at current prices of approximately \$2, 000. On the NVIDIA Tesla© K10 the number of execution cores is $n = 3072$ and is greater than $m \sum_{i=1}^{M} L_i = 2,926$ so the total computation time $\tau(I)$ to process a single image I in the corpus will be as in (4), which for this example is approximately 0.851 sec. using Matlab with no optimization. We have not yet done so but we expect that transforming the code from Matlab to C and rewriting it with parallel processing code optimization can yield an average $\tau(I)$ which is lower by several orders of magnitude.

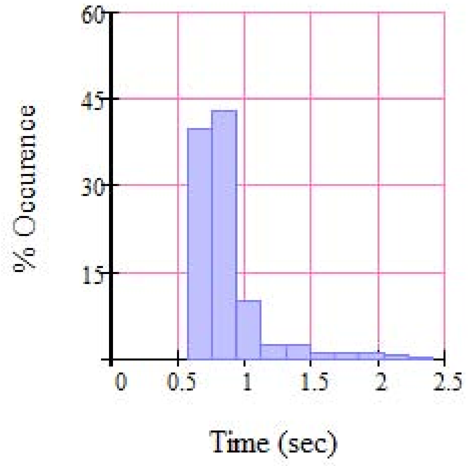


Fig. 4. Histogram of the computational times $\tau_j^{(i)}$, $1 \le j \le m$, $1 \le i \le M$, $m = 266$, $M = 3$. The mean is 0.851.

9 Conclusion

We introduced a method for automatically defining and measuring features of colored images.The method is based on a universal image distance that is measured by computing the complexity of the string-representation of the two images and their concatenation. An image is represented by a feature-vector which consists of the distances from the image to a fixed set of small image prototypes, defined once by a user. There is no need for any sophisticated mathematical-based image analysis or pre-processing since the universal image distance regards the image as a string of symbols which contains all the relevant information of the image. The simplicity of our method makes it very attractive for fast and scalable implementation, for instance on a specific-purpose hardware acceleration chip. We applied our method to supervised and unsupervised machine learning on satellite images. The results show that standard machine learning algorithms perform well based on our feature-vector representation of the images.

Acknowledgements. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Tesla K20 GPU used for this research.

References

1. Chester, U., Ratsaby, J.: Universal distance measure for images. In: 2012 IEEE 27th Convention of Electrical Electronics Engineers in Israel (IEEEI), pp. 1–4 (November 2012)
2. Canty, M.J.: Image Analysis, Classification and Change Detection in Remote Sensing: With Algorithms for Envi/Idl. CRC/Taylor & Francis (2007)
3. Lillesand, T.M., Kiefer, R.W., Chipman, J.W.: Remote sensing and image interpretation. John Wiley & Sons (2008)
4. Sayood, K., Otu, H.H.: A new sequence distance measure for phylogenetic tree construction. Bioinformatics 19(16), 2122–2130 (2003)
5. Ziv, J., Lempel, A.: On the complexity of finite sequences. IEEE Transactions on Information Theory 22(3), 75–81 (1976)
6. Ratsaby, J., Sirota, V.: Fpga-based data compressor based on prediction by partial matching. In: 2012 IEEE 27th Convention of Electrical Electronics Engineers in Israel (IEEEI), pp. 1–5 (November 2012)
7. Ratsaby, J., Zavielov, D.: An fpga-based pattern classifier using data compression. In: Proc. of 26th IEEE Convention of Electrical and Electronics Engineers, Israel, Eilat, November 17-20, pp. 320–324 (2010)
8. Kaspi, G., Ratsaby, J.: Parallel processing algorithm for Bayesian network inference. In: 2012 IEEE 27th Convention of Electrical Electronics Engineers in Israel (IEEEI), pp. 1–5 (November 2012)
9. Cilibrasi, R., Vitanyi, P.: Clustering by compression. IEEE Transactions on Information Theory 51(4), 1523–1545 (2005)
10. Deza, M., Deza, E.: Encyclopedia of Distances. Series in Computer Science, vol. 15. Springer (2009)

Faster Algorithms for Tree Similarity Based on Compressed Enumeration of Bounded-Sized Ordered Subtrees

Kunihiro Wasa[1], Kouichi Hirata[2], Takeaki Uno[3], and Hiroki Arimura[1]

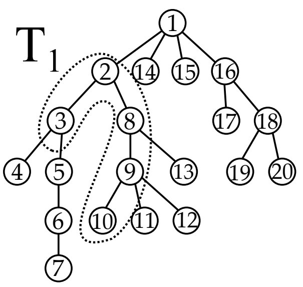
[1] Hokkaido University, N14 W9, Sapporo 060-0814, Japan
{wasa,arim}@ist.hokudai.ac.jp
[2] Kyushu Institute of Technology, Kawazu 680-4, Iizuka 820-8502, Japan
hirata@ai.kyutech.ac.jp
[3] National Institute of Informatics, 2-1-2 Hitotsubashi, Tokyo 101-8430, Japan
uno@nii.jp

Abstract. In this paper, we study efficient computation of tree similarity for ordered trees based on compressed subtree enumeration. The *compressed subtree enumeration* is a new paradigm of enumeration algorithms that enumerates all subtrees of an input tree T in the form of their compressed bit signatures. For the task of enumerating all compressed bit signatures of k-subtrees in an ordered tree T, we first present an enumeration algorithm in $O(k)$-delay, and then, present another enumeration algorithm in constant-delay using $O(n)$ time preprocessing that directly outputs bit signatures. These algorithms are designed based on bit-parallel speed-up technique for signature maintenance. By experiments on real and artificial datasets, both algorithms showed approximately 22% to 36% speed-up over the algorithms without bit-parallel signature maintenance.

1 Introduction

Similarity search is a fundamental problem in modern information and knowledge retrieval [14]. In particular, we focus on *tree similarity* between two trees, which plays a key role in a number of information and knowledge retrieval problems from semi-structured data such as similarity search, clustering, recommendation, and classification for structured data in the real world [2–4, 8, 13].

In this paper, we study the efficient computation of the frequency-based tree similarities using classes of ordered trees. *Ordered trees* are rooted trees which have total ordering among siblings. They are useful for modeling semi-structured documents such as HTML and XML, chemical compounds, natural language data, and Web access logs. In one direction, ordered tree similarities based on substructures have been extensively studied [9–11], where the focuses are on substructures of restricted forms such as paths [9] and q-grams [10]. In other direction, Kashima and Koyanagi [8] presented an efficient dynamic programming algorithm to compute the *ordered subtree kernel* of two ordered trees T_1 and T_2



i	Subtree S_i	Signature B_i	Freq. f_i
1	$S_1 = \{1, 2, 14, 15, 16\}$	(()()()())	1
2	$S_2 = \{1, 2, 14, 16, 17\}$	(()()(()))	6
3	$S_3 = \{2, 3, 8, 9, 13\}$	(()(()()))	5
4	$S_4 = \{2, 3, 8, 9, 10\}$	(()((())))	10
5	$S_5 = \{1, 2, 3, 14, 15\}$	((())()())	6
6	$S_6 = \{2, 3, 5, 8, 13\}$	((())(()))	8
7	$S_7 = \{2, 3, 4, 5, 8\}$	((()())())	7
8	$S_8 = \{8, 9, 10, 11, 12\}$	((()()()))	1
9	$S_9 = \{1, 2, 3, 8, 9\}$	((()(())))	5
10	$S_{10} = \{1, 2, 3, 16, 17\}$	(((()))())	13
11	$S_{11} = \{1, 2, 3, 4, 8\}$	(((())()))	5
12	$S_{12} = \{1, 2, 3, 16, 18\}$	(((()())))	6
13	$S_{13} = \{1, 2, 8, 9, 11\}$	((((()))))	5

Fig. 1. An ordered tree T_1, the corresponding k-subtrees, and the feature vector $\phi^{S_k}(T)$ for T_1, where $k = 5$. The set of nodes surrounded by a dashed circle indicates the subtree $S_4 = \{2, 3, 8, 9, 10\}$, which has occurrences $S_4^1 = \{1, 15, 16, 18, 19\}$ and $S_4^2 = \{2, 3, 8, 9, 12\}$ in T isomorphic to itself.

in $O(|T_1| \times |T_2|)$ time using general ordered subtrees of *unbounded size*. Besides the efficient DP algorithms for ordered trees of unbounded size [8], some authors pointed out the usefulness of the semi-structured features using bounded sized substructures [12]. However, it does not seem easy to extend the DP algorithm [8] for *bounded sized* ordered trees.

The *enumeration-based approach* [2,12,16] is another way of computing such a tree distance based on a general class of substructures, which is a simple and flexible approach that one uses a *pattern enumeration algorithm* [2,17,19], to find all substructures contained in an input data to construct a feature vector, and then to solve a variety of tasks for information retrieval, data mining, and machine learning using similarity measure obtained from the constructed feature vectors. One problem in this approach is the high computational complexity of enumerating all small substructures. Hence, our goal is to devise efficient algorithms for frequency-based similarity by employing the recent development of efficient enumeration and mining algorithms for semi-structured data [2,17,19].

In this paper, we study efficient computation of tree similarity between two ordered trees using as features the class of *bounded sized ordered subtrees in unrestricted shape*. We present two new efficient algorithms for enumerating the *compressed bit-signatures* of all ordered k-subtrees in an input ordered tree using bit-parallel speed-up technique. The first one runs in $O(k)$ time per signature, and the second one runs in constant time per signature using $O(n)$ time preprocessing [1,7]. From these compressed signatures, we can quickly compute the tree similarity between two ordered trees. We note that these algorithm are the first *compressed pattern enumeration algorithms* [18] for a subclass of trees and graphs. They directly enumerate the compressed representation of all substructures by incrementally constructing their compressed form on-the-fly without encoding/decoding.

Finally, we ran the experiments on real and artificial datasets to evaluate the proposed methods. We observed that the improved versions of the algorithms equipped with our bit-parallel speed-up technique showed around 36% speed-up from the algorithm without speed-up.

Organization of This Paper: In Sec.2, we give the definitions of the tree similarities with ordered k-subtrees. In Sec. 3, we present the first algorithm using at most k-subtrees, and in Sec.4, the second algorithm using exactly k-subtrees. In Sec.5, we ran experiments to evaluate these algorithms, and in Sec.6, we conclude this paper.

2 Preliminaries

In this section, we give basic definitions and notation on the tree similarities over ordered trees. We denote by $|A|$ the number of elements in a set A. For every integers $i \leq j$, we denote by $[i, j] = \{i, i+1, \ldots, j\}$.

Trees and k-Subtrees: An *ordered tree* is a rooted tree $T = (V, E, r)$ with a node set $V = V(T) = \{1, \ldots, n\}$, an edge set $E = E(T)$, and the root $r = \mathrm{root}(T)$ such that there exists a fixed ordering among children $Ch(u)$ of each internal node $u \in T$. The *size* of T is $|T| = |V|$. We assume the standard definitions of the parent, children, leaves, and paths [5]. We denote by $pa(v)$ the parent of node v, $Ch(v)$ the set of all children of v, and $Lv(T) \subseteq V(T)$ the set of all leaves of T. The *border set* is the set $Bd(S) = \{\, y \in Ch(x) \mid x \in S, y \notin S \,\}$, that is, the set of all nodes that are not contained in S, but are children of some nodes in S. We denote by $\mathcal{T}$ the countable set of all ordered trees.

A *subtree with size k* of T, or simply a *k-subtree*, is any connected subgraph $T[S] = (S, E(S), \mathrm{root}(S))$ of T induced in a subset $S \subseteq V(T)$ consisting of exactly k nodes of T. Since such a subtree can be completely specified by a connected node set S, we identify any subset $S \subseteq V(T)$ with the subtree $T[S]$ if it is clear from the context. We denote by $\mathcal{S}_k(T)$ the set of all distinct k-subtrees appearing in T modulo isomorphism. A k-subtree $S \in \mathcal{S}_k(T)$ *appears in* T if there is some subset $U \subseteq V(T)$ such that $T[U]$ is isomorphic to $T[S]$. We assume that the nodes are ordered as $v_1 \leq \cdots \leq v_n$ by the preorder traversal of T [5].

Bit Signatures: As the succinct representation of ordered trees, the *balanced parentheses representation* (BP) [1] of an ordered tree T of k nodes is a bit sequence $BP(T) = b_{2k-1} \cdots b_0$ defined by the depth-first traversal of T starting from $\mathrm{root}(T)$ the left and right parentheses, "`(`" $= 0$ and "`)`" $= 1$, when it visits a node at the first and last times, respectively. We call $BP(T)$ the bit signature of T. For example, the BP of a tree $S_3 = \{2, 3, 8, 9, 13\}$ of size 5 is "`(()(()()))`".

For each node $v \in T$, $lpos(v)$ and $rpos(v) \in [0, 2k-1]$ denotes the bit positions for the left and right parentheses in $BP(S)$ corresponding to v, respectively. For any subset $R \subseteq S$, $RPOS(R)$ denotes the bit-vector $X \in \{0,1\}^{2k}$ such that, for every $v \in S$, $X[rpos(v)] = 1$ iff $v \in R$. The *compressed subtree enumeration* on T is the task of enumerating all subtrees in T in the form of bit signature.

Tree Similarity: In this subsection, we give the tree similarity for ordered trees [8,11]. Let T be an input ordered tree and $\mathcal{S} = \{S_1, \ldots, S_M\} \subseteq \mathcal{T}$, $M \geq 1$, be a class of possible subtrees in T. Each elements of $\mathcal{S}$ are called *a subtree-feature*. The *subtree-feature vector* of T based on $\mathcal{S}$ is the vector $\phi^{\mathcal{S}}(T)$ of the number of counts that subtrees of $\mathcal{S}$ appear in T. Below, we will omit the superscript $\mathcal{S}$ if it is clear from context. Formally, the *subtree-feature vector* $\phi(T)$ for T based on $\mathcal{S}$ is defined by

$$\phi(T) = (f_1(T), \ldots, f_M(T)) \in \mathbb{N}^M, \tag{1}$$

where for every $i \in [1, M]$, $f_i(T) = Occ(S_i, T)$ is the number of all occurrences of the i-th subtree S_i in T. Then, we consider the the tree similarity $Sim(T, T')$ between T and T' in one of the following forms [14]:

– L_p-tree distance for every $p = 1, 2, \ldots$:

$$Sim(T, T') = ||\phi(T) - \phi(T')||_p = \left(\sum_i |f_i(T) - f_i(T')|^p \right)^{1/p}. \tag{2}$$

– Cosine-tree distance:

$$Sim(T, T') = Cosine(\phi(T), \phi(T')) = \frac{\sum_i f_i(T) \times f_i(T')}{(\sum_i f_i(T))^{\frac{1}{2}} \times (\sum_i f_i(T'))^{\frac{1}{2}}} \tag{3}$$

Once the feature vectors $\phi(T)$ and $\phi(T')$ are computed by a subtree enumeration algorithm for class $\mathcal{S}$, $Sim(T, T')$ can be computed in linear time in the length of the vectors. In Fig.1, we show examples of an ordered tree T_1, the corresponding k-subtrees, and the feature vector $\phi^{\mathcal{S}_k}(T_1)$ for T_1 based on all k-subtrees for $k = 5$.

Model of Computation: We assume the Word RAM [1,7] with standard bitwise Boolean and arithmetic operations ("+" and "$*$") on $w = \Theta(\log n)$ bits registers including *bitwise* AND "&", *bitwise* OR "|", *bitwise* NOT "$\sim$", *left shift* "$\ll$", and *right shift* "$\gg$", where n is an input size. We write a constant variable-length bit-vector as "1011". In this paper, a bit vector of length L is written as $X = b_{L-1} \cdots b_0 \in \{0, 1\}^L$, where the MSB b_{L-1} and LSB b_0 come in this order. For every i, we define the length and i-th bit of B by $|B| = L$ and $B[i] = b_i$, respectively.

An *enumeration algorithm* $\mathcal{A}$ receives an instance of size n and outputs all of m solutions without duplicates (See, e.g. [6]). For a polynomial $p(\cdot)$, $\mathcal{A}$ is of $O(f(n))$-delay using preprocessing $p(n)$ if the *delay*, which is the maximum computation time between two consecutive outputs, is bounded by $f(n)$ after preprocessing in $p(n)$ time.

3 Enumeration of At-Most k-Subtrees in a Tree

In Sec.3 and Sec.4, we present algorithms for computing the subtree-feature vector of T based on efficient compressed subtree enumeration.

Algorithm 1. The algorithm EnumAtMost for computing the feature vector H for the bit signatures of all subtrees with at most k nodes in an input tree T

```
 1: procedure EnumAtMost($T, k$)
 2:     $H \leftarrow \emptyset$;                    // A hash table $H$ representing a feature vector
 3:     for $r \leftarrow 1, \ldots, n$ do
 4:         Initialize bit-vectors $B$ and $X$;
 5:         RecAtMost($\{r\}, T, k$);
 6:     return $H$;
 7: procedure RecAtMost($S, T, k$)
 8:     If $H[S]$ is defined, then $H[S] \leftarrow H[S] + 1$ else $H[S] \leftarrow 1$;
 9:     Output BS($S$);
10:     If $|S| = k$, then return;
11:     for each extension point $v$ on $RMB(S)$ do
12:         Attach a new leaf $u$ to $v$ as the youngest child;
13:         Let $S \cup \{u\}$ be the resulting subtree;
14:         RecAtMost($S \cup \{u\}, T, k$);
```

The first algorithm that we present in this section is the compressed enumeration version of the constant delay enumeration algorithm for uncompressed subtrees of at most size k in an ordered tree by [2, 15, 19].

The Outline of the Enumeration Algorithm: In Algorithm 1, we show our algorithm EnumAtMost for at most k-subtrees and its subprocedure RecAtMost. This is a simple backtrack algorithm, which starts from a singleton tree as 0-subtree, and recursively expands the current $(i-1)$-subtree by attaching a new node u to some node $v = pa(u)$ on the current rightmost branch $RMB(S)$ to generate a new i-subtrees, until its size i becomes k (See [2]). Then, we say that some node $v \in T \setminus S$ *can be added to* S as a child of a node $u = pa(v)$ on $RMB(S)$ if v is the younger than any child of u contained in S. Such a parent node v on $RMB(S)$ is called the *extension point* and u is called the associated new child. If there is no such a node u, then the algorithm backtracks to the parent subtree. The *extension point set* is the set $XP(S) \subseteq RMB(S)$ of all extension points of S. The next lemma gives the characterization of $XP(S)$.

Lemma 1. *For any $u \in RMB(S)$, $u \in XP(S)$ iff there exists some $v \in T \setminus S$ such that (i) $v > \max(S)$ and (ii) v is younger than the youngest child of u in S.*

Example 1. For the 5-subtree $S_3 = \{2, 3, 8, 9, 10\}$ in Fig.1, $Lv(S_4) = \{3, 10\}$ and $Bd(S_4) = \{4, 5, 11, 12, 13\}$, $RMB(S_4) = \{2, 8, 9, 10\}$, and $XP(S_4) = \{8, 9\}$.

We will show how to incrementally maintain the extension set $XP(S)$ by growing S. For a singleton tree S consisting with the root $r = root(S)$ only, if r has a child in T then $XP(S) = \{r\}$, and otherwise $XP(S) = \emptyset$. For a subtree with more than one nodes, we have the next lemma.

Lemma 2. *Let S be any k-subtree of T with $k \geq 2$. Suppose that $k \geq 2$ and a k-subtree $R = S \cup \{v\}$ is obtained from a $(k-1)$-subtree S by attaching a new* *child v to its extension point $u = pa(v) \in XP(S)$. Then, $XP(R)$ is the set of vertices that satisfies the following (a)–(c):*

(a) For the parent, $pa(v) \in XP(R)$ iff v has a properly younger sibling in T.
(b) For the child, $v \in XP(R)$ iff v has some child in T.
(c) For any old extension point $x \in XP(S)$ other than $pa(v)$, $x \in XP(R)$ iff x is an ancestor of $pa(v)$.

In condition (c) of the above lemma, we note that any extension point x is either an ancestor or a descendant of $pa(v)$ in $XP(S)$ since $XP(S)$ is a subset of $RMB(S)$, a branch in S.

Fast Update of Bit Signatures: In our bit-parallel implementation of EnumAtMost, for each k-subtree, we maintain two bit-vectors of length $2k$, $B = BP(S)$ and $X = RPOS(XP(S))$, that represent the current subtree S and its extension point set $XP(S)$, respectively. For simplicity, we first describe the algorithm with bit-vectors whose length is no larger than the word length $w = \Theta(\log n)$. We efficiently update the bit-vectors B and X as follows.

Let $i \geq j$ and $ONE^{\ell}_{i:j} = 0^{i-1}1^{j-1+1}0^{\ell-j} \in \{0,1\}^{\ell}$ be the bit-mask of length ℓ whose i to j bits are filled with 1 bits and the other bits are filled with 0 bits, which can be computed by shift and subtraction in constant time for $i, j = O(\log n)$.

First, we initialize the bit-vectors B and X for the sets $S = \{r\}$ and $XP(S)$ by the following code: $B \leftarrow$ "01"; **if** r has a child on T **then** $X \leftarrow$ "01" **else** $X \leftarrow$ "00";

Next, the following code correctly updates the bit-vectors B and X when we compute the extension point set $XP(S \cup \{v\})$ for the new subtree $S \cup \{v\}$ from $XP(S)$ for the old one S, where $q = rpos(v)$ and $\ell = len(B)$:

– B is updated as follows.
$$B \leftarrow (B \,\&\, ONE^{\ell}_{\ell-1:q+1}) \ll 2 \;\mid\; (\text{“01”} \ll q) \;\mid\; (B \,\&\, ONE^{\ell}_{q:0});$$
– X is updated as follows.
```
X ← { ("1" ≪ q − 1) if v has a properly younger sibling }   // a parent
  | { ("1" ≪ q) if v has some child };                      // a child
  | (X & ONE^ℓ_{q−1:0}) ≫ 2                                 // others
```

From Lemma 2, we have the following lemma.

Lemma 3 (Update in the small subtree case). If $2k \leq w$, the above codes correctly updates the bit-vectors B and X in constant time using $O(1)$ words.

Lemma 4 (Update in the large subtree case). If $k = O(2^w)$, the bit-vectors B and X can be correctly updated in constant time using $O(k/w)$ words.

Proof. Proof sketch: We represent bit-vectors B and X as a doubly linked list of $b = O(\log n)$-bits blocks, each of which are maintained to store consecutive bits of length $\lceil b/2 \rceil < \ell \leq b$ (bits) similarly to [7]. In the update of B and X, we need to update only constant number of blocks, and thus, takes constant time. □

Theorem 1 (Compressed enumeration of at most k-subtrees). *For every $k \geq 1$, Algorithm 1 enumerates all compressed representations of the at most k-subtrees appearing in an ordered tree T in $O(1)$ time per solution, generated on the bit-vector $B \in \{0,1\}^{2k}$, using $O(k/w)$ words of space in addition to the space for an enumeration algorithm.*

From the above theorem, we observe that for every $k \geq 1$, all compressed representations of exact k-subtrees in T can be enumerated in $O(k)$ time per compressed representation using the same amount of space as above.

4 Enumeration of Exact k-Subtrees in a Tree

The second algorithm that we present in this section is the compressed enumeration version of the constant delay enumeration algorithm for uncompressed k-subtrees in an ordered tree by Wasa *et al.* [17].

The Outline of the Algorithm: The basic idea of Wasa *et al.'s* algorithm is as follows: Given a k-subtree S in an input tree T, we can obtain the other k-subtree S' from S by deleting one node from S and adding one node to S.

By repeating this process recursively using backtracking, for each node r in T, starting from the lexicographically least k-subtree with r as its root, we can enumerate all k-subtrees with root r appearing in T by recursively transforming the current k-subtree by the above process. This algorithm runs in $O(1)$ time per k-subtree by maintaining the node lists $DL(S) \subseteq Lv(S)$ and $AL(S) \subseteq Bd(S)$ to delete and to add, respectively. In Algorithm 2, we show our algorithm EnumExact and its subprocedure RecExact.

Fast Update of Bit Signatures: In the implementation with bit-operations, we use three bit-vectors B, L, and $A \in \{0,1\}^*$, where B is the *BP-vector* as defined in the previous section, L is the *leaf-vector* representing the set of leaves to delete, and A is the *add-vector* representing the set of nodes to add.

In this section, we give the efficient method for updating bit-vectors using bit parallel technique. A node v is an *exact extension point* in S if one of children of v can be attached to S. We define the sets $AL(S)$ and $DL(S)$ of nodes to add and to delete, and the set $EXP(S)$ of exact extension points by

$$DL(S) = \{\, x \in Lv(S) \mid x < minbord(S) \,\}. \tag{4}$$

$$AL(S) = \{\, x \in Bd(S) \mid x > maxleaf(S) \,\}. \tag{5}$$

$$EXP(S) = \{\, x \in S \mid x = pa(v), v \in AL(S) \,\}. \tag{6}$$

We give the following recurrence relation for $Lv(S)$, $Bd(S)$, and $EXP(S)$. In this subsection, $\leq$ denotes the DFS-ordering on $\mathcal{T}$.

Lemma 5. *Then, set is defined for any subset S.*

(a) If $S = \mathcal{I}_k$ is an initial k-subtree rooted at u, then $AL(\mathcal{I}_k)$ is the set $XP(\mathcal{I}_k)$ of all extension points, and $DL(\mathcal{I}_k)$ is the set $Lv(\mathcal{I}_k)$ of all leaves.

Algorithm 2. The algorithm EnumExact for computing the feature vector H for the bit signatures of all subtrees with exactly k nodes in an input tree T based on constant delay enumeration

```
 1: procedure EnumExact($T, k$)
 2:     $H \leftarrow \emptyset$;                  // A hash table $H$ representing a feature vector
 3:     Number the nodes of $T$ by the DFS-numbering;
 4:     Compute the initial $k$-subtree $\mathcal{I}_k$;
 5:     Initialize the related lists and pointers;
 6:     RecExact($\mathcal{I}_k, B, L, X; T, k$);
 7:     return $H$;

 8: procedure RecExact($S, B, L, X; T, k$)
 9:     Output $\mathtt{BS}(S)$;   $p \leftarrow \mathtt{MSB}(L)$;
10:     for each $\ell \in \mathsf{DelList}(S)$ do
11:         for each $\beta \in \mathsf{AddList}(S)$ such that $\beta \notin Ch(\max(Lv(S)))$ do
12:             $S \leftarrow \mathsf{Child}_1(S, \ell, \beta)$ by updating the related lists and pointers;
13:             Update bit sequences $B, L, X$;
14:             RecExact($S, B, L, X; T, k$);
15:             $S \leftarrow \mathcal{P}^1(S)$ by restoring the related lists and pointers;
16:             Restore bit sequences $B, L, X$;
17:             Modify $X$;
18:         Proceeds $p$;                  // to the next leaf position in $L$
19:     if $S$ is a $k$-pre-serial tree then
20:         $S \leftarrow \mathsf{Child}_2(S)$ by updating the related lists and pointers;
21:         Update bit sequences $sig(S), A, L$;
22:         RecExact($S, B, L, X; T, k$);
23:         $S \leftarrow \mathcal{P}^2(S)$ by restoring the related lists and pointers;
24:         Restore bit sequences $sig(S), A, L$;
```

(b) Let S be any k-subtree, $v \in AL(S)$, and $u \in DL(S)$ such that v is not a child of u. For any node x, the following conditions hold:

(i) $S' = (S \setminus \{u\}) \cup \{v\}$.

(ii) $Lv(S') = \{\, x \in Lv(S) \mid x \neq u \,\} \cup \{\, x \in T \setminus Lv(S) \mid Ch(x) \cap S = \{u\} \,\}$.

(iv) $DL(S') = \{\, x \in DL(S) \mid x < u \,\} \cup \{\, x \in T \setminus DL(S) \mid Ch(x) \cap S = \{u\} \,\}$.

(vi) $AL(S') = \{\, x \in AL(S) \mid x > v \,\} \cup \{\, x \in T \setminus AL(S) \mid x \notin Lv(T) \,\}$.

During the enumeration, the algorithm explicitly maintains the lists $Lv(S)$, $Bd(S)$, and $EXP(S)$. Using these lists and the pointers to the maximum leaf $maxleaf(S)$ and to the minimum border node $minbord(S)$, the algorithm implicitly represents the lists $DL(S)$ and $AL(S)$.

Next, we consider the generation of children of type I from Line 10 to Line 18 in Algorithm 2, and give the bit-parallel implementation of the update procedure for bit-vectors B, L, X, and pointers p and q to them, while the lists $Lv(S)$, $Bd(S)$ and $EXP(S)$ are maintained by the algorithm. During the enumeration, we maintain B, L, and X such that $B = BP(S)$, $L[rpos(v)] = 1$ iff $v \in Lv(S)$, and $X[rpos(v)] = 1$ iff $v \in EXP(S)$ for every $v \in T$. For initialization, we set

$B = BP(\mathcal{I}_k)$, $L = RPOS(Lv(\mathcal{I}_k))$, and $X = RPOS(EXP(\mathcal{I}_k))$ in $O(k)$ time by traversing the initial k-subtree $\mathcal{I}_k$).

Definition 1 (Update for children of type I). Suppose that we generate a child k-subtree $S' = (S \setminus \{u\}) \cup \{v\}$ of type I from the parent S. Then, we update the bit-vectors B, L and X as follows, where $p = rpos(u)$ and $\ell = len(B)$:

- The right position $q = rpos(v)$ can be computed from the bit-vector X using MSB by the following code: $q \leftarrow \mathtt{MSB}(X)$;
- B is updated by deleting the two bits "01" from right position p for node u, and inserting the two bits "01" for node v at right position $q - 1$.
  $B \leftarrow (B \ \&\ ONE^{\ell}_{\ell-1:p+2}) \mid (B \ \&\ ONE^{\ell}_{p-1:q+1}) \gg 2;$
  $B \leftarrow B \mid (\text{“01”} \ll q) \mid (B \ \&\ ONE^{\ell}_{q:0});$
- L is updated similarly to B. In addition, the two bits surrounding the delete position for u are overwritten with "01":
  $L \leftarrow (L \ \&\ ONE^{\ell}_{\ell-1:p+3}) \mid (L \ \&\ ONE^{\ell}_{p-1:q+1}) \ll 2 \mid (\text{“01”} \ll q)$
  $\mid (L \ \&\ ONE^{\ell}_{q-1:0}) \mid \{\, (\text{“01”} \ll p+1) \textbf{ if } Ch(pa(u)) = \{u\} \,\};$
- X is updated similarly to B. In addition, the two bits surrounding the delete position for u are overwritten with "01":
  $X \leftarrow (X \ \&\ ONE^{\ell}_{q-1:0}) \mid \{\, (\text{“1”} \ll q) \textbf{ if } v \text{ has a child } \}$
  $\mid (\text{“1”} \ll q-1) \textbf{ if } (\exists \text{younger sibling } r \text{ of } v)\ r \notin S;$

  Moreover, after the for-loop at line 17, we update X by deleting the extension point at the highest position one by one:
  $X \leftarrow X \ \&\ (\sim(\text{“1”} \ll (\mathtt{MSB}(X) - 1))) \textbf{ if } v \text{ has no younger sibling in } T;$
- We proceed the pointer $p = rpos(u)$ at line 18 by: $p \leftarrow \mathtt{MSB}(L \ \&\ ONE^{\ell}_{1:p-1});$

Next, the code from Line 19 to Line 24 generates the children of type II updating the bit-vectors B, L, and X.

Definition 2 (Update for children of type II). Suppose that we generate a child k-subtree $S' = (S \setminus \{u\}) \cup \{v\}$ of type II from the parent S. Then, we update the bit-vectors B, L and X as follows:

- The right position q can be computed $q \leftarrow \mathtt{MSB}(X)$;
- B is updated by deleting the most left bit of B and the most right bit of B, and inserting the two bits "01" for node v position q.
  $B \leftarrow (B \ \&\ ONE^{2k}_{2k-2:q+1}) \ll 1 \mid (\text{“01”} \ll q-1) \mid (B \ \&\ ONE^{2k}_{q:1}) \gg 1;$
- L is updated similarly to B. In addition, the right position bit of the inserting node v, is overwritten with "0".
  $L \leftarrow (L \ \&\ ONE^{2k}_{2k-2:q+1}) \ll 1 \mid (\text{“01”} \ll q-1) \mid (L \ \&\ ONE^{2k}_{q-1:1}) \gg 1;$
- X is updated by overwriting bits in the left of q with "0". In addition, two bits are overwritten with "1" if corresponding nodes satisfy some conditions.
  $X \leftarrow (X \ \&\ ONE^{2k}_{q-1:1}) \gg 1 \mid \{\, (\text{“1”} \ll q-1) \textbf{ if } v \text{ has a child; } \}$
  $\mid (\text{“1”} \ll q-2) \textbf{ if } (\exists \text{younger sibling } r \text{ of } v)\ r \notin S;$

In the small tree case that $2k \leq w$, it follows from Lemma 5 that the above procedure correctly updates the data structure in constant time per iteration using $O(1)$ words. In the large tree case that $k = O(2^w)$, a similar discussion to Lemma 4 shows that the procedure also run in constant time using $O(k/w)$ words. Therefore, we have the following theorem.

Theorem 2 (Compressed enumeration of exact k-subtrees). *Let T be an input tree and k be a positive integer. The algorithm* EnumExact *in Algorithm 2 enumerates all compressed representations of the exact k-subtrees appearing in T in $O(1)$ time per compressed representation, generated on the bit-vector $B \in \{0,1\}^{2k}$, using $O(k/w)$ words of space in addition to the space for an enumeration algorithm.*

From the above theorem, we obtained a constant-delay algorithm for compressed enumeration for exact k-subtrees, which improves on the $O(k)$-delay algorithm in Sec. 3 by a factor of $O(k)$.

5 Experiments

In the experiments, we compared the running time of the algorithms in Sec. 3 and Sec. 4 on artificial and real datasets. We implemented in C++ the algorithms EnumAtMost in Sec.3 and EnumExact in Sec.4, denoted by Atmost(α) and Exact(α), respectively, where α indicates the types of algorithms as follows:

- "Enum" enumerates subtrees without printing them.
- "Naive" is the original algorithm that first enumerates a subtree and then computes its bit signature.
- "Fast" is the modified algorithm that directly enumerates the bit signature of a subtree with bit-parallel signature maintenance.

The algorithms were complied by g++ 4.2.1 and were run on a PC (CPU Intel® Xeon(R) 3.6GHz, 34GB RAM) operating on Ubuntu OS 13.04.

Comparison of Algorithms on Real Data: As input, we use a phylogenetic tree of influenza virus of $n = 4240$ nodes, which was constructed from virus data in NCBI Influenza Virus Resource[1] by neighbor-joining method. In Fig.2, we show the running time of algorithms for computing the feature vector $\phi(T)$ of an input tree T varying the size of subtrees for $k = 15$ to 19. From this figure, for each of Exact and Atmost, the fast version (Fast) was faster than the naive version (Naive). For example, the speedup ratio for $k = 19$ were 36% for Exact, and 22% for Atmost. It depends on the type of update α which is faster between Exact and Atmost. In the case of Exact, the overhead of computing bit signatures over enumeration only (Enum) are 6 times for Fast and 9.5 times for Naive.

Comparison of Algorithms on Artificial Data: We used a artificial tree with size $n = 35$, which has depth one and consists of a root node and 34 leaves.

[1] http://www.ncbi.nlm.nih.gov/genomes/FLU/

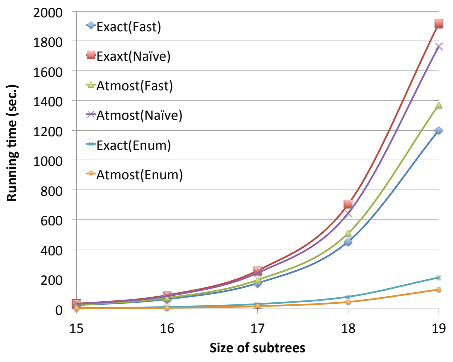


Fig. 2. The running time against the subtree size k on the real phylogenetic tree with $n = 4240$ nodes

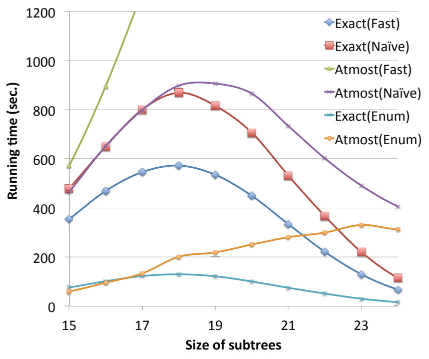


Fig. 3. The running time against the subtree size k on the artificial tree with $n = 35$ nodes and depth one

Fig.3 shows the result of experiment. The fastest algorithm was Exact(Fast), which was 34% faster than Exact(Naive) for $k = 18$.

Computing the Gram Matrix of a Set of Trees: To evaluate the usefulness of our algorithms in the context of tree mining [8, 12], we applied Exact(Fast) to similarly matrix computation [10, 11]. We computed $M = 70,532$ dependency trees, one tree per one Japanese sentence, in total size $1,140,098$ nodes and 3.8 MB from a Japanese newspaper corpus[2] in 44.9 MB by CaboCha.[3] Their average and standard deviation sizes are 16.16 and 12.65 (nodes). Applying Exact(Fast) to this dataset with $k = 4$, we computed the $M \times M$-similarly matrix for 4-subtrees using cosine-tree distance in $1,276.12$ seconds, where only 0.1% (1.61 seconds) of the time was spent for computing feature vectors and 99.9% for matrix computation.

Summary of Experimental Results: Overall, the proposed method (Fast) achieved around 22% to 30% speedup over the naive method (Naive). From the last experiment, the proposed method seems to have reasonable performance for data mining from middle size datasets.

6 Conclusion

In this paper, we studied the tree similarity based on bounded-sized ordered subtrees using fast compressed k-subtree enumeration. We presented two speed up techniques for bit signature generation based on bit-parallel approach. It is an interesting future problem to extend this work for various types of subtrees will be another future research problem including unordered subtrees.

[2] `http://www.ndk.co.jp/yomiuri/e_yomiuri/e_index.html`
[3] `http://code.google.com/p/cabocha/`

Acknowledgements. The authors would like thank anonymous reviewers for their comments which improved the correctness and the presentation of this paper very much, and thank Shin-ichi Nakano, Kunihiko Sadakane, and Tetsuji Kuboyama for helpful discussions and comments. This research was partly supported by MEXT Grant-in-Aid for Scientific Research (A), 24240021, FY20122015, and Grant-in-Aid for JSPS Fellows (25·1149).

References

1. Zaki, M.J.: Efficiently mining frequent trees in a forest. In: Proc. KDD 2002, pp. 71–80 (2002)
2. Asai, T., Abe, K., Kawasoe, S., Arimura, H., Sakamoto, H., Arikawa, S.: Efficient substructure discovery from large semi-structured data. In: Proc. SDM 2002 (2002)
3. Chim, H., Deng, X.: A new suffix tree similarity measure for document clustering. In: Proc. WWW 2007, pp. 121–130 (2007)
4. Collins, M., Duffy, N.: Convolution kernels for natural language. In: Proc. of Advances in Neural Information Processing Systems, NIPS, pp. 625–632 (2001)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. The MIT Press (2001)
6. Goldberg, L.A.: Polynomial space polynomial delay algorithms for listing families of graphs. In: Proc. ACM STOC 1993, pp. 218–225. ACM (1993)
7. Jansson, J., Sadakane, K., Sung, W.-K.: CRAM: Compressed random access memory. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part I. LNCS, vol. 7391, pp. 510–521. Springer, Heidelberg (2012)
8. Kashima, H., Koyanagi, T.: Kernels for semi-structured data. In: Proc. 19th ICML 2002, pp. 291–298. Morgan Kaufmann Publishers Inc. (2002)
9. Kimura, D., Kuboyama, T., Shibuya, T., Kashima, H.: A subpath kernel for rooted unordered trees. In: Huang, J.Z., Cao, L., Srivastava, J. (eds.) PAKDD 2011, Part I. LNCS, vol. 6634, pp. 62–74. Springer, Heidelberg (2011)
10. Kuboyama, T., Hirata, K., Aoki-Kinoshita, K.F.: An efficient unordered tree kernel and its application to glycan classification. In: Washio, T., Suzuki, E., Ting, K.M., Inokuchi, A. (eds.) PAKDD 2008. LNCS(LNAI), vol. 5012, pp. 184–195. Springer, Heidelberg (2008)
11. Kuboyama, T., Hirata, K., Kashima, H., Aoki-Kinoshita, K.F., Yasuda, H.: A spectrum tree kernel. Information and Media Technologies 22(2), 292–299 (2007)
12. Kudo, T., Maeda, E., Matsumoto, Y.: An application of boosting to graph classification. In: Proc. NIPS 2004 (2004)
13. Lakkaraju, P., Gauch, S., Speretta, M.: Document similarity based on concept tree distance. In: Proc. 19th ACM HT 2008, pp. 127–132 (2008)
14. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to information retrieval. Cambridge University Press (2008)
15. Nakano, S.: Efficient generation of plane trees. IPL 84(3), 167–172 (2002)
16. Tsuda, K., Kudo, T.: Clustering graphs by weighted substructure mining. In: Proc. 23rd ICML, pp. 953–960. ACM (2006)
17. Wasa, K., Kaneta, Y., Uno, T., Arimura, H.: Constant time enumeration of bounded-size subtrees in trees and its application. In: Gudmundsson, J., Mestre, J., Viglas, T. (eds.) COCOON 2012. LNCS, vol. 7434, pp. 347–359. Springer, Heidelberg (2012)
18. Xin, D., Han, J., Yan, X., Cheng, H.: Mining compressed frequent-pattern sets. In: Proc. VLDB 2005, pp. 709–720 (2005)
19. Zaki, M.J.: Efficiently mining frequent trees in a forest. In: Proc. KDD 2002, pp. 71–80 (2002)

Principal Directions-Based Pivot Placement

Fabrizio Angiulli and Fabio Fassetti

DIMES Department, University of Calabria, Rende, Italy
{f.angiulli,f.fassetti}@dimes.unical.it

Abstract. Determining a good sets of pivots is a challenging task for metric space indexing. Several techniques to select pivots from the data to be indexed have been introduced in the literature. In this paper, we propose a pivot placement strategy which exploits the natural data orientation in order to select space points which achieve a good alignment with the whole data to be indexed. Comparison with existing methods substantiates the effectiveness of the approach.

1 Introduction

The *similarity search* in metric spaces [7,12,14] is a fundamental task in a huge set of fields. In particular, *range queries*, which are of interest here, take as input the query object q and a radius R, and return all the objects of the dataset lying within distance R from q.

One of the main techniques for *indexing* objects from a metric space, is the *pivoting* based one [13,10,6,7,14,2]. In such approaches, the idea is to select a certain number of objects, called *pivots*. Due to the reverse triangle inequality, given two generic objects x and y, their distance cannot be smaller than the distance between x and p minus the distance between y and p, for any other object p. Hence, in order to answer to a range query, this lower bound can be exploited to discard those objects which do not lie within distance R from the query object q, namely whose lower bound is greater than R. By sketchily summarizing, at indexing time all the pairwise distances among the objects of the dataset and the pivots are stored in the index. At query time, first of all, the distances between the query object q and all the pivots are computed. Then, a *candidate selection phase* follows, which is accomplished by selecting those objects which are not discarded by exploiting the lower bounds computed through the pivots. The true neighbors of q are eventually retrieved by a *filtering phase* consisting in computing the actual distances among q and each candidate object.

In the best case, the objects returned by the candidate selection phase coincide with the query answer. However, minimizing the number of the spurious objects is a hard challenge. Usually, the greater the number of pivots, the smaller the number of spurious objects in the candidate set. By keeping fixed the number of pivots, it is known that a clever selection of pivots can drastically improve index performances. Given a dataset and an integer k, the *pivot selection problem* accounts for selecting a good set of k objects to be employed as pivots [1]. Normally, pivots are selected among the set of objects to be indexed. In some scenarios this may be the unique option, since it can be difficult to figure what a reasonable object out of those actually belonging to the data at hand can be. This is precisely the assumption made by all pivot selection techniques so far introduced in the literature [4,11,5].

Authors of [4] propose three different techniques, detailed in the following. According to the *Selection of N Random Groups* strategy, N groups each consisting in k pivots are randomly chosen from the dataset. For each group, the quality of the set of pivots is measured and the group scoring the maximum value of quality is returned. As for the *Incremental Selection* strategy, the idea is to randomly select N objects from the dataset. Then, the object (among the N selected ones) that alone scores the maximum value of quality is chosen as the first pivot p_1. The second pivot is the object p_2 such that maximizes the quality of the set of pivots $\{p_1, p_2\}$ and so on until k pivots are chosen. Finally, the *Local Optimum Selection* strategy consists in selecting an initial set of k pivots and a set of A pairs of objects. For N' times a so-called *victim* in the set of pivots is singled out and replaced by a better pivots chosen in a sample of X dataset objects. In order to individuate the victim, it is built a $A \times k$ matrix M, where $M(i,j)$ is the distance between the objects of the i^{th} pair computed through the pivot p_j. For each row, the maximum (d_{M}) and the second maximum (d_{M2}) values are computed. The *contribution* of each pivot p_j is computed as the sum over the A pairs of the difference between d_{M} and d_{M2}, if d_{M} is achieved in the column j and 0 otherwise. The *victim* is the pivot scoring the lowest contribution.

In [11] the *Sparse Spatial Selection* technique is proposed. The approach is based on the idea that if the pivots are well-distributed in the metric space they are able to discard more objects. The set of pivots is initially a singleton consisting in the first dataset object. Then, for each dataset object, this is chosen as a new pivot if and only if its distance from any pivot currently in the set is greater than $M\alpha$ where M is the maximum distance between any pair of dataset objects and α is a parameter whose empirically proven good value is 0.4. Such a method has next been extended in [5] where not only new pivots are added to the current set, but it is also checked if some pivot has become redundant and, in such a case, it is replaced by a better one. This latter task is accomplished by a estimating the contribution of an object in a fashion similar to the *Local Optimum Selection* approach.

As a main contribution, in this work we take a different perspective by considering the whole object domain in order to single out pivots, that is pivots can be objects that do not belong to the current dataset. We call this instance of the problem as *pivot placement problem.* Specifically, we consider here as reasonable scenario the case in which the object domain coincides with the d-dimensional Euclidean space $\mathbb{R}^d$. We describe a technique singling out the most promising directions which pivots should lie on. To the best of our knowledge this is the first work considering the pivot placement problem.

The remainder of the paper is organized as follows. Section 2 describes the pivot placement technique. Section 3 compares the approach here introduced with competitors. Finally, Section 4 draws the conclusions.

Algorithm 1: PPP(D, k)

```
    Input: $D$, set of objects; $k$, number of required pivots
1:  $n = |D|$
    // Compute small clusters
2:  $K = K_0$ // number of small clusters
3:  $C = GetSmallClusters(D, K)$
    // Compute intra-cluster directions
4:  $N = \frac{K(K+1)}{2}$ // number of directions
5:  for $i = 1$ to $K$ do
6:      $\mathbf{v}_i = \mathrm{PCA}(C_i)$
7:      $w_i = \frac{|C_i|(|C_i|-1)}{n(n-1)}$
8:      $\mathbf{c}_i = \frac{1}{|C_i|}\sum_{x \in C_i} x$
    // Compute inter-cluster directions
9:  $l = K + 1$
10: for $i = 1$ to $K - 1$ do
11:     for $j = i + 1$ to $K$ do
12:         $w_l = \frac{|C_i|(|C_j|-1)}{n(n-1)/2}$
13:         $\mathbf{v}_l = \frac{\mathbf{c}_i - \mathbf{c}_j}{\|\mathbf{c}_i - \mathbf{c}_j\|}$
14:         $l = l + 1$
    // Compute angles between directions
15: for $i = 1$ to $N$-1 do
16:     for $j = i$ to $N$ do
17:         $\alpha_{i,j} = \arccos(|\langle \mathbf{v}_i, \mathbf{v}_j \rangle|)$
18:         $\alpha_{j,i} = \alpha_{i,j}$
    // Perform prioritized fixed-width clustering
19: $\theta = \theta_0$ // cone angle
20: $\langle \ell, c \rangle = \mathrm{PFWC}(\boldsymbol{\alpha}, w, \theta)$
    // Determine the best pivots
21: $M = \frac{1}{n}\sum_{i=1}^{n} x_i$
22: $L = \max_{i=1}^{n} \|x_i - M\|$
23: $T = T_0$ // pivot displacement
24: for $i = 1$ to $\min\{c, k\}$ do
25:     $J_i = \{j : \ell_j = i\}$
26:     $p_i = M - T \cdot L \cdot \frac{\sum_{j \in J_i} w_j \mathbf{v}_j}{\sum_{j \in J_i} w_j}$
27: return $p$
```

Algorithm 2: PFWC($\boldsymbol{\alpha}$, w, θ)

```
    Input: $\boldsymbol{\alpha}$, pairwise angles; $w$, direction weights; $\theta$, angle threshold
    Output: $\ell$, labels (cluster numbers) assigned to directions; $c$, number of obtained clusters
1:  for $i = 1$ to $N$ do
2:      for $j = 1$ to $N$ do
3:          $\beta_{i,j} = 0$
4:          if $\alpha_{i,j} \le \frac{\theta}{2}$ then
5:              $\beta_{i,j} = 1$
6:  $m = N$
7:  $c = 0$
8:  while $m > 0$ do
9:      $c = c + 1$
10:     $\varrho = \boldsymbol{\beta} \cdot w$
11:     $i = \arg\max_i \varrho_i$
12:     for $j = 1$ to $N$ do
13:         if $\beta_{i,j} = 1$ then
14:             $m = m - 1$
15:             $\ell_j = c$
16:             $\beta_{j,\cdot} = 0$
17:             $\beta_{\cdot,j} = 0$
18: return $\langle \ell, c \rangle$
```

2 Principal Directions-Based Pivot Placement Algorithm

In this section, the PPP (for Principal directions-based Pivot Placement) algorithm is presented. The pseudo-code of the algorithm is reported in figure. It receives in input the dataset D and the number k of required pivots.

First, data is partitioned into a number K of homogeneous clusters $C_1, \ldots, C_K$, also called *small clusters* in the sequel. With this aim, the K-means algorithm is employed, that outputs a controlled number of prototypes having the property of minimizing the average distance to the associated groups. The small clusters are then exploited to determine directions connecting the data objects. A set $\mathbf{v}_1, \ldots, \mathbf{v}_N$ of $N = \frac{K(K+1)}{2}$ *directions*, that are versors associated with either a single small cluster ($1 \le i \le K$) or a pair of small clusters ($K < i \le N$), is populated. Each direction has also a *weight* w_i which represents the significance of the direction.

Directions $\mathbf{v}_i$ associated with single small clusters intend to capture the main direction along which the cluster objects spread. This direction is naturally captured by the *first principal component* of the cluster which is computed by exploiting Principal Component Analysis (PCA). PCA is [9] an orthogonal linear transformation to a new coordinate system such that the greatest variance by any projection of the data comes to lie on the first coordinate (called the first principal component). The weight w_i of the direction $\mathbf{v}_i$ consists in the number of pairwise objects of the cluster. As for the directions $\mathbf{v}_l$ $(K < l \leq N)$ associated with pairs C_i and C_j of different small clusters, they are defined in the terms of the vector linking the center of mass $\mathbf{c}_i$ of C_i and $\mathbf{c}_j$ of C_j, and the associated weight is given by the number of distinct pairs consisting of objects coming from the two clusters. Once intra-cluster and inter-cluster directions have been obtained, together with their importance, the matrix $\boldsymbol{\alpha}$ of pairwise angles $\alpha_{i,j}$ $(1 \leq i, j \leq N)$ formed by the directions $\mathbf{v}_i$ and $\mathbf{v}_j$ is computed, where $\alpha_{i,j} = \arccos(|\langle \mathbf{v}_i, \mathbf{v}_j \rangle|)$, in order to be exploited to determine directions along which pivots will be placed.

With this aim directions are grouped according to a *prioritized fixed width clustering strategy* (see algorithm PFWC in figure). Specifically, at each main iteration of PFWC, a seed direction $\mathbf{v}_i$ is selected and all the directions $\mathbf{v}_j$ (not already assigned to a group) such that $\alpha_{i,j} \leq \theta/2$ are assigned to the same group. These are the directions lying in the double cone having apex in the origin and axis collinear to $\mathbf{v}_i$ and forming an angle between surface and axis of $\theta/2$ radians. Seed directions are selected by assigning a *priority* ϱ_i $(1 \leq i \leq N)$ to those not already grouped and, then, by selecting the direction scoring the maximum priority. Priorities are computed as $\varrho = \boldsymbol{\beta} \cdot w$, that is to say ϱ_i is the sum of the weights associated with the directions that will become part of the group obtained by using direction $\mathbf{v}_i$ as seed.

Let M and L be the center of mass and the radius, respectively, of the dataset D. Moreover, let T a positive number, also called *displacement*, which is used to locate the hyper-sphere $\mathcal{S}$ of radius $T \cdot L$ centered in M. The c groups of directions returned by PFWC are finally exploited in order to place pivots. Pivot p_i $(1 \leq i \leq k)$ is obtained from the i-th group of directions, that are the directions $\mathbf{v}_j$ such that $\ell_j = i$. Specifically, p_i corresponds to one of the two points located at the intersection of the surface of the sphere $\mathcal{S}$ and the straight line passing through M whose direction is given by the mean of the versors in the i-th group.

3 Experiments

We used some collections of data available in the *Metric Spaces Library* [8] and in the *UCI KDD Repository* [3]: *COLOR* (112,682 points with 112 features), *LANDSAT* (275,465 points with 60 features), and *NASA* (40,150 points with 20 features). The PPP method has been compared with the *Selection of Random Groups* (RAND), *Incremental Selection* (INCR), *Local Optimum Selection* (LOCAL), and *Sparse Spatial Selection* (SSS) strategies. The parameters of PPP employed are the following: number of small clusters $K_0 = 100$, cone angle

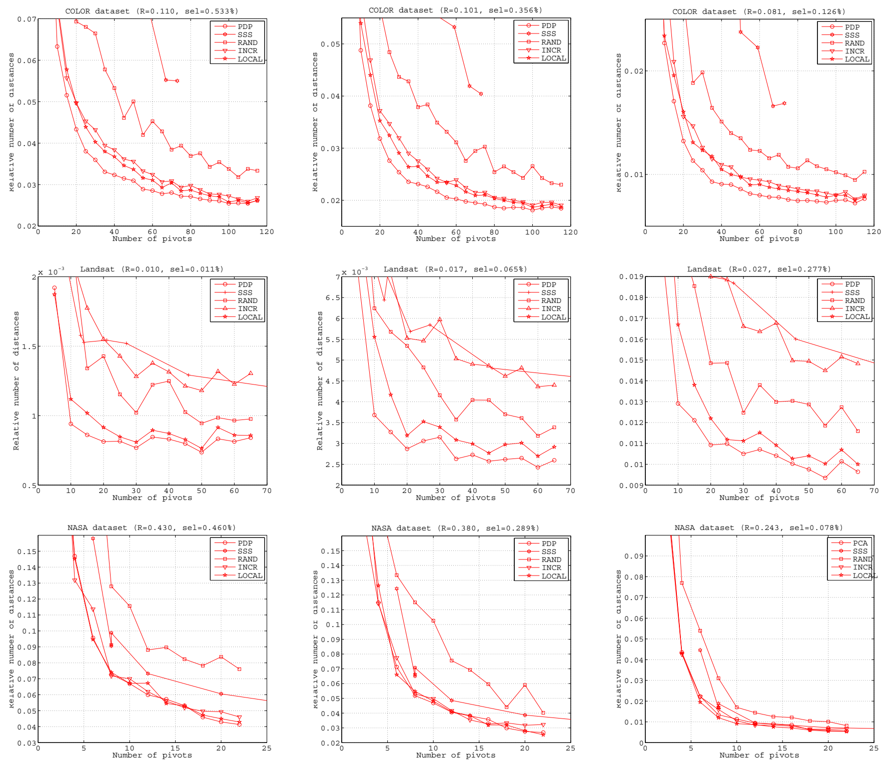


Fig. 1. Experimental results

$\theta_0 = \pi/3$, and pivot displacement $T_0 = 10$. As for the parameters of the competitors, we set them to the values suggested in [4]. Specifically, for RAND the number of sets of pivots is $N = 50$; for INCR the size of the sample is $N = 50$; for LOCAL the number of pairs A is set to 100,000, the number of iterations N' is set to k and the size of the sample is $X = N - 1$; finally, for SSS α has been set so that the number of selected pivots were close to k.

We performed a number of range query searches. Specifically, for each dataset, we considered three different radiuses. Radius values are such that on average the 0.01%, 0.05% and 0.1% of the dataset objects are selected. As for the queries, 5,000 objects have been picked out at random from the dataset. Experimental results are reported in Figure 1. We varied the number of pivots up to about the number of dimensions (reported on the x-axis) and compared the number of distances computed during both the *candidate selection* and the *filtering phase*, reported on the y-axis. Figures highlight that PPP performs better than competitors on the COLOR and LANDSAT datasets, and comparably to the other methods (INCR and LOCAL) on the NASA dataset. Experiments are encouraging, since they confirm that PPP may exhibit state-of-the-art performances as a method for pivot selection.

4 Conclusions

We addressed the pivot placement problem and provided a suitable algorithm to deal with it. Experiments confirmed that the proposed method, based on the idea of placing the pivots in the space by determining directions which achieve the better alignment with the whole dataset, improves indexing effectiveness. As for the future work, we are going to preform a more extensive experimental campaign, including further datasets and a comprehensive parameter sensitivity analysis. Also, theoretical assessment of the proposed strategy and extending it to general metric spaces is of interest.

References

1. Angiulli, F., Fassetti, F.: Indexing uncertain data in general metric spaces. IEEE Trans. Knowl. Data Eng. 24(9), 1640–1657 (2012)
2. Ares, L., Brisaboa, N., Esteller, M., Pedreira, O., Places, A.: Optimal pivots to minimize the index size for metric access methods. In: International Workshop on Similarity Search and Applications (SISAP), pp. 74–80 (2009)
3. Bache, K., Lichman, M.: UCI machine learning repository (2013)
4. Bustos, B., Navarro, G., Chávez, E.: Pivot selection techniques for proximity searching in metric spaces. Pattern Recognition Letters 24(14), 2357–2366 (2003)
5. Bustos, B., Pedreira, O., Brisaboa, N.R.: A dynamic pivot selection technique for similarity search. In: ICDE Workshops, pp. 394–401 (2008)
6. Chávez, E., Marroquín, J.L., Baeza-Yates, R.A.: Spaghettis: An array based algorithm for similarity queries in metric spaces. In: Symp. on String Processing and Information Retrieval (SPIRE), pp. 38–46 (1999)
7. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.: Searching in metric spaces. ACM Computing Surveys 33(3), 273–321 (2001)
8. Figueroa, K., Navarro, G., Chávez, E.: Metric spaces library (2007), `http://www.sisap.org/Metric_Space_Library.html`
9. Jolliffe, I.T.: Principal Component Analysis, 2nd edn. Springer (October 2002)
10. Micó, L., Oncina, J., Vidal, E.: 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)
11. Pedreira, O., Brisaboa, N.R.: Spatial selection of sparse pivots for similarity search in metric spaces. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 434–445. Springer, Heidelberg (2007)
12. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann Publishers Inc. (2005)
13. Yianilos, P.N.: Data structures and algorithms for nearest neighbor search in general metric spaces. In: ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 311–321 (1993)
14. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. Advances in Database Systems, vol. 32. Springer (2006)

Pivot Selection Strategies for Permutation-Based Similarity Search

Giuseppe Amato, Andrea Esuli, and Fabrizio Falchi

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo",
via G. Moruzzi, 1, Pisa 56124, Italy
`firstname.lastname@isti.cnr.it`

Abstract. Recently, permutation based indexes have attracted interest in the area of similarity search. The basic idea of permutation based indexes is that data objects are represented as appropriately generated permutations of a set of pivots (or reference objects). Similarity queries are executed by searching for data objects whose permutation representation is similar to that of the query. This, of course assumes that similar objects are represented by similar permutations of the pivots.

In the context of permutation-based indexing, most authors propose to select pivots randomly from the data set, given that traditional pivot selection strategies do not reveal better performance. However, to the best of our knowledge, no rigorous comparison has been performed yet. In this paper we compare five pivots selection strategies on three permutation-based similarity access methods. Among those, we propose a novel strategy specifically designed for permutations. Two significant observations emerge from our tests. First, random selection is always outperformed by at least one of the tested strategies. Second, there is not a strategy that is universally the best for all permutation-based access methods; rather different strategies are optimal for different methods.

Keywords: permutation-based, pivot, metric space, similarity search, inverted files, content based image retrieval.

1 Introduction

Given a set of objects C from a domain $\mathcal{D}$, a *distance function* $d : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$, and a query object $q \in \mathcal{D}$, a similarity search problem can be generally defined as the problem of finding a subset $S \subset C$ of the objects that are closer to q with respect to d. Specific formulations of the problem can, for example, require to find the k closest objects (k-nearest neighbors search, k-NN).

Permutation-based indexes have been proposed as a new approach to approximate similarity search [1,8,11,20]. In permutation-based indexes, data objects and queries are represented as appropriate permutations of a set of pivots $P = \{p_1 \dots p_n\} \subset \mathcal{D}$. Formally, every object $o \in \mathcal{D}$ is associated with a permutation Π_o that lists the identifiers of the pivots by their closeness to o, i.e., $\forall j \in \{1, 2, \dots, n-1\},\ d(o, p_{\Pi_o(j)}) \leq d(o, p_{\Pi_o(j+1)})$, where $p_{\Pi_o(j)}$ indicates the pivot at position j in the permutation associated with object o. For convenience, we denote the position of a pivot p_i, in the permutation of an object $o \in \mathcal{D}$, as $\Pi_o^{-1}(i)$ so that $\Pi_o(\Pi_o^{-1}(i)) = i$.

The similarity between objects is approximated by comparing their representation in terms of permutations. The basic intuition is that if the permutations relative to two objects are similar, i.e. the two objects *see* the pivots in a similar order of distance, then the two objects are likely to be similar also with respect to the original distance function d.

Once the set of pivots P is defined it must be kept fixed for all the indexed objects and queries, because the permutations deriving from different sets of pivots are not comparable. A selection of a "good" set of pivots is thus an important step in the indexing process, where the "goodness" of the set is measured by the effectiveness and efficacy of the resulting index structure at search time.

The paper is structured as follows. In Section 2 we discuss related work. Section 3 presents the strategies being compared. The tested similarity search access methods are presented in Section 4. Section 5 describes the experiments and comments their results. Conclusion and future work are given in Section 6.

2 Related Work

The study of pivot selection strategies for access methods usually classified as *pivot-based* [25] has been an active research topic, in the field of similarity search in metric spaces, since the nineties. Most access methods make use of pivots for reducing the set of data objects accessed during similarity query execution. In an early work by Shapiro [23], it was noticed that good performance were obtained by locating pivots *far away* from data clusters. In [24,18,5], following this intuition, several heuristics were proposed to select pivots between the *outliers* and far away from each other. Pivot selection techniques that maximize the mean of the distance distribution in the pivoted space were exploited in [7]. It was also argued that while good pivots are usually outliers, the reverse is not true. In [22,6], the problem of dynamic pivot selection as the database grows is faced. In [17] Principal Component Analysis (PCA) has been proposed for pivot selection. Principal components (PC) of the dataset are identified by applying PCA on it (actually a subset to make the method computationally feasible) and the objects in the dataset that are best aligned with PC vectors are selected as pivots.

Works that use permutation-based indexing techniques have mostly performed a random selection of pivots [1,8,11] following the observation that the role of pivots in permutation-based indexes appears to be substantially different from the one they have in traditional pivot-based access methods and also because the use of previous selection strategies did not reveal significant advantages. At the best of our knowledge, the only report on the definition of a specific selection techniques for permutation-based indexing is in [8], were it was mentioned that no significant improvement, with respect to random selection, was obtained by maximizing or minimizing the Spearman Rho distance through a greedy algorithm.

3 Pivot Selection Strategies

Permutation based access methods use pivots to build permutation that represent data objects. This paper compares four promising selection strategies used in combination with different permutation based indexes to make a comprehensive evaluation, and also to identify the specific features that can be exploited in the various cases.

As the baseline we tested the **random (rnd)** strategy, which samples pivots from the dataset following a uniform probability distribution.

3.1 Farthest-First Traversal (FFT)

A very well known topic in metric spaces is the *k-center* NP-hard problem that asks: given a set of objects C and an integer k, find a subset P of k objects in C that minimizes the largest distance of any object in C from its closest object in P. FFT (so called by Dasgupta in [9]) finds a solution close to optimal by selecting an arbitrary object $p_1 \in C$ and choosing, at each subsequent iteration, the object $p_i \in C$ for which $\min_{1 \leq j \leq i} d(p_i, p_j)$ is maximum. In [14], it has been proved that FFT achieves an approximation, with respect to the optimal solution, of at most a factor of 2. Note that FFT actually tries to maximize the minimum distance between the pivots, which intuitively could be a desirable property of the resulting pivot set. The computational cost of this algorithm is $O(n|C|)$, where n is the number of requested pivots.

3.2 *k*-Medoids (kMED)

Originally proposed in [15], k-medoids is a partitional clustering algorithm that tries to minimize the average distance between objects and selected cluster medoids. k-medoids is very similar to k-means. The difference is that it uses objects from the dataset as representatives of the centers of the clusters rather than computing centroids, which could be not possible in general metric spaces. Moreover, k-medoids is also more robust to noise and outliers because it minimizes the distances instead of their square. While FFT minimizes the largest distance of an object from its closest pivot, k-medoids minimizes the average distance of the objects from their closest pivot.

3.3 Pivoted Space Incremental Selection (PSIS)

In [7] several strategies for selecting pivots were proposed and tested considering the average distance of the transformed space obtained leveraging on the set of selected pivots and on the triangle inequality of the original metric space [25]. The presented algorithms try to maximize the average distance in the pivoted space defined as: $D_P(x, y) = \max_{1 \leq i \leq n} |d(x, p_i) - d(y, p_i)|$.

The goal of the proposed algorithm is to have good lower bounds D_P for the original distance d. Bustos et al. observed that the chosen pivots are outliers but that not all outliers are good pivots for maximizing the average D_P. The overall best between the proposed methods is the incremental selection technique. This technique greedily selects the first and subsequent pivots maximizing D_P on a set of pairs of objects in C.

3.4 Balancing Pivot-Position Occurences (BPP)

While the other pivot selection approaches mainly originate from the literature on similarity search in metric spaces and clustering, in this section we propose an algorithm specifically intended for permutation-based access methods. The intuition suggests that each pivot should appear in the various positions in the permutations uniformly. In fact, if a pivot $p_i \in P$ appears in the same position in all the permutations, such pivot is useless.

Let $c(p_i, j) = |\{\Pi_o : \Pi_o^{-1}(i) = j\}|$ be the number of permutations where p_i appears in position j. The mean value of $c(p_i, j),\ 1 \leq j \leq n$ is independent of the specific set of pivots and is always equal to $|C|/n$. BPP tries to minimize the deviation of $c(p_i, j)$ values from their mean. The algorithm starts by randomly selecting a set $P \in C$ of $\hat{n} > n$ candidate pivots and evaluating the permutations for all the objects $o \in C$ (or a subset $S \subset C$). At each iteration, the algorithm evaluates the effect of removing each $p_i \in P$ (or a fixed number t of candidate pivots) on the distribution of $c(p_i, j)$ and removes the pivot for which the minimum average standard deviation is obtained. The algorithm ends when the number of candidate pivots satisfies the request, i.e. $|P| = n$.

In [1] it was observed that the first pivots in the permutations, i.e. the nearest to the object, have been proved to be more relevant. Thus, in our experiments, we applied this general algorithm considering $c(p_i, j)$ for $1 \leq j \leq l$ where l is the actual length of the permutation we are considering. The complexity of the algorithm is thus $O(\hat{n}|S|)$ for initialization using the distance d, and $O(t\hat{n}^2|S|)$ for the iterative selection where the cost is the evaluation of each candidate pivot occurrence in the permutations.

4 Similarity Access Methods

We have compared the pivot selection strategies on three permutation based index structures that reasonably cover the various approaches adopted in literature by methods based on permutations.

4.1 Permutations Spearman Rho (PSR)

The idea of predicting the closeness between elements comparing the way they "see" a set of pivots was originally proposed in [8]. As distance between permutations, Spearman Rho, Kendall Tau and Spearman Footrule [12] were tested. Spearman Rho revealed better performance. Given two permutations Π_x and Π_y, Spearman Rho is defined as:

$$S_\rho(\Pi_x, \Pi_y) = \sqrt{\sum_{1 \leq i \leq n} (\Pi_x^{-1}(i) - \Pi_y^{-1}(i))^2}$$

When a k-NN search is performed, a candidate set of results of size $k > k'$ is retrieved considering the similarity of the permutations based on S_ρ (in our experiments we fixed $k' = 10k$). This set is then reordered considering the original distance d. In [8] an optimal incremental sorting [21] was used to improve efficiency when the candidate set of results to be retrieved using the Spearman Rho is not known in advance. In this work we just perform a linear scan of the permutations defining the size of the candidate set in advance.

As already mentioned, the most relevant information of the permutation Π_o is in the very first, i.e. nearest, pivots. Thus, we decided to test also truncated permutations. In this case we used the Spearman Rho distance with location parameter $S_{\rho,l}$ defined in [12], which is intended for the comparison of top-l lists. $S_{\rho,l}$ differs from S_ρ for the use of an inverted truncated permutation $\tilde{\Pi}_o^{-1}$ that assumes that pivots further than $p_{\Pi_o(l)}$ from o being at position $l + 1$. Formally, $\tilde{\Pi}_o^{-1}(i) = \Pi_o^{-1}(i)$ if $\Pi_o^{-1}(i) \leq l$ and $\tilde{\Pi}_o^{-1}(i) = l + 1$ otherwise.

4.2 MI-File

The Metric Inverted File approach (MI-File) [2,1] uses an inverted file to store relationships between permutations. It also uses some approximations and optimizations to improve both efficiency and effectiveness. The basic idea is that entries (the lexicon) of the inverted file are the pivots P. The posting list associated with an entry $p_i \in P$ is a list of pairs $(o, \Pi_o^{-1}(i))$, $o \in C$, i.e. a list where each object o of the dataset C is associated with the position of the pivot p_i in Π_o.

As already mentioned, in [1] it was observed that truncated permutations can be used without huge lost of effectiveness. MI-File allows truncating the permutation of both data and query objects independently. We denote with l_x the length of the permutation used for indexing and with l_s the one used for searching (i.e. the length of the query permutation).

The MI-File also uses a strategy to read just a small portion of the accessed posting lists, containing the most promising objects, further reducing the search cost. The most promising data objects in a posting list, associated with a pivot p_i for a query q, are those whose position of the pivot p_i, in their associated permutation, is closer to the position of p_i in the permutation associated with q. That is, the promising objects are the objects o, in the posting list, having a small $|\Pi_o^{-1}(i) - \Pi_q^{-1}(i)|$. To control this, a parameter is used to specify a threshold on the maximum allowed position difference (*mpd*) among pivots in data and query objects. Provided that entries in posting lists are maintained sorted according to the position of the associated pivot, small values of *mpd* imply accessing just a small portion of the posting lists.

Finally, in order to improve effectiveness of the approximate search, when the MI-File execute a k-NN query, it first retrieves $k \cdot amp$ objects using the inverted file, then selects, from these, the best k objects according to the original distance. The factor $amp \geq 1$, is used to specify the size of the set of candidate objects to be retrieved using the permutation based technique, which will be reordered according to the original distance, to retrieve the best k objects.

The MI-File search algorithm computes incrementally a relaxed version of the Footrule Distance with location parameter l between the query and data objects retrieved from the read portions of the accessed posting lists.

4.3 PP-Index

The Permutation Prefix Index (PP-Index) [10,11] associates each indexed object o with the *short* prefix Π_o^l, of length l, of the permutation Π_o. The permutation prefixes of the indexed objects are indexed by a *prefix tree* kept in main memory. All the indexed objects are serialized sequentially in a *data storage*, kept on disk, following the lexicographic order defined by the permutation prefixes.

At search time the permutation prefix Π_q^l of the query q is used to find, in the prefix tree, the smallest subtree which includes at least $z \geq k$ candidates (z is a parameter of the search function). All the $z' \geq z$ candidates that are included in that subtree, i.e., $o_1 \ldots o_{z'}$, are then retrieved from the data storage and sorted, using a max-heap of k elements, by their distance $d(q, o_i)$, thus determining the approximated k-NN result.

A key property of PP-Index is that any subtree of the prefix tree maps directly into a single sequence of *contiguous* objects in the data storage. The sequential access to disk is crucial for the search efficiency. For example, in our experimental setup, random access read from disk of data representing 10,000 objects from the test dataset (described in Section 5.1) takes 87.4 seconds, while a sequential read of the same number of objects takes 0.14 seconds. Computing 10,000 distances between objects in the test dataset takes only 0.0046 seconds, which indicates how having good disk access patterns is the key aspect for efficiency.

The approach of PP-Index to similarity search is close to the one of M-Index [19] , which uses permutation prefixes to compute a mapping of any object to a real number that is then used as the key to sequentially sort the indexed objects in a secondary memory data structure such as a sequential file of a B$^+$-tree.

Both PP-Index and M-Index share many intuitions with the Locality-Sensitive Hashing (LSH) model [13,20]. For example, following the same principle of Multi-Probe LSH [16], the PP-Index adopts a *multiple-query* strategy that generates additional queries by performing local permutations on the original permutation prefix of the query object, i.e. retrieving additional candidates that are still close to the query because their permutation prefix differ only for a swap in a pair of adjacent pivots. The first pair that is swapped is the one that has the minimum difference of distances with respect to the query, i.e. $\min_j(d(q, p_{\Pi_q(j+1)}) - d(q, p_{\Pi_q(j)}))$, and so on. Note that it may happen that some of the additional queries end up in selecting the same subtree of other queries, so that the number of sequences of candidates objects accessed on disk may be less than the number of queries.

5 Experiments

5.1 Experimental Settings

Datasets and Groundtruth: Experiments were conducted using the CoPhIR dataset [4], which is currently the largest multimedia metadata collection available for research purposes. It consists of a crawl of 106 millions images from the Flickr photo sharing website. We have run experiments by using as the distance function d a linear combination of the five distance functions for the five MPEG-7 descriptors that have been extracted from each image. As weights for the linear combination we have adopted those proposed in [3]. As the ground truth, we have randomly selected 1,000 objects from the dataset as test queries and we have sequentially scanned the entire CoPhIR to compute the exact results.

Evaluation Measures: All the tested similarity search techniques re-rank a set of approximate result using the original distance. Thus, if the k-NN results list $\tilde{\mathcal{R}}_k$ returned by a search technique has an intersection with the ground truth $\mathcal{R}_k$, the objects in the intersection are ranked consistently in both lists. The most appropriate measure to use is then the *recall*: $|\tilde{\mathcal{R}}_k \cap \mathcal{R}_k|/k$. In the experiments we fixed the number of results k requested to each similarity search techniques to 100 and evaluated the *recall@r* defined as $|\tilde{\mathcal{R}}_r \cap \mathcal{R}_r|/r$ where $\tilde{R}_r$ indicates the sub-list of the first r results in $\tilde{R}_k$ $(1 \leq r \leq k)$. Note that, being the two lists consistently ordered, $\tilde{\mathcal{R}}_k \cap \mathcal{R}_r \subset \tilde{\mathcal{R}}_r$ always holds and thus $\tilde{\mathcal{R}}_r \cap \mathcal{R}_r = \tilde{\mathcal{R}}_k \cap \mathcal{R}_r$, i.e. none of the results in $\tilde{\mathcal{R}}_k$ after the r-th position can give a contribute to *recall@r*. Given that the queries were selected from the dataset and that all the tested access methods always found them, we decided to remove each query from the relative approximate result list. In fact, not removing them would result in artificially raising the *recall@r* for small values of r.

The average query cost of each tested technique was measured adopting a specific cost model that will be specified in Section 5.2.

Selection Techniques Parameters: Given previous results reported in [2,1,10,11] we decided to use 1,000 pivots. The parameters used for each selection strategy were selected so that they required almost the same time to be computed (about 10 hours):

- FFT: We selected the pivots among a subset of 1 million randomly selected objects performing at each iteration 100,000 tries for selecting the added pivot.
- kMED: We performed the clustering algorithm on a subset of 1 million randomly selected objects.
- PSIS: We randomly selected 10,000 pairs of objects from the dataset and performed 10 trials at each iteration.
- BPP: We randomly selected a set of 10,000 candidate pivots and tested them on 100,000 randomly selected objects performing at each iteration no more than 100 trials for selecting the pivot to be removed.

5.2 Results

For all the tested similarity access methods we show a pair of figures. On the first one we report *recall@r* obtained by the various selection strategies keeping the parameters of the access method settings. Even if the parameters are fixed, the use of different sets of pivots results in different average query cost which can not be inferred from this figure. For this reason, in the second figure we report an orthogonal evaluation that compares the *recall*@10 versus the query cost while varying some parameters of the access methods.

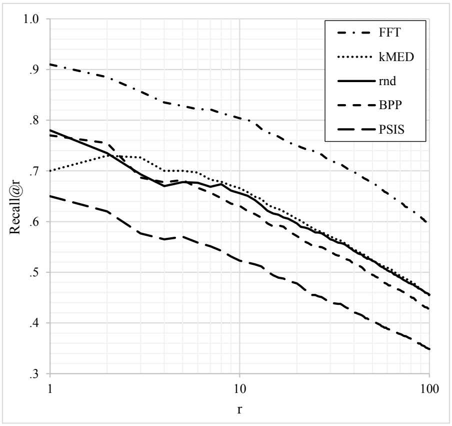


Fig. 1. *Recall@r* obtained by PSR for l=100 varying r

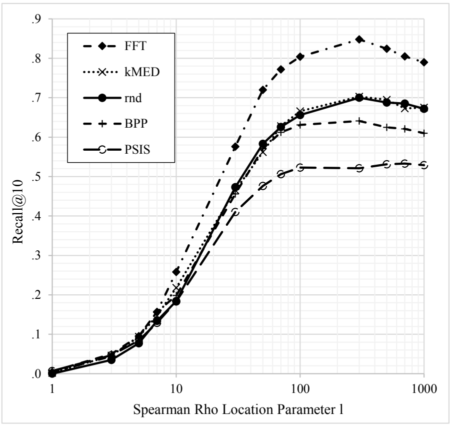


Fig. 2. *Recall@10* obtained by PSR for various location parameters

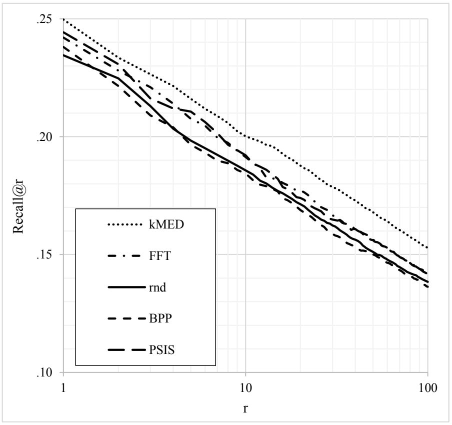


Fig. 3. *Recall@r* varying r obtained by the PP-Index using the multiple-query search (eight additional queries)

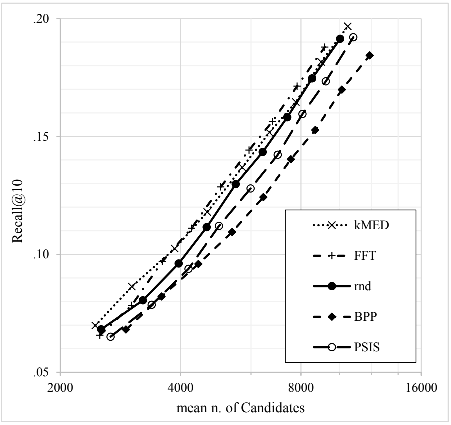


Fig. 4. *Recall@10* versus the number of candidates accessed (z') by the PP-Index when using the multiple-query search method with zero (lower left corner) to eight (upper right) of additional queries

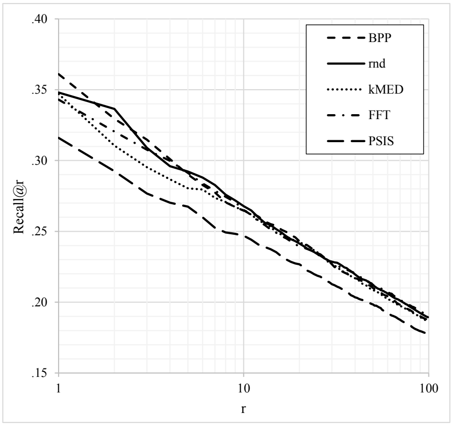


Fig. 5. *Recall@r* obtained by the MI-File using $l_s = 5$, varying the number of retrieved objects r from 1 to 100

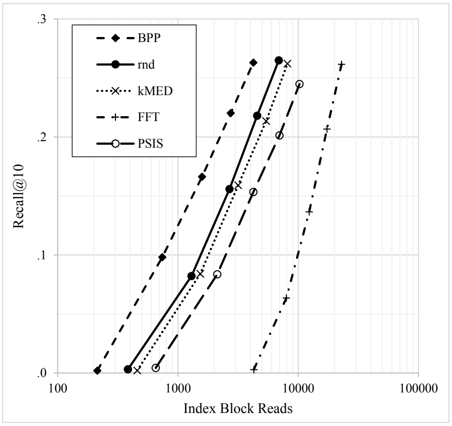


Fig. 6. *Recall@10* obtained by MI-File ranging l_s from 1 to 5

PSR: In Figure 1 we report the *recall@r* obtained by PSR for location parameter $l = 100$. The results show that FFT outperforms the other techniques in terms of effectiveness. PSIS performs significantly worse than all the others while the rest of the strategies obtained very similar results. In Figure 2 we tested various values of location parameter l, which directly impacts the query cost by reducing the index size and the permutation comparison cost. The results confirm that FFT significantly outperforms the others but also reveal that the differences are more relevant when l is closer to n, i.e. when more complete permutations are used. For values of l greater than 100, none of the techniques reported significant variations. The values of l used for the results reported in Figure 1 was chosen according to this observation.

PP-Index: Following the results of [11], we tested the PP-index by setting the length of the prefixes l to 6, and the values of z to 1,000. We tested both single- and multiple-query search, exploring a range of additional queries from 1 to 8. As the reference configuration we have chosen the one using a multiple-query search method with eight additional queries (nine total). As already noted in Section 4.3, some of the additional queries may result in selecting the same subtree of candidates. In fact, only 4.61 sequential blocks of candidates are accessed on disk on average for the above configuration.

Figure 3 shows that the PP-Index obtains its best results when using the kMED strategy, which is clearly better than the other strategies. FFT and PSIS form a group of second best strategies, followed by rnd and BPP, which are the worst performing ones. With respect to the other tested access methods, the PP-Index resulted to be more robust (or less sensitive) to the change of the pivot selection strategy. The recall curves for the various strategies have an almost identical slope and there is only an average difference of 1.3% between the best and worst strategies, almost constant across all the recall levels.

For the PP-Index, we have measured the query cost induced by the various strategies in terms of number of candidate objects selected by the queries on the prefix tree. Figure 4 shows that the best two strategies with respect to the recall/cost tradeoff are kMED and FFT, followed by rnd and PSIS, with BPP being the worst one. On the nine queries setup BPP needs about 20% more candidates to score a slightly worse recall than FFT. Again, the differences between the various strategies are smaller than those observed for the other access methods.

Note that the X axis of Figure 4 has a logarithmic scale. The almost straight lines indicate that the number of candidates grows with a logarithmic trend as more queries are used with the multiple-query search strategy, while the recall grows linearly, indicating that the multiple-query strategy has a very convenient recall/cost trend.

In summary, the kMED strategy resulted to be the best one, resulting in higher recall at a competitive cost.

MI-File: MI-File was tested indexing data objects using the closest 100 pivots ($l_x = 100$). Queries were executed ranging the number of closest pivots from 1 to 5, i.e. $l_s \in \{1, \dots 5\}$ (see Section 4.2). The maximum allowed position difference among pivots in data and query objects was 5 ($mpd = 5$). The size of the set of candidate objects retrieved was set to be 50 times k, ($amp = 50$).

Figure 5 shows the results obtained using l_s fixed to 5. For $r < 10$, BPP and rnd reveal better performance, while for $r > 10$ all the strategies almost overlap, except PSIS that is always the worst.

Figure 6 shows the results varying l_s from 1 to 5. Larger values of l_s imply larger number of disk blocks reads. It can be seen that once a target recall value is fixed, the cost needed by the MI-File to achieve such recall, varies significantly among the strategies. The cost needed to achieve a specific recall using the BPP method is one order of magnitude smaller than using the FFT method. For instance, the cost needed to obtain a $recall@10$ of 0.26 is 3,000 disk block reads using BPP, while the same recall requires 25,000 disk block reads using FFT.

The BPP method is overall the one offering the best performance with MI-File. The recall value obtained using $l_s = 5$ is mostly at the top. The cost needed to execute queries is significantly lower than all the other methods. This can be explained by the fact that, as discussed in Section 3.4, the BPP strategy has been designed to distribute the positions of the various pivots uniformly across the various permutations. This means that the posting lists of the MI-File are well balanced and that they tend to contain blocks of entries, related to the same pivot position, of equal size. As a consequence, there are no posting lists that are very long and that are also mostly accessed for any query, simultaneously improving effectiveness and efficiency.

6 Conclusion and Future Work

In this paper we compared five pivots selection strategies on three permutation-based access methods. For all the tested access methods we found at least one strategy that significantly outperforms the random selection. Another interesting point is that there is not a strategy that is universally the best for all the access methods. The PSR method, i.e. the sequential scan of the permutations adopting the Spearman Rho with location parameter l distance, largely benefited from the use of FFT. For PP-Index the best strategy has been kMED even if the performance differences are small. The novel proposed BPP strategy significantly outperformed the others when used in combination with the MI-File. This means that even if all the tested access methods are permutation-based, they significantly differ in the way they exploit the permutation space.

The CoPhIR collection is one of largest non-synthetic collections available for experiments on similarity search and its objects have a relatively high dimensionality. The results we have observed on this collection should thus be a good reference for practical applications that have similar characteristics (e.g., large collections of images). We are planning to extend the comparison on other collections with different characteristics in terms of data type, collection size and dimensionality. For the future we also plan to expand the comparison to other data structures, such as the M-Index [19], and to test novel strategies that make use of information on the queries, e.g., from a query log (as suggested in [11]).

References

1. Amato, G., Gennaro, C., Savino, P.: Mi-file: Using inverted files for scalable approximate similarity search. Multimedia Tools and Applications- An International Journal (November 2012) (online first)
2. Amato, G., Savino, P.: Approximate similarity search in metric spaces using inverted files. In: Proceedings of the 3rd International Conference on Scalable Information Systems, InfoScale 2008, pp. 28:1–28:10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, Brussels (2008)
3. Batko, M., Falchi, F., Lucchese, C., Novak, D., Perego, R., Rabitti, F., Sedmidubsky, J., Zezula, P.: Building a web-scale image similarity search system. In: Multimedia Tools and Applications
4. Bolettieri, P., Esuli, A., Falchi, F., Lucchese, C., Perego, R., Piccioli, T., Rabitti, F.: Cophir: a test collection for content-based image retrieval. CoRR, abs/0905.4627 (2009)
5. Brin, S.: Near neighbor search in large metric spaces. In: Proceedings of 21th International Conference on Very Large Data Bases, VLDB 1995, Zurich, Switzerland, September 11-15, pp. 574–584. Morgan Kaufmann (1995)
6. Bustos, B., Pedreira, O., Brisaboa, N.: A dynamic pivot selection technique for similarity search. In: IEEE 24th International Conference on Data Engineering Workshop, ICDEW 2008, pp. 394–401 (2008)
7. Bustos, B., Navarro, G., Chávez, E.: Pivot selection techniques for proximity searching in metric spaces. Pattern Recogn. Lett. 24(14), 2357–2366 (2003)
8. Chávez, E., Figueroa, K., Navarro, G.: Effective proximity retrieval by ordering permutations. IEEE Trans. Pattern Anal. Mach. Intell. 30(9), 1647–1658 (2008)
9. Dasgupta, S.: Performance guarantees for hierarchical clustering. In: Kivinen, J., Sloan, R.H. (eds.) COLT 2002. LNCS(LNAI), vol. 2375, pp. 351–363. Springer, Heidelberg (2002)

10. Esuli, A.: Mipai: Using the pp-index to build an efficient and scalable similarity search system. In: SISAP, pp. 146–148 (2009)
11. Esuli, A.: Use of permutation prefixes for efficient and scalable approximate similarity search. Information Processing & Management 48(5), 889–902 (2012)
12. Fagin, R., Kumar, R., Sivakumar, D.: Comparing top k lists. In: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2003, pp. 28–36. Society for Industrial and Applied Mathematics, Philadelphia (2003)
13. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: Proceedings of 25th International Conference on Very Large Data Bases, VLDB 1999, pp. 518–529 (1999)
14. Gonzalez, T.F.: Clustering to minimize the maximum intercluster distance. Theor. Comput. Sci. 38, 293–306 (1985)
15. Kaufman, L., Rousseeuw, P.J.: Finding groups in data: an introduction to cluster analysis. John Wiley and Sons, New York (1990)
16. Lv, Q., Josephson, W., Wang, Z., Charikar, M., Li Multi-probe, K.: lsh: efficient indexing for high-dimensional similarity search. In: Proceedings of the 33rd International Conference Very Large Data Bases, VLDB 2007, Vienna, Austria, pp. 950–961 (2007)
17. Mao, R., Miranker, W.L., Miranker, D.P.: Dimension reduction for distance-based indexing. In: Proceedings of the Third International Conference on SImilarity Search and APplications, SISAP 2010, pp. 25–32. ACM, New York (2010)
18. Micó, M.L., Oncina, J., Vidal, E.: 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. Novak, D., Batko, M., Zezula, P.: Metric index: An efficient and scalable solution for precise and approximate similarity search. Inf. Syst. 36(4), 721–733 (2011)
20. Novak, D., Kyselak, M., Zezula, P.: On locality-sensitive indexing in generic metric spaces. In: Proceedings of the Third International Conference on SImilarity Search and APplications, SISAP 2010, pp. 59–66. ACM, New York (2010)
21. Paredes, R., Navarro, G.: Optimal incremental sorting. In: In Proc. 8th Workshop on Algorithm Engineering and Experiments (ALENEX), pp. 171–182. SIAM Press (2006)
22. Pedreira, O., Brisaboa, N.R.: Spatial selection of sparse pivots for similarity search in metric spaces. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 434–445. Springer, Heidelberg (2007)
23. Shapiro, M.: The choice of reference points in best-match file searching. Commun. ACM 20(5), 339–343 (1977)
24. Yianilos, P.N.: Data structures and algorithms for nearest neighbor search in general metric spaces. In: Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1993, pp. 311–321. Society for Industrial and Applied Mathematics, Philadelphia (1993)
25. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search - The Metric Space Approach. Advances in Database Systems, vol. 32, pp. 1–191. Kluwer (2006)

Quantized Ranking for Permutation-Based Indexing

Hisham Mohamed and Stéphane Marchand-Maillet

Université de Genève, Geneva, Switzerland
{hisham.mohamed,stephane.marchand-maillet}@unige.ch

Abstract. Similarity search, translating into the nearest neighbor search problem, finds many applications for information retrieval and visualization, machine learning and data mining. The large volume of data that typical applications should handle imposes to find approximate solutions for the similarity search problem. Permutation-based indexing is one of the most recent techniques for approximate similarity search. Objects are represented by lists ordering their distances to a set of selected reference objects, following the idea that two neighboring objects have the same surrounding. In this paper, we propose a quantized representation of the permutation lists with its related data structure for effective retrieval. Our novel permutation-based indexing strategy is built to be fast, memory efficient and scalable without excessively sacrificing on search precision. This is experimentally demonstrated in comparison to existing proposals using several large-scale dataset of millions of documents and different dimensions.

Keywords: Metric Permutation table, Quantized Ranking, Permutation-Based Indexing, Approximate Similarity Search, Large-Scale Indexing.

1 Introduction

Similarity search [1] aims to extract the most similar objects to a given query. It is a fundamental operation for many applications, such as text plagiarism to track the similarity between an article against a database of texts, multiple genome comparison to find all the similarities between one or more genes, and multimedia retrieval to find the most similar picture or video to a given example.

Various indexing/search structures have been proposed to perform similarity search over high dimensional data. Unfortunately, it turns out that the search complexity increases when the dimensional increases, due to the curse of dimensionality [2]. Whenever the data dimension exceeds 8 to 20, K-d tree and similar data structures require scanning a large part of the database [3]. Accordingly, their performance becomes similar to the brute-force linear search techniques [4, 5].

In metric spaces, there are two common search scenarios, namely the range query and the K-nearest neighbor search. Several techniques have been developed for improving the performance in responding such queries, by decreasing the final number of direct distance calculations [5]. One of the most promising approaches is to perform approximate searching [6, 7]. This offers solutions to improve the retrieval time at the price of a slightly reduced precision.

Permutation-based indexing [8, 9] is a recent technique for approximate search in metric spaces. The idea is to represent each object by a list of permutations of selected reference items based on distance information. The information of similarity between any two objects is then derived by comparing the two corresponding permutation lists (see section 3 for details). In this work, we propose a new technique to reduce memory usage and improve the search precision and the running time. Our proposed technique is based on a quantized representation of the permutation lists. We then map our technique to a novel data structure (Metric Permutation Table (MPT)). We show that it is effective than the recent implementations for permutation based indexing [9–13].

The rest of the paper is organized as follows. In section 2, we review the literature related to similarity search and, in particular to permutation-based indexing. In section 3, we introduce our modeling for permutation-based indexing and build a formal justification for our data structure from here. Section 4 proposes the data structure and details the associated indexing and searching procedures. Our setup is evaluated in section 5 on a large dataset. Section 6 finally discusses our contributions.

2 Related Work

We use the principles of permutation-based indexing to efficiently perform approximate similarity search in large-scale datasets.

The idea of *permutation-based indexing* was first proposed in [8, 9]. It relies on the selection and ordering of reference objects by their distance to database objects. Order permutations are used as encoding to estimate the real ordering of distance values between the submitted query and the objects in the database (see section 3 for details). Amato and Savino [9, 13] introduced the metric inverted files (MIF), as a disk based data structure to store the permutations in an inverted file. The inverted list for a reference object stores the object-id and the ranking of this reference object relative to all objects in the database. Then, Spearman Footrule Distance (SFD) is used to estimate the similarity between the query and the database objects. To further reduce the storage, each object is encoded using the only nearest reference points. Mohamed and Marchand-Maillet [14] proposed a distributed implementation of the MIF, through three levels of parallelization. They were able to achieve high speed up compared to the sequential implementation.

In [15], authors propose the *prefix permutation index* (PP-Index). The PP-Index only stores the prefix of the permutations in the main memory and the data objects are saved in blocks on the hard disk. Objects which share the same prefix are saved close to each other. Hence, once a query is submitted all the objects in the block sharing the same prefix are used for a direct distance calculation with the query. The PP-Index requires less memory than the (MIF) and uses the permutations prefix in order to quickly retrieve the relevant objects, in a hash-like manner. The proposed technique initially gives a lower recall value. An improvement is made using multiple queries. When a query is received, the permutation list for the query is computed. Different combination of the query permutation list are then submitted to the tree as new queries.

In [12], authors proposed the *Neighborhood Approximation* (NAPP). The main idea of the NAPP is to try to get the objects located in the intersection region of reference objects shared between the query and database objects. This data structure is similar to the MIF except for data compression on the inverted lists for saving storage.

Mohamed and Marchand-Maillet [11] further proposed the MSA a memory-based, fast and an effective data structure for storing the permutations. The main idea is to concatenate the permutation lists in one string and use the suffixes of this string to retrieve the relevant objects in an effective way.

Our proposal here builds from the original permutations-based indexing and uses quantization as an approximation.

3 Indexing Model

We first detail our formal view of permutation-based indexing on which we build our contribution. Section 4 then proposes the corresponding algorithmic setup for this scheme.

Given a set of N objects o_i, $D = \{o_1, \ldots, o_N\}$, we identify each object o_i as a point x_i in some m-dimensional space R^m within which a distance function $d(.,.)$ applies. A range query (q, ε) from a point $q \in R^m$ (object q) aims at locating all points x_i (objects o_i in D) such that $d(q, x_i) \leq \varepsilon$. With no loss of generality, in our formal model, a K-nearest neighbour query is seen as a range query with an appropriate range value ε^* so as to return K results.

The definition of a set $R = \{r_1, \ldots, r_n\}$ of n reference points and their distance to all points is known to carry sufficient information to generate a set of points $\hat{x}_i$ approximating the original set (eg using FastMap [16]). The strategy to set the number and the position of the reference points is not discussed here and may be found in several studies [17–19]. From this point on, an object o_i refers to both the database object and the corresponding point $x_i \in R^m$, however best appropriate.

In [8], the use of a set R of reference objects (called pivots) is motivated by the intuitive (necessary and sufficient) assumption that for any query q, neighboring objects o_i will view the reference set R in the same way as q does. Conversely, if the distance ordering of the set R from two objects is similar, then these objects should be close one to another. We express this formally using the following definitions.

Definition 1. *Given a set $R = \{r_1, \ldots, r_n\}$ and an object o, we define the* ordered list *of R relative to o, $L(o, R)$, as the ordering of elements in R with respect to their increasing distance from o:*

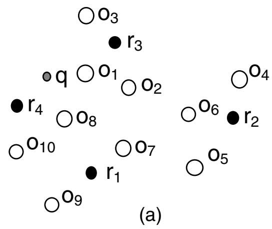
$$L(o, R) = \{r_{i_1}, \ldots, r_{i_n}\} \text{ such that } d(o, r_{i_j}) \leq d(o, r_{i_{j+1}}) \ \forall j = 1, \ldots, n-1$$

Then, for any $r \in R$, $L(o, R)_{|r}$ indicates the position of r in $L(o, R)$. In other words, $L(o, R)_{|r} = j$ such that $r_{i_j} = r$. Further, given $\tilde{n} > 0$, $\tilde{L}(o, R)$ is the pruned ordered list *of the $\tilde{n}$ first elements of $L(o, R)$.*

Figure 1(b) gives the ordered lists $L(o_i, R)$ for D and R illustrated in Figure 1(a).

In K-NN similarity queries, we are interested in ranking objects (to extract the K first elements) and not so much in the actual inter-object distance values. As discussed above, permutation-based indexing approximates distance calculations by assuming that the distances between objects will be approximated in terms of their ordering and then comparing the ordered lists of these objects. Here, we consider the Spearman Footrule Distance (d_{SFD}) between ordered lists. Formally,

$$d(q, o_i) \overset{\text{rank}}{\simeq} d_{\text{SFD}}(q, o_i) \qquad (1)$$



L(o1,R) =(r3,r4,r1,r2) L(o2,R) =(r3,r2,r4,r1)
L(o3,R) =(r3,r4,r1,r2) L(o4,R) =(r2,r3,r1,r4)
L(o5,R) =(r2,r1,r3,r4) L(o6,R) =(r2,r3,r1,r4)
L(o7,R) =(r1,r2,r4,r3) L(o8,R) =(r4,r1,r3,r2)
L(o9,R) =(r1,r4,r2,r3) L(o10,R) =(r4,r1,r3,r2)
L(q,R) =(r4,r3,r1,r2)

(b)

L̃(o1,R) =(r3,r4) L̃(o2,R) =(r3,r2)
L̃(o3,R) =(r3,r4) L̃(o4,R) =(r2,r3)
L̃(o5,R) =(r2,r1) L̃(o6,R) =(r2,r3)
L̃(o7,R) =(r1,r2) L̃(o8,R) =(r4,r1)
L̃(o9,R) =(r1,r4) L̃(o10,R) =(r4,r1)
L̃(q,R) =(r4,r3)

(c)

Fig. 1. Data setup (a) ∘ data objects o_i; • reference objects r_j; the gray circle is the query object q (b) Ordered lists for all the objects $L(o_i, R)$ using full permutations. (c) Pruned ordered lists for all the objects $\tilde{L}(o_i, R)$ with $\tilde{n} = 2$.

where,

$$d_{\text{SFD}}(q, o_i) = \sum_{j=1}^{n} |L(q, R)_{|r_j} - L(o_i, R)_{|r_j}|$$

As mentioned in [8], the extent of the approximation made is difficult to assess due to the non-continuous nature of ranking.

It is intuitively clear that objects may be encoded relatively to a reduced set of reference objects only. Several criteria may be introduced to prune the list of reference objects from the query. In practice, we use a locality criterion. This is supported in the neighbor search problem by the fact that NN-search is essentially a local process. However, reference objects will typically be chosen to cover the complete object space. For a given query q, nearest reference objects from q will suffice to discriminate objects as a response to the query.

Such a locality criterion may be translated on ranking $L(o_i, R)_{|r_j}$ by pruning lists $L(o_i, R)$ to a length $\tilde{n}$ into $\tilde{L}(o_i, R)$, including all reference objects r_j such that $L(o_i, R)_{|r_j} \leq \tilde{n}$. This corresponds to considering only the $\tilde{n}$ closest reference objects to encode o_i.

Thus, Eq.(1) further translates into

$$d(q, o_i) \overset{\text{rank}}{\simeq} \sum_{\substack{r_j \in \tilde{L}(q,R) \\ r_j \in \tilde{L}(o_i,R)}} |\tilde{L}(q, R)_{|r_j} - \tilde{L}(o_i, R)_{|r_j}| \tag{2}$$

Figure 1(c) gives the ordered lists $\tilde{L}(o_i, R)$ for $\tilde{n} = 2$ and D and R illustrated in Figure 1(a).

Eq.(2) is then the base for the construction of our indexing scheme. The main idea is to simplify further the distance approximation by quantizing the ranks within the ordered lists into $B \leq \tilde{n}$ intervals (buckets). We define

$$b_{ij} = \left\lceil \frac{B}{\tilde{n}} . \tilde{L}(o_i, R)_{|r_j} \right\rceil \quad \text{and} \quad b_{qj} = \left\lceil \frac{B}{\tilde{n}} . \tilde{L}(q, R)_{|r_j} \right\rceil \tag{3}$$

as regular quantization scheme. Eq.(2) now transforms into

$$d(q, o_i) \overset{\text{rank}}{\simeq} \sum_{\substack{r_j \in \tilde{L}(q,R) \\ r_j \in \tilde{L}(o_i,R)}} |b_{qj} - b_{ij}| \tag{4}$$

In the next section, we detail the corresponding data structure and the corresponding indexing and search procedures.

4 Practical Setup

The previous section proposes a formal model for efficient reference-based approximate similarity search. The model is built around the approximation of distance-based ranking, extending the permutation-based indexing strategy. We show here how we construct an efficient data structure to support our indexing strategy and explicit the search procedure, based on this data structure.

4.1 Indexing

Eq.(3) above defines a bucket index b_{ij} as the quantized rank of reference object r_j with respect to object o_i. Considering how these factors will be used, it is useful to center the construction of our data structure on the fact of efficiently answering the selection

$$r_j \in \tilde{L}(q, R) \quad \text{and} \quad r_j \in \tilde{L}(o_i, R)$$

used in Eq.(4).

At query time, $\tilde{L}(q, R)$ will be computed. We therefore need to characterise the set

$$\{o_i \ \text{ s.t } \ r_j \in \tilde{L}(o_i, R), \forall r_j \in \tilde{L}(q, R)\}$$

which is the typical use case of an inverted file. Since we quantize $\tilde{L}(o_i, R)_{|r_j}$ index into B buckets, we construct the inverted file at the time of quantization. More formally the indexing procedure is given in Algorithm 1, using the data structure presented in Figure 2a, which we call the metric permutation table (MPT).

Algorithm 1

```
IN: D of size $N$, $R$ of $n$ and $\tilde{n} \leq n$ and $B \leq \tilde{n}$
OUT: $MPT$
1. For $o_i \in D$
2.     Build $\tilde{L}(o_i, R)$
3.     For $r_j \in \tilde{L}(o_i, R)$
4.         $b_{ij} = \lceil \frac{B}{\tilde{n}} . \tilde{L}(o_i, R)_{|r_j} \rceil$ (Eq.(3))
5.         Store the ID $i$ of $o_i$ in the list $l$ of bucket number $b_{ij}$
```

Indexing Optimization. We propose two further optimisation steps that help reducing costs in memory and disk access.

Compression: Since any given object o_i may appear in several buckets. The direct storage of objects IDs within the buckets is inefficient. Buckets store lists of IDs (large integers) and may thus be efficiently compressed using delta encoding [20]. This strategy consists in replacing a series of large integers (given values) by a series of smaller integers (their successive difference). For example, the sequence $\{562895, 562896, 562900,$

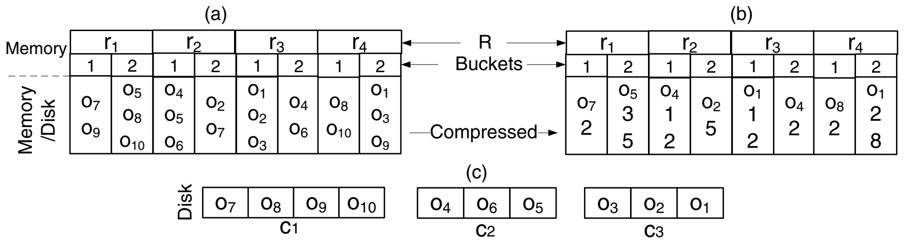


Fig. 2. The MPT data structure

$562910\}$ requires 16 bytes (4 bytes per value) and may be replaced by $\{562895, 1, 4, 10\}$ that may be stored using 4 bytes for the first value and 2 bytes for each subsequent value (hence 10 bytes in total). This simple strategy is applied individually to buckets, leading to significant memory savings. Our compressed data structure is presented in Figure 2(b). This compressed lists can be saved on the hard-disk if they can not be fitted in the main memory.

Clustering: We cluster the original objects into a given number of clusters using K-Means. Our motivation is to optimize disk access overhead. This overhead depends on the size of the database and the hard disk. We cluster the database into C clusters $(C_1, \dots, C_c)$ with the idea that objects sharing the same clusters will be accessed together. At query time, a reduced number of clusters should be accessed, containing the necessary points for further filtering (direct distance calculation, DDC). Figure 2(c) shows the clustered data for points in figure 1(a).

4.2 Searching

Equation (4) is the base for our search procedure. Essentially, it counts the discrepancy in quantized ranking from common reference objects between each object and the query. In practice, we further reduce the access to buckets. Instead of a *soft* (sum) counting of the difference in bucket index, we count the co-occurrences of each object with the query in the same bucket or neighbouring buckets. That is, each object o_i scores

$$s_i = \left|\left\{r_j \in \tilde{L}(q, R) \text{ such that } (r_j \in \tilde{L}(o_i, R) \text{ and } |b_{ij} - b_{qj}| \leq 1)\right\}\right| \qquad (5)$$

This computation is efficiently supported by the above data structure. Objects o_i are then sorted according to their decreasing s_i scores. This sorted candidate list provides an approximate ranking of the database objects relative to the submitted query. This approximate ranking can be improved by direct distance calculation (DDC). Similar to the proposals in [14] (DDC factor) and [13] (amplification factor), for a K-NN query we apply DDC on the $K_c = \Delta.K$ first objects in our sorted candidate list and call $\Delta > 1$ the *DDC factor*. The use of this parameter will be explored in our performance evaluation (section 5).

The search procedure uses the above data structure as described in Algorithm 2.

Algorithm 2

```
IN: $q$, $R$ of size $n$, $\Delta$
OUT: Sorted Objects list: out
1.  Create a list of counters $s[0 \dots N]$ and set them to $0$
2.  Build $\tilde{L}(q, R)$
3.  For $r_j \in \tilde{L}(q, R)$
4.      $b_{qj} = \lceil \frac{B}{\tilde{n}}.\tilde{L}(q, R)_{|r_j} \rceil$ (Eq.(3))
5.      For $k = b_{qj} - 1$ to $b_{ij} + 1$
6.          start_Obj = $l[k]$[0]
7.          For $i = 1$ to $l[k].size()$
8.              s[start_Obj + $l[k]$[i]]++
9.              start_Obj=start_Obj+$l[k]$[i]
10. sort(s)
11. $K_c = K_{NN} \times \Delta$
12. $A \leftarrow s(0, K_c)$
13. out=calc_distance$(A, K_c, q)$
14. $sort(out)$
```

First the pruned ordered list $\tilde{L}(q, R)$ is constructed (Line 2). Equation (3) is used to characterise the active buckets (Line 4-5), which are decompressed (Line 6-9) to get the stored object IDs. For each such object, a count of co-occurrence is computed using Eq.(5). Line 10 sorts the candidate list in decreasing order of their score(counters s). Final DDC filtering is performed in lines 11-13.

The average complexity for accessing the reference lists and the buckets is $O(1)$. While it is $O(z)$ for the lists of each bucket, where z is the average size of a list. Hence, the total average searching complexity is $O(\tilde{n}z)$.

5 Results

Our data structure was implemented in C++. The experiments were run on a 2.70GHz CPU with 128Gb of memory and linked with 512GB storage capacity. We measure recall (RE), position error (PE) [5], and the response time. In 5.1, we measure the general performance of our data structure on a dataset of about 9-millions color features (84-dimensional) extracted from the 12-million ImageNet corpus [21][1]. In section 5.2, we compare the performance of our proposed structure to that proposed in [9–13], based on the datasets reported in these publications.

5.1 General Performance

Recall and Position Error. The selection of the number of buckets B and the parameter for direct distance calculation Δ affects the performance of the structure (response time and memory usage) and the precision of the search. These parameters mainly depend on the dataset and the distribution of the reference objects. We observe empirically that $B = 5$ and $\Delta = 40$ are good trade off values between performance and efficiency.

[1] The feature dataset comprises 9'175'426 elements of 84 dimensions, a set of 100 NN queries with ground truth and is available by contacting the authors.

Two sets of reference objects of size $n = 1'000$ and $n = 2'000$ were selected using the distributed references selection technique proposed in [11]. This greedy selection technique ensures that the minimum distance between any two reference objects is larger than a given threshold value. This leads to an appropriate coverage of the dataset by the selected reference objects.

The experiments were thus performed fixing the number of buckets $B = 5$ and $\Delta = 40$ for pruned ordered lists of sizes $\tilde{n} = \{10, 20, 50\}$ for $n = 1000$ and $n = 2000$ reference objects and for the 1-NN, 10-NN and 100-NN search scenarios. Figures 3(a) and 3(b) show the average RE and PE using 100 queries, selected randomly from the dataset.

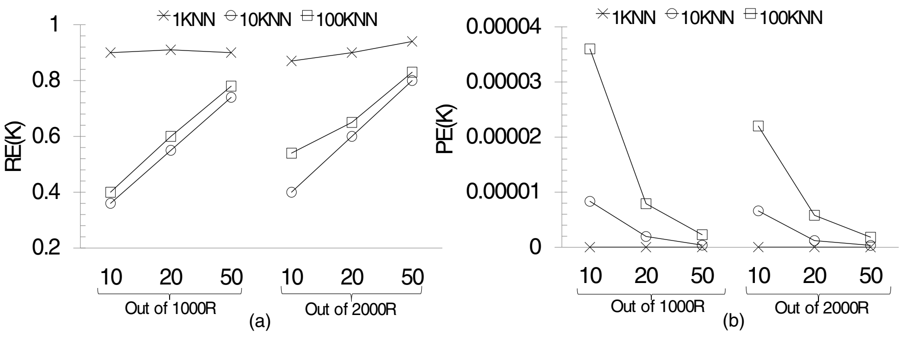


Fig. 3. Using $\Delta = 40$ $B = 5$ a) RE using $\tilde{n} = \{10, 20, 50\}$ nearest reference objects out of $n = 1000$ and $n = 2000$ for 1-NN, 10-NN and 100-NN queries. b) PE using $\tilde{n} = \{10, 20, 50\}$ nearest references out of $n = 1000$ and $n = 2000$ for 1-NN, 10-NN and 100-NN queries.

We note that RE increases while PE decreases when the size of the pruned lists $\tilde{n}$ increases. This shows that $\tilde{n}$ directly impacts the quality of approximation (as suggested in Eq.(2)).

When comparing the RE and PE for the same number of nearest reference objects $\tilde{n}$ out of an increasing total number of reference objects n, we see that the RE increases and PE decreases. This is reasonable, since a higher n indicates a more dense set of reference objects and therefore a lower number of objects located in the same bucket for each reference object, thus decreasing the number of false positive. For example, for 10-NN, if $\tilde{n} = 50$ out of $n = 1000$ and $n = 2000$ reference objects, RE= 0.74 and RE= 0.8, respectively. The number of direct distance calculation K_c for 1-NN, 10-NN and 100-NN queries is 40, 400 and 4000, respectively. This is about 0.0004%, 0.004% and 0.04% of the number of database objects. Hence, using our technique, we significantly decrease the number of direct distance calculations between the query and the database objects even with a small set of reference objects.

Indexing Time, Searching Time and Memory Usage. Table 1 shows the average indexing time, searching time, uncompressed and compressed memory based on 100 queries.

Table 1. Average indexing time, searching time, uncompressed and compressed memory for the MPT based on 100 queries

n	$\tilde{n}$	Indexing Time	Searching Time	Uncompressed memory	Compressed memory
1000R	10	52min	0.37s	367MB	252 MB
	20		0.48s	720MB	427 MB
	50		0.75s	1773MB	955 MB
2000R	10	102min	0.3s	370MB	267 MB
	20		0.4s	719MB	442 MB
	50		0.62s	1773MB	973 MB

When the number of reference objects n increases, the indexing time consistently increases. On the other hand, the value of $\tilde{n}$ does not affects the indexing time, since the costly operations essentially depend on n.

The size of the lists assigned to each bucket decreases as n increases, which leads to a decreasing running time (at the cost of a more rough approximation). The searching time however increases along with the number of nearest reference objects $\tilde{n}$.

The last two columns in Table 1 show the uncompressed and compressed memory usage. Using the delta technique, we are able to achieve around 50% decrease in memory usage. This percentage decreases when the number of reference objects increases. The main reason is that the distribution of the objects changes. When the number of reference objects increases consecutive objects are not anymore close to each other (they are located in different lists), so that the compression technique is less effective.

In summary, increasing the number n of reference objects increases the memory usage but decreases the running time and improves the RE and PE.

5.2 Comparative Experiments

It is difficult to run exhaustive comparative tests with previous work since each proposal generally tests its implementation on a specific dataset, which is not always made available. And even if so, often not all parameters of the experiments are reported.

We compare MPT to earlier proposals (NAPP [12], Perms [22], PP-Index [15], MIF [9]). At the time of writing, we did not gain access to the CoPhIR dataset, so our evaluation focuses on the available datasets.

In [12], the authors made a comparison based on the recall between their implementation and previous implementations using the color histogram dataset available on the SISAP website [23]. This dataset is a set of 112,544 color histogram (112-dimensional vector) from an image database. We use the same dataset and compare our results with that summarized in [12]. Figure 4(a) shows the recall RE for the 30-KNN scenario using different numbers n of reference objects. In [12], the number of nearest reference objects is $\tilde{n} = 7$ for NAPP, PP-Index, Perms and MIF with at maximum 3000 direct distance calculations for NAPP. For the PP-Index, query expansion is not considered.

We compare to MPT using the same values for n and $\tilde{n}$ with $B = 7$ and $\Delta = \{40, 100\}$, so that K_c is between 1200 and 3000 for the 30-NN scenario. MPT using

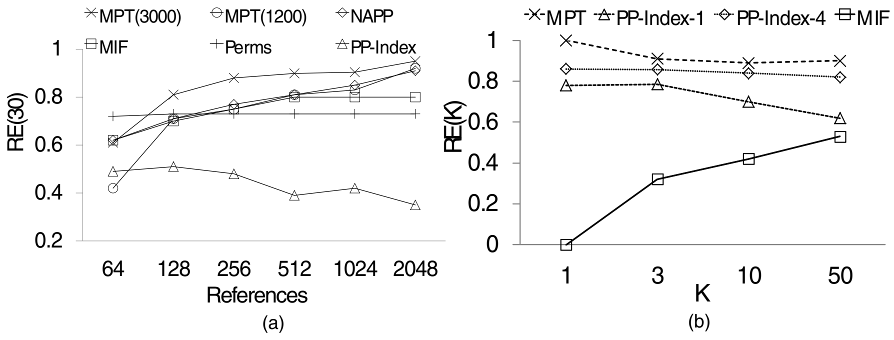


Fig. 4. a) Recall comparison between MPT, MIF, NAPP, Perms and PP-Index for top 30-NN. The recall values for these structures are taken from [12] (our experiments are based on averaging 200 queries). b) Recall comparison between MPT, MIF and PP-Index for 1, 3, 10 and 50 top K-NN. The Recall values for PP-Index and MIF are taken from [15] (our experiments are based on averaging 100 queries).

$K_c = 1200$ ($\Delta = 40$) was able to achieve the same performance as NAPP [12] using 3000 DDC. Our MPT is then 2.5 faster, thanks to the bucketing technique we propose. NAPP uses the full range of nearest reference objects and compares it to the full range of nearest reference objects of the query, leading to a high number of false positive objects. Since in MPT, we care only about the shared buckets, we achieve a higher recall while saving calculations. The best searching time for NAPP is 0.0054s, opposed to 0.0031s for MPT using 2048 reference objects.

In [15] author proposed a comparison between PP-Index and the MIF [9] using a smaller dataset available from the UCI knowledge Discovery in Database Archive [2]. The dataset consists of 50,000 HSV color histograms (32-dimensional vector) from an image database. We use the same dataset and compare our results to the results published in [15] (Figure 4b). MIF indexing used 50 out of a set of 100 reference objects. PP-index indexing used 50 references with a prefix length of 6 (experiments were made using queries with an expansion factor = 4). Our MPT indexing used 50 out of 100 nearest reference objects selected randomly from the dataset. With no query expansion, our MPT data structure outperforms the PP-Index (PP-Index-1 and PP-Index-4) and the MIF for different numbers of K-NN. Our proposal is also more effective in terms of indexing and searching time than PP-Index and MIF. MPT requires 2 seconds for indexing and 0.009s for searching, while PP-Index requires 4.9s for indexing and 0.01s and 0.02s for searching using 1 and 4 queries respectively. For MIF, it needs around 1 min for indexing.

We finally compare our proposal to the MSA proposed in [11] using the same dataset of 4’594’734 visual shape features (21-dimensional), extracted from the 12-million ImageNet corpus [21]. RE and PE were computed using the nearest $\tilde{n} = \{50, 250, 500, 750, 1000\}$ reference objects out of totals of $n = \{100, 500, 1000, 1500, 2000\}$,

[2] `http://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.data.html`

respectively. RE$= 0.7$ for 100-NN using 1000 nearest references and no DDC. Our proposed MPT achieves a higher recall value using less nearest reference objects. For only $\tilde{n} = 50$ out of $n = 1000$ reference objects, the recall value is 0.8 with $\varDelta = 4$ ($K_c = 400$ objects have been compared to the query). Further, our data structure requires only 0.13s for the search while MSA needs 1.9s. The main saving is made in not using the SFD but an approximate version of its induced ranking Eq.(4) and (5). Memory saving is also significant with 490MB needed for MPT while MSA requires 17GB (due to the large number of reference objects needed to avoid DDC).

6 Conclusion

We present a new strategy for approximate search in metric spaces, based on permutation-based indexing. We develop a formal model where we exhibit the approximation made by quantizing the ranking of values of distance between objects and reference objects. A candidate selection strategy for answering a K-NN query is detailed and implemented with the support of our data structure, called the Metric Permutation Table (MPT).

The evaluation is performed using standard and large-scale datasets. We demonstrate empirically the validity of our claims by comparing the performance of our setup with that of state-of-the-art methods.

We now work at improving the estimation of the number of buckets with respect n and $\tilde{n}$. We also wish to relate the performance of our strategy with various techniques for selecting the set of reference objects, which is critical to obtain an accurate encoding of the neighboring information.

Acknowledgment. This work is jointly supported by the Swiss National Science Foundation (SNSF) via the Swiss National Center of Competence in Research (NCCR) on Interactive Multimodal Information Management (IM2) and the European COST Action on Multilingual and Multifaceted Interactive Information Access (MUMIA) via the Swiss State Secretariat for Education and Research (SER).

References

1. Jagadish, H.V., Mendelzon, A.O., Milo, T.: Similarity-based queries. In: Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 1995, pp. 36–45. ACM, New York (1995)
2. Samet, H.: Foundations of multidimensional and metric data structures. The Morgan Kaufmann series in computer graphics and geometric modeling. Elsevier/Morgan Kaufmann (2006)
3. Chávez, E., Navarro, G., Baeza-Yates, R.A., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. 33(3), 273–321 (2001)
4. Lee, D.T., Wong, C.K.: Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. Acta Inf. 9, 23–29 (1977)
5. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. Advances in Database Systems, vol. 32. Springer (2006)

6. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, STOC 1998, pp. 604–613. ACM, New York (1998)
7. Patella, M., Ciaccia, P.: Approximate similarity search: A multi-faceted problem. J. of Discrete Algorithms 7(1), 36–48 (2009)
8. Gonzalez, E., Figueroa, K., Navarro, G.: Effective proximity retrieval by ordering permutations. IEEE Transactions on Pattern Analysis and Machine Intelligence 30(9), 1647–1658 (2008)
9. Amato, G., Savino, P.: Approximate similarity search in metric spaces using inverted files. In: Proceedings of the 3rd International Conference on Scalable Information Systems, InfoScale 2008, pp. 28:1–28:10. ICST, Brussels (2008)
10. Esuli, A.: Mipai: Using the pp-index to build an efficient and scalable similarity search system. In: Proceedings of the 2009 Second International Workshop on Similarity Search and Applications, SISAP 2009, pp. 146–148. IEEE Computer Society, Washington, DC (2009)
11. Mohamed, H., Marchand-Maillet, S.: Metric Suffix Array For Large-Scale Similarity Search. In: ACM WSDM 2013 Workshop on Large Scale and Distributed Systems for Information Retrieval, Rome, IT (February 2013)
12. Tellez, E.S., Chávez, E., Navarro, G.: Succinct nearest neighbor search. Inf. Syst. 38(7), 1019–1030 (2013)
13. Amato, G., Gennaro, C., Savino, P.: Mi-file: using inverted files for scalable approximate similarity search. In: Multimedia Tools and Applications (2012)
14. Mohamed, H., Marchand-Maillet, S.: Parallel Approaches to Permutation-Based Indexing using Inverted Files. In: 5th International Conference on Similarity Search and Applications (SISAP), Toronto, CA (August 2012)
15. Esuli, A.: Pp-index: Using permutation prefixes for efficient and scalable approximate similarity search. In: Proceedings of LSDSIR 2009, pp. 1–48 (2009)
16. Faloutsos, C., Lin, K.I.: Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. SIGMOD Rec. 24(2), 163–174 (1995)
17. Bustos, B., Pedreira, O., Brisaboa, N.: A dynamic pivot selection technique for similarity search. In: Proceedings of the First International Workshop on Similarity Search and Applications, SISAP 2008, pp. 105–112. IEEE Computer Society, Washington, DC (2008)
18. Ares, L.G., Brisaboa, N.R., Esteller, M.F., Pedreira, O., Places, A.S.: Optimal pivots to minimize the index size for metric access methods. In: Proceedings of the 2009 Second International Workshop on Similarity Search and Applications, SISAP 2009, pp. 74–80. IEEE Computer Society, Washington, DC (2009), doi:10.1109/SISAP.2009.21
19. Bustos, B., Navarro, G., Chávez, E.: Pivot selection techniques for proximity searching in metric spaces. Pattern Recognition Letters 24(14), 2357–2366 (2003)
20. Smith, S.W.: The scientist and engineer's guide to digital signal processing. California Technical Publishing, San Diego (1997)
21. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR 2009 (2009)
22. Téllez, E.S., Chávez, E., Camarena-Ibarrola, A.: A brief index for proximity searching. In: Bayro-Corrochano, E., Eklundh, J.-O. (eds.) CIARP 2009. LNCS, vol. 5856, pp. 529–536. Springer, Heidelberg (2009)
23. Figueroa, K., Navarro, G., Chávez, E.: Metric spaces library (2007), `http://www.sisap.org/Metric_Space_Library.html`

Extreme Pivots for Faster Metric Indexes

Guillermo Ruiz[1], Francisco Santoyo[1], Edgar Chávez[2], Karina Figueroa[1], and Eric Sadit Tellez[1]

[1] Universidad Michoacana de San Nicolás de Hidalgo, México
{gruiz,psantoyo}@dep.fie.umich.mx,
{karina,sadit}@fismat.umich.mx
[2] Universidad Nacional Autónoma de México
elchavez@matem.unam.mx

Abstract. Pivot tables are popular for *exact* metric indexing. It is well known that a large pivot table produces faster indexes. The rule of thumb is to use as many pivots as the available memory allows for a given application. To further speedup searches, redundant pivots can be eliminated or the scope of the pivots (the number of database objects covered by a pivot) can be reduced.

In this paper, we apply a different technique to speedup searches. We assign objects to pivots while, at the same time, enforcing proper coverage of the database objects. This increases the discarding power of pivots and in turn leads to faster searches. The central idea is to select a set of *essential* pivots (without redundancy) covering the entire database. We call our technique *extreme pivoting* (EP).

A nice additional property of EP is that it balances performance and memory usage. For example; using the same amount of memory, EP is faster than the List of Clusters and the Spatial Approximation Tree. Moreover, EP is faster than LAESA even when it uses less memory.

The EP technique was formally modeled allowing performance prediction without an actual implementation. Performance and memory usage depend on two parameters of EP, which are characterized with a wide range of experiments. Also, we provide automatic selection of one parameter fixing the other. The formal model was empirically tested with real world and synthetic datasets finding high consistency between the predicted and the actual performance.

1 Introduction

We are interested in the *proximity search problem*, where a metric space $(\mathcal{X}, d)$ is given. For a finite subset S of $\mathcal{X}$, and an element $q \in \mathcal{X}$, the goal is to find elements in S near q. One proximity query of interest is *range query*, given $r \geq 0$ we seek for $(q, r) = \{s \in S \mid d(s, q) \leq r\}$. It is also useful the notion of k *nearest neighbor query* (KNN), given an integer $k > 0$ find k nearest elements of S to q. KNN queries are equivalent to range queries if the search radius can be bounded. The ball centered at q with radius r is named the *query ball* (an analogous definition exists for KNN queries). The set $\mathcal{X}$ is often referred as the *universe* of objects, S is called the *database*, and q is called the *query object*.

The proximity search problem can be trivially solved with a sequential scan. This solution makes sense when the set S is small and the distance function d is cheap.

For large datasets and/or expensive distance functions, metric indexing takes advantage of transitive properties of the metric to avoid a sequential scan. This problem has important practical applications in machine learning, pattern recognition, statistics, bioinformatics and textual and multimedia information retrieval, to name a few. There are several books and surveys (Chavez et al. [1], Samet [2], Zezula et al. [3]) describing with detail the different discarding rules, we will assume some familiarity of the reader with the subject.

There are two popular approaches to metric indexing, namely *pivot based indexes* and *compact partitions*. The later divides the data in spatially coherent regions, and at query time, regions without intersection with the query ball are discarded. The pivoting scheme consists in an implicit contractive mapping, where a distance $\delta(x, y)$ with the property $\delta(q, x) \leq d(q, x) \forall x \in \mathcal{X}$ is built using a set of objects (named pivots).

1.1 Related Work

For a long time, the main measure for performance comparison has been the number of distance computations to answer a query. The rationale behind this measure is because distances are the most expensive operation in a query. However, when measuring the actual time for answering a query it can be a surprise that a good index under this measure could be very slow when measuring the time. This may be led by a combination of a cheap distance and an expensive indexing structure.

AESA [4] stores $O(n^2)$ distances, and hence the construction cost is of the same order. At query time it performs a constant number of distance computations for a fixed database. However, they compute a linear number of arithmetic and logical operations at query time. A linear size restriction of the same idea is presented in LAESA [5]. The table of pivots comes from this initial idea.

Chavez et al. [1], proved that any pivot based metric index requires at least a logarithmic (on the database size) number of pivots (randomly selected from the database); however, the base of the logarithm depends on the intrinsic dimension of the database, needing larger indexes as the intrinsic dimension increases. This optimal number of pivots could not fit in main memory; hence, the rule of thumb is to use as much pivots as they fit.

Many of the pivots used are redundant if they are selected randomly. Hence, pivot selection became popular. Bustos et al. [6] present different strategies to this matter, showing that the proper selection of pivots is driven by datasets and queries. Remarkably, randomly chosen pivots can be good enough in any case. Finally, Celik [7,8] presents the priority vantage point method (Kvp). The Kvp is a structure where only the K most promising pivots are stored per object. It experimentally shows that the most promising pivots are those either near or far from the object.

Another approach consists in making a compact partition of the data. Here, two indexes are representative. The Spatial Approximation Tree (SAT) [9] is a metric tree where each node c is selected from S; c has $N(c)$ nodes, defined as $u \in N(c)$ if $d(u, c) < d(u, v), v \in N(c)$, for all $u \in S \setminus \{c\}$. This procedure is recursively repeated for each $u \in N(c)$ with S as the set items in $S \setminus \{N(c) \cup \{c\}\}$ having u as its closer object in $N(c)$. Since $N(c)$ depends on the construction order, there are many $N(c)$. The author proposes to review S in the natural order imposed by the distance value of each object to c. This index has good performance and no construction parameters (other than the order of the objects in $N(c)$); this simplicity of use makes the SAT a fair choice when there is not much knowledge of the database. Chavez and Navarro [10] present a robust and memory efficient alternative, dubbed as the List of Clusters (LC). The LC needs linear memory and up to quadratic construction time. At equality of memory, it remains unbeatable on datasets with very high intrinsic dimensionality.

2 Extreme Pivots

An Extreme Pivoting (EP) Index consists on a set of *pivot groups* (PG). Each group is a tuple $(\mathbb{P}, \alpha)$, where $\alpha > 0$ and $\mathbb{P}$ is a subset of S. For every $p \in \mathbb{P}$ there exists a set $A(p) \subset S$ such that for every $x \in A(p)$ we have that $|\mu_p - d(p,x)| \geq \alpha$. Here, μ_p is for the expected value of $d(u,p)$ for all $u \in S$. We must ensure that $\bigcup_{p \in \mathbb{P}} A(p) = S$ and $A(p_i) \cap A(p_j) = \emptyset$ for $i \neq j$; hence, both $\mathbb{P}$ and α define a partition of S. The proper selection of these parameters is deferred and discussed in upcoming sections. Let $p \in \mathbb{P}$, define $\overleftarrow{A}(p) = \{u \in A(p) \mid d(u,p) \leq \mu_p - \alpha\}$, and $\overrightarrow{A}(p) = \{u \in A(p) \mid d(u,p) \geq \mu_p + \alpha\}$ as depicted by Figure 1. For u in S, let $\mathsf{piv}(u)$ be the pivot such that $u \in A(p)$.

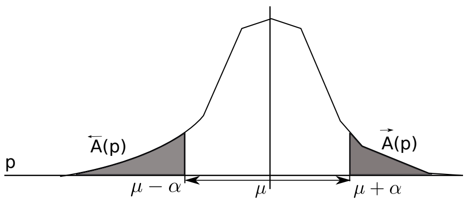


Fig. 1. The regions being controlled by α, from the perspective of pivot p

The elements of $A(p)$ will be called associates of p. Our construction depends critically on α, and in turn, the optimum value of α depends on the distribution of objects in the database. We will estimate α using some probabilistic assumptions, for now, please notice that when $\alpha = 0$, then just a pivot is required and our approach degrades to a pivot in a pivot table. If α is arbitrarily large, then $A(p)$ will be arbitrarily small. The parameter α governs more than just the size of associates of a pivot, as we will see.

Given an element u and a pivot p, let $a = Pr(u \in A(p))$, a is the probability of covering u with the pivot p. If $a \leq 1$ is a fixed value for a fixed database, then the number of pivots per group is $m = \frac{\ln{(an+1)}}{\ln{(1/(1-a))}}$.

In this expression, m is a function of a (or more precisely, it depends on α). The proper value of a is tightly coupled with the searching performance (based on the discarding probability induced by a pivot group), which is estimated in the following paragraphs.

Given a query (q,r), the probability that u is not discarded by ℓ pivots is

$$Pr(|d(q,p_1) - d(u,p_1)| \leq r, \ldots, |d(q,p_\ell) - d(u,p_\ell)| \leq r).$$

In order to simplify the analysis, we suppose all objects to be described by independent identical distributed random variables (i.i.d.r.v). Therefore, the previous expression can be rewritten as follows

$$Pr(|d(q,p) - d(u,p)| \leq r)^{\ell}.$$

To bound the above probability we will use the distribution of distances from the pivot to all the database elements.

In addition to i.i.d.r.v. assumption we assume that probability functions are symmetric around the mean. Let X be the random variable such that $X(u) \sim d(\mathsf{piv}(u), u)$, and let Y be $Y(u) \sim d(u,v)$ for $u, v \in \mathcal{X}$. Thus, the probability function of X is defined in the extremes of Y, i.e., $X(u)$ corresponds to the shaded area in Figure 1, and $Y(u)$ matches the whole region.

Since we assume symmetry, then for every $u \in \overleftarrow{A}(\mathsf{piv}(u))$ there exists an element $v \in \overrightarrow{A}(\mathsf{piv}(v))$ such that $X(u) + X(v) = D$ with D a constant. We assume as well that for every element u there exists an element v such that $Y(u) + Y(v) = D$. Using these assumptions, $E[X] = E[Y] = D/2$, hence $E[X - Y] = 0$. Let σ_X^2 and σ_Y^2 be the variance of X and Y respectively.

Now, $Pr(|d(p,u) - d(p,q)| > r) = Pr(|X - Y| > r)$ and, using the Chebychev inequality[1] we have $Pr(|X - Y| > r) < (\sigma_X^2 + \sigma_Y^2)/r^2$.

The probability that a given u would not be discarded by its covering pivot $\mathsf{piv}(u)$, is $1 - Pr(|X - Y| > r) \geq 1 - (\sigma_X^2 + \sigma_Y^2)/r^2$.

The number of distances the algorithm would compute is

$$\text{cost} = m\ell + n\left(1 - Pr\left(|X - Y| > r\right)\right)^{\ell} \tag{1}$$

$$\geq m\ell + n\left(1 - \frac{\sigma_X^2 + \sigma_Y^2}{r^2}\right)^{\ell} \tag{2}$$

for a database of size n, and using m pivots for each of ℓ pivot groups. We obtained a lower bound for the average number of distances computed using this technique. We could minimize this last equation to get the optimum values for m and ℓ.

Fixing m, we obtain the optimal number of groups ℓ as

$$\ell^{\star} = \frac{\ln m/n - \ln\ln(1/s)}{\ln(s)}, \tag{3}$$

where $s = 1 - \frac{\sigma_X^2 + \sigma_Y^2}{r^2}$.

And using this optimal ℓ we can compute the minimum cost

$$\text{cost}^{\star} \geq \frac{m\left(\ln\frac{n}{m} + \ln\ln(1/s) + 1\right)}{\ln(1/s)} \tag{4}$$

$$= m\log_{1/s}\frac{n}{m} + o(m\ln(1/s)). \tag{5}$$

[1] For a random variable Z with mean μ_Z and variance σ_Z^2, $Pr(|Z - \mu_Z| > \epsilon) < \sigma_Z^2/\epsilon^2$.

This expression is similar to the cost obtained by Chavez et al. [1] for randomly selected pivots; i.e., $cost \geq \ln n + \ln \ln (1/t) + 1/\ln (1/t)$, with $t = 1 - 2\sigma_Y^2/r^2$. This expression gets an optimal number of pivots,

$$\begin{aligned} k^\star &= \frac{\ln n + \ln \ln(1/t)}{\ln(1/t)} && (6) \\ &= \log_{1/t} n + o(\ln(1/t)). && (7) \end{aligned}$$

Using random pivots (as analyzed by Chavez et al), the query cost depends on the database and the query radius, that is why the only way to improve the search speed is to increment ℓ. Our analysis also depends of σ_X, a parameter that we set at construction time, adjusting α. Please remember that $\sigma_X^2 = E[(d(u,p) - \mu_p)^2]$, and by construction $|d(u,p) - \mu_p| \geq \alpha$, thus $\sigma_X^2 \geq \alpha^2$. So, we have the chance to make adjustments to get better results on average, once we know the database. Also, in our cost equation, we can set the ℓ and m parameters (m depends on α) at construction time, ℓ can control the memory needed by the index. A greater m will reduce the probability of not discarding an element, even on fixed memory setups.

3 EP Table

A simple implementation of EP is a table. A set of ℓ pivot groups will be called an *EP Table*. Each group is computed as follows given the number of pivots m, and the number of instances (groups) ℓ. The construction consists on creating ℓ groups using Algorithm 1, which was sketched and analyzed in the previous section.

Algorithm 1. Randomized construction of the EP Table

```
Input: The input database $S = \{u_1, \ldots, u_n\}$, and the number of pivots $m$
Output: The set of pivots $P$, and the array $g$ of $n$ tuples $(\mathsf{piv}(u), d(u, \mathsf{piv}(u)))$ $\forall u \in S$.
1: Select a random pivot $p_1$, $P \leftarrow P \cup \{p_1\}$
2: Initialize $g[1,n] = (p_1, d(u_1,p_1)), (p_1, d(u_2,p_1)), \ldots, (p_1, d(u_n,p_1))$
3: for $i = 2$ to $m$ do
4:     Select $p_i$ randomly from $S$, $P \leftarrow P \cup \{p_i\}$
5:     for $j = 1$ to $n$ do
6:         $g[j] = (p_i, d(u_j,p_i))$ if $|d(u_j,p_i) - \mu| > |d(u_j, \mathsf{piv}(u_j)) - \mu|$.
7:     end for
8: end for
```

3.1 Optimizing α

In the previous construction, all parameters are assumed fixed. We can optimize the parameters using the model and the analysis described previously. The optimal α is achieved maximizing the probability of discarding an object u, which is approximated by $1 - (\sigma_X^2 + \sigma_Y^2)/r^2$. Using this expression, we observe that $\sigma_X = \sqrt{r^2 - \sigma_Y^2} \geq \alpha$.

This σ_X value implies that for high intrinsic dimensional datasets, α will be large, and it will not be useful (because it will produce a very large m). In this case, a suboptimal α value can be used and the whole performance could be improved using several pivot groups, i.e., increasing ℓ.

A better option consists in fixing ℓ controlling how much memory is used by the metric index. Once fixed ℓ, we can approximate the optimal m numerically. For this purpose we use Expression 2 as detailed in Algorithm 2. Here, the idea is to be incrementing m by one, and stop the algorithm whenever the derivative of Expression 2 becomes zero or positive. This procedure will create a single group, so it must be called ℓ times.

Algorithm 2. Numerically optimized construction of the EP-Table

```
Input: The input database $S = \{u_1, u_2, \cdots, u_n\}$, and the number of groups $\ell$.
Output: The set of pivots $P$, and the array $g$ of $n$ tuples $(\mathsf{piv}(u), d(u, \mathsf{piv}(u)))$ $\forall u \in S$.
 1: Estimate $\sigma_Y^2$ and $r^2$.
 2: Define $prev \leftarrow n$.
 3: Select a random pivot $p_1$, $P \leftarrow P \cup \{p_1\}$
 4: Initialize $g[1, n] = (p_1, d(u_1, p_1)), (p_1, d(u_2, p_1)), \cdots, (p_1, d(u_n, p_1))$
 5: Define $m \leftarrow 1$
 6: Compute $cost_1 = m\ell + n(1 - (\sigma_X^2 + \sigma_Y^2)/r^2)^\ell$. {At any step, $\sigma_X^2$ is computed with the
    current tuples in $g$}.
 7: while True do
 8:   Select $p_i$ randomly from $S$, $P \leftarrow P \cup \{p_i\}$
 9:   for $j = 1$ to $n$ do
10:     $g[j] = (p_i, d(u_j, p_i))$ if $|d(u_j, p_i) - \mu| > |d(u_j, \mathsf{piv}(u_j)) - \mu|$.
11:   end for
12:   $m \leftarrow m + 1$.
13:   Update $\sigma_X^2$ with the current tuples in $g$.
14:   $cost_i = m\ell + n\left(1 - (\sigma_X^2 + \sigma_Y^2)/r^2\right)^\ell$.
15:   if $cost_i \geq cost_{i-1}$ then
16:     stop loop
17:   end if
18: end while
```

It is important to notice that this procedure depends heavily on the estimated values σ_Y^2 and r^2. Also, for real world databases the i.i.d.r.v. assumption can be far from true. For this reason, we add a damping constant $\beta \leq 1$ for the discarding probability in lines 6 and 14, resulting on $cost_i = m\ell + n\left(1 - \beta(\sigma_X^2 + \sigma_Y^2)/r^2\right)^\ell$. The precise value of β is dependent on how much both the database and the query set disparate from the i.i.d.r.v. assumption.

4 Experimental Performance

In this section we present the performance as a function of the dimension, and for different standard databases used by the community. As usual, vector spaces are indexed without using the coordinates. Results are reported in both time and the fraction of the database revised for a given query.

— **Nasa** This database is a collection of 40150 vectors of dimension 40 obtained from the SISAP project (`http://www.sisap.org`). It uses L_2 as distance function.
— **Colors** The second benchmark is a set of 112682 color histograms (112-dimensional vectors) from SISAP, under the L_2 distance.
— **CoPhIR-1M** We use a subset of the CoPhIR database, of one million objects selected from the CoPhIR project [11]. Each object is a 208-dimensional vector and we use the L_1 distance.
— **RVEC** We generate random vectors in the unitary cube, in five dimensions 4, 8, 12, 16 and 20.

Each plot represents 256 nearest neighbor queries. The query object was not indexed. We used four searching algorithms as baseline for comparison.

1. The *Sequential* scan to bound the searching time when the dimension is large.
2. The *LAESA*, the standard pivot table.
3. The *List of Clusters* (LC) [10], which until now it holds the best performance for equality of memory (on the right setup).
4. The fourth baseline is a version of Navarro's *SAT* [9] built using a random order. The SAT is probably the best index having no parameters.

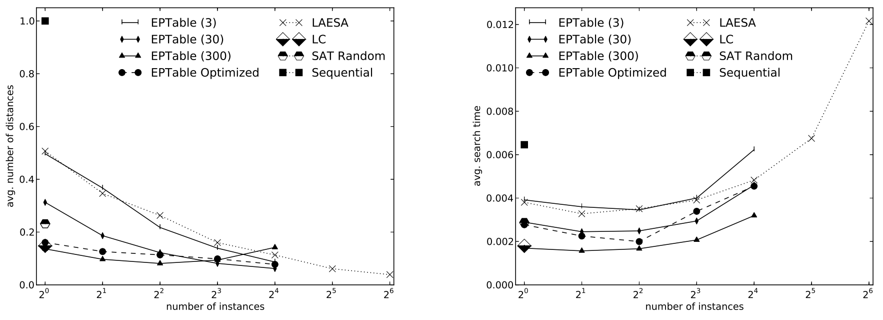
We show that with a simple tweak (essentially adjusting the number of groups) we can be faster than LAESA using as much as 64 pivots, for a fraction of the memory usage, or several times faster than LAESA using the same index size. The LC is reported as the best setup for the given dataset among the bucket sizes $16, 32, 64, 128, 256, 512, 1024$, and 2048. We tested EP Table with $3, 10, 30, 100, 300$, and 1000 pivots per instance, and up to 16 instances; we fix $\beta = 0.8$ (Section 3.1) in the numerically optimized EP Table in order to diminish the effect of the i.i.d.r.v. assumption. The optimal value of β can be very tricky, however, values around 1 produce good indexes without reducing the performance on well known distributions, and 0.8 seems to be simply enough for most setups. A detailed study on β is beyond the scope of this paper.

The algorithms were implemented in C# with the Mono framework[2]. Algorithms and indexes are available as open source software in the *natix* library[3]. All experiments were executed in a 4x quadcore Intel Xeon 2.40 GHz workstation with 32GiB of RAM, running CentOS Linux without exploiting the multicore architecture.

4.1 Performance of Our Indexes per Database

Figure 2 shows the performance for the Nasa database. The left side of the figure presents the performance as the fraction of the database revised. The EP Table is the best option, specially in setups with few instances and few pivots per instance. As shown in Figure 2(b), the search speed decreases when either the number of instances or the number of pivots per instance is large. This slowdown is because the internal computation

[2] `http://www.mono-project.org`
[3] `http://github.com/sadit/natix/`



(a) Average ratio of evaluated distances (b) Average search time

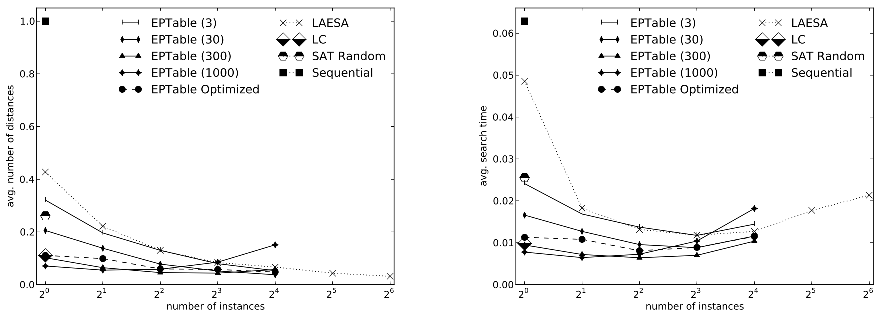
Fig. 2. Performance comparison of the nearest neighbour search in the Nasa database. We show curves only for $m = 3, 30$, and 300 in order to simplify the analysis of the figure.

increases. Please notice that the performance of the numerically optimized EP Table is very close to the best setups (on both review and real time).

In practical applications we need to optimize both memory and real query time, not the amount of computed distances. At some point, increasing the number of instances to reduce the fraction of the database revised, slowdowns the query time. Please notice that the EP Table remains as the fastest index, faster than our baselines. The speed difference is especially noticeable in setups with few instances and many pivots.

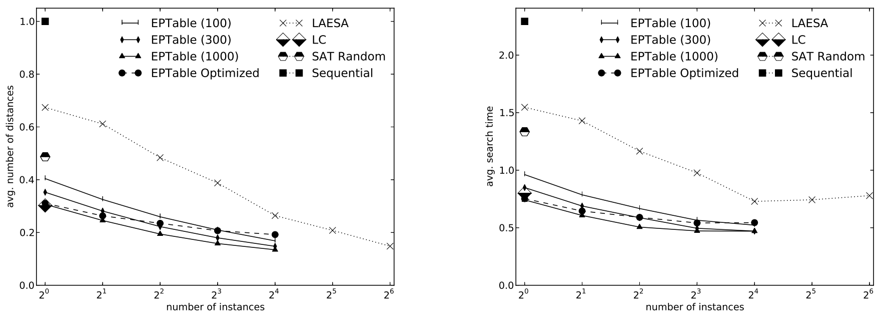
The performance of Colors is similar to the one of Nasa when the instance is numerically optimized. Figure 3(a) compares the fraction of the database revised. Notice that EP Table with few instances and several pivots per instance produce very fast indexes, surpassing the performance of LAESA, LC, and SAT. The real time, Figure 3(b) shows a similar performance for real time; however, the performance degrades for a large number of instances and pivots since the distance function is not very expensive and the internal cost becomes of matter.

The last real world database is CoPhIR-1M. Figure 4 shows how LAESA and EP Table shows a decreasing tendency as the number of instances increases. The optimum number of instances is not reached, because the database is large and the distance function is more expensive. However, both indexes increase the searching time at some point. Compared to SAT, EP Table is faster even using a single instance. As compared with LC, EP Table is faster, and equally faster with a single instance, however the LC requires more than 15 thousand centers, and EP Table is fixed with 1000 pivots, which means that EP Table is faster to construct (this is really important on databases with a costly distance function). In any case, it is possible to allow the EP Table to use more pivots per group to achieve faster indexes. In CoPhIR-1M, as in other databases, the numerically optimized EP Table is very close to the best setups of EP Table, however it decreases its performance as the number of instances increase. This can be an effect of our i.i.d. assumption. In contrast, the performance of the optimized EP Table in synthetic databases (e.g. RVEC-20) easily surpass LAESA (see Figure 5). This last fact is



(a) Average ratio of evaluated distances (b) Average search time

Fig. 3. Performance comparison of the nearest neighbour search in the Colors database. We show curves only for $m = 3, 30, 300$, and 1000 in order to simplify the analysis of the figure.



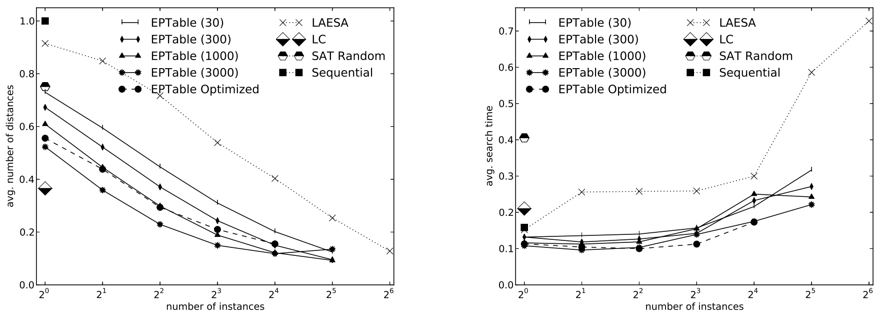
(a) Average ratio of evaluated distances (b) Average search time

Fig. 4. Performance comparison of the nearest neighbor search in the CoPhIR-1M database. We show curves only for $m = 100, 300$, and 1000 in order to simplify the analysis of the figure.

not surprising, since all the objects were generated exactly with the same random process as we have modeled. It is important to notice that for this high intrinsic dimensional dataset with a cheap distance function, our EP Table is faster than other indexes.

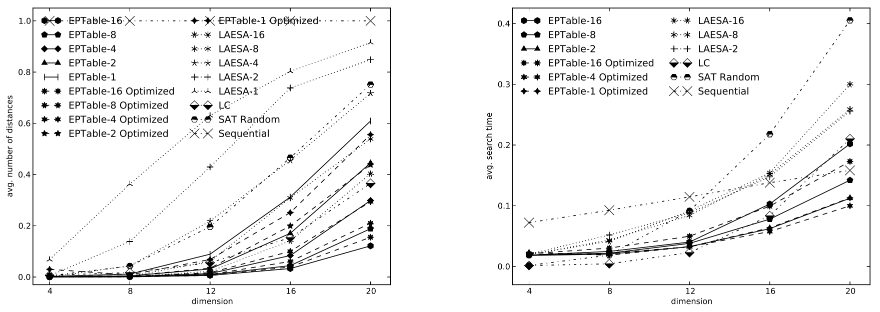
4.2 The Effect of the Dimension on the Search Performance

The curse of dimensionality stands for the odd situation where a (clever) index is slower than a plain sequential scan of the data, it is well documented in the literature [1]. The last experiment consist in testing how the indexes handle the dimensionality. We used random vectors of several dimensions in RVEC-*. Figure 6(a) shows the fraction of the database revised. Each point of EP Table represents the best performance setup in that dimension. This setup consists on six different number of pivots per instance $(3, 10, 30, 100, 300, \text{and } 1000)$. In the same figure, the *optimized* versions of EP Table



(a) Average ratio of evaluated distances (b) Average search time

Fig. 5. Performance comparison of the nearest neighbour search in the RVEC-20 database. We show curves only for $m = 30, 300, 1000$, and 3000 in order to simplify the analysis of the figure.



(a) Average ratio of evaluated distances (b) Average search time

Fig. 6. Performance comparison of the nearest neighbor search in terms of the dimensionality (random vectors). On the right side, we ommit some curves in order to improve the readability.

shows the performance for the index numerically optimized for the desired number of instances, see Section 3.1. Also, the figure shows the performance for LAESA with several pivots (1, 2, 4, 8, and 16), implying that both LAESA and EP Table use about the same amount of memory.

It is remarkable that EP Table outperforms all indexes, and it rapidly surpasses the performance of LAESA. Please also notice that EP Table is better in the majority of the setups. Also, notice that SAT rapidly degrades its performance with increasing dimension. Remarkably, the LC is quite robust (please remember that we show only the best setup for each dimension); however, EP Table surpasses both SAT and LC in most configurations. The numerically optimized EP Table shows a similar performance than the best non optimized EP Table and surpasses the performance of LAESA, SAT, and LC (even on a single instance and large dimensions). These experimental results verify the closeness of our theoretical analysis to the real performance.

Measuring the query time (Figure 6(b)), EP Table (and the numerically optimized version) produces the fastest indexes, specially in large dimensions. However, the order of the time curves is not the same than the order found measuring the amount of evaluated distances. This also remarks that minimizing the number of distances not necessarily produces faster indexes.For example, in this experiment, the faster indexes are those having a medium number of instances.

5 Conclusions

We presented EP Table, a new index for proximity searching. It can be seen as a generalization of the pivot based indexes. The indexing technique incorporates a model which proved to be accurate if the distance distribution from the objects to the pivots is known. This has been proved with uniform multidimensional synthetic data, and we postulate that once characterized the distance distributions, the same performance boost exhibited with the synthetic database will be attained with real world databases. The resulting index may have a fixed number of instances, with each instance using essentially one machine word per database element. We have shown that EP Table surpasses the performance of both pivot tables and compact partitioning indexes.

We are working on incorporating the cost of the distance function and the size of the database in the analysis and the pivot optimization. Also, the β parameter needs a deeper study to fully understand its functionality and capabilities, this is one of our current research trends. Finally, we are studying how to substitute the table search with faster structures using the internal structure of the partition to speed up searches.

References

1. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. 33(3), 273–321 (2001)
2. Samet, H.: Foundations of Multidimensional and Metric Data Structures, 1st edn. The Morgan Kaufman Series in Computer Graphics and Geometic Modeling. Morgan Kaufmann Publishers, University of Maryland at College Park (2006)
3. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search - The Metric Space Approach, 1st edn. Advances in Database Systems, vol. 32. Springer (2006)
4. Vidal Ruiz, E.: An algorithm for finding nearest neighbours in (approximately) constant average time. Pattern Recognition Letters 4, 145–157 (1986)
5. Micó, M.L., Oncina, J., Vidal, E.: A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. Pattern Recogn. Lett. 15, 9–17 (1994)
6. Bustos, B., Navarro, G., Chávez, E.: Pivot selection techniques for proximity searching in metric spaces. Pattern Recognition Letters 24(14), 2357–2366 (2003)
7. Celik, C.: Priority vantage points structures for similarity queries in metric spaces. In: Shafazand, H., Tjoa, A.M. (eds.) EurAsia-ICT 2002. LNCS, vol. 2510, pp. 256–263. Springer, Heidelberg (2002)
8. Celik, C.: Effective use of space for pivot-based metric indexing structures. In: SISAP 2008: Proceedings of the First International Workshop on Similarity Search and Applications (sisap 2008), pp. 113–120. IEEE Computer Society, Washington, DC (2008)

9. Navarro, G.: Searching in metric spaces by spatial approximation. The Very Large Databases Journal (VLDBJ) 11(1), 28–46 (2002)
10. Chávez, E., Navarro, G.: A compact space decomposition for effective metric indexing. Pattern Recogn. Lett. 26, 1363–1376 (2005)
11. Bolettieri, P., Esuli, A., Falchi, F., Lucchese, C., Perego, R., Piccioli, T., Rabitti, F.: CoPhIR: a test collection for content-based image retrieval. CoRR abs/0905.4627v2 (2009)

Quicker Similarity Joins in Metric Spaces

Kimmo Fredriksson[*] and Billy Braithwaite

School of Computing, University of Eastern Finland
firstname.lastname@uef.fi

Abstract. We consider the join operation in metric spaces. Given two sets A and B of objects drawn from some universe $\mathbb{U}$, we want to compute the set $A \bowtie B = \{(a,b) \in A \times B \mid d(a,b) \leq r\}$ efficiently, where $d : \mathbb{U} \times \mathbb{U} \to \mathbb{R}^+$ is a metric distance function and $r \in \mathbb{R}^+$ is user supplied query radius. In particular we are interested in the case where we have no index available (nor we can afford to build it) for either A or B. In this paper we improve the Quickjoin algorithm (Jacox and Samet, 2008), based on the well-know Quicksort algorithm, by (i) replacing the low level component that handles small subsets with essentially brute-force nested loop with a more efficient method; (ii) showing that, contrary to Quicksort, in Quickjoin unbalanced partitioning can improve the algorithm; and (iii) making the algorithm probabilistic while still obtaining most of the relevant results. We also show how to use Quickjoin to compute k-nearest neighbor joins. The experimental results show that the method works well in practice.

1 Introduction

Many important problems in information and multimedia retrieval, pattern recognition, data mining and computational biology, to name a few, can be seen as proximity or similarity searching in metric spaces. Metric space is a pair $(\mathbb{U}, d)$, where $\mathbb{U}$ is an universe of objects, and $d(\cdot,\cdot)$ is a distance function $d : \mathbb{U}\times\mathbb{U} \to \mathbb{R}^+$. The distance function is *metric*, if it satisfies for all $x, y, z \in \mathbb{U}$

$$\begin{aligned} &d(x,y) = 0 \text{ iff } x = y \text{ (reflexivity)},\\ &d(x,y) = d(y,x) \text{ (symmetry)},\\ &d(x,y) \leq d(x,z) + d(z,y) \text{ (triangular inequality)}. \end{aligned}$$

In the point of view of the applications, we have some subset $S \subseteq \mathbb{U}$ of objects, $|S| = n$, and we are interested in the proximity of the objects towards themselves, or towards some query objects. The most fundamental type of query is *range query*: retrieve all objects in the database S that are within a certain similarity threshold r to the given query object q, that is, compute $R(S,q) = \{o \in S \mid d(q,o) \leq r\}$. Another common query is to retrieve the k-nearest neighbors of q in S. A large number of different data structures and query algorithms have been proposed, see e.g. [1–3].

[*] Corresponding author.

In this paper we consider *similarity join*, a query that has several solutions in vector spaces (a particular case of a metric space), but only a few methods exist for general metric spaces. More precisely, the *range* join operation for two sets A and B is defined as

$$A \underset{r}{\bowtie} B = \{(a,b) \in A \times B \mid d(a,b) \leq r, a \in A, b \in B\}.$$

Sometimes we may be interested in *self join*, which is defined simply as $A \bowtie A$. The two basic solutions to solve join queries are to use two nested loops and compute all the pairwise distances, or to utilize an index build to support range queries in either set. In this paper we are interested in the case where we have no index available, nor it is feasible to build such an index for various reasons, see Sec. 2 for justifications of this assumption. Still we aim to do much better than the nested loop approach.

Finally, we note that in general metric spaces the (black-box) distance function is the only way to distinguish between the objects. Moreover, the distance function is often very expensive to evaluate (consider e.g. comparing documents or images). Hence the usual complexity measure is the number of distance function evaluations required to answer the query. For example, the basic solution to compute $A \bowtie B$ by using two nested loops has therefore the complexity $\Theta(|A|\,|B|)$.

Our contributions are as follows. We improve the Quicksort [4] inspired Quickjoin algorithm [5] in three ways. (1) The original algorithm is based on partitioning the data set to smaller pieces, and the small pieces are joined essentially by brute-force. We discuss several method to improve that phase. The result can also be used as is rather than just being a component of Quickjoin. (2) We show that, contrary to Quicksort, in Quickjoin (controllably) mildly unbalanced partitioning can improve the algorithm. (3) We show how to make the algorithm probabilistic, to gain faster execution with a price of possibly missing a small fraction of the relevant results. All the improvements can be combined into a single algorithm. We also sketch a method to adapt Quickjoin to handle k-nearest neighbor joins. The experimental results show that the running time of the algorithm is substantially improved in practice.

The paper is organized as follows. In Sec. 2 we briefly cover some previous work on similarity joins, giving a more detailed look of the Quickjoin algorithm in Sec. 2.1, as this is the basis of our work. Section 3 describes our improvements to Quickjoin, and in Sec. 4 we adapt the algorithm to compute k-NN joins. Section 5 gives experimental results and Sec. 6 concludes.

2 Previous Work

As already mentioned, the basic solution is to compute all the pairwise distances, which leads to quadratic complexity (Alg. 4 gives the pseudocode). To get subquadratic time on average, one can take the smaller of the two data sets, preprocess it to use (almost) any of the various off-the-self indexes [1–3] supporting range queries, and query the objects from the other set from the index (see Alg. 7). This method can be reasonably good, if the time for building the index is subquadratic, which is often but not always true (e.g. AESA [6] needs quadratic time to build the index, effectively solving the self join problem while doing so). However there is often a trade-off between the efficiency of the build time and query time, an illustrative example of this is the List of Clusters [7].

For spatial joins in vector spaces there are many specific methods that support the join operation, some but not all of them are based on indexing. While it is possible to map general metric spaces to vector spaces, the applicability of these methods are limited. For discussion and references, see [5].

To the best of our knowledge, there are only two indexing based methods to solve similarity joins in general metric spaces. The first is eD-index [8] (refinement of its predecessor, D-index [9]), which effectively uses the strategy to build an index supporting range queries, and then use it repeatedly to solve the join query. The eD-index is a tree structure based on search-separable buckets. Basically the index uses ball partitioning technique, but it replicates objects that are at distance of at most r from the partition boundaries to the "exclusion" set. This guarantees that there always exist a bucket for any pair (a, b) if $d(a, b) \leq \epsilon$ (it is assumed that $r \leq \epsilon$). Quickjoin uses somewhat similar strategy, although it is not an indexing technique.

More recently, an index called List of Twin Clusters [10] (extension of List of Clusters [7]) was proposed. This method indexes *two* different datasets simultaneously, using compact ball partitioning technique. Their method support range searching in either indexed set, and (range) join operation between the two indexed sets. They also considered k-nearest neighbor joins.

Finally, Quickjoin algorithm [5] solves the range join efficiently without any index. We cover the algorithm in detail in Sec. 2.1. Some of the various *pros* and *cons* of range searching in general, eD-index, LTC and Quickjoin are as follows:

– While using off-the-self range searching methods immediately offers a wide variety of solutions, they are not necessarily better than Quickjoin.
– Sometimes the datasets can be large, and building an index may be costly if it does not fit into main memory. Quickjoin uses relatively little memory.
– If one wants to join a certain dataset repeatedly against several others (or using different query radii), then the use of some good indexes with high construction cost may be amortized.
– The indexing techniques support other queries besides join as well.
– While eD-index has some attractive properties, it is dependent on the parameters ρ and ϵ, and the index need to be rebuilt for query radii $r > \epsilon$. The authors considered only a self join operation.
– LTC supports join only for two fixed datasets, and need to be rebuilt for joining against a third set, unless one resorts to the basic range searching strategy. The construction cost is also high.
– Quickjoin does not directly support k-nearest neighbor joins (however, see Sec. 4).

On the other hand, as shown in Secs. 3.1 and 5, Quickjoin can be improved by combining it with indexing only small subsets of the input.

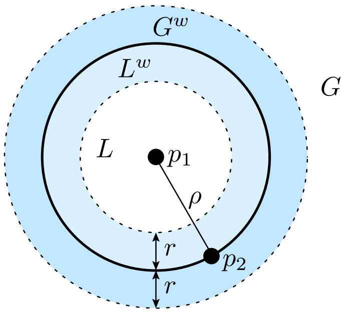


Fig. 1. Quickjoin ball partitioning. The input data set S is partitioned to four sets. Note that $L \cup G = S$, $L^w \subseteq L$, $G^w \subseteq G$ and $L \cap G = L^w \cap G^w = \emptyset$.

2.1 Quickjoin

We concentrate on self join. The general case of $A \bowtie B$ is easily solved by setting $S = A \cup B$ and solving $S \bowtie S$, and adding a simple check that does not report a pair (u, v) if u and v both belong to either in A or B.

Algorithms 1–4 show the complete pseudocode (except for SelfJoinBF, which is almost identical to JoinBF shown in Alg. 4). In high level, the algorithm is based on partitioning the data set S recursively into small subsets, until the subset size is at most some small constant c, in which case the algorithm switches to a brute-force nested loop approach. The (ball) partitioning algorithm divides the set into four partly overlapping subsets. To this end, the algorithm chooses (randomly) two *pivots* p_1 and p_2 from S, and uses the distance $\rho = d(p_1, p_2)$ and the pivot p_1 to define a ball that partitions the space in two. More precisely, the algorithm computes

$$L = \{u \in S \mid d(u, p_1) < \rho\} \tag{1}$$
$$G = \{u \in S \mid d(u, p_1) \geq \rho\}. \tag{2}$$

Figure 1 illustrates (note: the original paper used inequalities "$\leq$" and "$>$" for L and G respectively). Both sets are then joined recursively, separately. However, there may be some object $u \in L$ and another object $v \in G$ such that $d(u, v) \leq r$. The objects near the partition boundary are therefore replicated into two *window* partitions:

$$L^w = \{u \in S \mid \rho - r \leq d(u, p_1) < \rho\} = \{u \in L \mid \rho - r \leq d(u, p_1)\} \tag{3}$$
$$G^w = \{u \in S \mid \rho \leq d(u, p_1) \leq \rho + r\} = \{u \in S \mid d(u, p_1) \leq \rho + r\}. \tag{4}$$

These are again joined against each other recursively, using Alg. 3 (a slight variation of Alg. 1). Note that this algorithm computes $L^w \bowtie G^w$, and not $(L^w \cup G^w) \bowtie (L^w \cup G^w)$, as $L^w \bowtie L^w$ and $G^w \bowtie G^w$ are already solved as a part of $L \bowtie L$ and $G \bowtie G$. All four partitions can be computed with $O(|S|)$ distance computations with a single pass through S.

Alg. 1. QuickJoin (S, r)

```
if $|S| \leq c$ then return SelfJoinBF $(S, r)$
$p_1 \leftarrow$ Pivot$(S)$
$p_2 \leftarrow$ Pivot$(S \setminus \{p_1\})$
$\rho \leftarrow d(p_1, p_2)$
$(L, G, L^w, G^w) \leftarrow$ Partition$(S, p_1, r, \rho)$
$R \leftarrow$ QuickJoin$(L, r) \cup$ QuickJoin$(G, r) \cup$ QuickJoinWin$(L^w, G^w, r)$
return $R$
```

Alg. 2. Partition (S, p, r, ρ)

```
$L \leftarrow \{u \in S \mid d(u, p) < \rho\}$
$G \leftarrow \{u \in S \mid d(u, p) \geq \rho\}$
$L^w \leftarrow \{u \in L \mid \rho - r \leq d(u, p_1)\}$
$G^w \leftarrow \{u \in S \mid d(u, p_1) \leq \rho + r\}$
return $(L, G, L^w, G^w)$
```

Alg. 3. QuickJoinWin (S_1, S_2, r)

```
if $|S_1| + |S_2| \leq c$ then return JoinBF $(S_1, S_2, r)$
$p_1 \leftarrow$ Pivot$(S_1 \cup S_2)$
$p_2 \leftarrow$ Pivot$((S_1 \cup S_2) \setminus \{p_1\})$
$\rho \leftarrow d(p_1, p_2)$
$(L_1, G_1, L_1^w, G_1^w) \leftarrow$ Partition$(S_1, p_1, r, \rho)$
$(L_2, G_2, L_2^w, G_2^w) \leftarrow$ Partition$(S_2, p_1, r, \rho)$
$R \leftarrow$ QuickJoinWin$(L_1, L_2, r) \cup$ QuickJoinWin$(G_1, G_2, r)$
$W \leftarrow$ QuickJoinWin$(L_1^w, G_2^w, r) \cup$ QuickJoinWin$(G_1^w, L_2^w, r)$
return $R \cup W$
```

Alg. 4. JoinBF (S_1, S_2, r)

```
$R \leftarrow \emptyset$
for $i \leftarrow 1$ to $|S_1|$ do
    for $j \leftarrow 1$ to $|S_2|$ do
        if $d(S_1[i], S_2[j]) \leq r$ then $R \leftarrow R \cup \{(S_1[i], S_2[j])\}$
return $R$
```

The algorithm was shown to make $O(n(1+w)^{\log n}) = O(n^{1+\log(1+w)})$ distance computations on average, where $w \leq 1$ is the fractional average size of the window partitions. They gave some experimental evidence that w is nearly linearly dependent on r (however, see also Sec. 3.3), the total time approaching $O(n^2)$ for large r. The worst case is $O(n2^n)$, i.e. when w approaches 1 and the recursion depth approaches n (very unbalanced partitioning). The algorithm can use generalized hyperplane partitioning instead of ball partitioning, can partition the data to more subsets, can base the partitioning in coordinate data in case of vector spaces, and it can be made to perform reasonably well on secondary memory as well. For more details refer to [5].

3 Improved Quickjoin

We give three improvements to Quickjoin. The first two can in principle also be used as is to solve $A \bowtie B$, without being a building block for Quickjoin.

3.1 Handling Small Subsets

The Quickjoin algorithm spends most of its time in Alg. 4 (depending also on the cut-off constant c). There are many possibilities to improve it.

Our first alternative is to use pivot based strategy. Given some small constant $k < |S_1|$, we first compute the distances $d(S_1[i], S_2[j])$, for $1 \leq i \leq k$ and $1 \leq j \leq |S_2|$, saving the distance to $P[i,j]$ and reporting a match if $P[i,j] \leq r$. In the next phase the items $S_1[i]$ for $1 \leq i \leq k$ are used as pivots to filter out objects from S_2. That is, given $S_1[l]$ for $l > k$, we first compute all distance $d(S_1[l], S_1[i])$, again for all $1 \leq i \leq k$. The triangular inequality can then be used to discard some object, in particular, if $|d(S_1[i], S_1[l]) - d(S_1[i], S_2[j])| > r$ for some i (note that these distances are known at this point), then we know that $d(S_1[l], S_2[j]) > r$ without actually evaluating the distance. Alg. 5 gives the pseudocode. The number of distance computations is still $O(|S_1|\,|S_2|)$ in the worst case, but note that the algorithm needs also some extra CPU time (the time besides evaluating the distance function) as compared to Alg. 4.

Another approach, which is also very simple to implement, has lower extra CPU time and has lower memory footprint, is to adapt the technique proposed in [11]. Assume that we are interested in $d(q,p)$, and that we know $d(q,q')$, where $q, q' \in S_1$, and $p \in S_2$, and $d_L(p)$ and $d_U(p)$ denote the known lower and upper bound distances to $d(q,q')$.

Lemma 1 ([11]). *Let $d_L(p) \leq d(p,q') \leq d_U(p)$, and $e = d(q,q')$ and $l = \max\{e - d_U(p), d_L(p) - e, 0\}$ and $u = e + d_U(p)$. It holds that $l \leq d(q,p) \leq u$.*

In our case we initialize $d_L(p) = 0$ and $d_U(p) = \infty$ for all $p \in S_2$, and maintain $e = d(q,q')$ by using $q = S_1[i]$ and $q' = S_1[i-1]$. The algorithm takes each $q \in S_1$ in turn and matches it against all $p \in S_2$. Based on Lemma 1, we can immediately report a matching pair (q,p) if $u \leq r$ (in practice this is rather unlikely), or deduce that (q,p) is not in our result if $l > r$ (in practice this is where the improvement comes from), both without ever evaluating $d(q,p)$. Only if $r \leq l$ we must evaluate the actual distance, and in this case we also update $d_L(p) = d_U(p) = d(q,p)$. Otherwise we set $d_L(p) = l$ and $d_U(p) = u$. In either case, we effectively have now $d_L(p) \leq d(q,p) \leq d_U(p)$, ready to handle the next item from S_1. Alg. 6 shows the pseudocode.

As the method is used to filter single objects, rather than balls with covering radius as in the original algorithm, as a part of an index, it works better in our context. In the worst case the algorithm is slightly slower than Alg. 4, but not asymptotically, and in practice it is substantially better. Moreover, both Alg. 6 and Alg. 5 can easily be made probabilistic, contrary to Alg. 4, see Sec. 3.2. Also, the two algorithms could be combined, but we leave this avenue as a future work.

Alg. 5. JoinPivots (S_1, S_2, r)

```
$R \leftarrow \emptyset$
**for** $i \leftarrow 1$ **to** $k$ **do for** $j \leftarrow 1$ **to** $|S_2|$ **do**
    $P[i,j] \leftarrow d(S_1[i], S_2[j])$
    **if** $P[i,j] \leq r$ **then** $R \leftarrow R \cup \{(S_1[i], S_2[j])\}$
**for** $i \leftarrow k+1$ **to** $|S_1|$ **do**
    **for** $l \leftarrow 1$ **to** $k$ **do** $D[l] \leftarrow d(S_1[l], S_1[i])$
    **for** $j \leftarrow 1$ **to** $|S_2|$ **do**
        $f \leftarrow$ false
        **for** $l \leftarrow 1$ **to** $k$ **do if** $|P[l,j] - D[l]| > r$ **then** $f \leftarrow$ true; **break**
        **if** NOT $f$ AND $d(S_1[i], S_2[j]) \leq r$ **then** $R \leftarrow R \cup \{(S_1[i], S_2[j])\}$
**return** $R$
```

Alg. 6. JoinDC (S_1, S_2, r)

```
**for** $j \leftarrow 1$ **to** $|S_2|$ **do** $d_L[j] \leftarrow 0$; $d_U[j] \leftarrow \infty$
$R \leftarrow \emptyset$; $e \leftarrow \infty$
**for** $i \leftarrow 1$ **to** $|S_1|$ **do**
    **if** $i > 1$ **then** $e \leftarrow d(S_1[i], S_1[i-1])$
    **for** $j \leftarrow 1$ **to** $|S_2|$ **do**
        $d_L[j] \leftarrow \max\{e - d_U[j], d_L[j] - e, 0\}$
        $d_U[j] \leftarrow e + d_U[j]$
        **if** $d_U[j] \leq r$ **then** $R \leftarrow R \cup \{(S_1[i], S_2[j])\}$
        **else if** $d_L[j] \leq r$ **then**
            $d_L[j] \leftarrow d_U[j] \leftarrow d(S_1[i], S_2[j])$
            **if** $d_U[j] \leq r$ **then** $R \leftarrow R \cup \{(S_1[i], S_2[j])\}$
**return** $R$
```

Note that the efficiency of Alg. 6 depends partly on how close $S_1[i-1]$ and $S_1[i]$ are to each other. That is, the permutation of the set S_1 affects the complexity. One possible way to improve the algorithm is to permute the set so that $\sum d(S_1[i-1], S_1[i])$ is minimized. However, deciding if a given permutation is optimal in this sense is NP-complete problem even for vector spaces with two dimensions, which is easily seen by reducing from TSP on Euclidean plane. We use a simple greedy heuristic that selects the next query object from a small sample at each step, which improves the algorithm in some cases.

Finally, we note that any known indexing technique with a reasonably low construction cost supporting range queries can be used to replace Alg. 4. If the construction cost is subquadratic, the total average time including the queries will be as well. Alg. 7 gives a generic code. In our experiments we used Vantage Point Tree (VPT) [12].

In Sec. 5 we experimented with replacing JoinBF (Alg. 4) with the presented alternatives. Similarly we replaced SelfJoinBF with the above methods modified to compute self join. The modifications are trivial, so we omit the details.

Alg. 7. JoinRQ (S_1, S_2, r)

```
1  if $|S_1| > |S_2|$ then $S_1 \leftrightarrow S_2$
2  $I \leftarrow$ IndexBuild($S_1$); $R \leftarrow \emptyset$
3  for $i \leftarrow 1$ to $|S_2|$ do $R \leftarrow R \cup$ IndexSearch($I, S_2[i], r$)
4  return $R$
```

3.2 Making It Probabilistic

In [13] a general probabilistic method was given to "stretch the triangle inequality". In indexing point of view, the normal triangle inequality allows us to discard any element u in the database that satisfies $|d(u,p) - d(q,p)| > r$, where q is the query object, and p is some pivot object, without evaluating $d(u,q)$. "Stretching" means that we will now discard u if $|d(u,p) - d(q,p)| > (1 - \varepsilon) \times r$ holds, for some $0 \leq \varepsilon < 1$. They show that this method can dramatically reduce the number of distance computations, while keeping the error probability low, i.e. only a small fraction of relevant results is missed. To apply this idea to Alg. 5, we replace the line 9 with:

```
9        if $|P[l,j] - D[l]| > (1 - \varepsilon) \times r$ then $f \leftarrow$ true; break
```

Similarly, we can reduce the number of distance computations by replacing the line 8 of Alg. 6 with:

```
8      else if $d_L[j] \leq (1 - \varepsilon) \times r$ then
```

As shown by the experimental results, the algorithm is very tolerant to quite large values of ε. Similar adjustment can be often made for Alg. 7, we used the the method described in [14].

3.3 Optimizing by Unbalancing

Recall that the average time for Quickjoin is $O(n(1+w)^{\log n})$. The exponent $\log n$ comes from the fact that the depth of the recursion is $O(\log n)$ if the partitioning is balanced. See also [5, Sec. 3.1, p. 13] for a discussion how to enforce a balanced partition. It should be obvious that the complexity decreases as the depth of the recursion is decreased. In fact, if the ratio between the size of the partitions is $\alpha : (1 - \alpha)$, for some constant $0 < \alpha \leq \frac{1}{2}$, the depth of the recursion is $O(\log_{1/(1-\alpha)} n)$, which is $O(\log_2 n)$ for $\alpha = \frac{1}{2}$. However, the complexity depends also on w, and that *decreases* (for a constant r) when the partition becomes unbalanced, towards L, i.e. when the recursion depth *increases* in a certain way. This should be apparent on Fig. 1; when ρ decreases, the "volume" of the window partitions (the "rings") decrease. Note that this assumes ball partitioning, for generalized hyperplane partitioning balanced partitioning seems to be the best strategy. We experimented with this idea simply by replacing the line 4 in both QuickJoin and QuickJoinWin with:

```
4    $\rho \leftarrow \beta \times d(p_1, p_2)$
```

where β is some constant ≤ 1. As shown in Sec. 5 decreasing β improves the time complexity of the algorithm up to a certain point. We leave more detailed analysis for a future work.

4 *k*-NN Join

We now show how Quickjoin can be adapted to compute the k-nearest neighbor joins. More precisely, we want to compute a subset $A \bowtie B \subset A \times B$, such that $|A \bowtie B| = k$ and for all $(a, b) \in A \bowtie B$ and $(u, v) \in A \times B \setminus A \bowtie B$ it holds that $d(a, b) \leq d(u, v)$.

Disregarding k-NN joins, there there exists few specific algorithms for "normal" k-NN queries, retrieving the set k-NN(q), the k closest object to the query object q. However, there are two related general methods to use range queries to compute the k-NN query that can be applied to almost any index supporting range queries [1]. The first is to start with $r = \infty$ and update (decrease) r as more and more objects from the index are compared against q, namely, keeping r equal to the largest distance to the k nearest neighbors seen so far. The second strategy is to start with some small r, and doubling (or using some smaller constant factor) it until the range query retrieves at least k objects. Our method is also based on range joins, and can be seen as a combination of the two alternatives above.

Our solution is to run the range join algorithm twice. The first time we use $r = 0$. This effectively means that the window partitions are empty, and the cost is $O(n \log n)$ distance evaluations on average for the partitioning process. However, the algorithm still makes calls to SelfJoinBF, when the size of the subset is $\leq c$. We modify this so as to keep track (with a heap) of the k smallest distances (and the corresponding object pairs) encountered. Thus, when the algorithm ends, we have approximation of the true result, i.e. some relevant object pairs are likely missed (and replaced by some inferior pairs). For the second pass of the range join we use $r := r'$, where r' is the largest distance between the object pairs found in the first pass. Note that r' is the upper bound for the largest distance between the object pairs of the true result, if the first pass computes at least k distances in SelfJoinBF. This can be guaranteed by using $c \geq \lceil 4k/n \rceil + 1$ in the first pass, assuming self-join and balanced partitioning. (It is also easy to handle wildly unbalanced partitioning by simply stopping the recursion early.) The second pass can use whatever value for c. It is also easy to further optimize the algorithm by always keeping r as the largest distance between the object pairs found so far, effectively decreasing it over time. Clearly the second pass dominates the running time. Notice that the variants that improve over JoinBF may skip distance computations, hence reducing r over time may work less effectively.

5 Preliminary Experimental Results

We have implemented the algorithms in C and ran experiments with different datasets comparing the performance of our variants with varying parameters against the original Quickjoin. We made experiments with random vectors in uniformly distributed unitary cube (this allows us to easily control the dimension and the size of the input data), *nasa* dataset of 40,150 feature vectors in 20-dimensional space (although the intrinsic dimension is much lower), and with a dictionary of 69,069 English words. Both real world datasets were obtained from `http://www.sisap.org/library/dbs/`. For vector data we used Euclidean distance and for strings edit distance.

We have many parameters to test out. The problem specific parameters are r, n, dimension, or specific dataset. In addition we have several parameters controlling the behavior of the algorithm itself, the bucket size (c in Algs. 1 and 3), which algorithm to use to handle the buckets (Alg. 4, 6, 5 or 7), ε to control the probabilistic variant(s) and β to control the balance of the recursion. Alg. 5 also depends on the number of pivots to use. It is not feasible to test all possible combinations systematically, so we test them mostly separately, picking some default or promising values for the free parameters.

We report only the number of distance evaluations made, neglecting the actual running times, which grow approximately linearly with the number of distance evaluations. For easier comparison among the different datasets, we report the numbers as $(e/b) \times 100\%$, where e is the number of actual distance evaluations, $b = n(n-1)/2$ is the number of distance evaluations needed by the naïve brute-force approach for self join, where n is the size of the dataset. For Alg. 5 we used simply $k = \max(1, \min(16, |S_1|/8))$ pivots. The value of k can have a large impact on the performance (as well as the choice of the pivots), but we did not make any efforts to optimize this.

Figure 2 shows for random vector space the effects of the bucket size c and the balance factor β (for $c = 30$) when using either Alg. 4 (JoinBF = BF), 5 (JoinPivots = Pivots), 6 (JoinDC = DC) or 7 (JoinRQ = RQ) to handle the small subsets in QuickJoin and QuickJoinWin. We omit the result for optimizing the query order for Alg. 6, as the results were usually just slightly worse or better than for plain Alg. 6. In general it seems to be good only for high dimensions and large query radii. All alternatives to BF beat it. For experiment with huge values of c, see Fig. 4. Combined with unbalanced partitioning we can almost double the performance as compared to the original algorithm.

Figure 3 shows the effect of the dimension, while keeping the number of results retrieved constant. Here we used JoinBF to handle the small subsets in all cases. Again, unbalancing the partitioning helps considerably.

Figure 4 shows the results for probabilistic search. BF, which is not probabilistic, is shown just for reference. For fixed c the number of distance evaluations decrease approximate linearly with increasing ε. For $\varepsilon \gtrapprox 0.4$ all the methods start quickly miss relevant results. Pivots is able to prune the number of distance evaluations the most, but it is also the least accurate. Note however, the the number of pivots used greatly affect this [13], the more pivots used, the more relevant results are missed in general, although the algorithm also gets faster. The reason is quite clear, the more pivots, the more opportunities to make a

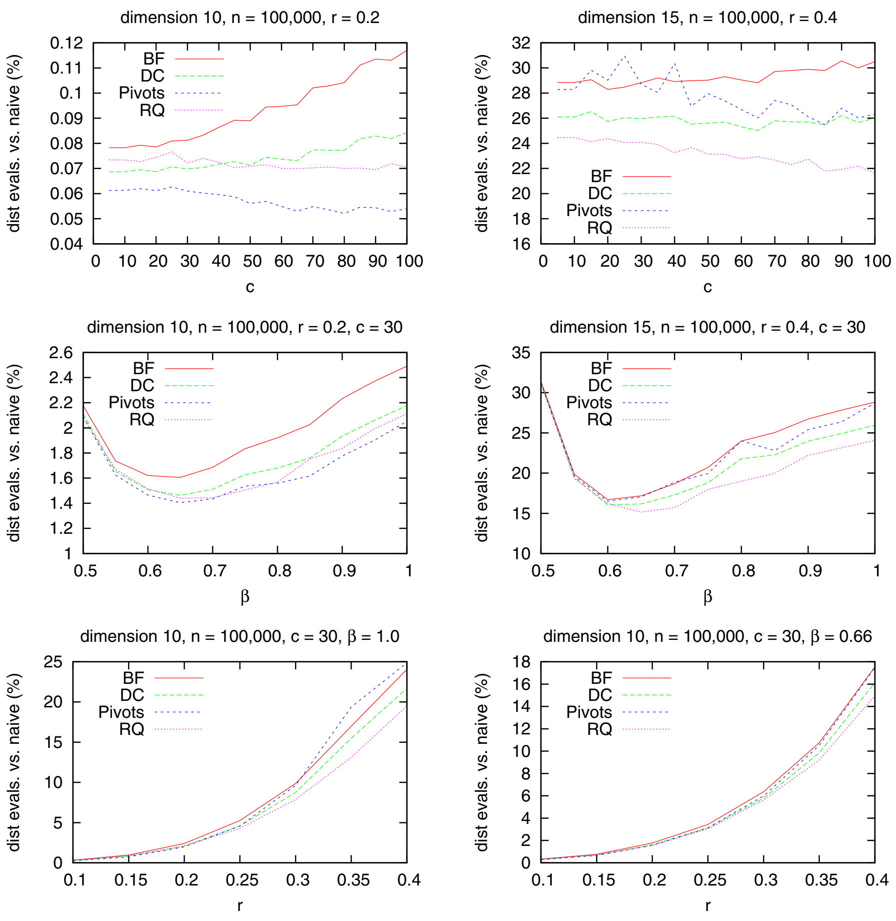


Fig. 2. Top row: the effect of c. Middle: optimum unbalance is close to $\beta \approx \frac{2}{3}$. Bottom row: the effect of increasing radius, for $\beta = 1.0$ and $\beta = 0.66$ (dimension = 10).

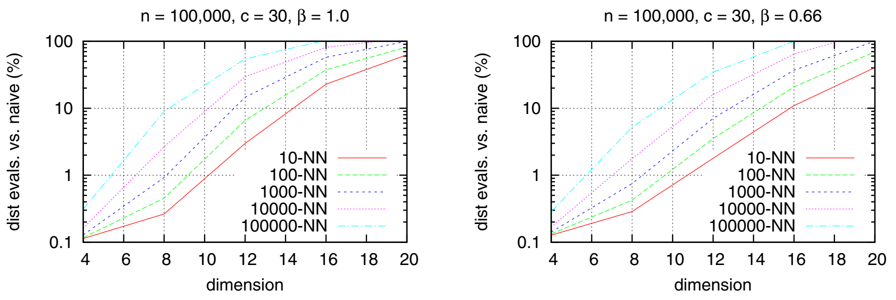


Fig. 3. Performance for k-NN for different dimensions and for $\beta = 0.66$ and $\beta = 1.0$

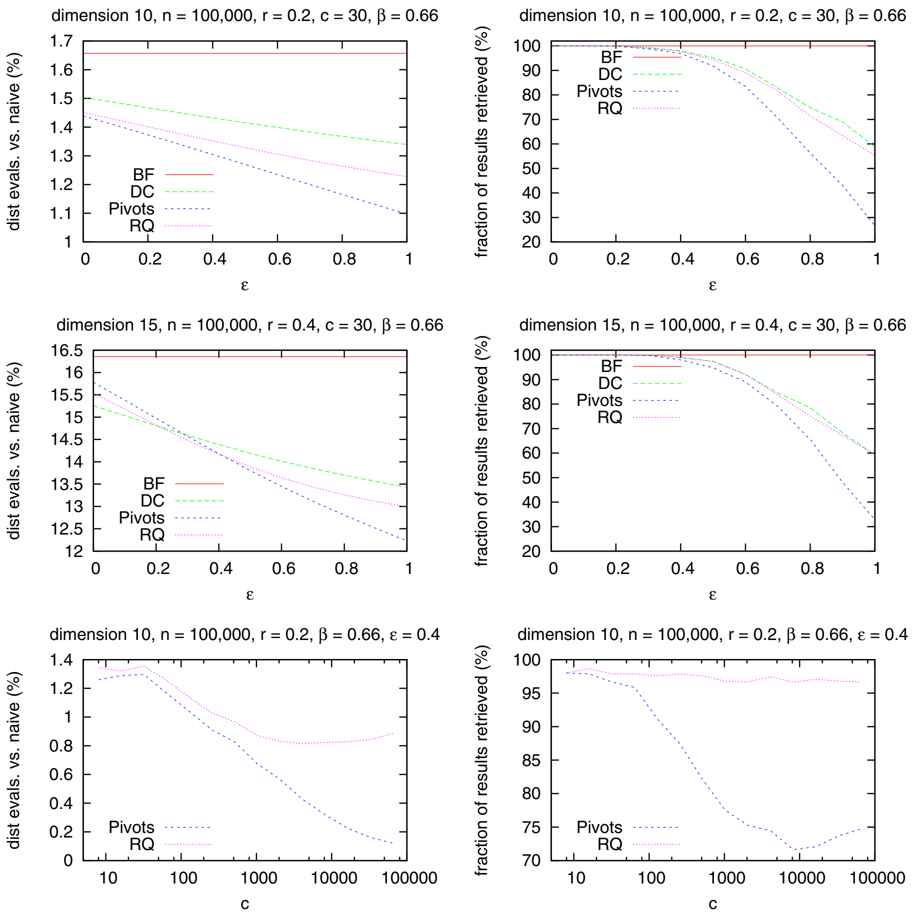


Fig. 4. Probabilistic search. Top and middle rows: the number of distance evaluations and the results retrieved as a function of ε. Bottom row: increasing c close to n, with $\varepsilon = 0.4$. In all cases $\beta = 0.66$.

(wrong) decision to filter out some object. The bucket size c also affects the performance. For larger c the algorithm gets faster, but especially Pivots starts to miss even more relevant results (for fixed ε).

Finally, Fig. 5 repeats some of the experiments on real data. The results are similar, but small discrete range for radii flattens out the plot for β on string data somewhat. For the same reason, we omit the plots for probabilistic search on string data, since e.g. for $r = 2$ any $0.5 \leq \varepsilon < 1$ is effectively the same. Note also that the balance factor (β) plot for Nasa dataset deviates from the plot with random data. This is mostly due to the relatively low intrinsic dimension of the Nasa dataset.

We mention some additional results without plots. When combining all the improvements and comparing against plain Quickjoin, we can get respectable

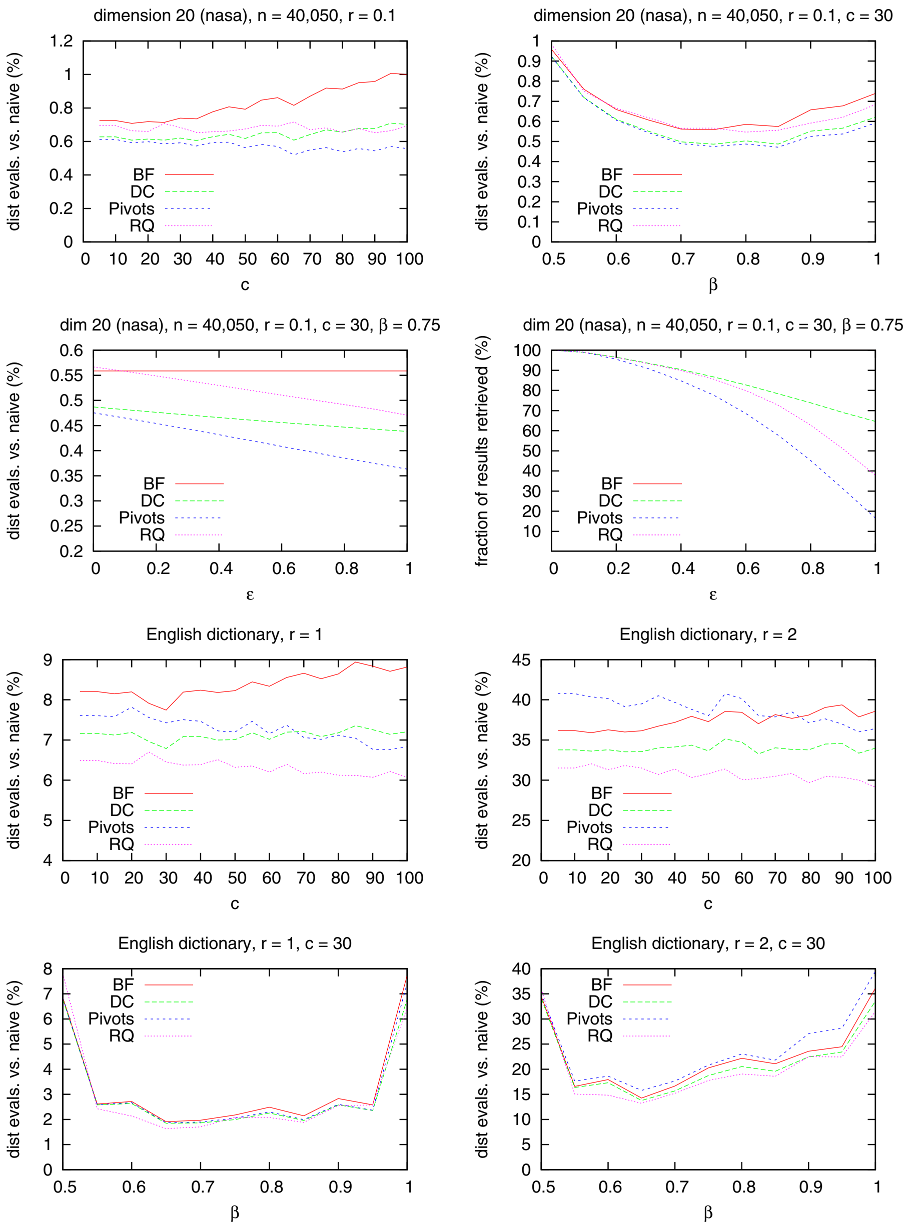


Fig. 5. Performance on real data, Nasa and English dictionary

improvements. For example, using the English dictionary and $r = 2$, $c = 100$, $\beta = 0.66$ and RQ, we get about 2.6 fold improvement against standard Quickjoin ($c = 30$, $\beta = 1.0$, BF). Similarly for random vector space of dimension 15,

$n = 100,000$, $r = 0.4$, $c = 1000$, $\beta = 0.66$, $\varepsilon = 0.4$ and RQ, we get 3.2 fold improvement against Quickjoin, while still retrieving 99.8% of the results.

6 Concluding Remarks and Future Work

We have shown three ways to improve Quickjoin algorithm. The algorithms work well in practice, improving the performance by up to 3 fold. We plan to make more comprehensive experiments in the future. Many improvements are still possible. For example, using range queries to handle the small buckets seems promising, but there are many other alternatives to the structures we have considered here. It would be interesting to tailor some index specifically for Quickjoin, so that the build and search times are balanced, and keeping the memory consumption low. Saving some distance evaluations during the partitioning process seems also possible, using technique similar to Alg. 6.

References

1. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquin, J.: Searching in metric spaces. ACM Computing Surveys 33(3), 273–321 (2001)
2. Hjaltason, G., Samet, H.: Index-driven similarity search in metric spaces. ACM Transactions Database Systems 28(4), 517–580 (2003)
3. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann Publishers Inc., San Francisco (2005)
4. Hoare, C.A.R.: Quicksort. Comput. J. 5(1), 10–15 (1962)
5. Jacox, E.H., Samet, H.: Metric space similarity joins. ACM Trans. Database Syst. 33(2) (2008)
6. Vidal, E.: An algorithm for finding nearest neighbors in (approximately) constant average time. Pattern Recognition Letters 4, 145–157 (1986)
7. Chávez, E., Navarro, G.: A compact space decomposition for effective metric indexing. Pattern Recognition Letters 26(9), 1363–1376 (2005)
8. Dohnal, V., Gennaro, C., Zezula, P.: Similarity join in metric spaces using eD-index. In: Mařík, V., Štěpánková, O., Retschitzegger, W. (eds.) DEXA 2003. LNCS, vol. 2736, pp. 484–493. Springer, Heidelberg (2003)
9. Dohnal, V., Gennaro, C., Savino, P., Zezula, P.: D-index: Distance searching index for metric data sets. Multimedia Tools Appl. 21(1), 9–33 (2003)
10. Paredes, R., Reyes, N.: Solving similarity joins and range queries in metric spaces with the list of twin clusters. Journal of Discrete Algorithms (JDA) 7, 18–35 (2009)
11. Fredriksson, K.: Exploiting distance coherence to speed up range queries in metric indexes. Information Processing Letters 95(1), 287–292 (2005)
12. Uhlmann, J.: Satisfying general proximity/similarity queries with metric trees. Information Processing Letters, 175–179 (1991)
13. Chávez, E., Navarro, G.: Probabilistic proximity search: Fighting the curse of dimensionality in metric spaces. Information Processing Letters 85, 39–46 (2003)
14. Fredriksson, K.: Engineering efficient metric indexes. Pattern Recognition Letters (PRL) 28(1), 75–84 (2007)

Evaluation of Different Metrics for Shape Based Image Retrieval Using a New Contour Points Descriptor

María Teresa García-Ordás, Enrique Alegre,
Oscar García-Olalla, and Diego García-Ordás

University of León, León, Spain
{mgaro,enrique.alegre,ogaro,dgaro}@unileon.es
http://pitia.unileon.es/varp

Abstract. In this paper, an image shape retrieval method was evaluated using Euclidean, Intersect, Hamming and Cityblock distances and different kinds of k-nearest neighbours classifiers such as the original kNN, mean distance kNN and Weighted kNN. Shapes were described using a new method based on the description of the contour points, CPDH36R, obtaining better results than with the original CPDH shape descriptor. The efficiency in the retrieval was tested using Kimia99, Kimia25, MPEG7 and MPEG2 datasets obtaining an 84% of success rate in Kimia25, 94% in Kimia99, 91% in MPEG2 and 82% in MPEG7 datasets using our CPDH36R method, cityblock distance and original kNN against the 68%, 91%, 74% and 59% respectively obtained using the original CPDH. The greatest difference between the original method and our proposal can be seen clearly using MPEG2 dataset. Another advantage of our retrieval method, apart from the success rate, is the computational cost which is clearly better than the one achieved with the original Earth Mover Distance classifier used in the CPDH original method.

Keywords: image retrieval, shape description, kNN, contour.

1 Introduction

Image Retrieval is a technique that consists on searching and retrieving images from an image dataset. More and more images are stored on the internet everyday, so it is necessary to develop new image retrieval methods that ensure high accuracy dealing with million of images. This process can be divided into two well defined steps: The image description and the retrieval method.

In the last few years, many research groups have been working on new image description and retrieval methods based on different features such as texture, colour and shape. In [1], Zakariya et al. proposed a method for images retrieval that combines features based on texture color and shape with variable weights. For the retrieval process they used a simple k-nearest neighbours, taking into account different values of k. Zhang et al. proposed in [2] a retrieval system based on shape and texture watermarks which shows the advantages of combining different features in datasets with colour images.

However, there are multiple datasets based only in the objects contour. For that reason, it is important to find shape descriptors that achieve good performance by themselves. In [3], Ayyalasomayajula et al. proposed a method based on DCT (discrete cosine transform) for occluded image retrieval. Also, several methods based on DCT were proposed in the last few years for shape based image retrieval [4,5].

Other shape descriptor methods have been evaluated with promising results such as ICA Zernike Moment and top 80 matches as retrieval method [6], wavelet based shape features with L1 Norm Distance Function [7] or global feature space representation of contours [8].

Methods based on nearest neighbours are highly used in image retrieval systems. In [9], Zheng et al. proposed an improvement to optimize the kNN method that decrease the search of the nearest neighbour in more than a 98% retaining the same accuracy. In 2009, Zhang et al. proposed a method for image retrieval called "bayesian information fusion" based on nearest neighbours combined with an estimate likelihood [10]. Weighted kNN is a really interesting algorithm widely used by many researchers. In [11] a kNN was developed based on the kernel difference weighted. Recently, Gu et al. proposed in [12] a kNN variant based on a semi-supervised weighted Mahalanobis distance metric.

Our descriptor, CPDH36R, is based on the one developed by Shu et al. [13] which is a shape contour descriptor for shape matching and retrieval. They used the EMD (Earth Movers Distance) metric as matching method but we have selected another matching method, taking into account the work developed by Graham and his group [14], kNN variants and 4 different distance metrics, resulting in a save of time in the retrieval process.

The rest of the paper is organised as follows. In section 2, CPDH original method and our contribution are described. Used classifiers are described in section 3. Datasets and results are shown in section 4 and finally, in section 5, our conclusions are shown.

2 Methods

2.1 CPDH Method

The original method, Contour Points Distribution Histogram (CPDH), was proposed by Shu et al. [13]. The first step is to obtain the points representing the contour of the image under polar coordinates. The contour points that are used to describe the image, can be obtained by different methods. In this case, they used the standard Canny operator to detect the object boundary. See [15]. Once the points are extracted, the centroid is set as the origin. After that, they construct the minimum circumscribed circumference and the region defined by that circumference is divided into several bins using some concentric circumferences and equal interval angles. The final step is to construct the CPDH descriptor taking into account the number of points belonging to each division, and the angle and radius of each division.

2.2 CPDH36R Method

In our case, we used the method proposed by Belongie et al. [15], obtaining a contour like the one shown in figure 1.

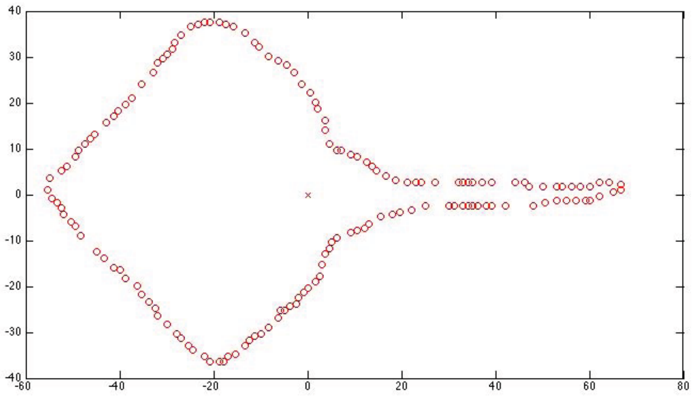


Fig. 1. Contour points extraction using Belongie method

This contour can be represented as a points collection as

$$P = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\} \\ /(x_i, y_i) \in \mathbb{R}^2 \tag{1}$$

where m denotes the number of points on the contour. After that, the centroid of the previous points is obtained using the expression 2 in order to obtain the minimum shape circumscribed circumference.

$$(x_c, y_c) = \frac{\sum_{i=1}^{m}(x_i, y_i)}{m} \tag{2}$$

The centroid is set as the origin and the points collection, P, is translated into polar coordinates using ec. 3

$$P = \{(\rho_1, \alpha_1), (\rho_2, \alpha_2), ..., (\rho_m, \alpha_m)\} \in \mathbb{R}^2 \tag{3}$$

where

$$\rho_i = \sqrt{(x_i - x_c)^2 + (y_i - y_c)^2} \tag{4}$$

and

$$\alpha_i = \begin{cases} \arctan(\frac{(y_i-y_c)}{(x_i-x_c)} & if x > 0,\ y \geq 0 \\ \arctan(\frac{(y_i-y_c)}{(x_i-x_c)} + 2\pi & if x > 0,\ y < 0 \\ \arctan(\frac{(y_i-y_c)}{(x_i-x_c)} + \pi & if x < 0 \\ \arctan(\frac{\pi}{2}) & if x = 0,\ y > 0 \\ \arctan(\frac{3\pi}{2}) & if x = 0,\ y < 0 \end{cases} \tag{5}$$

So the circumscribed circumference has as centre the centroid C (x_c, y_c) and radius $\rho_{max} = max(\rho_i), i = 1, 2, ..., m$.

Then, the circle defined by the circumscribed circumference is divided into several bins partitioning its area into $u \times v$ bins being u the parts in which ρmax is divided and v the sectors in which the circle is divided. The result of this process is shown in figure 2.

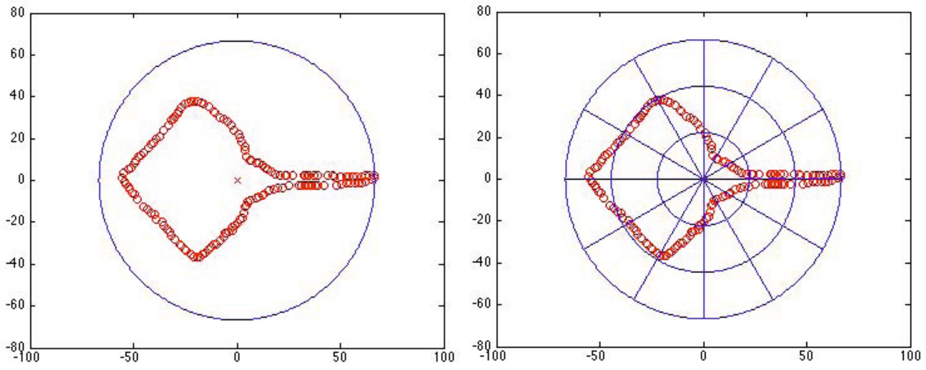


Fig. 2. Circumference circunscribed to the image contour (left) and circle partition in 36 bins (v=12, u=3) (Right)

After these previous steps, we can build the descriptor by counting the number of points belonging to each bin. The original CPDH is composed by a triplet for each bin $H_i = (\rho_i, \theta_i, n_i)$ where ρ_i denotes the radius of the concentric circumferences, θ_i denotes the angle space and n_i denotes the number of points located in the bin r_i.

The authors of CPDH, used the EMD classifier in order to make their descriptor invariant to rotation. The main disadvantage of EMD is its slowness, thus one of our motivations in developing Contour Points Distribution Histogram Radius using 36 features (CPDH36R) was the necessity of making a new descriptor containing just the most relevant features, and for this reason obtaining even better hit rates than with the base-line method but being much more efficient in terms of computational cost.

The proposed CPDH36R descriptor uses the first and third element of each of the previous triplets, that are the radius and number of points in the bin

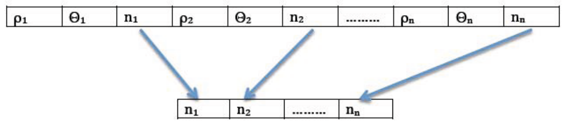


Fig. 3. First step to construct CPDH36R: 36SEQ extraction

respectively, obtaining the 36SEQ shown in figure 3. Radius are normalized following equation 6

$$r_p = \frac{\rho_{1..u}}{\rho_{max}} \tag{6}$$

Where ρ_{max} is the biggest radius and $p \in [1..36]$. CPDH36R is created multiplying the number of points of each bin (figure 3) by its corresponding normalized radius (first element of the triplet), that contains just the most relevant features for kNN classification given that the second element is constant to all the images because the number of circumference divisions is always the same. See figure 4. As shown in section 3, this proposal combined with kNN outperforms the CPDH original descriptor with EMD.

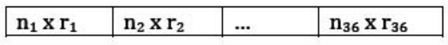


Fig. 4. CPDH36R constructed by multiplying the number of points of each bin by its corresponding normalized radius

3 kNN Classifiers

Three different kNN classifiers were used. Original kNN, mean distance kNN in which the average distance between the query and the samples is calculated and the class is assigned taking in account this distance, and finally weighted kNN in which we want to weigh the nearest neighbours taking in account its assigned weight.

3.1 Mean Distance kNN

The main idea of using mean distance kNN is to assign the query sample to the class whose mean distance to the query sample is smaller, instead of assign it to the most represented class. In figure 5, we can see that although the most represented class is the X class, four samples against three, the query sample belongs to '−' class because the mean distance is smaller.

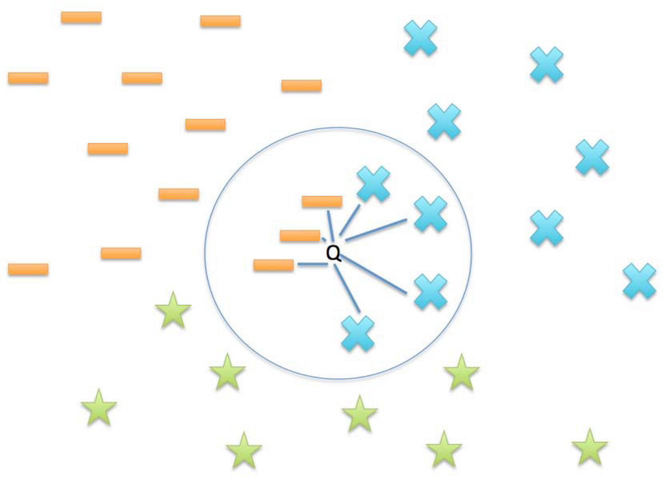


Fig. 5. kNN mean distance example taking k=7

3.2 Weighted kNN

In weighted kNN, a weight is assigned to each sample. This weight w_i is the difference between the farthest sample from the query and the proper distance from the query sample. See equation 7

$$w_i = |d_{max} - d_i| \tag{7}$$

Each class is weighted adding the weight of each sample of the class. Finally, the query sample belongs to the heavier class.

4 Experiments and Results

4.1 Datasets

Four datasets were used to test our descriptor. Kimia25 and Kimia99 datasets, the original MPEG7 and a subset of MPEG7 dataset composed of 400 images from MPEG7 called MPEG2. Kimia25 is composed of 6 classes. Each of them contains four shapes except the "hand class" which is composed of 5 samples. Kimia99 is composed of 9 classes, each one containing 11 shapes. Kimia dataset with 25 and 99 images are shown in figure 6. MPEG7 contains 70 classes with 20 samples for each of them. MPEG2 is an MPEG7 subset composed for 20 of the classes of MPEG7 dataset. Some examples of the shapes contained in MPEG7 and MPEG2 datasets are shown in figure 7.

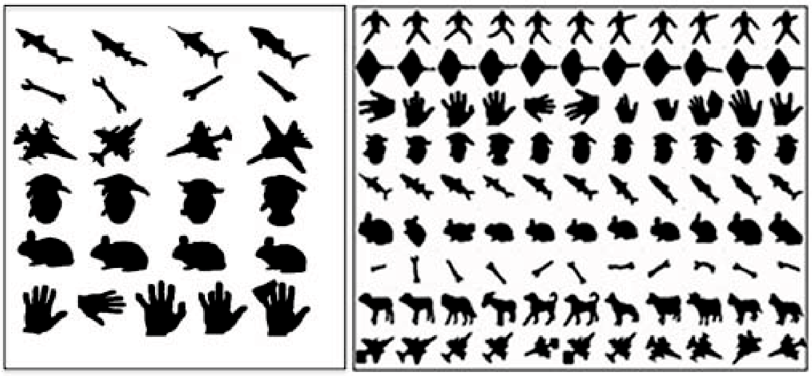


Fig. 6. Kimia dataset composed by 25 samples divided in 6 classes on the left and Kimia dataset composed by 99 samples divided in 11 classes on the right

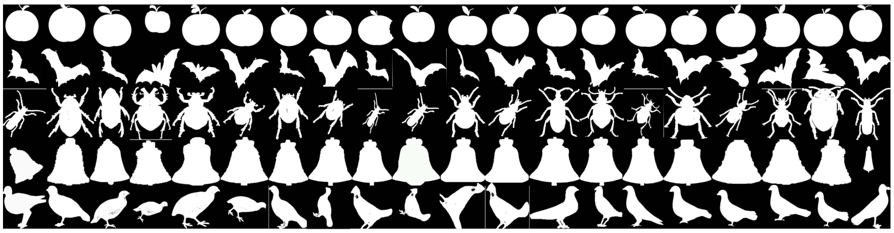


Fig. 7. Example of 4 classes with 20 samples of MPEG2 and MPEG7 datasets

4.2 Results

Our proposal was classified using kNN, kNN Mean and Weigthed kNN with euclidean, intersect, hamming and cityblock distances and they were tested on 4 datasets: kimia25, kimia99, MPEG7 and MPEG2 datasets. Classification results for each dataset using the original kNN with k=1 are shown in tables 1 and 2. Using Kimia25 is the only case in which the best result was achieved using EMD because of the small number of samples, (see table 1) obtaining a 90% of succes rate against our 84%. In table 1, we can see that results are better testing the descriptors against kimia99 dataset than against kimia25 because of the small number of samples and in most cases they are slightly better with our proposal CPDH36R than with the original CPDH except when using Euclidean distance. In table 2 we can see the efficiency of our retrieval system. Using MPEG7 dataset, once again, our proposal CPDH36R outperforms the other one with all the distances except with Hamming. Using MPEG2 dataset our method outperforms the other one classifying with all the distances and achieving a 91.25% of success rate using cityblock distance. Although on simpler dataset as Kimia25, the original CPDH with EMD offered better results, in the more

Table 1. Kimia25 and Kimia99 classification using CPDH original and CPDH36R method with k=1

	Kimia25		Kimia99	
Classifier	CPDH	CPDH36R	CPDH	CPDH36R
kNN Euclidean	76%	72%	**96.96%**	91.91%
kNN Intersect	60%	80%	83.83%	93.93%
kNN Hamming	72%	76%	91.91%	94.94%
kNN Cityblock	68%	**84%**	91.91%	**94.94%**
EMD	**90%**	-	86.41%	-

Table 2. MPEG7 and MPEG2 classification using CPDH original and CPDH36R method with k=1

	MPEG7		MPEG2	
Classifier	CPDH	CPDH36R	CPDH	CPDH36R
kNN Euclidean	**64.85%**	81.29%	78.25%	90.25%
kNN Intersect	50.36%	82.07%	67.50%	91.25%
kNN Hamming	66.71%	66.29%	**85.00%**	85.00%
kNN Cityblock	59.14%	**82.14%**	74.75%	**91.25%**
EMD	-	-	44.25%	-

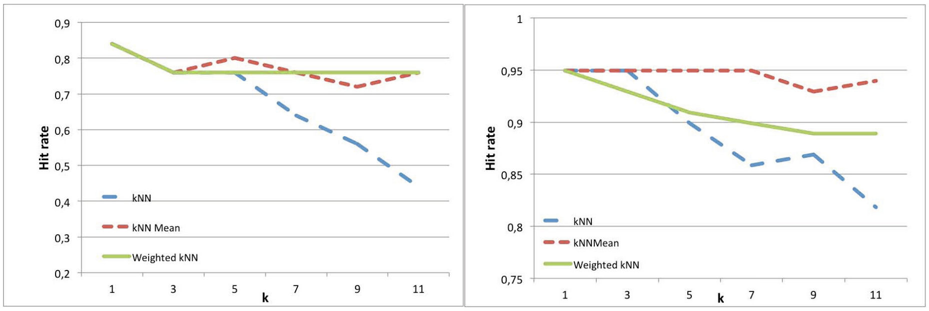


Fig. 8. Hit rate of the different kNN methods applied on the CPDH36R descriptor using Kimia25 dataset (left) and Kimia99 dataset (right)

complex dataset, CPDH36R has obtained a significant improvement of the hit rate against the original method with either EMD or kNN.

In figures 8 and 9 we can see the results with the other two classifiers, kNN mean distance and weighted kNN for different values of k and Cityblock distance. As we can see in all the figures, using original kNN, the retrieval hit rate is decreasing for higher values of k. This problem is solved using kNN mean distance or weighted kNN in which the hit rate variations is less than using the original kNN. In figure 8, using Kimia25 dataset, we can see that the original kNN performance is really bad for higher values of k, obtaining more stable results using kNN mean distance or weighted kNN. In the rest of the datasets, 8, 9, it is shown that the most stable results are achieved using kNN mean distance.

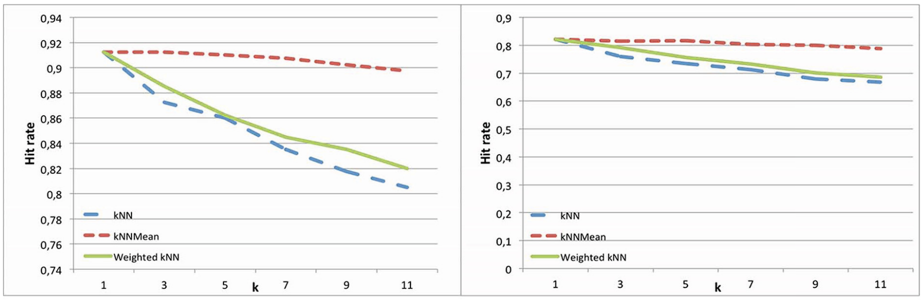


Fig. 9. Hit rate of the different kNN methods applied on the CPDH36R descriptor using MPEG2 (left) and MPEG7 (right) datasets

5 Conclusions

In this paper, a new method was proposed in order to retrieve image shapes. The original method, CPDH descriptor, was classified using EMD distance, a very slow and time consuming method. Our descriptor can be classified using kNN with classical metrics such as Euclidean, Intersect, Hamming and Cityblock distances obtaining results in a few seconds while with EMD and these same datasets it takes days. The tests carried out using the original CPDH method using the classical metrics offered worse results than our proposal CHPD36R. Our proposal was tested using 4 datasets: kimia25, kimia99, MPEG7 and MPEG20 obtaining really good results achieving the highest differences in the more challenging dataset, the MPEG7.

Acknowledgment. This work has been supported by grant DPI2012-36166 and via the pre-doctoral FPU fellowship program from the Spanish government.

References

1. Zakariya, S.M., Ali, R., Ahmad, N.: Combining visual features of an image at different precision value of unsupervised content based image retrieval. In: 2010 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), pp. 1–4 (2010)
2. Zhang, H., Chen, H., Yu, F.-X., Lu, Z.-M.: Color image retrieval system based on shape and texture watermarks. In: 2012 Second International Conference on Instrumentation, Measurement, Computer, Communication and Control (IMCCC), pp. 573–576 (2012)

3. Ayyalasomayajula, P., Grassi, S., Farine, P.: Retrieval of occluded images using dct phase and region merging. In: 2012 19th IEEE International Conference on Image Processing (ICIP), pp. 2441–2444 (2012)
4. He, D., Gu, Z., Cercone, N.: Efficient image retrieval in dct domain by hypothesis testing. In: 2009 16th IEEE International Conference on Image Processing (ICIP), pp. 225–228 (2009)
5. Seyedin, S., Fauzi, M., Anuar, F.: 1-d dct shape feature for image retrieval. In: 2009 IEEE International Conference on Signal and Image Processing Applications (ICSIPA), pp. 431–436 (2009)
6. Mei, Y., Androutsos, D.: Robust affine invariant shape image retrieval using the ica zernike moment shape descriptor. In: 2009 16th IEEE International Conference on Image Processing (ICIP), pp. 1065–1068 (2009)
7. Jian, M., Xu, L.: Trademark image retrieval using wavelet-based shape features. In: International Symposium on Intelligent Information Technology Application Workshops, IITAW 2008, pp. 496–500 (2008)
8. Khalid, S.: Robust shape matching using global feature space representation of contours. In: 2012 International Conference on Computing, Networking and Communications (ICNC), pp. 724–728 (2012)
9. Zheng, W.-M., Lu, Z.-M., Burkhardt, H.: Fast progressive image retrieval schemes based on updating enhanced equal-average equal-variance k nearest neighbour search in vector quantised feature database. In: 2007 6th International Conference on Information, Communications Signal Processing, pp. 1–5 (2007)
10. Zhang, R., Guan, L.: Multimodal image retrieval via bayesian information fusion. In: IEEE International Conference on Multimedia and Expo, ICME 2009, pp. 830–833 (2009)
11. Zuo, W., Lu, W., Wang, K.Q., Zhang, H.: Diagnosis of cardiac arrhythmia using kernel difference weighted knn classifier. In: Computers in Cardiology, pp. 253–256 (2008)
12. Gu, F., Liu, D., Wang, X.: Semi-supervised weighted distance metric learning for knn classification. In: 2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering (CMCE), vol. 6, pp. 406–409 (2010)
13. Shu, X., Wu, X.-J.: A novel contour descriptor for 2d shape matching and its application to image retrieval. Image and Vision Computing 29(4), 286–294 (2011)
14. McNeill, G., Vijayakumar, S.: Hierarchical procrustes matching for shape retrieval. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, vol. 1, pp. 885–894 (2006)
15. Belongie, S., Malik, J., Puzicha, J.: Shape matching and object recognition using shape contexts. IEEE Transactions on Pattern Analysis and Machine Intelligence 24, 509–522 (2001)

Evaluation of LBP Variants Using Several Metrics and kNN Classifiers

Oscar García-Olalla, Enrique Alegre, María Teresa García-Ordás, and Laura Fernández-Robles

University of León, León, Spain
{ogaro,enrique.alegre,mgaro,l.fernandez}@unileon.es
http://pitia.unileon.es/VARP

Abstract. In this paper, we demonstrate that the Adaptive Local Binary Pattern with oriented Standard deviation (ALBPS) method outperforms the original local binary pattern (LBP) as well as some of its most recent variants: Adaptive Local Binary Pattern (ALBP), Complete Local Binary Pattern (CLBP) and Local Binary Pattern Variance (LBPV). All the descriptors have been tested using two different dataset, KTH-TIPS 2a, a challenging multiclass dataset for material recognition and a binary sperm dataset for vitality classification. Three variants of the non parametric method of nearest neighbours combined with four metric distances have been used in the retrieval step in order to draw a more decisive conclusion. Best results were achieved when describing the images with ALBPS in both datasets. In regard to the KTH-TIPS 2a, the best performance is obtained using the weighted kNN with a 61.47% of hit rate using ALBPS and Chi Square distance, outperforming the ALBP in 1,07% and the original LBP in 6,76%. In relation to the binary sperm dataset, the best result was obtained with ALBPS and a kNN classifier (k=9), reaching a 72.66% of hit rate using the Chi Square metric, outperforming the original LBP in 22,47% and the ALBP in 1,22%. In the latter case, the weighted kNN did not improve the results achieved using just kNN. Taking this results into account, we can determine that ALBPS has more discriminant power for image retrieval than the rest of the tested LBP variants in different image contexts.

Keywords: ALBPS, LBP, kNN, weighted kNN, image retrieval.

1 Introduction

Texture analysis and retrieval is a challenging open problem in the computer vision field. For this reason, nowadays many different methods and algorithms are being proposed in order to deal with it. It is considered an important task to solve because multiple fields require texture analysis to classify images. For example, there are a lot of tasks that can be automatized using texture analysis in fabric environments. Behravan et al [1], proposed a hybrid scheme for on-line detection and classification of textural fabric defects. Paniagua-Paniagua et al developed a method to outperform the quality detection in the cork industry [2]. Another field of application is the sperm assessment such as in porcine industry. Few works deal with making this process automatic. In general, computer-based systems designed for semen analysis tasks should reliably segment the sperm heads [3], extract the features that better describe them and classify those patterns in order to estimate how many damaged spermatozoa are present in the sample. It is possible to find a number of works using texture or shape analysis to classify the spermatozoon heads as intact or damaged. Those approaches evaluate the acrosome integrity in different ways, sometimes using complex descriptors such as the Curvelet transform [4] and, other times, evaluating a broad range of texture and based-moments descriptors [5]. However there are few works that try to classify the vitality of boar sperm. In recent works, Alegre et al. [6,7] obtained a hit rate of 76.80% using texture descriptors on images captured at 100×.

Retrieval step is as important as the description of the image. One of the most widely used classifiers is k Nearest Neighbours (kNN). There are a lot of works based on improvements of the classic kNN. In 2011, a multi-label text categorization based on fuzzy similarity and k Nearest Neighbors was carried out by Jung et al [8]. Moreover, metric distances are crucial components in matching processes. Several distances have been used in the past years in order to get the higher performance of the classifier. Weinberger et al. [9] proposed Mahanalobis learned distance to simulate the Support vector machine behaviour using nearest neighbours.

Recently, a lot of Local Binary Pattern (LBP) [10] variants have been proposed in order to make this method adaptive as shown in the next relevant approaches. In [11], Li et al introduced a new LBP method invariant to scale and rotation using adaptive textons. García-Olalla et al [12] proposed a new LBP method combining Adaptive Local Binary Pattern with the standard deviation of the image along different orientations (ALBPS), achieving a 85.63% of hit rate classifying the vitality of boar sperm. However, this method has just been evaluated with Support Vector machines for a binary classification task, leaving a gap about its performance in other kind of scenarios. Furthermore, no more recent LBP variants were tested to describe the spermatozoa heads in order to compare their results.

In this paper we demonstrate the robustness of ALBPS evaluating its performance using two texture datasets with different goals against LBP and some state-of-the-art LBP variants (ALBP, CLBP, LBPV). The first dataset, KTH-tips 2a, is composed of images belonging to 11 classes for materials recognition whereas the sperm dataset used in [12], is focused on a concrete binary classification problem where classes are very close to each other. The good behaviour of ALBPS using the parametric classifier SVM was showed in [12]. On the contrary, in this work several modifications of the most acknowledged non parametric classifier, Nearest Neighbours, have been tested with several distance metrics.

The rest of the paper is organised as follows. In section 2, the method used and the retrieval system is described. Experiments and datasets are show in section 3 and finally, in section 4, conclusions are discussed.

2 Methodology

2.1 Texture Description

Local Binary Pattern. Local Binary Pattern (LBP) [13] is an algorithm used to describe the texture of grayscale images extracting their local spatial structure. Given a pixel, a pattern code is computed by comparing this pixel with the value of its neighbours:

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c)2^p \ , \ s(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases} \tag{1}$$

where g_c is the value of the central pixel, g_p is the value of its neighbour p, P are the number of neighbours and R is the radius of the neighbourhood.

After LBP is obtained for each pixel, in this work, a histogram with just $P + 2$ bins is built in order to describe the whole image instead of the typical 256 element histogram. This is due to the use of the uniform LBP which allows just $P + 2$ values for the histogram.(see equation 2)

$$LBP_{P,R}^{riu2} = \begin{cases} \sum_{p=0}^{P-1} s(g_p - g_c)2^p \text{ if } & U(LBP_{P,R}) \leq 2 \\ P + 1 & \text{otherwise} \end{cases} \tag{2}$$

where $U(LBP_{P,R})$ is defined as the number of spatial transitions in the pattern.

Adaptive Local Binary Pattern. Guo et al. [14] proposed an adaptive descriptor based on Local Binary Pattern motivated by the lack of information about the orientation in the classical Local Binary Pattern algorithm. The method that they presented takes into account the mean and the standard deviation along different orientations over all the pixels in order to make the matching more robust against local spatial structure changes. To minimize the variations of the mean and standard deviation of the directional differences, Guo et al. proposed a scheme to minimize the directional difference $|g_c - w_p * g_p|$ adding to equation 1 an extra parameter w.

The objective function is defined as follows:

$$w_p = arg_w \min \left\{ \sum_{i=1}^{N} \sum_{j=1}^{M} |g_c(i,j) - w \cdot g_p(i,j)|^2 \right\} \tag{3}$$

where w_p is the weight element used to minimize the directional difference and N and M are the number of rows and columns in the image respectively. Each weight w_p is estimated along one orientation $2p\pi/P$ for the whole image.

Therefore, adding this extra parameter w, the ALBP method is defined as:

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - w_p \cdot g_c)2^p \ , \ s(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases} \tag{4}$$

Adaptive Local Binary Pattern with Standard deviation (ALBPS). The main disadvantage of the previous proposal is the need of including statistical information in the matching step. Adding this information to the descriptor instead of to the classifier allows us to combine the ALBP approach with all kind of classifiers. In [12], García-Olalla et al proposed a combination of ALBP with standard deviation features called ALBPS. The standard deviation vector σ is obtained using the equation 5.

$$\sigma_p = \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{M}(g_c(i,j) - g_p(i,j) - \mu_p)^2/(M \cdot N)} \tag{5}$$

where N and M are the numbers of rows and columns respectively, $g_c(i,j)$ is the center pixel at position (i,j), $g_p(i,j)$ is the neighbourhood of $g_c(i,j)$ lying along orientation $2p\pi/P$ with radius R and μ_p the oriented mean obtained by:

$$\mu_p = \sum_{i=1}^{N}\sum_{j=1}^{M}|g_c(i,j) - g_p(i,j)|/(M \cdot N) \tag{6}$$

ALBPS descriptor is formed by concatenating the $P+2$ bins histogram values of the uniform LBP approach together with the P-dimensional standard deviation vector, yielding a descriptor of $2P+2$ features being P the size of the neighbourhood.

Local Binary Pattern Variance. LBPV [15], is a proposal by Guo et al which consists of a combination of LBP and a contrast distribution method. First, the uniform LBP is calculated on the whole image. Then, the variance of the image is used as an adaptive weight to adjust the contribution of the LBP code in the histogram calculation. The LBPV histogram is computed as:

$$LBPV_{P,R}(k) = \sum_{i=1}^{N}\sum_{j=1}^{M} w(LBP_{P,R}(i,j),k), k \in [0,K] \tag{7}$$

where k is each bin of the histogram, K the maximum value of LBP and w is defined as:

$$w(LBP_{P,R}(i,j),k) = \begin{cases} VAR_{P,R}(i,j), & LBP_{P,R}(i,j) = k \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

where $VAR_{P,R}$ is the variance of the neighbourhood.

$$VAR_{P,R} = \frac{1}{P}\sum_{p=0}^{P-1}(g_p - u)^2 \tag{9}$$

where u is the mean over the different neighbours: $u = 1/P \sum_{p=0}^{P-1} g_p$.

Completed Local Binary Pattern. In [16], Guo et al proposed another method called CLBP to generalize and complete the classical LBP. In that method, a local region is represented by its center pixel and a local difference sign - magnitude transform called LDSMT. LDSMT decomposes the image local structure into two complementary components: the difference signs and the difference magnitudes. In order to code both components, they proposed two operators, CLBP-Sign (CLBP_S) and CLBP-Magnitude (CLBP_M). Since all of them are in binary format, they can be combined to form the final CLBP histogram. CLBP_S is equal to the classical LBP histogram defined in 1, and CLBP_M is defined in the equation 10.

$$CLBP\_M_{P,R} = \sum_{p=0}^{P-1} t(m_p, c)2^p \ , \ t(x,c) = \begin{cases} 1 \text{ if } x \geq c \\ 0 \text{ if } x < c \end{cases} \tag{10}$$

where c is a threshold determined adaptively.

Finally, CLBP is obtained either concatenating or merging both operators. In our case, we have chosen the concatenation due to its higher similarity with ALBPS method.

2.2 Nearest Neighbours Variants

Appart from the standard kNN, two variants of the original kNN have been used in this work, mean distance kNN and weighted kNN.

Mean Distance kNN. The main idea of this metric is to assign the query sample to the class whose k nearest neighbours are closer to it. To do that, the k nearest neighbours are selected obtaining the mean distance of each classes. Finally, we assign the query sample to the class of the set with lower mean distance.

Weighted kNN. In weighted kNN, each sample has a weight w_p depending on the query selected which is the difference between the farthest sample from the query and the proper distance from the query sample. See equation 11.

$$w_p = |d_{max} - d_p| \tag{11}$$

Each class is weighted adding the weight of each sample of the class. Finally, the query sample belongs to the heaviest class.

3 Experiments

3.1 Datasets

ALBPS has been evaluated using two different datasets. A binary and gray scale sperm head dataset used to classify them as dead or alive and the multi class KTH-tips 2a texture dataset for material recognition [17].

KTH-tips 2a Dataset. This dataset is composed by several images for material categorization [17]. It contains 11 materials (lettuce, brown bread, white bread, aluminium, corduroy, cork, cotton, cracker, linen, wood and wool) with a number of images for four different samples from each material. All the samples were taken at 3 poses, 4 different illumination conditions and 9 scales. All this variations make a very challenging dataset. In figure 1(a), we can see some examples of some textures under different conditions.

Boar Spermatozoa Dataset. This set of images has been captured in CENTROTEC, an Artificial Insemination Center that is a University of Leon spin-off. The sperm was obtained from boars of three different races: Piyorker, Large White and Landrace. The dataset contains 351 dead and 450 alive spermatozoa heads in gray scale.

A 3×3 texture range filter had been applied on the whole dataset in order to reduce the non-informative areas and therefore facilitate the subsequent dataset description and classification. Figure 1(b) shows examples of dead and alive spermatozoon heads.

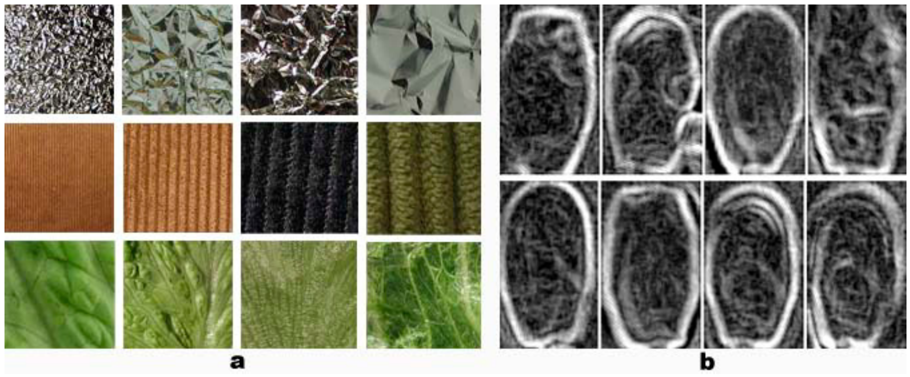


Fig. 1. (a) Examples of the KTH-tips 2a dataset. In the first row, different images of the aluminium texture. In the second, corduroy images and in the third, lettuce leaves under different scales and illumination. (b) Sperm dataset. The first row shows dead spermatozoon heads and the second row alive ones.

3.2 Experimental Setup

KTH-Tips 2a Dataset. The experimental setup used for the KTH-tips 2a dataset is the standard protocol developed by Caputo et al and used in several works [17,18]. Basically, it takes one of the samples of each material for test and the rest for training, which conforms a more challenging setup than using random images for test and training. In this work, we carried out four classifications using this method to increase its robustness, one classification for each sample in the test class. The final hit rate is obtained as the mean of the hit rates in each iteration. We show the comparison of the three k-NN approaches shown previously (standard kNN, mean distance kNN and Weighted kNN).

Boar Spermatozoa Dataset. Again the three k-NN approaches have been used to classify the spermatozoa images as dead or alive. We have used the one versus all paradigm which consists of classifying each image taking into account the rest of the images in the dataset. This process has been repeated using several values of neighbours (1, 3, 5, 7, 9 and 11).

3.3 Multiclass Dataset Results

NN Variants Evaluation. Three different classifiers based on the Nearest Neighbours method have been implemented in order to evaluate the different LBPs descriptors proposed in this paper. Four distances have been selected due to their good general behaviour: euclidean, intersect distance, Chi square and Cityblock. In the figure 2 we can observe the mean performance along different values of k of all the classifiers for each pair of LBP descriptor and distance. As we can see, the best results are obtained using the Weighted kNN for all the test except for the LBP-Chi square in which standard kNN performs better.

LBP Variants Performance with Weighted NN. Once we have determined the best classifier (Weighted KNN), a comparison between the proposed ALBPS and the rest of LBP methods evaluated was carried out. In figure 3, we can see the mean performance along different k values of the descriptors using different distances and the weighted kNN classifier. It can be appreciated that the proposed ALBPS outperforms all the others methods with all the distances except in the euclidean experiment where ALBPS and ALBP get the same results. The best performance was obtained using Chi Square and ALBPS with a 60.23% of hit rate.

Best k Neighbour Value. As it has been proven, the best result was achieved using Chi Square and ALBPS. However, this result is the mean value along different numbers of neighbours. In figure 4, we can see the performance of ALBPS for all the k values and all the distances evaluated. The results show that the best distance remains the Chi square with a hit rate of 61.47% for k=1.

In table 1, we can see the results achieved using the weighted kNN method for all the distances highlighting the best k for each method.

3.4 Binary Dataset Results

NN Variants Evaluation. The same Nearest Neighbours methods have been also evaluated in order to classify the boar sperm heads as dead or alive. The best descriptor of the multi class experiment (ALBP) was compared with the

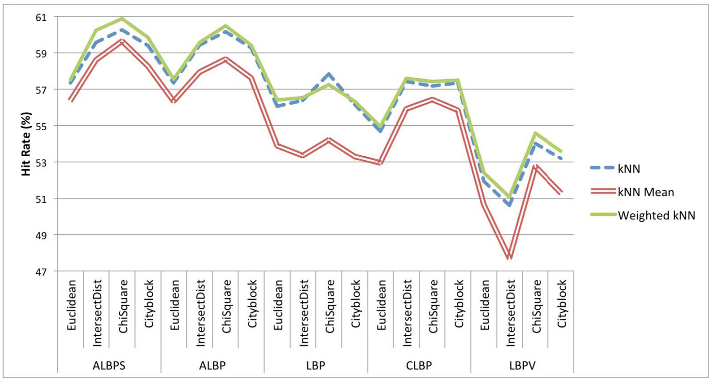


Fig. 2. Results achieved on the KTH-tips 2a dataset using different classifiers. The hit rate value corresponds with the mean hit rate along different values of k (1, 3, 5, 7, 9, 11) for all the pairs Descriptor-Distance.

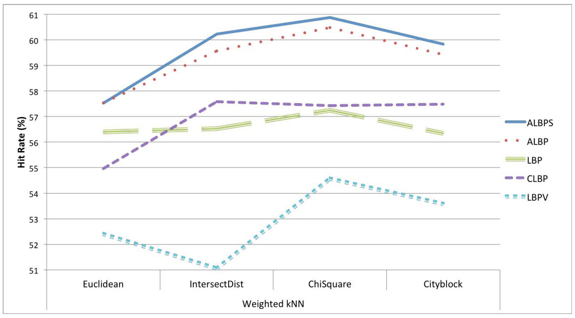


Fig. 3. Results achieved on the KTH-tips 2a dataset using different descriptors and the weighted kNN classifier. The hit rate value corresponds with the mean hit rate along different values of k (1, 3, 5, 7, 9, 11) for each distance.

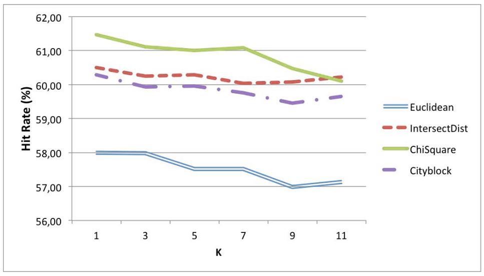


Fig. 4. Results achieved on the KTH-tips 2a dataset using weighted kNN and different values of k and distances

Table 1. Hit rate in % using weighted kNN with several distances on the KTH tips 2a dataset

Classifier method		ALBPS	ALBP	LBP	CLBP	LBPV
kNN weighted	Euclidean	58.00	58.00	56.90	55.18	52.97
	Intersect	60.50	60.06	56.88	58.31	51.47
	ChiSquare	**61.47**	60.82	57.58	58.08	54.78
	Cityblock	60.29	59.85	56.57	58.12	53.79

original LBP method and our ALBPS approach. In figure 5, we can observe the performance of different configurations of ALBP and our ALBPS proposal using three distances: Euclidean, Chisquare and Cityblock. In this experiment we have excluded the Intersect distance due to its poor performance. As in the previous section, the results show the mean performance along different values of k. As we can see, in the binary classification problem, the weighted classifier does not improve the original kNN whereas the kNN with mean distances obtains lower performance in all cases. As the simple kNN has lower computational cost than the weighted kNN, we selected the former one as the best classifier.

LBP Variants Performance with Weighted NN. The proposed ALBPS method has been compared with the ALBP method and the original LBP in order to determine its performance. In figure 6, we can see the results obtained by these descriptors using kNN and different distance metrics. ALBPS outperforms the others in all the experiments, achieving an improvement of 1.05% with respect to ALBP and a 18,80% respect the original LBP method. This figure shows that the Chi Square distance, as well as in the multi class dataset, proves to outperform the rest.

Best k Neighbour Value. In order to determine the best configuration, In figure 7 we show the performance of ALBPS for all the k values and all the distances evaluated. Results show that the best distance remains the Chi square with a hit rate of 72,66% for k=9.

In table 2, the results obtained using kNN with all the distances choosing the best k for each descriptor are numerically presented.

Table 2. Hit rate in % using kNN on the sperm dataset

Classifier method		ALBPS	ALBP	LBP
	Euclidean	71.03	71.41	43.82
kNN	ChiSquare	**72.66**	71.79	59.30
	Cityblock	72.16	71.54	43.80

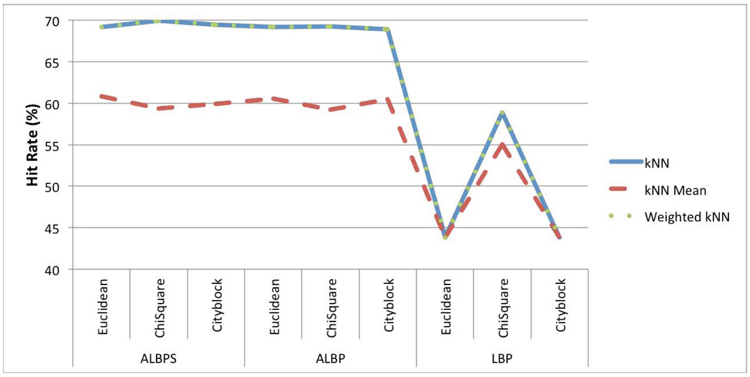


Fig. 5. Results achieved on the boar sperm dataset using different descriptors and the simple kNN classifier. The hit rate value corresponds with the mean hit rate along different values of k (1, 3, 5, 7, 9, 11) using different distances.

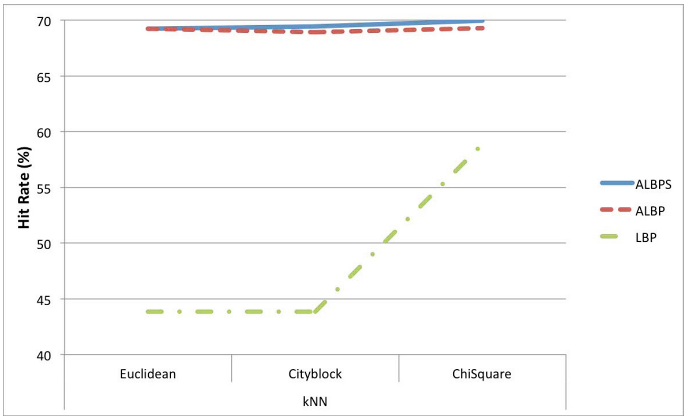


Fig. 6. Results achieved on the boar sperm dataset using ALBP, ALBPS and the original LBP. The hit rate value corresponds with the mean hit rate along different values of k (1, 3, 5, 7, 9, 11) for all the pairs Descriptor-Distance.

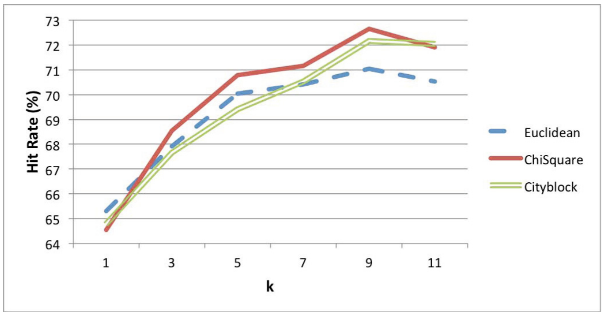


Fig. 7. Results achieved on the boar sperm dataset using kNN with different k values and distances

4 Conclusions

In this paper, we have demonstrated that the method ALBPS[12] outperforms recent LBP descriptors variants, such as ALBP, CLBP and LBPV as well as the classical LBP method for several kNN-based methods with multiple distances on two datasets: the multi class KTH-tips 2a dataset for material recognition and a binary dataset for vitality sperm classification. With regard to the multi class dataset, the results show that the best descriptor was ALBPS using a weighted kNN and the ChiSquare distance achieving a 61.47% of hit rate. Moreover, for the binary dataset, ALBPS proved to outperform the rest of the methods reaching a hit rate of 72.66%. However in the latter dataset, the weighted kNN does not show any improvement with respect to the standard kNN, being the simple kNN method with k=9 and ChiSquare distance the best classifier. Taking into account all the results obtained, we can conclude that adding standard deviation information to the feature descriptor, as proposed with ALBPS, the accuracy was incremented on both datasets with all combinations of evaluated non parametric classifiers and metrics. These results, together with the results showed in García-Olalla et al work [12] using SVM, give a positive insight on the use of ALBPS in different scenarios.

Acknowledgments. This work has been supported by grants DPI2009-08424 and DPI2012-36166 and via the pre-doctoral FPU fellowship program from the Spanish government.

References

1. Behravan, M., Boostani, R., Tajeripour, F., Azimifar, Z.: A hybrid scheme for online detection and classification of textural fabric defects. In: Second International Conference on Machine Vision, ICMV 2009, pp. 118–122 (December 2009)
2. Paniagua-Paniagua, B., Vega-Rodriguez, M.A., Bustos-Garcia, P., Gomez-Pulido, J.A., Sanchez-Perez, J.M.: Advanced texture analysis in cork quality detection. In: 2007 5th IEEE International Conference on Industrial Informatics, vol. 1, pp. 311–315 (June 2007)
3. Gonzalez-Castro, V., Alegre, E., Morala-Arguello, P., Suarez, S.: A combined and intelligent new segmentation method for boar semen based on thresholding and watershed transform. International Journal of Imaging 2, 70–80 (2009)
4. González-Castro, V., Alegre, E., García-Olalla, O., García-Ordás, D., García-Ordás, M.T., Fernández-Robles, L.: Curvelet-based texture description to classify intact and damaged boar spermatozoa (Aveiro), pp. 448–455 (2012)
5. Alegre, E., González-Castro, V., Aláiz-Rodríguez, R., García-Ordás, M.T.: Texture and moments-based classification of the acrosome integrity of boar spermatozoa images. In: Computer Methods and Programs in Biomedicine (2012)
6. Alegre, E., García-Olalla, O., González-Castro, V., Joshi, S.: Boar spermatozoa classification using longitudinal and transversal profiles (LTP) descriptor in digital images. In: Aggarwal, J.K., Barneva, R.P., Brimkov, V.E., Koroutchev, K.N., Korutcheva, E.R. (eds.) IWCIA 2011. LNCS, vol. 6636, pp. 410–419. Springer, Heidelberg (2011)

7. Alegre, E., García-Ordás, M.T., González-Castro, V., Karthikeyan, S.: Vitality assessment of boar sperm using N concentric squares resized (NCSR) texture descriptor in digital images. In: Vitrià, J., Sanches, J.M., Hernández, M. (eds.) IbPRIA 2011. LNCS, vol. 6669, pp. 540–547. Springer, Heidelberg (2011)
8. Jiang, J.-Y., Tsai, S.-C., Lee, S.-J.: Fsknn: Multi-label text categorization based on fuzzy similarity and k nearest neighbors. Expert Systems with Applications 39(3), 2813–2821 (2012)
9. Weinberger, K.Q., Blitzer, J., Saul, L.K.: Distance metric learning for large margin nearest neighbor classification. In: NIPS. MIT Press (2006)
10. Ojala, T., Pietikainen, M., Harwood, D.: Performance evaluation of texture measures with classification based on kullback discrimination of distributions. In: Proceedings of the 12th IAPR International Conference on Pattern Recognition, ICPR 1994 (1994)
11. Li, Z., Liu, G., Yang, Y., You, J.: Scale- and rotation-invariant local binary pattern using scale-adaptive texton and subuniform-based circular shift. IEEE Transactions on Image Processing 21, 2130–2140 (2012)
12. García-Olalla, O., Alegre, E., Fernández-Robles, L., García-Ordás, M.T.: Vitality assessment of boar sperm using an adaptive LBP based on oriented deviation. In: Park, J.-I., Kim, J. (eds.) ACCV Workshops 2012, Part I. LNCS, vol. 7728, pp. 61–72. Springer, Heidelberg (2013)
13. Ojala, T., Pietikainen, M., Harwood, D.: A comparative study of texture measures with classification based on featured distributions. Pattern Recognition 29(1), 51–59 (1996)
14. Guo, Z., Zhang, L., Zhang, D., Zhang, S.: Rotation invariant texture classification using adaptive lbp with directional statistical features. In: 2010 17th IEEE International Conference on Image Processing (ICIP), pp. 285–288 (September 2010)
15. Guo, Z., Zhang, L., Zhang, D.: Rotation invariant texture classification using lbp variance (lbpv) with global matching. Pattern Recognition 43(3), 706–719 (2010)
16. Guo, Z., Zhang, L., Zhang, D.: A completed modeling of local binary pattern operator for texture classification. IEEE Transactions on Image Processing 19(6), 1657–1663 (2010)
17. Caputo, B., Hayman, E., Mallikarjuna, P.: Class-specific material categorisation. In: ICCV (2005)
18. Chen, J., Shan, S., He, C., Zhao, G., Pietikainen, M., Chen, X., Gao, W.: Wld: A robust local image descriptor. PAMI (2010)

Evaluation of Jensen-Shannon Distance over Sparse Data

Richard Connor[1], Franco Alberto Cardillo[2], Robert Moss[1], and Fausto Rabitti[2]

[1] Department of Computer and Information Sciences,
University of Strathclyde, Glasgow, G1 1XH, United Kingdom
[2] ISTI (Information Science and Technology Institute)
National Research Council of Italy, Via Moruzzi 1, 56124 Pisa, Italy
{richard.connor,robert.moss}@strath.ac.uk,
{franco.alberto.cardillo,fausto.rabitti}@isti.cnr.it

Abstract. Jensen-Shannon divergence is a symmetrised, smoothed version of Küllback-Leibler. It has been shown to be the square of a proper distance metric, and has other properties which make it an excellent choice for many high-dimensional spaces in $\mathbb{R}^*$.

The metric as defined is however expensive to evaluate. In sparse spaces over many dimensions the Intrinsic Dimensionality of the metric space is typically very high, making similarity-based indexing ineffectual. Exhaustive searching over large data collections may be infeasible.

Using a property that allows the distance to be evaluated from only those dimensions which are non-zero in both arguments, and through the identification of a threshold function, we show that the cost of the function can be dramatically reduced.

1 Introduction

Jensen-Shannon divergence is the name given in [8] to a divergence function probably first identified in [10]. It is a simple derivation from Küllback-Leibler [7] yet is positive, symmetric, bounded, and well-defined in the presence of zero values.

Two authors [4,9] have independently established that one form of Jensen-Shannon divergence is the square of a proper metric. Since then the metric has attracted some more interest in both statistics and information theory, and deeper analysis e.g. [5] shows that it has some properties that, in short, should lend it to being an excellent semantic distance function in many contexts.

The fact that a form exists which is a proper metric immediately leads to the possibility of its use within metric indexing techniques. However many probabilistic spaces are high-dimensional and sparse, and typical Intrinsic Dimensionality [1] is very high: metric indexing techniques are unlikely to be effective.

In this paper we show a way of significantly reducing the cost of similarity search using Jensen-Shannon, showing how an equivalent metric can be derived which requires access only to the intersecting dimensions of the objects being compared. This allows a much more efficient evaluation, and in particular an evaluation which can be performed over inverted indices, thus also subject to parallel evaluation.

2 Definitions and Algebraic Derivations

Jensen-Shannon divergence is defined in terms of Küllback-Leibler divergence:

$$JS(v,w) = \tfrac{1}{2}KL(v,m) + \tfrac{1}{2}KL(w,m)$$

where m is the vector mean of v and w. If logs are taken to base two, then the outcome is bounded in $[0,1]$.

Some simple algebra gives some other forms of interest for the same function:

$$JS(v,w) = H(m) - \tfrac{1}{2}H(v) - \tfrac{1}{2}H(w) \tag{1}$$

where H is Shannon's entropy function. This can be evaluated as:

$$JS(v,w) = \tfrac{1}{2}\sum_i \bigl(v_i \log(v_i) + w_i \log(w_i) - (v_i + w_i)\log \tfrac{1}{2}\,(v_i + w_i)\bigr) \tag{2}$$

From this also can be derived:

$$JS(v,w) = 1 - \tfrac{1}{2}\sum_i \mathcal{F}(v_i, w_i) \tag{3}$$

for a kernel function $\mathcal{F}$ defined by

$$\mathcal{F}(x,y) = h(x) + h(y) - h(x+y)$$

where $h(x) = -x\log_2(x)$.

From this form it may be observed that the evaluation of JS can be achieved with reference only to those dimensions where v_i and w_i are *both* non-zero. A similar form to Equation 3 was given in [3] where the observation was made that this could give an efficient evaluation, but was not quantified.

2.1 Threshold Calculation

If the purpose of the distance calculation is as a part of a threshold search, the threshold requirement (using the proper metric form) is:

$$\sqrt{1 - \frac{1}{2}\sum_i \mathcal{F}(v_i, w_i)} < t$$

for threshold t. The function $\mathcal{F}$ can be seen as a similarity accumulator, reaching the value of 2 for perfect similarity, and the term $2t^2$ can be viewed as the maximum shortfall which may occur in order for the threshold t not to be exceeded.

A cost-saving strategy may be used based on this observation. At any point of the iterative calculation, if it can be determined that it is impossible for the value of $\sum_i \mathcal{F}(v_i, w_i)$ to reach the threshold of $2 - 2t^2$, then the calculation may be abandoned.

If stage k of the calculation is considered:

$$\sum_i \mathcal{F}(v_i, w_i) = \sum_{i=1..k} \mathcal{F}(v_i, w_i) + \sum_{i=k+1..n} \mathcal{F}(v_i, w_i)$$

the value of the left hand term is known, and an upper bound for the right-hand term can be found using the Jensen inequality, as $\mathcal{F}$ is a convex function:

$$\sum_{i=j..k} \mathcal{F}(v_i, w_i) \leq \mathcal{F}\left(\sum_{i=j..k} v_i, \sum_{i=j..k} w_i\right) \tag{4}$$

where the value $\sum_{i=k+1..n} v_i$ is simply the complement of $\sum_{i=1..k} v_i$. Therefore, at any stage k of the calculation, the following inequality can be tested:

$$\sum_{i=1..k} (\mathcal{F}(v_i, w_i)) + \mathcal{F}\left(1 - \sum_{i=1..k} v_i, 1 - \sum_{i=1..k} w_i\right) < 2 - 2t^2$$

and, if the outcome is true, the final distance calculation will be greater than t.

3 Evaluation

3.1 Definitions

For each test sparse vectors and inverted indices were implemented in a straightforward manner, such that no zero values are stored. Based on the above observations, five different versions of the metric over sparse vector spaces were tested[1]:

Definition 1. An algorithm based on Equation 2, accessing all dimensions of the sparse vectors being compared.

Definition 2. An algorithm based on Equation 3. The algorithm iterates through all nodes of each argument vector, but no calculation is performed if the dimension is not present in both vectors.

Definition 3. The same algorithm as Defn. 2, but at each stage the current accumulator value is checked against a calculated threshold derived from Equation 4, and the calculation is abandoned when possible.

Definition 4. The accumulation of values is performed over inverted index data structures. The calculation proceeds one dimension at a time, with a separate accumulator being maintained for each object in the set

Definition 5. Again over the inverted index structures, this time maintaing a threshold based on Equation 4; if the accumulator for any object in the set drops below the required threshold, this is set to -1 and no further calculations are performed over other dimensions of that vector.

The thresholds in Definitions 3 and 5 cause each test to return 10^{-5} of the data.

[1] All implementations are in Java; the source code is available from the authors.

3.2 Framework

For each data set tested, 10^5 separate objects were considered and each one measured against the other members of the set to perform 10^{10} calculations. All code was implemented in Java and executed on a 1.8 GHz Intel Core i7 processor with 4GB of memory; all nonessential processes and network access were disabled. All data structures fitted within the Java heap. Each test was repeated until the standard error of the mean time measured was less than 1%, with a garbage collection being called between each iteration.

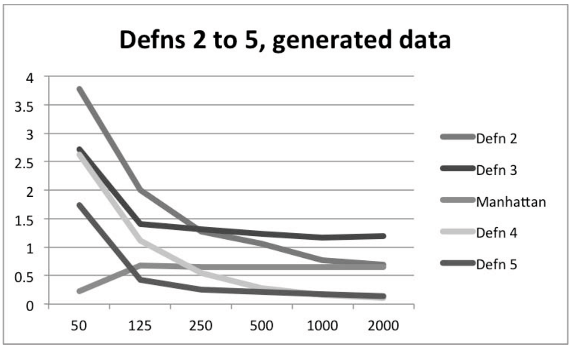


Fig. 1. Cost per calculation using the different implementations

3.3 Generated Spaces

To test the mechanisms over sparse Cartesian spaces a number of generated spaces were used. For each of these, the generator was set to populate a mean of 50 dimensions within all of those available, with the maximum number of dimensions being set between 50 (i.e. a dense space) and 2000.

3.4 Real Spaces

We used the following data sets: *colors*, taken from the colors.ascii file of the SISAP data collection; *english*, taken from the English.dic file of the SISAP collection, from which vectors are generated by the probabilistic technique given in [2]; *occs*, a file of occupations taken from census data using the same generation technique; and *MF-eh*, *MF-ht* taken from the MIR-flickr collection [6]. The key characteristics of these sets is given in Table 1.

Results. The results of applying the five techniques to the different data sets are given in Table 2. Definitions 2 to 5 over the generated data are repeated in the graph shown in Figure 1. As well as giving the costs for the five definitions, the cost of Manhattan distance over the sparse representations is also shown.

The results certainly vindicate the techniques described; it is notable that for every truly sparse data set, Jensen-Shannon evaluated by Definition 5 outperforms Manhattan distance, clearly because less data is being moved though the processor. It is equally interesting to note that, even for non-sparse data, the use of inverted indices with a threshold cutoff performs an order of magnitude better than doing per-object comparisons. While the threshold cutoff is highly effective for some data sets, it is much less so for others for reasons we do not yet fully understand.

Table 1. Data set characteristics

	Data set				
Implementation	colors	english	occs	MF-eh	MF-ht
Total Dimensions	78	483	865	150	43
Mean non-zero dimensions	40.1	16.0	38.3	142.9	43.0
IDIM	5.79	87.6	74.7	5.57	2.37

Table 2. Time (μs) per distance calculation

	Generated Sets						Real data sets				
Implementation	50	125	250	500	1000	2000	colors	english	occs	MF-eh	MF-ht
Defn. 1	3.926	5.325	5.641	6.003	5.952	6.099	3.556	1.838	4.229	11.638	3.391
Defn. 2	3.775	1.995	1.276	1.061	0.772	0.692	2.416	0.410	1.180	10.558	3.335
Defn. 3	2.711	1.404	1.307	1.232	1.162	1.195	1.396	1.139	1.157	2.158	1.934
Manhattan	0.222	0.675	0.652	0.655	0.651	0.649	0.256	0.206	0.429	0.646	0.197
Defn. 4	2.622	1.109	0.550	0.286	0.159	0.103	1.631	0.185	0.603	7.420	2.275
Defn. 5	1.739	0.422	0.253	0.208	0.178	0.140	0.205	0.152	0.199	1.127	0.260

4 Conclusions and Further Work

In this paper we have shown how two algebraic deductions from the Jensen-Shannon distance can be used to give a very significant cost saving in its evaluation. The inverted index implementation is also perfectly suited to parallelisation, and in particular can use parallel threads on a graphics accelerator rather than specialist hardware. In combination, we believe this metric is made much more accessible.

We have not yet fully investigated the threshold cutoff. In particular, the results shown here evaluate the threshold at every stage of the algorithms, which applies a significant cost for little benefit at the early stages. Different collections behave in different ways, but it should be easily possible to determine a better strategy when the calculation is made only when there is a significant chance of aborting the calculation.

Acknowledgements. The authors would like to acknowledge funding from the UK Research Councils for the related project EP/G012407/1. Richard Connor was supported by the National Research Council of Italy (CNR) for a Short-term Mobility Fellowship (STM), which funded a stay at ISTI-CNR in Pisa where much of this work was done.

References

1. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. 33(3), 273–321 (2001)
2. Connor, R.C.H., Simeoni, F., Iakovos, M., Moss, R.: Towards a universal information distance for structured data. In: Ferro, A. (ed.) SISAP, pp. 69–77. ACM (2011)
3. Dagan, I., Lee, L., Pereira, F.C.N.: Similarity-based models of word cooccurrence probabilities. Mach. Learn. 34(1-3), 43–69 (1999)
4. Endres, D.M., Schindelin, J.E.: A new metric for probability distributions. IEEE Transactions on Information Theory 49(7), 1858–1860 (2003)
5. Fuglede, B., Topsoe, F.: Jensen-shannon divergence and hilbert space embedding. In: Proceedings of International Symposium on Information Theory, ISIT 2004, p. 31 (2004)
6. Huiskes, M.J., Lew, M.S.: The mir flickr retrieval evaluation. In: MIR 2008: Proceedings of the 2008 ACM International Conference on Multimedia Information Retrieval. ACM, New York (2008)
7. Küllback, S., Leibler, R.A.: On information and sufficiency. Ann. Math. Statist. 22(1), 79–86 (1951)
8. Lin, J.: Divergence measures based on the shannon entropy. IEEE Transactions on Information Theory 37(1), 145–151 (1991)
9. Österreicher, F., Vajda, I.: A new class of metric divergences on probability spaces and and its statistical applications. Ann. Inst. Statist. Math. 55, 639–653 (2003)
10. Radhakrishna Rao, C.: Diversity: Its measurement, decomposition, apportionment and analysis. Sankhyā: The Indian Journal of Statistics, Series A (1961-2002) 44(1), 1–22 (1982)

A Multi-way Divergence Metric for Vector Spaces

Robert Moss and Richard Connor

Department of Computer and Information Sciences,
University of Strathclyde, Glasgow, G1 1XH, United Kingdom
{robert.moss,richard.connor}@strath.ac.uk

Abstract. The majority of work in similarity search focuses on the efficiency of threshold and nearest-neighbour queries. Similarity join has been less well studied, although efficient indexing algorithms have been shown. The multi-way similarity join, extending similarity join to multiple spaces, has received relatively little treatment.

Here we present a novel metric designed to assess some concept of a mutual similarity over multiple vectors, thus extending pairwise distance to a more general notion taken over a set of values. In outline, when considering a set of values X, our function gives a single numeric outcome $D(X)$ rather than calculating some compound function over all of $d(x, y)$ where x, y are elements of X.

$D(X)$ is strongly correlated with various compound functions, but costs only a little more than a single distance to evaluate. It is derived from an information-theoretic distance metric; it correlates strongly with this metric, and also with other metrics, in high-dimensional spaces. Although we are at an early stage in its investigation, we believe it could potentially be used to help construct more efficient indexes, or to construct indexes more efficiently.

The contribution of this short paper is simply to identify the function, to show that it has useful semantic properties, and to show also that it is surprisingly cheap to evaluate. We expect uses of the function in the domain of similarity search to follow.

Keywords: distance metric, multi-way divergence.

1 Introduction

Much of the research on similarity search focuses on the similarity (or distance) between two vectors. For many situations, however, ascertaining the mutual similarity of a set of vectors would be useful. Applications could be, for example, similarity joins, clustering, cluster analysis, and potentially many more that are dependent upon these techniques.

The calculation of density, or multi-way divergence as the analogue to distance, has been rarely used and is typically calculated through some compound function over the set of pair-wise distances within the set of vectors: for example, the mean intra-set distance or the mean distance from each vector to a centroid.

In this paper, we derive a function to calculate the divergence of a set of vectors that is based on an existing distance function. This new metric, which is grounded in information theory, avoids the problem of repeated calls to the distance metric through a direct, calculable notion of multi-way divergence without relying on approximation. It reuses the notion of complexity offered in the definition of the original distance metric and reverts to this definition when the size of the set is two. It is bounded, giving a maximum value when there is no commonality, allowing absolute comparisons to be made. And finally, it is only a little more expensive than a single distance to evaluate.

We show that multi-way divergence offers a cheaper alternative to compound functions whilst giving similar, or better, semantic properties. Although we are at an early stage in our investigation, we believe it could be applied usefully to construct new metric indices or to execute more complex queries, such as similarity joins, efficiently.

2 Related Work

The notion of a multi-way distance metric is not new, motivation coming from geometry and topology. Recently a few papers have analysed the generalisation to multi-way for any existing metric [3, 5, 6]. They consider whether various axioms are observed in these generalisations. For example, a metric space with a simple dyadic metric has the following axioms:

$$\begin{aligned} d(x,y) &= 0 \Leftrightarrow x = y \\ d(x,y) &= d(y,x) \\ d(x,y) &\leq d(x,z) + d(y,z) \end{aligned}$$

The first of these, is simply generalised to $D(x_1, \ldots, x_n) = 0$ if and only if all x_i are equal; the second to $D(x_{\pi(1)}, \ldots, x_{\pi(n)}) = D(x_1, \ldots, x_n)$ for every permutation π of $\{1, 2, \ldots, n\}$; while the third, triangle inequality, has been generalised in numerous ways [6], such as polyhedron inequality:

$$(n-1) \cdot D(x_1, \ldots, x_n) \leq \sum_{i=1}^{n} D(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{n+1})$$

These generalised axioms are interesting in their own right, and the first two are desirable properties of a multi-way divergence function, but it is not clear if polyhedron inequality aids similarity search at this stage.

Deza and Rosenberg introduced the multi-way extension of the star distance in [3], while perimeter distance [6] gives a geometrical “average distance”. Both of these, however, are compound functions and do not fundamentally look at generalising any specific metric.

3 Structural Entropic Divergence

We consider structural entropic distance (SED) proposed by [2] as a metric over trees, and shown as a vector distance in [1]. This metric operates over probability vectors where the sum of a vector's components equals 1. It is a ratio of the complexity of the mean vector to the geometric mean of complexities of individual vectors, where complexity is defined in terms of Shannon entropy [4] and is the amount of information required to describe the vector.

$$D(x_1, x_2) = \frac{C(\frac{x_1+x_2}{2})}{\sqrt{C(x_1)C(x_2)}} - 1$$

where $C(x) = b^{-\sum_i x_i \log_b x_i}$. They observe that the complexity of the mean vector will be the same as each individual complexity when both vectors are equal; that when the vectors have no intersecting components, the complexity of the mean vector is equal to the sum of individual complexities; and, for any other point, the complexity of the mean vector lies between these bounds.

3.1 Generalisation to a Multi-way Function

We observe a generalisation of this function to multiple arguments (MSED), while preserving the essence of these properties. Considering the numerator first, we can easily extend to multiple arguments as follows:

$$C\left(\frac{x_1 + x_2}{2}\right) \Rightarrow C\left(\sum_{i=1}^{n} \frac{x_i}{n}\right)$$

Now consider the denominator, a geometric mean, this too we extend to the following form:

$$\sqrt{C(x_1)C(x_2)} \Rightarrow \sqrt[n]{\prod_{i=1}^{n} C(x_i)}$$

The ratio of these two terms turns out to give a function in the range $[1, n]$, which can be scaled back into $[0, 1]$:

$$D(x_1, \ldots, x_n) = \frac{1}{n-1} \cdot \left(\frac{C(\frac{1}{n}\sum_{i=1}^{n} x_i)}{\sqrt[n]{\prod_{i=1}^{n} C(x_i)}} - 1 \right)$$

This generalised function remains a ratio of the complexity of a centroid to the geometric mean of its neighbours' complexities, and has the following properties:

– If all vectors are identical it gives 0
– If all vectors are different it gives 1
– All other inputs give a value between these bounds

Consider now the metric space axioms described in Section 2. SED has already been shown to be a proper metric in [2]. For MSED, we have stated that the lower bound is achieved when all elements are the same, so the first axiom holds. Since all operations involved in both the numerator and denominator are commutative, total symmetry holds too. We do not yet know whether polyhedron inequality holds.

4 Evaluation

We compared MSED to three other measures of multi-way divergence, based both on the SED metric and on Euclidean distance. Tuples of 2, 4, 8, 16 and 32 were chosen over randomly generated 5 and 15 dimensional probability vectors.

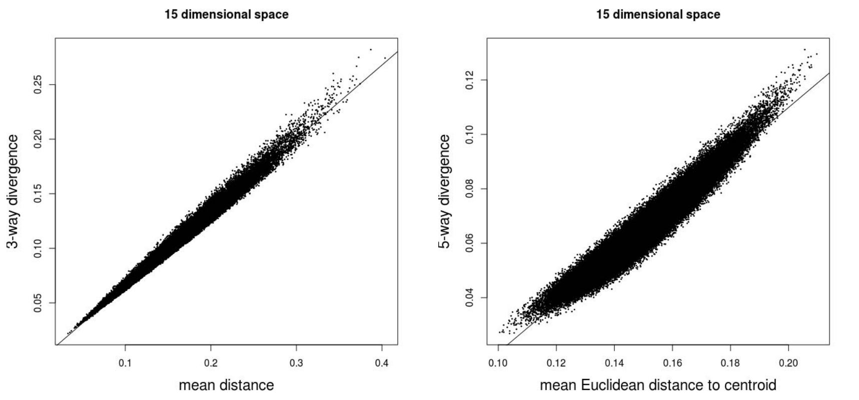
Table 1. Pearson's correlation coefficients for MSED

	PCC				
5 dimension SED	2-tuple	4-tuple	8-tuple	16-tuple	32-tuple
mean intra-cluster distance	1	0.9892284	0.9918476	0.9935051	0.9947683
mean distance to a centroid	0.9976659	0.9929966	0.9942112	0.9946998	0.9951615
max intra-cluster distance	1	0.9015272	0.8148037	0.7273526	0.6234713
5 dimension Euclidean					
mean intra-cluster distance	0.9245572	0.9587758	0.966835	0.9741714	0.9770738
mean distance to a centroid	0.9245572	0.9558159	0.9616505	0.964643	0.9684536
max intra-cluster distance	0.9245572	0.8799614	0.7991186	0.7164785	0.6081154
15 dimension SED					
mean intra-cluster distance	1	0.9910392	0.9927492	0.9932659	0.9945081
mean distance to a centroid	0.9979758	0.994636	0.995117	0.9945673	0.9949007
max intra-cluster distance	1	0.8558546	0.764015	0.6177353	0.5374743
15 dimension Euclidean					
mean intra-cluster distance	0.943713	0.9630723	0.9692825	0.9727428	0.97439
mean distance to a centroid	0.943713	0.9618546	0.9676469	0.9700614	0.9718063
max intra-cluster distance	0.943713	0.8333269	0.7624881	0.6385571	0.5282632

The best correlation comes with the mean distance to a centroid. This method is calculated by making a centroid using the mean vector then averaging all distances in the cluster to it. Since MSED is a ratio of the complexity of a centroid to the geometric mean of individual complexities, there is much more in common with this definition. Even when the comparison distance metric is Euclidean distance, a very strong correlation exists (figure 2b).

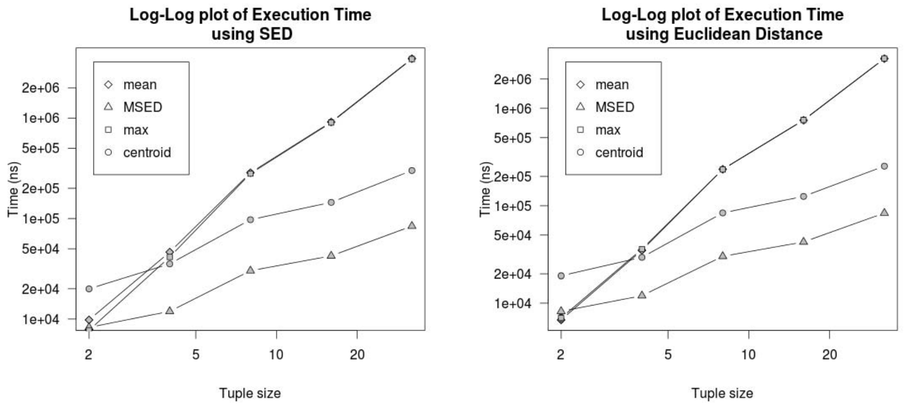
The mean intra-cluster distance also correlates well with MSED. Figure 2a shows the correlation with the mean intra-cluster distance, which appears to be strongest at the, more commonly used, lower end.

MSED correlates – less strongly than the other methods – with both SED and Euclidean maximum intra-cluster distance, and the correlation drops as the cluster size increases. Rather than assessing the mutual similarity, the maximum distance simply describes the two farthest points in the cluster. These two points must lie on the cluster perimeter and describe the spread of points across the space. Using only two points, however, fails to account for the spread in other dimensions, further verified by the drop in correlation in the higher dimensional space.



(a) triples with mean structural entropic distance (b) 5-tuples with mean euclidean distance to a centroid

Fig. 1. Correlation with divergence in 15-dimensional space



(a) Performance vs SED (b) Performance vs Euclidean Distance

Fig. 2. Performance

4.1 Performance

When many calculations are performed over a given metric space, the complexity of individual vectors need only be calculated once and stored for later use, thus amortising the cost of the calculation when multiple calls to divergence are required. The only calculation required is the complexity of the centroid, followed by some simple arithmetic, making the calculation really quite cheap.

The performance of the compound functions depend on the number of internal distance calls required: mean distance to centroid is linear, while mean and maximum intra-cluster distances are quadratic. We measured this to compare with MSED, Figure 2 shows the average time in nanoseconds to evaluate each function. While MSED clearly grows linearly it is approximately 3 times faster than mean distance to centroid using euclidean distance, and 4 times faster when using SED.

5 Conclusion

We have shown a formulation of MSED in a single calculation that is based upon information theory; that it is semantically comparable to other more expensive techniques that approximate mutual similarity through averaging, and that the cost to evaluate it is low in comparison with other approximations.

At this point, we find the divergence metric interesting in its own right, and have no very clear idea how it may be usefully deployed. However we have shown that it gives a useful semantic measure of some concept such as *density* within a set of objects, and yet is surprisingly cheap to evaluate compared with other approximations to this concept. We believe that this function will turn out to be useful in the domain of similarity search.

References

1. Connor, R., Moss, R.: A multivariate correlation distance for vector spaces. In: Navarro, G., Pestov, V. (eds.) SISAP 2012. LNCS, vol. 7404, pp. 209–225. Springer, Heidelberg (2012)
2. Connor, R., Simeoni, F., Iakovos, M., Moss, R.: A bounded distance metric for comparing tree structure. Inf. Syst. 36(4), 748–764 (2011)
3. Deza, M.-M., Rosenberg, I.G.: n-semimetrics. European Journal of Combinatorics 21(6), 797–806 (2000)
4. Shannon, C.E.: A mathematical theory of communication. SIGMOBILE Mob. Comput. Commun. Rev. 5(1), 3–55 (2001)
5. Warrens, M.J.: k-adic similarity coefficients for binary (presence/absence) data. Journal of Classification 26(2), 227–245 (2009)
6. Warrens, M.J.: N-way metrics. Journal of Classification 27(2), 173–190 (2010)

Optimal Distance Bounds for the Mahalanobis Distance

Tobias Emrich, Gregor Jossé, Hans-Peter Kriegel, Markus Mauder,
Johannes Niedermayer, Matthias Renz, Matthias Schubert, and Andreas Züfle

Institute for Informatics, Ludwig-Maximilians-Universität München
{emrich,josse,kriegel,mauder,niedermayer,
renz,schubert,zuefle}@dbs.ifi.lmu.de

Abstract. The Mahalanobis distance, or quadratic form distance, is a distance measure commonly used for feature-based similarity search in scenarios where features are correlated. For efficient query processing on such data effective distance-based spatial pruning techniques are required. In this work we investigate such pruning techniques by means of distance bounds of the Mahalanobis distance in the presence of rectangular spatial approximations. Specifically we discuss how to transform the problem of computing minimum and maximum distance approximations between two minimum bounding rectangles (MBRs) into a quadratic optimization problem. Furthermore, we show how the recently developed concept of spatial domination can be solved under the Mahalanobis distance by a quadratic programming approach.

1 Introduction

The most common distance measure for adaptive similarity search and metric learning is the Mahalanobis distance or quadratic form distance. It is frequently used for feature-based similarity search and relies on correlation matrices. The Mahanolobis distance allows us to integrate correlations into the distance computation while working with the original set of features and index structures, whereas the Euclidean distance often proves inadequate. Although it can be shown that any matrix under which the Mahalanobis distance is a metric is equivalent to a linear transformation of the feature space [9], transforming the database is usually not an option: the matrix describing correlations might vary between queries and generating index structures for all possible transformations is infeasible. There exists a wide variety of methods for learning Mahalanobis distances [10].

When processing similarity queries such as ϵ-range queries, k-nearest neighbor (kNN) queries, reverse-kNN queries in a feature space, an important efficiency aspect is early pruning of objects which cannot be part of the result set. Such pruning is facilitated by appropriate approximation techniques. For example, such approximations include minimal bounding rectangles (MBRs) of complex spatial polygons, MBRs of sets of data points and rectangular approximations of uncertain objects. During query execution costly distance functions or I/O operations can be avoided by using lower/upper bound distances on these approximations. For decades, the minimum and maximum distance between MBRs have been used to decide whether an object can be discarded (pruned) from any further consideration. Recently, it has been shown that these metrics can be improved under various settings using the concept of spatial domination [4].

In this work, we extend the concept of spatial domination to similarity search using the Mahalanobis distance. We start by reviewing methods to compute the minimum and maximum distance between a point and an MBR. Then we provide a solution for computing these bounds between two MBRs. Finally, we develop a spatial domination decision criterion based on the quadratic form distance and evaluate it experimentally.

2 Related Work

The Mahalanobis distance and its extensions [2,3] are a frequently used distance measure, particularly in the context of multimedia databases. To make this application feasible, techniques to improve its processing speed have been developed. In [1] the Mahalanobis distance is conservatively bounded by distance approximations based on parametrized MBRs and spheres. If the similarity matrix is query-independent, the underlying space can be transformed before indexing, reducing the problem to a query in Euclidean space [9]. In contrast, in this paper we assume that the similarity matrix is query-dependent. Examples for this scenario are also given in [9]. Recently, a specialized index structure for an extension of the Quadratic Form Distance has been proposed [6]. While such index structures can be useful for specialized applications, our spatial pruning method can be used in general R-trees, making it extremely versatile. A similarly widely applicable method is described in [8]. The authors determine the shortest distance using a gradient descent method while limiting the search to feasible points. In addition, they generate a lower bound of the Mahalanobis distance through a lower-dimensional approximation of the similarity matrix. However, the authors do not investigate the problem of computing the maximum distance and the problem of spatial domination. In [5] the authors aim at conservatively bounding the maximum Mahalanobis distance between each point and the sample's mean given that the applied similarity matrix is the sample's covariance.

3 Problem Definition

Formally, the Mahalanobis distance is defined as follows:

Definition 1. *Let $x, y \in \mathbb{R}^d$ be d-dimensional points (column-vectors) and $A \in \mathbb{R}^{d \times d}$ be a symmetric positive definite matrix. The Mahalanobis distance d_A between the two points x and y is defined by: $d_A(x, y) = \sqrt{(x - y)^T \cdot A \cdot (x - y)}$*

Minimum distance and maximum distance in the presence of MBR approximations are well-studied for L_p-norms [7]. In this work, we show how these distance bounds can be computed for the Mahalanobis distance. In particular, we show how to compute the minimum/maximum distance between point and MBR and between two MBRs.

Definition 2. *Let $x, y \in \mathbb{R}^d$ be a be d-dimensional point, $X, Y \subset \mathbb{R}^d$ be MBRs and $A \in \mathbb{R}^{d\times d}$ be a symmetric positive definite matrix. We define the following bounds:*

$$\begin{aligned}
d_A^{min}(x,Y) &= \min_{y\in Y} d_A(x,y) && \textit{(point-MBR minimum distance)}\\
d_A^{max}(x,Y) &= \max_{y\in Y} d_A(x,y) && \textit{(point-MBR maximum distance)}\\
d_A^{min}(X,Y) &= \min_{x\in X, y\in Y} d_A(x,y) && \textit{(MBR-MBR minimum distance)}\\
d_A^{max}(X,Y) &= \max_{x\in X, y\in Y} d_A(x,y) && \textit{(MBR-MBR maximum distance)}
\end{aligned}$$

Furthermore, we want to examine spatial domination under the Mahalanobis distance.

Definition 3. *Let $X, Y, R \subset \mathbb{R}^d$ be MBRs and $A \in \mathbb{R}^{d\times d}$ be a symmetric positive definite matrix. X spatially dominates Y w.r.t. R if:*

$$Dom_A(X, Y, R) := \forall x \in X, y \in Y, r \in R : d_A(x, r) < d_A(y, r)$$

Informally spatial domination checks if "X is definitely closer to R than Y" (see Figure 1(a) illustrating a Euclidean case of domination, while Figure 1(b) shows the same situation under a quadratic form transformation). This relation can be utilized for pruning in the context of many spatial query predicates (e.g. for nearest and reverse nearest neighbor queries). For example, if R approximates the query object for a nearest neighbor query then Y can safely be pruned if $Dom(X, Y, R)$ holds. The reason is that $Dom(X, Y, R)$ implies that X must be closer to R than Y. We refer to any criterion whose fulfillment implies domination as a domination decision criterion (DDC).

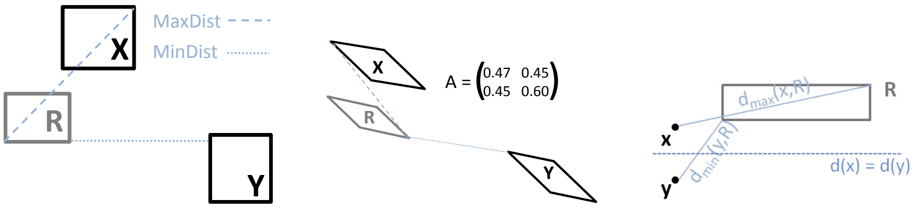
In the following, we will show that the minimum and maximum distances as well as the decision about spatial domination can be rewritten as quadratic optimization problems of the general form: $f(x) = \frac{1}{2}x^T \cdot Q \cdot x + c \cdot x$ subject to the constraint $x_{lb} < x < x_{ub}$. Using quadratic programming the minimum of $f(x)$ (with respect to x) satisfying the above constraints can be computed in polynomial time.

4 Minimum and Maximum Distance Bounds

Point and MBR: In order to find a lower bound for the Mahalanobis distance between point and MBR, we define:

$$d_A^{min}(x,Y) = \min_{y\in Y} \sqrt{(x-y)^T \cdot A \cdot (x-y)} \tag{1}$$

The constraint $y \in Y$ can be rewritten as $y_{\text{lb}} < y < y_{\text{ub}}$ where y_{lb} (y_{ub}) is the lower (upper) bound vector of Y consisting of the minimum (maximum) values of Y in each dimension. Since the square root does not affect the pruning decision, the function (see Equation 1) can be rewritten to match the optimization term:



(a) Domination on MBRs (b) Transformed Space (c) DDC^{MM} not decisive

Fig. 1. Minimum / Maximum Distance and Spatial Domination

$$\min_y \sqrt{(x-y)^T \cdot A \cdot (x-y)} \hat{=} \min_y [(x-y)^T \cdot A \cdot (x-y)] =$$
$$\min_y [x^T Ax - x^T Ay - y^T Ax + y^T Ay] = \min_y [-x^T Ay - y^T Ax + y^T Ay] \overset{\star}{=}$$
$$\min_y [-(Ay)^T x - y^T Ax + y^T Ay] = \min_y [y^T A^T x - y^T Ax + y^T Ay] =$$
$$\min_y [-y^T Ax - y^T Ax + y^T Ay] = \min_y [\frac{1}{2} \cdot y^T (2 \cdot A) y - 2 \cdot x^T A^T y]$$

The above holds due to the symmetry of A ($A = A^T$) and because transposing a scalar (namely $((Ay)^T x)^T$ at $\star$) can be ignored.

The maximum distance between point and MBR can be computed analogously by using $-A$, since

$$0 < \max_y d_A(x,y) = -\min_y \big((-(x-y)^T A(x-y))\big) = -\min_y d_{-A}(x,y).$$

MBR and MBR: Extending the above definitions of minimum and maximum distance to the case where $X, Y \subset \mathbb{R}^d$ are MBRs is straightforward. Given a (symmetric, positive definite) matrix A, the minimum distance of X and Y w.r.t. A is defined as:

$$d_A^{min}(X,Y) = \min_{x \in X, y \in Y} \sqrt{(x-y)^T \cdot A \cdot (x-y)} \tag{2}$$

Ignoring the square root the function to be minimized can be rewritten as follows:

$$(x-y)^T \cdot A \cdot (x-y) = (x^T A - y^T A) \cdot (x-y) =$$
$$x^T Ax - x^T Ay - y^T Ax + y^T Ay = \frac{1}{2} \cdot \begin{pmatrix} x \\ -y \end{pmatrix}^T \cdot \begin{pmatrix} 2A & 2A \\ 2A & 2A \end{pmatrix} \cdot \begin{pmatrix} x \\ -y \end{pmatrix}$$

Following from this observation we can define proper boundary conditions:

$$\begin{pmatrix} x_{\text{lb}} \\ -y_{\text{ub}} \end{pmatrix} \leq \begin{pmatrix} x \\ -y \end{pmatrix} \leq \begin{pmatrix} x_{\text{ub}} \\ -y_{\text{lb}} \end{pmatrix}$$

Again we are able to derive $d_A^{max}(X,Y)$ by replacing A with $-A$.

5 Domination Decision Criterion

To derive a DDC the minimum and maximum distance may be utilized as follows:

$$Dom_{(A)}(X, Y, R) \Leftarrow d_{(A)}^{max}(X, R) < d_{(A)}^{min}(Y, R) \tag{3}$$

However, this so called MinMax criterion (DDC$^{\text{MM}}$ or DDC$_A^{\text{MM}}$ if A is of particular interest) does not always yield the best possible result. To see this, it suffices to consider the simplified case where X and Y are points (as depicted in Figure 1(c)). In this example each point above the equi-distance-potential of x and y ($\{z \in \mathbb{R}^d \mid \|x - z\| = \|y - z\|\}$) is obviously closer to x than to y. Thus y is spatially dominated by x w.r.t. R, although this would not have been detected by DDC$^{\text{MM}}$ (Equation 3). Recently an optimal criterion was developed to correctly detect this relation and therefore enhancing the DDC$^{\text{MM}}$ under all L_p-Norms [4]. However this optimal criterion can not be directly applied under the Mahalanobis distance, since the matrix A implies a rotation and scaling of the original feature space. Figure 1(b) illustrates this transformation of the original example from Figure 1(a). Note that even for computing maximum and minimum distance between the transformed MBRs now different corners realize these distances.

Definition 4. *Let X, Y, R be MBRs, the optimal domination decision criterion under the quadratic form matrix A, DDC$_A^{OPT}$, is defined as:*

$$DDC_A^{OPT}(X, Y, R) := Dom_A(X, Y, R) = \forall x \in X, y \in Y, r \in R : d_A(x, r) < d_A(y, r)$$

This can be reformulated as follows:

$$\begin{gathered}\forall x \in X, \forall y \in Y, \forall r \in R : d_A(y, r)^2 - d_A(x, r)^2 > 0 \Leftrightarrow \\ \min_{x \in X, y \in Y, r \in R} [d_A(y, r)^2 - d_A(x, r)^2] > 0 \Leftrightarrow \\ \min_{x \in X, y \in Y, r \in R} [(y - r)^T \cdot A \cdot (y - r) - (x - r)^T \cdot A \cdot (x - r)] > 0\end{gathered}$$

The function to minimize can then be expanded:

$$\begin{gathered}y^T Ay - y^T Ar - r^T Ay + r^T Ar - (x^T Ax - x^T Ar - r^T Ax + r^T Ar) = \\ y^T Ay - y^T Ar - r^T Ay - x^T Ax + x^T Ar + r^T Ax = \\ \begin{pmatrix} x \\ -y \\ r \end{pmatrix}^T \cdot \begin{pmatrix} A & 0 & A \\ 0 & A & A \\ A & A & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ -y \\ r \end{pmatrix} = \frac{1}{2} \cdot \begin{pmatrix} x \\ -y \\ r \end{pmatrix}^T \cdot \begin{pmatrix} 2A & 0 & 2A \\ 0 & 2A & 2A \\ 2A & 2A & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ -y \\ r \end{pmatrix}\end{gathered}$$

The constraints are straightforwardly given by

$$\begin{pmatrix} x_{\text{lb}} \\ -y_{\text{ub}} \\ r_{\text{lb}} \end{pmatrix} \leq \begin{pmatrix} x \\ -y \\ r \end{pmatrix} \leq \begin{pmatrix} x_{\text{ub}} \\ -y_{\text{lb}} \\ r_{\text{ub}} \end{pmatrix}$$

By reformulating the dominance decision criterion under a quadratic form as above, we improve the selectivity and reduce the problem to an efficiently solvable quadratic programming problem as is demonstrated in the next section.

6 Experiments

Our goal is to reduce the number of candidates for a query predicate (k nearest neighbor, reverse k nearest neighbors, etc.) by efficiently pruning dominated objects and therefore avoiding costly I/O operations. To test our approach, we implemented the decision criteria DDC^{MM} and DDC^{OPT} in Matlab using the built-in solver for quadratic programming problems which relies on trust region optimization. In the first setting we randomly created MBRs within the unit cube as well as symmetric positive definite matrices and compared the overall decision power of DDC^{MM} and DDC^{OPT}, varying the maximal extent of the MBRs as well as the number of dimensions. As shown in Figure 2, DDC^{OPT} is superior to DDC^{MM}, but efficiency decreases as dimensionality and extent increase. In order to monitor the relative behavior of DDC^{MM} in comparison to DDC^{OPT} we tested in how many positive DDC^{OPT} cases DDC^{MM} would detect dominance as well (see Figure 3). The gain is striking: if the MBRs are relatively small, DDC^{OPT} is only slightly better than DDC^{MM}, but it outperforms DDC^{MM} by orders of magnitude when dimensionality increases and the MBRs expand. In the most demanding setting (6 dimensional MBRs with a maximal extent of 0.6) DDC^{MM} can only verify dominance in less than 1 percent of the cases where DDC^{OPT} detected dominance. Nevertheless, DDC^{OPT} is less costly than DDC^{MM}, as displayed in Figure 4 which shows the average runtime of one dominance decision. It is conceivable that the runtime results may be improved by using a different algorithm for the quadratic programming problem, such as an interior point method.

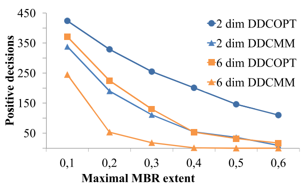


Fig. 2. Absolute number of positive dominance decisions (on 1000 random MBRs)

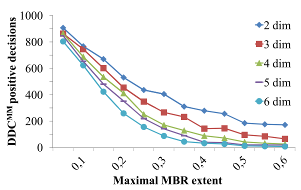


Fig. 3. DDC^{MM} pruning selectivity relative to 1000 positive DDC^{OPT} decisions

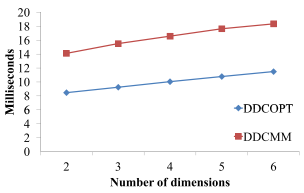


Fig. 4. Average runtime of DDC^{MM} and DDC^{OPT} in milliseconds

7 Conclusions

In this paper we have defined conservative minimum and maximum distance approximations for MBRs under the Mahalanobis distance by formulating the problam as a quadratic optimiziation problem under constraints. In addition, we have shown that the spatial domination problem can also be described as a quadratic optimization problem. Our experiments show that especially with large MBRs the spatial domination approach outperforms the traditional MinMaxdist approach by orders of magnitude.

We plan to extend this work into several directions. First we want to apply the proposed bounds to actual query predicates such as nearest- and reverse nearest neighbor queries in the presence of an index. Second we plan to consider the underlying geometry to find more efficient solutions than the methods presented here.

References

1. Warrens, M.J.: N-way metrics. Journal of Classification 27(2), 173–190 (2010)
2. Beecks, C., Uysal, M.S., Seidl, T.: Signature quadratic form distance. In: Proc. CIVR, pp. 438–445 (2010)
3. Beecks, C., Uysal, M.S., Seidl, T.: Similarity matrix compression for efficient signature quadratic form distance computation. In: Proc. SISAP, pp. 109–114 (2010)
4. Emrich, T., Kriegel, H.-P., Kröger, P., Renz, M., Züfle, A.: Boosting spatial pruning: On optimal pruning of MBRs. In: Proc. SIGMOD, pp. 39–50 (2010)
5. Gath, E.G., Hayes, K.: Bounds for the largest mahalanobis distance. Linear Algebra Appl. 419(1), 93–106 (2006)
6. Lokoc, J., Hetland, M.L., Skopal, T., Beecks, C.: Ptolemaic indexing of the signature quadratic form distance. In: Proc. SISAP, pp. 9–16 (2011)
7. Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In: Proc. SIGMOD, pp. 71–79 (1995)
8. Seidl, T., Kriegel, H.-P.: Efficient user-adaptable similarity search in large multimedia databases. In: Proc. VLDB, pp. 506–515 (1997)
9. Skopal, T., Bartos, T., Lokoc, J.: On (not) indexing quadratic form distance by metric access methods. In: Proc. EDBT, pp. 249–258 (2011)
10. Yang, L.: An overview of distance metric learning. Technical report, Department of Computer Science and Engineering, Michigan State University (2007)

Text Categorization via Similarity Search
An Efficient and Effective Novel Algorithm*

Hubert Haoyang Duan[1], Vladimir G. Pestov[1], and Varun Singla[2]

[1] University of Ottawa, Ottawa, Ontario K1N 6N5, Canada
{hduan065,vpest283}@uottawa.ca
[2] Indian Institute of Technology Delhi, Hauz Khas, New Delhi-110 016, India
ee5080429@ee.iitd.ac.in

Abstract. We present a supervised learning algorithm for text categorization which has brought the team of authors the 2nd place in the text categorization division of the 2012 Cybersecurity Data Mining Competition (CDMC'2012) and a 3rd prize overall. The algorithm is quite different from existing approaches in that it is based on similarity search in the metric space of measure distributions on the dictionary. At the preprocessing stage, given a labeled learning sample of texts, we associate to every class label (document category) a point in the space of question. Unlike it is usual in clustering, this point is not a centroid of the category but rather an outlier, a uniform measure distribution on a selection of domain-specific words. At the execution stage, an unlabeled text is assigned a text category as defined by the closest labeled neighbour to the point representing the frequency distribution of the words in the text. The algorithm is both effective and efficient, as further confirmed by experiments on the Reuters 21578 dataset.

1 Introduction

The amount of texts readily available in the world is growing at an astonishing rate; classifying these texts through machine learning techniques, promptly and without much human intervention, has thus become an important problem in data mining. Much research, in the field of supervised learning, has been done to find accurate algorithms to classify documents in a dataset to their appropriate categories, e.g. see [27] or [1] for a detailed survey of text categorization.

The most widely used model for text categorization is the Vector Space Model (VSM) [24]. Under this model, a data dictionary $\mathcal{T}$ consisting of unique words across the documents in the dataset is constructed. The documents are represented by real-valued vectors in the space $\mathbb{R}^{\mathcal{T}}$ with dimension equaling to the size of the dictionary. Given $t \in \mathcal{T}$, the t-th coordinate of a vector is the relative frequency of the word t in a given document. When some of the documents'

* This work has been partially supported by a 2012 NSERC Canada Graduate Scholarship and a 2013 Ontario Graduate Scholarship (Hubert Haoyang Duan), 2012–2017 NSERC Discovery Grant *"New set-theoretic tools for statistical learning"* (Vladimir Pestov), and the 2012 Mitacs Globalink Program (Varun Singla).

actual class labels are known and used for training, many well-known classifiers in supervised machine learning, such as SVM [6], k-NN [7], and Random Forest [3], can then be applied to categorize documents.

Text categorization decidedly comes across as a problem of detecting similarities between a given text and a collection of texts of a particular type. Although distance-based learning rules for text categorization, such as the k-nearest neighbour classifier, e.g. [18], are not new, they are currently based on the entire feature space, while any dimension reduction steps are done independently beforehand [27].

We aim to fill this gap by suggesting a novel supervised learning algorithm for text categorization, called the Domain-Specific classifier. It discovers specific words for each category, or domain, of documents in training and classifies based on similarity searches in the space of word frequency distributions supported on the respective domain-specific words.

For each class label, $j = 1, 2, \dots, k$, our algorithm extracts class, or domain, specific words from labeled training documents, that is, words that appear in the class j more frequently than in all the other document classes combined (modulo a given threshold). Now a given unlabeled document is assigned a label j if the normalized frequency of domain-specific words for j in the document is higher than for any other label.

To see that this classifier is indeed similarity search based, let $x_j \in \mathbb{R}^{\mathcal{T}}$ be a binary vector whose t-th coordinate is 1 if and only if t is domain-specific to j, and 0 otherwise. Normalize x_j according to the ℓ^p distance, and let a document be represented by a vector $w \in \mathbb{R}^{\mathcal{T}}$. Then the label assigned to w is that of the closest neighbour to w among $x_1, x_2, \dots, x_k$ with regard to the simplest similarity measure, the inner product on $\mathbb{R}^{\mathcal{T}}$. In other words, we seek to maximize the value of $\langle w, x_j \rangle$ over $j = 1, 2, \dots, k$. Notice that the well-known cosine similarity measure, cf. e.g. [27], corresponds to the special case $p = 2$.

This algorithm was first used in the 3rd Cybersecurity Data Mining Competition (CDMC 2012) to notable success, as the team of authors placed second in the text categorization challenge, and first in classification accuracy [19]. In addition, the classification performance of the algorithm was validated on a sub-collection of the popular Reuters 21578 dataset [16], consisting of single-category documents from the top eight most frequent categories, with the standard "modApté" training/testing split. In terms of accuracy, our classifier performs slightly better than SVM with a linear kernel, and is significantly faster.

This paper is organized as follows. Section 2 surveys common feature selection and extraction methods and classifiers considered in the text categorization literature. Section 3 explains the new Domain-Specific classifier in detail and casts it as a similarity search problem. Section 4 discusses results from the CDMC 2012 Data Mining competition and experiments from the competition and on the Reuters 21578 dataset. Finally, Section 5 concludes the paper with some discussion and directions for future work.

2 Related Work

In this section, we describe the VSM model and provide a brief survey on widely known methods for text categorization.

From this section onwards, following notation similar to [27], we let $\mathcal{D} = \{d_1, d_2, \ldots, d_n\}$ denote the dataset of documents, with size $n = |\mathcal{D}|$, and $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$ the data dictionary of all unique words from documents in $\mathcal{D}$, with size $m = |\mathcal{T}|$. Given a document d and a word $t \in \mathcal{T}$, $|d|$ denotes the number of words in d and $t \in d$ indicates that the word t is found in d.

2.1 Vector Space Model

The Vector Space Model (VSM) [24] is the most common model for document representation in text categorization. According to [1], there is usually a standard preprocessing step for the documents in $\mathcal{D}$, where all alphabets are converted to lowercase and all stop words, such as articles and prepositions, are removed. Sometimes, a stemming algorithm, such as the widely used Porter stemmer [20], is applied to remove suffices of words (e.g. the word "connection" $\rightarrow$ "connect").

In VSM, the data dictionary, consisting of all unique words that appear in at least one document in $\mathcal{D}$, is first constructed. Sometimes, n-grams, which are phrases of words, are also included in the dictionary; however, the benefit of these additional phrases is still up for debate [27]. Given the data dictionary, each document can be represented as a vector in the real-valued vector space with dimension equaling the size of the dictionary. Two common methods for associating a document to a vector are explained below.

The simplest method assigns to a document d the vector consisting of the relative term frequencies for d, see e.g. [11]. The second, known as the *tf-idf* method, assigns d to the vector consisting of the products of term and inverse document frequencies [23]. Mathematically speaking, a document d is mapped to a real-valued vector of length m: $d \longmapsto (w_1, w_2, \ldots, w_m) \in \mathbb{R}^m$, for

$$w_i = \frac{c(t_i, d)}{|d|} \quad \text{(frequency method)} \tag{1}$$

or

$$w_i = c(t_i, d) \log \left(\frac{n}{|\{d \in \mathcal{D} : t_i \in d\}|} \right) \quad (\textit{tf-idf} \text{ method}), \tag{2}$$

where $c(t_i, d)$ denotes the number of times the word t_i appears in d. Other representations include binary and entropy weightings and the normalized *tf-idf* method [1].

Once the documents are represented as vectors, the dataset can be interpreted as a data matrix M of size $(n \times m)$. However, a main challenge for text categorization is that the size of the data dictionary is usually immense so the data matrix is extremely high dimensional. Dimension reduction techniques must often be applied before classification to reduce complexity [1].

2.2 Feature Selection and Extraction Methods

Due to the potentially large size of the data dictionary, feature selection and extraction methods are often applied to reduce the dimension of the data matrix. Feature selection methods assign to each feature, a word in the data dictionary, a statistical score based on some measure of importance. Only the highest scored features, past some defined threshold, are kept and a lower dimensional data matrix is created from only these features. Some known feature selection methods in text categorization include calculating the document frequency, e.g. [31], mutual information [5], and χ^2 statistics, e.g. [26]. See e.g. [10] or [31] for a thorough study of text feature selection methods.

Feature extraction methods transform the original list of features to a smaller list of new features, based on some form of feature dependency. Common well-known feature extraction techniques, as surveyed in [22], are Latent Semantic Indexing (LSI) [8], Linear Discriminant Analysis (LDA) [28], Partial Least Squares (PLS) [32], and Random Projections [2].

2.3 Classification Algorithms

Well-known classifiers that have been applied to text categorization include the k-nearest neighbour classifier [18], Support Vector Machines [12], the Naive Bayes classifier [15], and decision trees [13]. We ask the reader to refer to indicated references, or to survey articles, such as [27], [11], and [1]. The paper [30] provides comparable empirical results on some of these classifiers.

The standard approach in literature for text categorization is that one, or more, feature selection or extraction technique is first applied to a data matrix, since the original data matrix is often extremely high-dimensional. A learning algorithm, independent of the dimension reduction process, is then used for classification [27]. The novel approach in this paper is that we consider a new classifier based only on extracted class specific words, which naturally reduces time complexity and the dimension of the dataset. In other words, the Domain-Specific classifier both performs dimension reduction and classifies, in consecutive and dependent steps.

3 The Domain-Specific Classifier

Our algorithm consists of two distinct stages: extraction of domain-specific words from training samples and classification of documents based on the closest labeled point determined by these domain-specific words.

3.1 Preprocessing Stage: Domain-Specific Words

Fix an alphabet Σ and denote Σ^* as the set of all possible "words" formed from Σ. A document d is then simply an ordered sequence of "words", $d \in (\Sigma^*)^{|d|}$, and the data dictionary $\mathcal{T}$ is a subset of Σ^*. Given a set of labeled documents, we can denote it as $\mathcal{D}_{\text{lab}} = \{(d_1, l_1), (d_2, l_2), \ldots, (d_n, l_n)\}$, where d_i is a document and $l_i \in \{1, 2, 3, \ldots, k\}$ is its label, out of a possible k different labels. In addition, we can partition $\mathcal{D}_{\text{lab}}$ into subsets of documents according to their labels:

$$\mathcal{D}_{\text{lab}} = \bigcup_{j=1}^{k} \mathcal{D}_{\text{lab}}^{j} \tag{3}$$

where $\mathcal{D}_{\text{lab}}^{j} = \{(d, l) \in \mathcal{D}_{\text{lab}} : l = j\}$ is the set of documents of label j. Then, for a particular label j and a word $t \in \mathcal{T}$ in the data dictionary, we denote $f_j(t)$ as the average proportion of times the word t appears in documents with label j:

$$f_j(t) = \frac{1}{|\mathcal{D}_{\text{lab}}^{j}|} \sum_{(d,j) \in \mathcal{D}_{\text{lab}}^{j}} \frac{c(t,d)}{|d|} \ . \tag{4}$$

Domain-specific words are those words which appear, on average, proportionally more often in one label type of documents in $\mathcal{D}_{\text{lab}}$ than other types.

Definition 1. *Let $\alpha \geq 0$. A word $t \in \mathcal{T}$ in the data dictionary is* domain *(or* class*) j* specific *if*

$$f_j(t) > \alpha \sum_{j' \neq j} f_{j'}(t). \tag{5}$$

This definition of domain-specific words depends on the parameter α and hence, so does the Domain-Specific classifier. As α increases from 0, the number of domain-specific words for each class label decreases; as a result, α can be thought of as a threshold parameter, and an optimal choice for α is determined through cross-validation using training data.

3.2 Classification Stage

Let now d be an unclassified document. We associate to it a vector $w = w_d \in \mathbb{R}^{\mathcal{T}}$ (a relative frequency distribution of words) as in Eq. (1), that is, for every $t \in \mathcal{T}$,

$$w(t) = \frac{c(t,d)}{|d|}. \tag{6}$$

Let j be a label. Denote $CS_j = CS_{j,\alpha}$ the set of domain-specific words to j. Define the total relative frequency of domain j specific words found in d:

$$w[CS_j] = \sum_{t \in CS_j} w(t) = \frac{1}{|d|} \sum_{t \in CS_j} c(t,d). \tag{7}$$

The classifier assigns to d the label j for which the following ratio is the highest:

$$j = \operatorname{argmax}_i \frac{w[CS_i]}{|CS_i|^{1/p}}. \tag{8}$$

Here, $p \in (0, \infty]$ is a parameter, which normalizes a certain measure with regard to the ℓ^p distance, cf. below in Section 3.3.

3.3 Space of Positive Measures on the Dictionary

A (positive) measure on a finite set $\mathcal{T}$ is simply an assignment $t \mapsto w(t)$ to every $t \in \mathcal{T}$ of a non-negative number $w(t)$; a probability measure also satisfies $\sum_{t \in \mathcal{T}} w(t) = 1$. Denote $M(\mathcal{T})$ the set of all positive measures on $\mathcal{T}$.

Fix a parameter $p \in (0, \infty]$. The following is a positive measure on $\mathcal{T}$:

$$x_j(t) = \begin{cases} \frac{1}{|CS_j|^{1/p}}, & \text{if } t \in CS_j, \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

If $p = 1$, we obtain a probability measure uniformly supported on the set of domain j specific words. In general, values of $p \in (0, \infty]$ correspond to different normalizations of the uniform measure supported on these words, according to the ℓ^p distance. (The case when $p = \infty$, that is, the ℓ^∞ distance, corresponds to non-normalized uniform measure.)

Among the similarity measures on $M(\mathcal{T})$, we single out the standard inner product

$$\langle w, v \rangle = \sum_t w_t v_t. \tag{10}$$

Notice that for every $w \in M(\mathcal{T})$ and each j,

$$\langle w, x_j \rangle = \frac{w[CS_j]}{|CS_j|^{1/p}}, \tag{11}$$

and for this reason, the classification algorithm (8) can be rewritten as follows:

$$j = \text{argmax}_i \, \langle w, x_i \rangle. \tag{12}$$

Our classifier is based on finding the closest point x_i to the input point w in the sense of the simplest similarity measure, the inner product.

The similarity workload is a triple $(\mathbb{U}, S, \mathbb{X})$, consisting of the domain $\mathbb{U} = M(\mathcal{T})$, the similarity measure $S(w, v) = \langle w, v \rangle$ equal to the standard inner product (a rather common choice in the problems of statistical learning, cf. [25]), and the dataset $\mathbb{X} = \{x_1, x_2, \ldots, x_k\}$ of normalized uniform measures corresponding to the text categories and domain-specific words extracted at the preprocessing stage.

Note that the well-known cosine similarity measure arises in the special case when the normalizing parameter is $p = 2$; hence, it is not necessary to consider this measure separately. Our experiments have shown that different datasets require different normalizing parameters for x_j, and that the optimal normalization depends on the sizes of the document categories; Section 5 includes a discussion on this topic.

4 Experiments and Results

This section details the experiments and results obtained for the Domain-Specific classifier, in the 2012 Cybersecurity Data Mining Competition and on the Reuters

21578 dataset. All of the programming for this section were done with standard packages in R [21] and with the specialized packages `e1071` [9] and `randomForest` [17], on a desktop running Windows 7 Enterprise, with a Intel i5 3.10 GHz processor and 4GB of RAM.

4.1 The 2012 Cybersecurity Data Mining Competition

The 3rd Cybersecurity Data Mining Competition (CDMC 2012) [19], associated with the 19th International Conference on Neural Information Processing (ICONIP 2012) in Doha, Qatar from November 12 - 15, 2012 [29], included three supervised classification tasks: electronic news (e-News) text categorization, intrusion detection, and handwriting recognition.

The Domain-Specific classifier was first developed by the team of authors for the e-News text categorization challenge, which required classifying news documents to five topics: business, entertainment, sports, technology, and travel. The documents were collected from eight online news sources. The words in these documents were obfuscated, and all punctuations and stop words removed. Here is a sample scrambled text document paragraph from the competition:

HUJR Xj gjXZMUXe fAJjAeK UO jwXeA URSek UYjmX xjI K SeeW eOWrjJeeR ZARWZDek UAk WDjkmzXZMe KXR UA eRReAXZUr BmeRXZjA RZAze zjOWUAZeR XjkUJ OmRX UzzjOWrZRx OjDe IZXx weIeD WejWre uxe OjRX RmzzeRRwmr RXUDXmWR OmRX

In total, 1063 e-News documents for training, each labeled as one of the $k = 5$ topics, were given for the goal of classifying 456 documents.

Table 1. Information on the dataset size for e-News classification task

Label j	Topic	# of documents
1	Business	205
2	Entertainment	215
3	Sport	193
4	Technology	223
5	Travel	227
	Training set	1063
	Classification task	456

4.2 Competition Experiments

After a pre-processing step, where all document words of length less or equal to 3 were removed, a data dictionary of all unique words from the training and classification documents, consisting of $m = 55822$ words, was constructed. The 1063 labeled documents were converted to vectors of length $m = 55822$, according to the Vector Space Model. Then, 5-fold cross-validation on the training dataset was performed to test the performance of the classifier.

For comparison purposes, the Support Vector Machines (SVM) classifier [6], using the Gaussian Radial Basis (GRB) and linear kernels with cost 10, and the

Random Forest classifier [3], using 50 trees, were also tested. The performance measures considered were classification accuracy and the F-Measure (F1) [1]. See Table 2, where the computation times for both the training and predicting stages are also indicated.

Table 2. Classification performance of the Domain-Specific classifier (DSC) through 5-fold cross validation on the training set, compared to SVM with the Gaussian Radial Basis (GRB) and linear kernels and Random Forest (RF)

α	DSC 0	DSC 0.25	DSC 0.5	DSC 0.75	DSC 1	DSC **2**	DSC 5	DSC 10	SVM GRB	SVM linear	RF
Accuracy	0.575	0.854	0.887	0.896	0.896	**0.915**	0.901	0.882	0.613	0.821	0.882
F1 Business	0.491	0.765	0.839	0.833	0.765	0.794	0.774	**0.868**	0.576	0.740	0.853
F1 Entertainment	0.400	0.925	0.915	0.896	0.928	**0.949**	0.926	0.897	0.530	0.784	0.845
F1 Sport	0.899	0.962	0.946	1.000	0.987	**1.000**	0.974	0.944	0.847	0.886	0.943
F1 Technology	0.548	0.767	0.850	0.825	0.871	**0.891**	0.891	0.848	0.597	0.785	0.842
F1 Travel	0.600	0.883	0.892	**0.947**	0.916	0.920	0.911	0.867	0.641	0.896	0.917

Computational Time

	DSC	SVM GRB	SVM linear	RF
Training stage	**1.5 secs**	3.99 mins	3.25 mins	3.96 mins
Predicting stage	0.6 secs	14.3 secs	16.9 secs	**0.4 secs**

The choice $\alpha = 2$ resulted in the best accuracy of 0.915 for the Domain-Specific classifier, and the optimal normalizing parameter was $p = 1$, corresponding to the choice of $x_1, x_2, \ldots, x_k$ normalized as probability measures uniformly supported on the domain-specific words. Consequently, these two values were used for the classification of the 456 documents in the competition. Note that the accuracy score and the F-Measure for 4 out of the 5 categories, for $\alpha = 2$, were higher than the respective scores obtained with SVM with the GRB and the linear kernels and Random Forest. Experiments had also shown that the Domain-Specific classifier was extremely fast and efficient, since distance calculations are only based on domain-specific words, not the entire data dictionary. As a result, no dimension reduction technique prior to classification was required.

4.3 Competition Results

The submissions for the three tasks for the 2012 Cybersecurity Data Mining Competition [19] were strictly evaluated based on the F-Measure with respect to each class label to determine the overall rankings. However, the classification accuracy scores for the three tasks were also sent to the participants.

The team of authors finished 1st in pure accuracy and 2nd in the e-News text categorization task with the Domain-Specific classifier, see Table 3. Overall, the team received 3rd place in the entire competition.

4.4 Experiments on the Reuters 21578 Dataset

The Reuters 21578 dataset, consisting of documents from the Reuters newswire in 1987 and categorized by Reuters Ltd. and Carnegie Group, Inc., is a clas-

Table 3. Results of the Domain-Specific classifier for the e-News task

	Label 1	Label 2	Label 3	Label 4	Label 5	Accuracy	Task Ranking
F-Measure	0.847	0.943	0.991	0.805	0.947	-	2nd
Accuracy	-	-	-	-	-	0.912	1st

sical benchmark for text categorization classifiers [16]. To further test the effectiveness and efficiency of the Domain-Specific classifier, we considered single-category documents from the top eight most frequent classes (known as the R8 subset) from the Reuters 21578 dataset and divided according to the standard "modApté" training/testing split. These documents were downloaded from [4].

Table 4 provides the category sizes for this dataset. Standard pre-processing of the dataset consisted of removing all stop words and words of length two or less; afterwards, the size of the dictionary of all unique words from the training and testing documents was $m = 22931$ words.

Table 4. Information on the Reuter 21578 dataset considered

Label j	Topic	# of training documents	# of testing documents
1	acq	1596	696
2	crude	253	121
3	earn	2840	1083
4	grain	41	10
5	interest	190	81
6	money-fx	206	87
7	ship	108	36
8	trade	251	75
	Total	5485	2189

In this case, evaluation of the Domain-Specific classifier, based on the accuracy, F-Measure, and computational time, has shown that $\alpha = 0.45$ and $p = \infty$ (non-normalized measures on domain-specific words) were optimal. The SVM, using the linear kernel and a class weight adjustment (2840 divided by the number of documents in each category) to address the varying sizes of the categories, and Random Forest, using 50 trees, classifiers were also tested to compare against our novel algorithm. Table 5 provides the classification results obtained by the Domain-Specific classifier at those values, SVM, and Random Forest.

The Domain-Specific classifier performed slightly better than SVM with the linear kernel, and better than Random Forest in terms of accuracy. With respect to the F-Measure, our classifier performed better than SVM for categories with large sizes, and better than Random Forest in 6 of the 8 categories, while SVM had a higher F-Measure on two of the smaller categories, undoubtably due to SVM's class weight adjustment. Computationally, our classifier ran considerably faster than SVM and Random Forest.

Table 5. Classification performance of the Domain-Specific classifier (DSC) compared to SVM with the linear kernel and Random Forest (RF) on the Reuters 21578 dataset

	Accuracy	F1 acq	F1 crude	F1 earn	F1 grain	F1 interest	F1 money-fx	F1 ship	F1 trade
DSC $\alpha = 0.45$	**0.952**	**0.961**	**0.954**	0.978	0.800	**0.857**	**0.859**	**0.836**	0.807
SVM linear	0.946	0.948	0.913	0.970	**0.900**	0.834	0.844	0.833	**0.947**
Random Forest	0.926	0.917	0.911	**0.983**	0.462	0.775	0.556	0.326	0.877

Computational Time

	DSC	SVM linear	RF
Training stage	**6.6 secs**	1.43 hours	54.55 mins
Predicting stage	2.3 secs	55.9 secs	**1.08 secs**

5 Conclusion

In this paper, we have introduced a novel text categorization algorithm, the Domain-Specific classifier, based on similarity searches in the space of measures on the data dictionary. The classifier finds domain-specific words for each document category, which appear in this category relatively more often than in the rest of the categories combined, and associates to it a normalized uniform measure supported on the domain-specific words. For an unlabeled document, the classifier assigns to it the category whose associated measure is most similar to the document's vector of relative word frequencies, with respect to the inner product. The cosine similarity measure arises as a special case corresponding to the ℓ^2 normalization.

Our classifier involves a similarity search problem in a suitably interpreted domain. We believe that this is the right viewpoint with the aim of further improvements. It is worthwhile noting that our algorithm is unrelated to previously used distance-based algorithms (e.g. the k-NN classifier [18]). The dataset in the similarity workload is completely different, and as a result, unlike most algorithms in text categorization, this classifier does not require any separate dimension reduction step beforehand.

The process of selecting domain-specific words in our algorithm is actually an implicit feature selection method which is class-dependent, something we have not seen before from a classifier in text categorization. For each class, instead of a centroid, we are choosing an outlier, a uniform measure supported on the domain-specific words, which is representative of this class and not of any other class. Not only does each uniform measure lead to a reduction in the dimension of the feature space (as most words are not domain-specific) for similarity calculations, it does so dependent of the class labels, since domain-specific words are chosen relative to all classes.

This algorithm was first developed for the 2012 Cybersecurity Data Mining Competition and brought the team of authors 2nd place in the text categorization challenge, and 1st place in accuracy. This is evidence that our algorithm outperformed many existing text categorization algorithms, as surveyed in Section 2. In addition, our algorithm was evaluated on a sub-collection of the Reuters

21578 dataset against two state-of-the-art classifiers, and shown to have a slightly higher classification accuracy than SVM, with a higher F-Measure for the larger categories, and overall performed better than Random Forest. Computationally, our classifier ran significantly faster than either, especially in the training stage.

The normalizing parameter p plays a significant role: it is to account for class imbalance. When there are categories with very few documents, $p = \infty$ should be used to avoid over-emphasizing the smaller categories; and small values of p should be used when the categories have roughly the same number of documents.

For future work, we hope to test the Domain-Specific classifier on biological sequence databases. Other definitions of domain-specific words can be investigated, for instance the one proposed in [14]. We would like to experiment with assigning non-uniform measures on the domain-specific words, for instance, by putting weights based on their relative occurrences or on α. Finally, we would like to extend the process of selecting domain-specific words to a general classification context, by defining class-specific features relative to the classes and performing classification on only these class-dependent features.

References

1. Aas, K., Eikvil, L.: Text Categorization: A Survey. In: Technical Report 941. Norwegian Computing Center (1999)
2. Bingham, E., Mannila, H.: Random projection in dimensionality reduction: Applications to image and text data. In: Proceedings of 7th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, KDD 2001, San Francisco, USA, pp. 245–250 (2001)
3. Breiman, L.: Random Forests. Machine Learning 45(1), 5–32 (2001)
4. Cardoso-Cachopo, A.: Datasets for single-label text categorization, `http://web.ist.utl.pt/acardoso/datasets`
5. Church, K.W., Hanks, P.: Word association norms, mutual information and lexicography. In: Proceedings of ACL 27, Vancouver, Canada, pp. 76–83 (1989)
6. Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning 20, 273–297 (1995)
7. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Transactions on Information Theory 13, 21–27 (1967)
8. Deerwester, S., Dumais, S.T., Harshman, R.: Indexing by Latent Semantic Analysis. Journal of the American Society for Information Science 41(6), 391–407 (1990)
9. Dimitriadou, E., Hornik, K., Leisch, F., Meyer, D., Weingessel, A.: e1071: Misc functions of the Department of Statistics (e1071), TU Wien. R package version 1.6 (2011), `http://CRAN.R-project.org/package=e1071`
10. Forman, G.: An Extensive Empirical Study of Feature Selection Metrics for Text Classification. Journal of Machine Learning Research 3, 1289–1305 (2003)
11. Ikonomakis, M., Kotsiantis, S., Tampakas, V.: Text Classification Using Machine Learning Techniques. WSEAS Transactions on Computers 4(8), 966–974 (2005)
12. Joachims, T.: Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In: Nédellec, C., Rouveirol, C. (eds.) ECML 1998. LNCS, vol. 1398, pp. 137–142. Springer, Heidelberg (1998)
13. Johnson, D.E., Oles, F.J., Zhang, T., Goetz, T.: A decision-tree-based symbolic rule induction system for text categorization. IBM Systems Journal 41(3), 428–437 (2002)

14. Keim, D.A., Oelke, D., Rohrdantz, C.: Analyzing document collections via context-aware term extraction. In: Horacek, H., Métais, E., Muñoz, R., Wolska, M. (eds.) NLDB 2009. LNCS, vol. 5723, pp. 154–168. Springer, Heidelberg (2010)
15. Kim, S.B., Rim, H.C., Yook, D.S., Lim, H.S.: Effective Methods for Improving Naive Bayes Text Classifiers. In: Ishizuka, M., Sattar, A. (eds.) PRICAI 2002. LNCS (LNAI), vol. 2417, pp. 414–423. Springer, Heidelberg (2002)
16. Lewis, D.D.: Test Collections, Reuters-21578, `http://www.daviddlewis.com/resources/testcollections/reuters21578/`
17. Liaw, A., Wiener, M.: Classification and Regression by randomForest. R News 2(3), 18–22 (2002)
18. Lim, H.-S.: Improving kNN Based Text Classification with Well Estimated Parameters. In: Pal, N.R., Kasabov, N., Mudi, R.K., Pal, S., Parui, S.K. (eds.) ICONIP 2004. LNCS, vol. 3316, pp. 516–523. Springer, Heidelberg (2004)
19. Pang, P.S., Ban, T., Kadobayashi, Y., Song, J., Huang, K.: The 3rd Cybersecurity Data Mining Competition (2012), `http://www.csmining.org/cdmc2012`
20. Porter, M.F.: An algorithm for suffix stripping. Program 14(3), 130–137 (1980)
21. R Development Core Team: R: A Language and Environment for Statistical Computer. R Foundation for Statistical Computing, Vienna, Austria (2008), `http://www.R-project.org` ISBN 3-900051-07-0
22. Radovanovic, M., Ivanovic, M.: Text Mining: Approaches and Applications. Novi Sad J. Math 38(3), 227–234 (2008)
23. Salton, G., McGill, M.J.: An Introduction to Modern Information Retrieval. McGraw-Hill (1983)
24. Salton, G., Wong, A., Yang, C.S.: A Vector Space Model for Automatic Indexing. Communications of the ACM 18(11), 613–620 (1975)
25. Schölkopf, B., Smola, A.: A Short Introduction to Learning with Kernels. In: Mendelson, S., Smola, A.J. (eds.) Advanced Lectures on Machine Learning. LNCS (LNAI), vol. 2600, pp. 41–64. Springer, Heidelberg (2003)
26. Schütze, H., Hull, D.A., Pedersen, J.O.: A Comparison of Classifiers and Document Representations for the Routing Problem. In: Proceedings of 18th ACM International Conference on Research and Development in Information Retrieval, SIGIR 1995, Seattle, USA, pp. 229–237 (1995)
27. Sebastiani, F.: Machine Learning in Automated Text Categorization. ACM Computing Surveys 34, 1–47 (2002)
28. Torkkola, K.: Linear Discriminant Analysis in Document Classification. In: Proceedings of 2001 IEEE ICDM Workshop on Text Mining, ICDM 2001, San Jose, USA, pp. 800–806 (2001)
29. Weichold, M., Huang, T.W., Lorentz, R., Qaraqe, K.: The 19th International Conference on Neural Information Processing, ICONIP 2012 (2012), `http://www.iconip2012.org`
30. Yang, Y., Liu, X.: A re-examination of text categorization methods. In: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1999, Berkeley, USA, pp. 42–49 (1999)
31. Yang, Y., Pedersen, J.O.: A Comparative Study on Feature Selection in Text Categorization. In: Proceedings of 14th International Conference on Machine Learning, ICML 1997, Nashville, USA, pp. 412–420 (1997)
32. Zeng, X.Q., Wang, M.W., Nie, J.Y.: Text Classification Based on Partial Least Square Analysis. In: Proceedings of ACM, Seoul, Korea, pp. 834–838 (2007)

Efficient Approximate Indexing in High-Dimensional Feature Spaces

Simone Santini [*]

Escuela Politécnica Superior
Universidad Autónoma de Madrid

Abstract. In this paper we present a fast approximate indexing method for high dimensional feature space that uses the error probability as an independent variable.

The idea of the algorithm is to define a low-dimensional feature space in which a significant portion of the inter-distance variance is concentrated, to search for the nearest neighborhood of the query in this space, and then to extend the search by a factor ζ to include a number of objects "near" this nearest neighborhood. We shall show that, under reasonable hypotheses on the distribution of items in the feature space, it is possible to derive a relation between the value ζ and the error probability.

We study the error probability and the complexity of the algorithm, validate the model using a data set of images, and show how the results can be used to design indexing schemes.

1 Introduction

Attempts to create fast indices in high dimensional metric feature spaces have been (possibly inevitably) hampered by the peculiar characteristics of the metric of such spaces. In particular, given a data base of N elements $D = \{q_1, \ldots, q_N\}$ in the space, a point x, and the distance between x and its nearest neighbor $\bar{\Delta} = \min_{q \in D}\{\Delta(x, q)\}$, it is known that a fraction $1 - \epsilon$ of the elements of D will be within a distance $\bar{\Delta} + \epsilon$ from x, where $\epsilon = \Theta(1/N)$ [11].

This phenomenon, colloquially known as the *curse of dimensionality* all but prevents the efficient application of *divide and conquer* techniques that partition the space. Tree-based techniques such as R- and R^*-trees [9,2], K-D-trees [3], SS-trees [13], M-trees [6], etc. or techniques based on variations of Linear Hashing such as *Interpolation-Based Index Maintenance (IBIM)* [4] perform reasonably well for spaces of limited dimensionality, but their performance degrades as the dimensionality of the space increases and, in very high dimensional spaces such as those typical of multimedia data, they are basically equivalent to a linear search, that is, they entail the determination of the distance between the query and all elements in the data base.

Finding the actual nearest neighborhood in high dimensional feature spaces seems inherently hard [5], and there has been a certain interest in approximate algorithms.

[*] This work was supported by the *Ministerio de Educacion y Ciencia* under the grant N. TIN2011-28538-C02, *Novelty, diversity, context and time: new dimensions in next-generation information retrieval and recommender systems.*

Approximate nearest neighbor algorithms [1] guarantee that they will return a point at a distance at most $(1+\epsilon)$ times that of the nearest neighbor, but have been shown to suffer the curse of dimensionality as well [7]. Probably Approximately Correct (PAC) algorithms [7] return a point with a relative error of $(1+\epsilon)$ with probability $(1-\delta)$, and ar more immune from the curse of dimensionality. The algorithm presented in this paper belongs to this family, but it is based on different assumptions than other common algorithms in this class. The algorithm in [7], for example, perform search in the full space, and gains efficiency by finding an early candidate and then using the probability distibution of the distance to cut further search. Here we use a *filtering* approach, similar to that of [12]: we perform a search in a low-dimensional feature space to retrieve a set of *candidates* among which we search the nearest neighbor in the full feature space. We show that, under reasonable hypotheses on the distribution of the points in the space, this allows us to guarantee that the nearest neighbor is found with a certain probability and that, in case it is not found, its distance from the actual nearest neighbor is limited with a given probability.

2 The Approximate Indexing Model

Consider a data base with N objects, each described by a m-dimensional feature vector. A standard procedure in data base indexing is to rotate the feature space in such a way that the variance of the data base is concentrated in a relatively small number of axes. Let $D \in \mathbb{R}^{N\times d}$ a matrix containing all the feature vectors. We can apply singular value decomposition [8] obtaining a decomposition $D = U\Sigma V'$, where $\Sigma = \mathrm{diag}(\sigma_1,\ldots,\sigma_d)$ is a $d\times d$ diagonal matrix containing the eigenvalues of D ($\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_d$), V is a $d\times d$ matrix containing the corresponding eigenvectors in the columns, and U is a $N\times d$ orthogonal matrix.

The matrix V defines a transformation into a new feature space in which the data base has variance σ_i in the ith axis. Since the eigenvalues are in descending order, most of the variance is concentrated in the lowest dimensions of the space. In many applications, this property is used for dimensionality reduction, discarding the last dimensions and using only the first ones as a reduced feature space.

This facilitates indexing, as it reduces the dimensionality of the feature space, but it presents two main disadvantages:

i) in order to guarantee acceptable results, the dimensionality reduction is in general insufficient to guarantee efficient search; that is, even in the reduced space, search is linear, and the algorithm must analyze the whole data base in order to determine the nearest neighbor;

ii) discarding the higher elements may result, in certain cases, in retrieving a false nearest neighbor; this is not necessarily a problem in most application (the method that we present here does the same), but a simple pruning of the feature space does not allow the designer to control or at least determine the probability of error.

The method that we present here is based on the same idea of operating in spaces of reduced dimensionality, but it permits to operate a first search on a much smaller space (thereby taking better advantage of fast search methods), and permits the designer to tune the method to reach a controllable compromise between efficiency and precision. We partition the feature space F into two orthogonal subspaces X and Y (viz. $F = X \oplus Y \approx X \times Y$) where X contains the first few axes of the transformed space, with a variance σ_X in the distribution of features, and Y contains the rest of dimensions with a variance σ_Y. We choose the dimension of X in such a way that

$$\nu = \frac{\sigma_X^2}{\sigma_Y^2} > 1 \tag{1}$$

We then proceeds as follows:

i) given a query point q, we find the nearest neighbor n using only the feature space X; let $d(q, n)$ be the distance between q and n in the sub-space X;

ii) we increase the square of the distance by an amount α, and we do a range query on the space X again for all items x such that $d^2(q, x) \le d^2(q, n) + \alpha$; let T be the set of items resulting from this search, i.e.

$$T_q = \{x | d^2(q, x) \le d^2(q, n) + \alpha\} \tag{2}$$

iii) We use the whole space, with a distance measure Δ, to find the closest neighbor to q in T_q:

$$r = \arg\min_{x \in T_q} \Delta^2(q, x) \tag{3}$$

If X has a small dimensionality, the first and the second steps can be done in time $|T_q| \log N$, while the third requires at most time $|T_q|$.

The hypothesis that we make is that, if $\nu > 1$ (viz. if the variance of the data base is sufficiently concentrated in X), most of the times the true nearest neighbor will be close enough to the one that we estimated in step i) using only the distance function of X, therefore even with a small value of α (that is, with a small T_q), we shall be able to retrieve the true nearest neighbor with sufficient probability.

Let distances be Euclidean. Let $d : X \times X \to \mathbb{R}^+$ be the distance in the space X, $\delta : Y \times Y \to \mathbb{R}^+$ the distance function in Y, and $\Delta : F \times F \to \mathbb{R}^+$ the distance function in F, with $\Delta^2 = d^2 + \delta^2$.

In step i), we do a search on X, and we find a "bogus" nearest neighbor, b, at a distance $\Delta^2(q, b) = \Delta_b^2 = d_b^2 + \delta_b^2$. Suppose we have missed the true nearest neighbor. Then there is an item t at a distance $\Delta_t^2 = d_t^2 + \delta_t^2$ with $\Delta_t^2 < \Delta_b^2$.

We have found b as the nearest neighborhood on X, which entails $d_b \le d_t$. Not only, we have checked all items that, on X, were within a distance $d_b^2 + \alpha$ from the query. Had the true nearest neighbor been within this distance on X, it would have been included in T_q and in step iii) we should have found it. If we have made a mistake, then $t \notin T_q$, that is, $d_t^2 > d_b^2 + \alpha$. The situation is illustrated in figure 1. The true nearest neighbor, t, is, in X, at a distance $d_t^2 > d_b^2 + \alpha$, therefore will not be included in T_q and will not be detected. Nevertheless, because of its very small component δ_t, its distance from the query is smaller than that of the detected nearest neighbor.

We are interested in the probability that this happens, that is, in the probability

$$\mathbb{P}_E = \mathbb{P}\{\Delta_t^2 < \Delta_b^2 | d_t^2 - d_b^2 > \alpha\} = \mathbb{P}\{d_t^2 - d_b^2 < \delta_b^2 - \delta_t^2 | d_t^2 - d_b^2 > \alpha\} \tag{4}$$

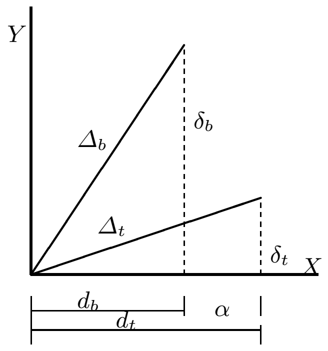


Fig. 1. A situation in which the algorithm leads to a wrong detection of nearest neighbor: the nearest neighbor that is being detected on X is b, with a distance $\varDelta_b$ from the query. In X, the true nearest neighbor is at a distance greater than $d_b^2 + \alpha$ from the query, so it is not retrieved. Nevertheless, the component δ, which is not considered in the determination of T_q, is much smaller than that of the detected nearest neighbor, so that $\varDelta_t < \varDelta_b$.

If the objects have Gaussian distribution in the data base, the distances have a χ^2 distribution with a number of degrees of freedom that depends on the dimension of the respective sub-spaces[1]. We shall make a significant approximation, and assume that the distances have a χ^2 distribution with two degrees of freedom, that is, that they are exponentially distributed. The variance of these distributions is the variance of the data bases in the respective sub-spaces, i.e.

$$p_d(u) = \frac{1}{2\sigma_X^2}\exp(-\frac{u}{2\sigma_X^2}) \quad p_\delta(u) = \frac{1}{2\sigma_Y^2}\exp(-\frac{u}{2\sigma_Y^2}) \tag{5}$$

The differences of distances that appear in $\mathbb{P}_E$ have a Laplace distribution:

$$\begin{aligned} p_{\Delta d}(u) &= \frac{1}{4\sigma_X^2}\exp(-\frac{|u|}{2\sigma_X^2}) \\ p_{\Delta\delta}(u) &= \frac{1}{4\sigma_Y^2}\exp(-\frac{|u|}{2\sigma_Y^2}) \end{aligned} \tag{6}$$

So we have

$$\begin{aligned} \mathbb{P}_E &= \int_\alpha^\infty dx p_{\Delta d}(x) \int_x^\infty dy p_{\Delta\delta}(y) \\ &= \int_\alpha^\infty dx \frac{1}{4\sigma_X^2}\exp(-\frac{x}{2\sigma_X^2}) \int_x^\infty dy \frac{1}{4\sigma_Y^2}\exp(-\frac{y}{2\sigma_Y^2}) \\ &= \frac{1}{2}\frac{\sigma_Y^2}{\sigma_X^2+\sigma_Y^2}\exp\left[-\frac{1}{2}\frac{\sigma_X^2+\sigma_Y^2}{\sigma_Y^2}\frac{\alpha}{\sigma_X^2}\right] \end{aligned} \tag{7}$$

[1] Not all data sets have a Gaussian distribution. However, as we shall see when comparing with sampled data, in many cases the distributions that we obtain assuming a Gaussian distribution do have a reasonable fit with the distribution of the actual data.

The parameter α is the additional distance that we consider in the space X. The parameter $\zeta = \alpha/\sigma_X^2$ can be seen as a distance expressed in units of σ_X^2, that is, as a form of natural distance, a number that does not depend on the unit that we use to measure distances and coordinates in X, but only on the distribution of the data base. With this definition we have:

$$\mathbb{P}_E = \frac{1}{2(1+\nu)} \exp\left[-\frac{1+\nu}{2}\zeta\right] \tag{8}$$

If we look for the nearest neighbor in X and then we "peek" forward for a (natural) distance ζ, this is the probability that we will make a mistake in the determination of the nearest neighbor.

In general, we are more interested in using this formula the other way around: given a target error probability p, we want to know how far we have to *peek* in the data base in order to attain it. This value is given by

$$\zeta = \frac{2}{1+\nu} \log \frac{1}{2(1+\nu)p} \tag{9}$$

3 Complexity

if X is a low-dimensional feature space, the search for the nearest neighbor in X takes time $\log N$. If there are m objects within a distance ζ from the nearest neighbor, we can find them in time m (this can be done, for example, using methods based on linear hashing [4]). A further time m goes into finding the nearest neighbor in the whole feature space among these objects (we assume that the whole feature space is high dimensional, so that no indexing method performs better than linear scan). The total execution time is therefore $\log N + 2m$. The first term is fixed, so in order to determine the complexity of the algorithm we have to estimate the value of m. We find the nearest neighbor at a distance d_b^2 from the query, and we keep searching for a distance α. That is, we put an element u in T_q if $d_u^2 - d_b^2 < \alpha$. With the distribution for the difference of distances given in 6, we have

$$\begin{aligned}
\mathcal{P}\{d_u^2 - d_b^2 < \alpha | d_u^2 > d_b^2\} &= \frac{\mathcal{P}\{d_u^2 - d_b^2 < \alpha, d_u^2 > d_b^2\}}{\mathcal{P}\{d_u^2 > d_b^2\}} \\
&= 2\frac{1}{4\sigma_X^2}\int_0^\alpha dx \, \exp[-\frac{x}{2\sigma_X^2}] \\
&= 1 - \exp[-\frac{\alpha}{2\sigma_X^2}] \\
&= 1 - \exp[-\frac{\zeta}{2}] \\
&\overset{\text{def}}{=} \pi(\zeta)
\end{aligned} \tag{10}$$

The probability that in a data base of N elements, n are at a distance greater than ζ is then

$$\begin{aligned}
\mathbb{P}_\zeta[n] &= \binom{N}{n} \pi(\zeta)^n (1-\pi(\zeta))^{N-n} \\
&\approx \frac{(N\pi(\zeta))^n}{n!} \exp[-N\pi(\zeta)]
\end{aligned} \tag{11}$$

where the Poisson approximation is valid for N large. Then

$$m(\zeta) = \mathbb{E}[\mathbb{P}_\zeta[n]] = N\pi(\zeta) = N\left(1 - \exp\left(-\frac{\zeta}{2}\right)\right) \tag{12}$$

If we are interested in the complexity for a prescribed error probability p, we plug in eq. (9) obtaining:

$$\begin{aligned} m(p) &= N\left(1 - \exp\left(-\frac{1}{1+\nu}\log\frac{1}{2(1+\nu)p}\right)\right) \\ &= N\left[1 - (2(1+\nu)p)^{\frac{1}{1+\nu}}\right] \\ &\stackrel{\text{def}}{=} N\mu(\nu, p) \end{aligned} \tag{13}$$

The value $\mu(\nu, p)$ represents the fraction of the data base that is necessary to search in order to obtain the nearest neighborhood with probability p.

$*$ $*$ $*$

Before proceeding to the validation of the model and to some general considerations on its predictions, we summarize here the results obtained. Remember that we divide the search space into a low-dimensional one in which the distance measures have a variance σ_X^2 and a high dimensional one in which the variance is $\sigma_Y^2 < \sigma_X^2$.

i) If we search the nearest neighbor in the low-dimensional space and we extend the search for a distance $\alpha = \zeta\sigma_X^2$ we find the nearest neighbor with a probability:

$$\mathbb{P}_S(\zeta) = 1 - \mathbb{P}_E(\zeta) = 1 - \frac{1}{2(1+\nu)}\exp\left(-\frac{1+\nu}{2}\zeta\right) \tag{14}$$

ii) Correspondingly, if we want to find the nearest neighbor with probability $1 - p$ we shall have to find it X and then extend the search up to a distance $\alpha = \zeta\sigma_X^2$, with

$$\zeta = \frac{2}{1+\nu}\log\frac{1}{2(1+\nu)p} \tag{15}$$

iii) Given a value of ζ, the average complexity of the search (assuming that search is efficient in the low-dimensional space) is

$$C(\zeta) = \log N + N\left(1 - \exp\left(-\frac{\zeta}{2}\right)\right) \tag{16}$$

iv) If we want to find if we want to find the nearest neighbor with probability $1 - p$, the complexity is

$$C(p, \nu) = \log N + N\left[1 - (2(1+\nu)p)^{\frac{1}{1+\nu}}\right] = \log N + N\mu(\nu, p) \tag{17}$$

4 Validation

For the purpose of model validation, we use the *Im2Text* data base [10]. The data base contains 1.000.000 annotated images, although in this case we didn't use the captions. All the statistical results are based samples of 10.000 images selected at random from the data base, with several independent sampling used in order to confirm that the values are significant.

As a feature vector, we used a block histogram on the luminance channel of the images. Images were resized to a standard size of 1024 by 1024 and divided into blocks of size 128 (8 by 8 blocks). For each block, the mean and variance of the histogram were computed and united to form a feature vector of 128 double precision numbers. Singular value decomposition was then applied so as to obtain a feature space (the *transformed space*) in which the variance was concentrated in a relatively small number of dimensions.

The distributions of interests for this data base are shown in figure 2 for one of the samples. All the distribution have been obtained through random samples of the data set [10] and cross-validated using other data sets: they are highly representative of the distributions that one obtains when applying singular value decomposition to actual data sets in a wide range of multimedia applications (which are the ones that motivated this work), and reasonably representative of other high-dimensional data sets.

Figure 2(a) shows the singular values obtained through singular value decomposition; figure 2(b) shows the distribution of the distance d^2 between images (computed over the whole feature space), and figure 2(c) shows the distribution of the distance difference $d_i^2 - d_j^2$. The latter is superimposed to the theoretical model that we use in this paper.

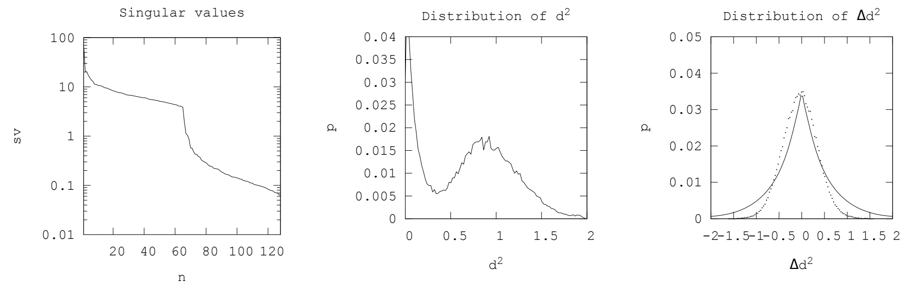


Fig. 2. The distributions of interest for the validation set: the singular values (a), the distribution of the inter-image distances d^2 (b), and the distribution of the distance differences $d_i^2 - d_j^2$ (c). The distance differences are superimposed to the theoretical Laplace distribution, for comparison purposes.

The coincidence is not perfect (for high dimensional feature spaces the actual difference is a Normal distribution). Moreover, the distribution of d^2 doesn't look at all like the exponential that the model hypothesizes. The distribution of d^2 for features of reduced dimensions (the first n components of the features in the transformed space) is shown in figure 3 This is not necessarily a problem for our model, as the distribution

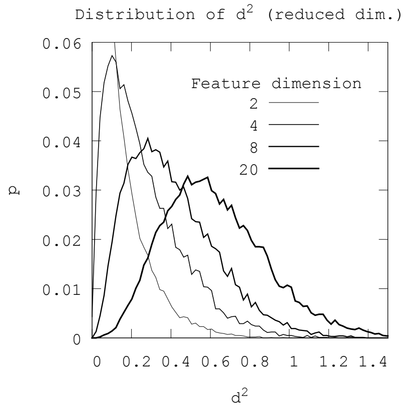


Fig. 3. The distribution of the square of the inter-image distance (d^2) for feature spaces of reduced dimensionality n

of d^2 never enters the equations: only the distribution of Δ^2 does and this, even in the worst case of a high dimensional feature space, is close enough to the actual distribution so that the advantage of a closed-form solution, afforded by the Laplace distribution outweighs the disadvantages of the imprecision of the model.

Figure 4 shows the comparison between the theoretical value of the error probability and the measured one as a function of ν for several values of ζ.

The model underestimates somewhat the actual performance for the highest values of ν (viz., in this experiments, when the initial search is done in a space of dimension 10 or more), as the experiment always finds the nearest neighbor. . As ν increases, the theoretical model converges rapidly to 1, so the model and the measurements converge once again (albeit trivially: the area of very high ν will result in searches that are just as inefficient as those carried out on the whole feature vector).

We have attempted a validation of the search complexity (viz. of the number $m(\zeta)$). The results obtained were formally significant, but the variance of the number of objects actually retrieved in the data base was so high that almost any reasonable estimation of m would have been correct. This might indicate that the complexity depends on parameters other than ν and ζ, parameters that, left uncontrolled in our validation, cause the high variance. For the purposes of the following section, we shall take a formalist point of view and assume that, since the prediction is well within the variance of the data, it can be assumed to be correct.

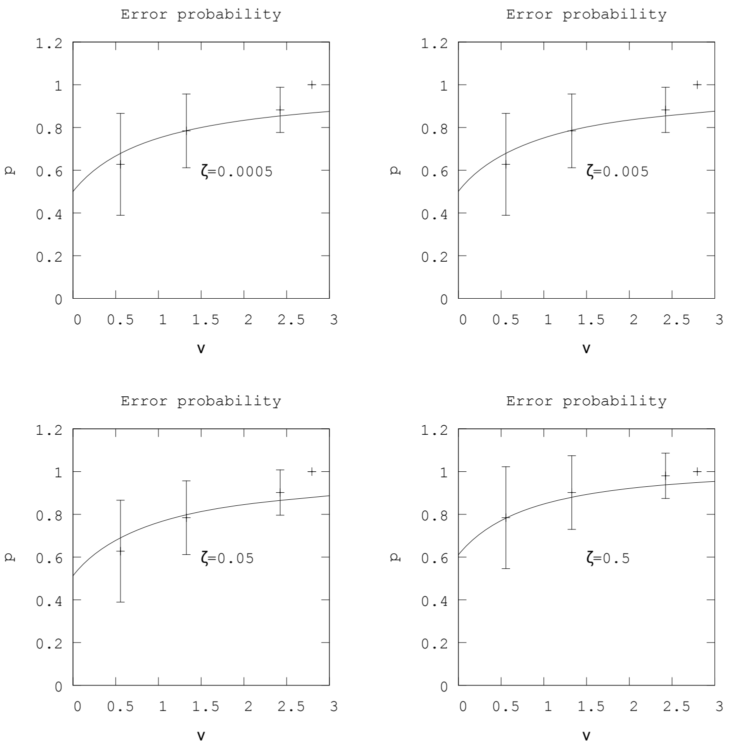


Fig. 4. Comparison between the error probability in finding the nearest neighbor predicted by the model and that measured on the Im2Text data base using the features described in the paper

5 Some Analytical Considerations

We begin by noticing that, in equation (14), if $\zeta = 0$, then

$$
\begin{aligned}
\mathbb{P}_E(0) &= \frac{1}{2(1+\nu)} \overset{\text{def}}{=} \mathbb{P}^{\uparrow}_E(\nu) \\
\mathbb{P}_S(0) &= \frac{1+2\nu}{2(1+\nu)} \overset{\text{def}}{=} \mathbb{P}^{\downarrow}_S(\nu)
\end{aligned}
\tag{18}
$$

$\mathbb{P}^{\downarrow}_S(\nu)$ is the minimal success probability (for a given value of ν) that requires looking beyond the nearest neighbor found in the first step of the algorithm. If the desired success probability is less than $\mathbb{P}^{\downarrow}_S(\nu)$ (or, equivalently, if the desired error probability is higher than $\mathbb{P}^{\uparrow}_E(\nu)$) then one can simply search for the nearest neighbor in the reduced feature space. This cut probability is shown in table 1 for various values of ν. As expected, for high values of ν it is often a good strategy to simply ignore the rest of the feature space (this is what $\zeta = 0$ amounts to) and do the search in the reduced space. For smaller ν and/or if a very high success probability is desired, then the full algorithm should be applied.

Table 1. The probability of success $\mathbb{P}_S^{\downarrow}$ beyond which it is convenient just to search the reduced feature space, as a function of ν

ν	0.2	0.5	1	2	4	8	16	20
$\mathbb{P}_S^{\downarrow}$	0.583	0.667	0.750	0.833	0.900	0.944	0.971	0.976

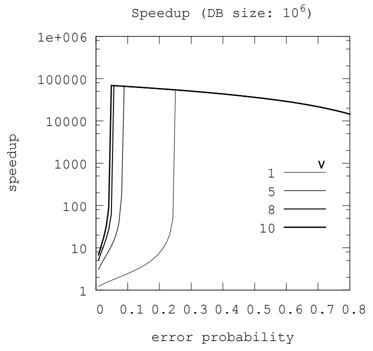


Fig. 5. Speedup of the method proposed here with respect to the baseline. The saturation point depends on the size of the data base, being roughly proportional to $N/\log N$. The values shown are for $N = 10^6$. After saturation the curve decreases as $N(1-p)/\log N$.

In order to compare the proposed method with a baseline, we consider a very simple method that also finds the nearest neighbor with a prescribed probability. The method is simply this: given the desired error probability p, reduce the data base considering a random sample of $N(1-p)$ objects, and search the nearest neighbor only in this reduced data base. It is easy to see that the search will yield the actual nearest neighbor with a probability p and even with a linear method, we have reduced the search time by a factor $(1-p)$.

The complexity of the method proposed here, for a prescribed error probability, is given by eq. (17). The speed-up of this method, with respect to to the baseline, is:

$$\rho(p,\nu) = \frac{N(1-p)}{\log N + N\mu(p,\nu)} = \frac{1-p}{\frac{\log N}{N} + \mu(p,\nu)} \tag{19}$$

For large data bases, $(\log N)/N$ is very small, so we can approximate the speed-up as

$$\rho(p,\nu) = \frac{1-p}{\mu(p,\nu)} = \frac{1-p}{1-(2(1+\nu)p)^{\frac{1}{1+\nu}}} \tag{20}$$

It is easy to see that, for all $\nu > 0$, $(2(1+\nu)p)^{\frac{1}{1+\nu}} > p$, from which we derive $\rho(p,\nu) > 1$, i.e. using the low-dimensional space as a filter always yields better performance (apart from the negligible factor $(\log N)/N$). Figure 5 plots the logarithm of the speed-up as a function of p for various values of ν. Note that in the graph the value of ρ as given

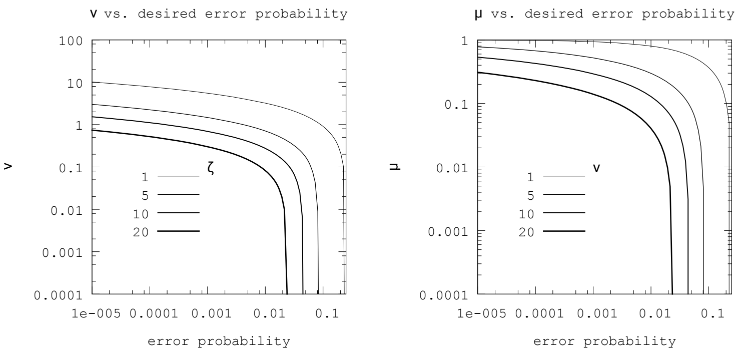


Fig. 6. Values of ζ (a) and μ (b) as a function of the desired error probability for several values of ν

by (20) goes to infinity. This doesn't happen in practice because, when $\mu(p, \nu)$ becomes zero, that is, for $p = \mathbb{P}_S^{\downarrow}(\nu)$, the term $(\log N)/N$ dominate. The speed up, therefore, increases with p according to (20) for $p < \mathbb{P}_S^{\downarrow}(\nu)$, and has a value $\rho = N(1-p)/\log N$ for $p > \mathbb{P}_S^{\downarrow}(\nu)$.

Finally, we propose some of the data to be used to the design of indexing solution using the method introduced here. For the designer, the most important independent variable is the error probability p. Given a system with a low dimensional space (with an efficient index) and a desired error probability p, the designer will derive the value of ζ used in the second step of the method of page 196 and then will estimate the complexity of the method, that is, the value of μ. Figure 6.a shows the predicted values of ζ as a function of the desired error probability for given values of ν. Figure 6.b shows the value of μ (which determines the execution time of the algorithm) as a function of the desired error probability for various values of ν.

6 Some Final Remarks

One of the major uncertainties of the model, as it stands now, is the high variance measured in the measurement of the number of accessed data items as a function of ζ for a given value of ν. This makes the model formally correct, but it makes its complexity predictions quite imprecise. The most likely explanation is that the complexity depends on variables other than ν and ζ and the fact that these variables are not controlled causes the high variance. A more detailed model is necessary to reveal what are these variables.

The algorithm, as it is, begins by looking for the nearest neighbor in the space X. This element is needed only in order to determine its distance d^2 from the query so that we can determine the range to search in X as $d^2 + \zeta\sigma_X^2$. It should be possible to avoid this step–and consequently eliminate the factor $\log N$ from the complexity expression–by deriving a statistical characterization of d^2. This would make the algorithm faster for a given value of ζ, but the additional uncertainty introduced by the statistical determination will probably require a larger ζ for a given target error probability. It will be necessary to analyze whether, and in what circumstances, this is advantageous.

References

1. Arya, S., Mount, D.M., Netanyahu, N.S., Silverman, R., Wu, A.Y.: An optimal algorithm for approximate nearest neighbor searching in xed dimensions. Journal of the ACM 45(6), 891–923 (1998)
2. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The r^*-tree: an efficient and robust access method for points and rectangles. In: Proceedings of the ACM SIGMOD International Conference on the Management of Data (1990)
3. Bentley, J.L.: K-D trees for semidynamic point sets. In: Proceedings of the Sixth ACM Annual Symposium on Computational Geometry (1990)
4. Burkhard, W.: Interpolation/based index maintenance. In: Proceedings of the ACM Symposium on Principles of database systems (PODS), pp. 76–89 (1983)
5. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Computing Surveys (CSUR) 33(3), 273–321 (2001)
6. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: Proceedings of the 23rd VLDB (Very Large Data Bases) Conference, Athens, Greece (1997)
7. Ciaccia, P., Patella, M.: Pac nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In: Proceedings of the International Conference on Data Engineering, ICDE (2000)
8. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. Journal of the American Society for Information Science 41(6), 391–407 (1990)
9. Gravila, D.M.: The R-Tree index optimization. In: Waugh, T., Healey, R. (eds.) Advances in GIS Research. Taylor & Francis (1994)
10. Ordoñez, V., Kulkarni, G., Berg, T.L.: Im2text: Describing images using 1 million captioned photographs. Neural Information Processing Systems (2011)
11. Pestov, V.: On the geometry of similarity search: dimensionality curse and concentration of measure. Information Processing Letters 73(1-2), 47–51 (2000)
12. Seidl, T., Kriegel, H.-P.: Optimal multi-step k-nearest neighbor search. In: Proceedings of the ACM SIGMOD, International Conference on Management of Data, Seattle, USA, pp. 154–165 (1998)
13. White, D., Jain, R.: Similarity indexing with the SS-tree. In: Proc. 12th IEEE International Conference on Data Engineering (1996)

Semi-supervised Tag Extraction in a Web Recommender System

Vasily A. Leksin[1] and Sergey I. Nikolenko[1,2,3]

[1] Surfingbird LLC, ul. Bazhova, d. 24, korp. 2, 129128 Moscow, Russia
`vasily.leksin@gmail.com`
[2] Laboratory of Internet Studies,
National Research University Higher School of Economics,
ul. Soyuza Pechatnikov, d. 16, 190008 St. Petersburg, Russia
[3] Steklov Mathematical Institute, nab. r. Fontanka, 27, St. Petersburg, Russia
`sergey@logic.pdmi.ras.ru`

Abstract. An important characteristic feature of recommender systems for web pages is the abundance of textual information in and about the items being recommended (web pages). To improve recommendations and enhance user experience, we propose to use automatic tag (keyword) extraction for web pages entering the recommender system. We present a novel tag extraction algorithm that employs semi-supervised classification based on a dataset consisting of pre-tagged documents and (for the most part) partially tagged documents whose tags are automatically mined from the content. We also compare several classification algorithms for tag extraction in this context.

Keywords: tag extraction, recommender systems, text mining.

1 Introduction

One characteristic feature of recommender systems dealing with web pages (e.g., *Surfingbird* or *StumbleUpon*) is that unlike many classical recommender systems with media or product items (Netflix, Amazon etc.) web pages as recommendation objects contain a lot of textual information. This textual information can augment pure collaborative filtering methods to improve recommendations, especially in cold start situations, and enhance user experience. One important aspect of improving user experience with content information is tagging. Tags let users easily find similar objects, quickly assess a new object and so on.

In one typical scenario of using tags in recommender systems, the tags are already in existence, and the purpose is to make recommendations, either product [1, 2] or social [3], by an existing set of tags assigned to users and/or items. Another class of algorithms includes using variations of collaborative filtering algorithms, based on either cooccurrence analysis [4], nearest neighbors [5], or matrix and/or tensor factorizations [6, 7], to recommend tags for users to apply to an item (e.g., for further use with a recommender system).

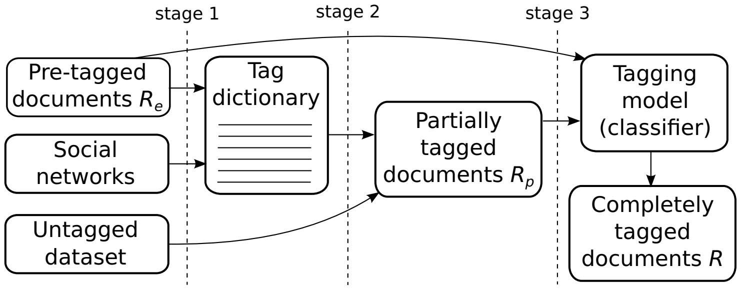

stage 1
stage 2
stage 3
Pre-tagged documents R_e
Social networks
Untagged dataset
Tag dictionary
Partially tagged documents R_p
Tagging model (classifier)
Completely tagged documents R

Fig. 1. General scheme of our tag extraction approach

However, in the case of a web page recommender system like *Surfingbird* the problem is not how to utilize an already existing wealth of information for recommendations but rather how to assign tags to web pages for further use: there are many pages, and collaborative filtering algorithms are oriented towards the "heavy tail" that would find personalized recommendations for each user. Thus, we cannot assume that users and/or moderators will cover the database with tags all by themselves, or at the very least we have to begin with a sufficiently good tag coverage for the existing database and then rely upon existing tag recommendation algorithms.

Most systems that use tags rely on either users or external structured ontologies (e.g., a movie database) to provide them and usually face problems like fuzzy string searching to cope with misprints and spelling variations. However, in the context of a large-scale content-based recommender system for web pages (or other texts) user-created tags are insufficient: new web pages are mostly added automatically, and a user who has already read a recommended web page is unlikely to spend time to tag it. Therefore, in this context we have to solve a harder problem: suggest tags for web pages based on content analysis for web pages and a "seed" training dataset R_e of tags for web pages that come pre-tagged (e.g., from well kept RSS streams). We call this problem *semi-supervised tag extraction*. In this work, we present a novel approach to semi-supervised tag extraction for web pages as recommender system objects and show promising empirical results.

2 Tag Extraction

Our algorithm for automated tag extraction consists of three main stages, as shown on Fig. 1:

(1) extract tags from the pre-tagged part of the dataset R_e and social networks;
(2) perform partial tag labeling for the untagged part of the datasets based on key phrase occurrence, getting a partially tagged dataset R_p;

Gadgets	Games	Books	Music	Movies
android	assassin creed	stories	bahh tee	the matrix
hardware	video games	albert camus	britney spears	pearl harbor
google	rally	o. henry	whitney houston	sherlock holmes
software	development	ryunosuke akutagawa	george watsky	apocalypse now
iphone	reviews	audiobook	rap	titanik
samsung	call of duty	steve jobs	slipknot	ocean's thirteen
apple	star wars	arkady gaidar	emma hewitt	comedy
ios	half-life	pierre gamarra	james blunt	south tablet
park pc	releases	biography	ellie white	avatar
smartphones	angry birds	guy endore	izzy johnson	the green mile

Fig. 2. Most popular tags from some of Surfingbird's categories (mostly translated from Russian)

(3) learn a tagging model (classifier) from $R_e \cup R_p$ and apply it to R_p, getting a completely tagged dataset as well as a model ready to tag new resources (web pages).

In this section, we deal with the first two stages.

On the first stage, *tag dictionaries* are constructed. In the Surfingbird recommender system, web pages are divided into thematic categories (e.g., "Gadgets", "Games", "Books", "Music", "Movies", or "TV"), so we construct dictionaries for each category independently. The first source of tags is the "seed" training dataset, in which the correct tags are provided by publishers in RSS streams. For many important categories, there is also a very convenient source of tags in the form of social networks. Many (about half) users log in through social networks (the most popular ones in Russia being *vkontakte* and *facebook*), and it becomes possible to mine information about a user's interests. Thus, we can extract a list of tag candidates provided by all users interested in a specific thematic category. After dictionaries have been constructed for each thematic category, we prune uninformative tags, i.e., tags that are either too rare or too popular (we choose a threshold for that purpose). We take the union of social networks tags and predefined tags as the total vocabulary of tags; a sample of the most popular tags in some popular categories is given in Fig. 2.

On the second stage, we search for occurrences of tags in the content of web pages for which tags are unknown (note that both individual words and phrases can become tags). To do so, we extract useful textual content from each web page, transform the tag phrase into a search query by making a conjunction of all words and use text search to find the corresponding web pages. Note that this step can be efficiently implemented on the database level with, e.g., the PostgreSQL full text search feature. For each search result, we check whether all words indeed occur as the tag phrase (e.g., that the document actually mentions "Stephen King" rather than Stephen Fry making fun of Elvis Presley); due to possible variations and different syntactic forms, this is done with an inexact string matching algorithm that would be infeasible to run on each document.

As a result of the first two stages, we get a list of tags T and a training dataset of resources R. The dataset consists of two parts: pre-tagged web pages R_e in the "seed" dataset where we know the tags exactly and partially tagged web pages R_p whose known tags are extracted from occurrences in content. However, there are many useful tags that can be assigned to a web page but do not actually occur on it; e.g., a web page from the "Books" category devoted to *Alice in Wonderland* could be supplied with tags "children's literature", "classical literature", or "Lewis Carroll" even if these phrases do not actually appear in the text. This is the subject of the third, main step in our approach.

3 Tag Recommendation

Consider the set $F = (T, R, Y)$, where T is a finite set (dictionary) of tags, $R = R_e \cup R_p$ is a finite set of resources (web pages) divided into exactly tagged resources R_e and partially tagged resources R_p, and $Y \subseteq T \times R$ is the available assignment of tags to resources. Let $T_r = \{t \in T \mid (t, r) \in Y\}$ denote the set of tags occurring in the content of a resource r. The semi-supervised tag recommendation problem is to find, for a given training set F, which of the tags $t \in T$ should match the resources $r \in R_p$ in addition to the ones already specified by Y. We pose this problem as a classification problem in the space of resource content, transforming each $r \in R$ into a bag of words and denoting by r_w the number of times a word w occurs in a resource r.[1] Tag recommendation here can be considered as a binary classification problem: does a given tag t match a given page r? We compare two different sets of resource features: word counts r_w and tf-idf weights

$$\text{tf-idf}(w, r, R) = \text{tf}(w, r)\text{idf}(w, R) = \frac{r_w}{\sum_{w \in W} r_w} \log \frac{|R|}{|\{r \in R \mid w \in r\}|}.$$

For the classification methods, in this work we concentrate on methods related to the general approach of support vector machines (SVM). We apply and compare the following classification algorithms.

1. *Regularized Least Squares Classification* (RLSC) [8] with linear kernel: solve a minimization problem with the square loss function, i.e., find the weights $\boldsymbol{w}$ that solve the following optimization problem:

$$\min_{\boldsymbol{w} \in \mathbb{R}^d} \frac{1}{2} \sum_{i=1}^{n} \|y_i - \boldsymbol{w}^\top \boldsymbol{x}_i\|_2^2 + \frac{\lambda}{2} \|\boldsymbol{w}\|_2^2.$$

2. *Support Vector Machine* (SVM) with l_2-SVM loss trained with the modified Newton method [9], i.e., find the weights $\boldsymbol{w}$ that that solve the following optimization problem:

$$\min_{\boldsymbol{w} \in \mathbb{R}^d} \frac{1}{2} \sum_{d \in D} l_2 \left(y_i \boldsymbol{w}^\top \boldsymbol{x}_i\right) + \frac{\lambda}{2} \|\boldsymbol{w}\|^2, \quad l_2(z) = \max\left(0, 1 - z^2\right);$$

[1] Note that although we discard word order information in classification, it may and will be used to construct the training set R_p.

3. *Multi-switch Transductive SVM* (MTSVM) [10], a semi-supervised version of SVM with l_2 loss function based on the following optimization problem:

$$\min_{\boldsymbol{w},y'}\left[\frac{\lambda}{2}\|\boldsymbol{w}\|^2+\frac{1}{2|D|}\sum_{i\in D}l_2\left(y_i\boldsymbol{w}^\top\boldsymbol{x}_i\right)+\frac{\lambda'}{2|U|}\|\boldsymbol{w}\|^2\right],\quad l_2(z)=\max(0,1-z^2),$$

$$\text{subject to }\frac{1}{|U|}\sum_{j\in U}\max\left(0,\operatorname{sign}\left(\boldsymbol{w}^\top\boldsymbol{x}_j\right)\right)=r,$$

where U is the unlabeled part of the data, y' are labels for unlabeled data that are also being optimized, and r is a predefined fraction of unlabeled data that is expected to have positive labels (r can be estimated from the labeled part of the data). MTSVM is reported to be very successful on large sparse partially labeled datasets [10].

4. *Deterministic Annealing Semi-supervised SVM* (DASVM) [10], a relaxation of the MTSVM problem with a loss function close to squared loss for large values of temperature (i.e., in the beginning of the annealing process) and converging to the l_2 loss function used in TSVM as temperature drops to zero. This approach lets DASVM overcome the problems related to the fact that l_2 loss is nonconvex: we begin with a convex squared loss function, where a global optimum is easy to find, and then gradually move to the nonconvex l_2 loss, hopefully staying in the global optimum and avoiding local ones. We refer to [10] for details of the training algorithm.

We consider the first two algorithms, RLSC and SVM, in order to compare traditional fully supervised approaches to tag recommendation to our semi-supervised approach. In particular, in [11] an SVM classifier with tf-idf features showed the best classification results in a tag recommendation system.

4 Evaluation

Experiments were conducted on the real life Surfingbird collection of web pages. To evaluate the performance of our method, we selected all web pages from the "Gadgets" category, among them 608 pre-tagged web pages in the "seed" dataset from RSS streams and 9,442 untagged web pages that were tagged on the first two stages of our algorithm. A tag is included in the dictionary if (i) it appears in the web page title, and (ii) it appears at least 5 times in the body of the web page. As a result, we selected 421 tags in the category "Gadgets".

To get training features for documents, we applied basic text mining procedures (extracting useful text, stop word removal, and lemmatization) for all web pages in the dataset. Then we discarded words that appear in the dictionary of tags from the resulting documents (an important step since otherwise a classifier would simply learn that a tag is characterized by words occurring in it). The final vocabulary consisted of 52,383 terms; each document was then converted to vectors of word counts r_w and tf-idf weights tf-idf(w,r,R). The training dataset consisted of all partially tagged web pages and randomly chosen 60% of web

Table 1. A comparison of different classification algorithms

Algorithm	RLSC	SVM	MTSVM	DASVM
Micro-F, term occurrence count features	0.21	0.25	**0.31**	0.30
Micro-F, tf-idf features	0.24	0.27	**0.35**	0.33
Micro-F, counts, tags not occurring in the text	0.13	0.15	0.20	**0.21**
Micro-F, tf-idf, tags not occurring in the text	0.14	0.17	0.21	**0.23**

pages from the "seed" dataset with known tags; the testing dataset included the remaining 40% of the "seed" dataset.

To test classifiers, we converted tag extraction to a set of binary classification problems. For fully supervised classification (RLSC and regular l_2-SVM), the problem was reduced to a binary classification task for each tag $t \in T$ by assigning labels from $\{-1, +1\}$ as follows: for $r \in R_e$, assign label $+1$ if r is labeled by t and -1 otherwise; for $r \in R_p$, assign label $+1$ if r is labeled by t (but no negative examples from R_p – our purpose here is to add more tags to R_p). For semi-supervised classification, those $r \in R_p$ that do not have t assigned to them are added to the training set as unlabeled data.

Table 1 shows microaverage F-measure (micro-F) values [12] for different classification algorithms. To understand the advantages of this method over key phrase extraction methods [13], we also computed the micro-F measure separately on those tags from the validation part of R_e that do *not* occur in the page's content, i.e., tags that generalize and augment the content. These results are shown in the bottom row of Table 1. The best classification quality was shown by semi-supervised methods in every case; tf-idf weights consistently outperform word counts as classification features. The results show an improvement over traditional approaches presented in, e.g., [11]. Importantly, even new tags that do not occur in the text are predicted with reasonable micro-F values comparable to other tag recommendation methods based on tf-idf weights.

5 Conclusion

We present a novel approach to tag extraction based on semi-supervised learning. Our approach works well even in the case of relatively small amounts of pretagged web pages (in our case, 5-10% of the total amount of web pages). Further work may include finding synonyms among tags with ideas similar to [14] to reduce tag space and thus improve the training set for classification. Another promising approach is to pair our method with topic modeling in a unified system similar to the ones proposed in [15, 16]; this will further enhance tag mining in the semi-supervised case.

Acknowledgements. This work has been supported by the Russian Presidential Grant Programme for Young Ph.D.'s, grant no. MK-6628.2012.1, Russian Presidential Grant Programme for Leading Scientific Schools grant no. NSh-

3229.2012.1, and Russian Fund for Basic Research grants 12-01-00450-a, 11-01-12135-ofi-m-2011, and 11-01-00760-a.

References

1. Guy, I., Zwerdling, N., Ronen, I., Carmel, D., Uziel, E.: Social media recommendation based on people and tags. In: Proceedings of the 33rd Annual ACM SIGIR Conference, pp. 194–201 (2010)
2. Sen, S., Vig, J., Riedl, J.: Tagommenders: Connecting users to items through tags. In: 18th International World Wide Web Conference, p. 671 (April 2009)
3. Zhou, T.C., Ma, H., King, I., Lyu, M.R.: UserRec: A user recommendation framework in social tagging systems. In: Proceedings of the 24th AAAI Conference on Artificial Intelligence, pp. 1486–1491 (2010)
4. Sigurbjörnsson, B., van Zwol, R.: Flickr tag recommendation based on collective knowledge. In: Proceedings of the 2nd ACM Conference on Recommender Systems, pp. 327–336 (2008)
5. Jäschke, R., Marinho, L., Hotho, A., Schmidt-Thieme, L., Stumme, G.: Tag recommendations in folksonomies. In: Kok, J.N., Koronacki, J., Lopez de Mantaras, R., Matwin, S., Mladenič, D., Skowron, A. (eds.) PKDD 2007. LNCS(LNAI), vol. 4702, pp. 506–514. Springer, Heidelberg (2007)
6. Rendle, S., Schmidt-Tieme, L.: Pairwise interaction tensor factorization for personalized tag recommendation. In: Proceedings of the 3rd ACM International Conference on Web Search and Data Mining, pp. 81–90 (2010)
7. Symeonidis, P., Nanopoulos, A., Manolopoulos, Y.: Tag recommendations based on tensor dimensionality reduction. In: Proceedings of the 2nd ACM Conference on Recommender Systems, pp. 43–50 (2008)
8. Rifkin, R.M., Yeo, G., Poggio, T.: Regularized least-squares classification. In: Advances in Learning Theory: Methods, Model and Applications. NATO Science Series III: Computer and Systems Sciences, vol. 1, pp. 131–154. IOS Press, Amsterdam (2011)
9. Keerthi, S.S., DeCoste, D.: A modified finite Newton method for fast solution of large scale linear SVMs. Journal of Machine Learning Research 6, 341–361 (2005)
10. Sindhwani, V., Keerthi, S.S.: Large scale semi-supervised linear svms. In: Proceedings of the 29th Annual ACM SIGIR Conference, pp. 477–484. ACM, New York (2006)
11. Illig, J., Hotho, A., Jäschke, R., Stumme, G.: A comparison of content-based tag recommendations in folksonomy systems. In: Wolff, K.E., Palchunov, D.E., Zagoruiko, N.G., Andelfinger, U. (eds.) KONT/KPP 2007. LNCS(LNAI), vol. 6581, pp. 136–149. Springer, Heidelberg (2011)
12. Fan, R.E., Lin, C.J.: A study on threshold selection for multi-label classification. Technical report, National Taiwan University (2007)
13. Medelyan, O., Frank, E., Witten, I.H.: Human-competitive tagging using automatic keyphrase extraction. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 2009), vol. 3, pp. 1318–1327 (2009)
14. Turney, P.D.: Mining the Web for Synonyms: PMI-IR versus LSA on TOEFL. In: Flach, P.A., De Raedt, L. (eds.) ECML 2001. LNCS(LNAI), vol. 2167, pp. 491–502. Springer, Heidelberg (2001)
15. Si, X., Sun, M.: Tag-LDA for scalable real-time tag recommendation. Journal of Computational Information Systems 6, 23–31 (2009)
16. Krestel, R., Fankhauser, P.: Personalized topic-based tag recommendation. Neural Computation 76(1), 61–70 (2012)

A Similarity Model for 3D Objects Based on Stable Sub-clouds

Markus Mauder[1], Peer Kröger[1], and Karl-Ludwig Schinner[2]

[1] Institute for Computer Science, Ludwig-Maximilians-Universität München, Germany
{mauder,kroeger}@dbs.ifi.lmu.de
[2] Opdi-Tex GmbH, Eresing, Germany
schinner@opdi-tex.de

Abstract. We present an idea of a novel similarity model for objects represented by 3D point clouds that were generated by scans of real-world objects. Various existing approaches find descriptive points on the object surface or extract features of groups of points. However, 3D object scans when conducted outside a lab environment often suffer from imprecisions and noise artifacts, which many existing approaches do not handle well. To better tolerate these imperfections, our model extracts stable sub-clouds from the input point cloud, which represent classes of adjacent sub-clouds sharing similar features. We demonstrate experimentally that features generated from these sub-clouds can be used to establish a measure of similarity between objects. We show preliminary results of an application of this technique to point clouds of models scanned from real-world objects and demonstrate that this technique has good potential to deal with imperfect data by showing how the computed distance relates to degrees of modification of the data. Our technique extracts features from particularly resilient portions of the object and is thus better able to accommodate deficiencies in the input data.

1 Introduction

Large numbers of 3D objects from devices like scanners or LIDAR sensors are now being generated in areas such as archeology or geography. Typically, this data consists of 3D point clouds representing real objects. Object recognition and 3D model acquisition, e.g. for classification and tagging, is a crucial step for processing this data. Since manual processing of such big amounts of data is not feasible, automatic methods for acquiring such models are required [2]. One key part of an automatic acquisition of objects is a similarity model for 3D point cloud representations that can be used by any classifier. The majority of 3D point cloud data exhibits a considerable amount of noise or artifacts due to limitations of the devices generating the raw data. Traditional similarity models (cf. [3]) often cannot handle noise effectively and accurately and, thus, are quite limited for automatic 3D model acquisition. In this paper, we sketch a novel similarity model for 3D point clouds that has the potential to be quite robust against those artifacts. It follows a data-partitioning approach and proposes a novel object partitioning, stable sub-clouds, that have descriptive power and can be used for extracting an object descriptor. The presented approach consists of three steps: First, the point cloud representing the entire object is partitioned into sub-clouds. Second, features are

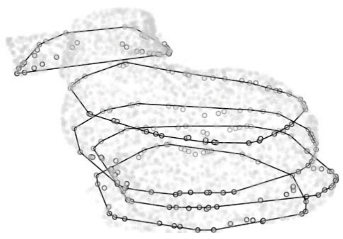


Fig. 1. Example of stable sub-clouds

extracted. Features of the most stable sub-clouds are generated to derive a descriptor of the original object. Third, a distance measure is defined to compute the similarity between any two objects. The main contribution is the computation of stable sub-clouds and their processing to derive a descriptive feature vector. This technique is similar in spirit to the image analysis technique *Maximally Stable Extremal Regions*[6] which identifies pixel sub-sets that are unchanged by environmental influence.

The presented object partitioning is based on the observation that adjacent cross sections through any object normally have very similar shapes. As the distance between two cross sections increases their shapes will become less similar. The rate of this change is an indication of the prevalence of this particular cross section's shape throughout the object. In the presented model, each cross section is represented by a sub-set of the model's points, which approximate the cross section's shape. These sub-clouds can be compared with one another to establish a rate of change at the current position. Comparing sub-clouds that are close is expected to yield a lower difference value than that of farther spaced sub-clouds. This difference is indicative of its suitability for matching to a sub-cloud from a differently represented version of the same object. We call the length of the range in which no sub-cloud differs to a reference sub-cloud by more than a threshold, the *stability* of the reference sub-cloud. Sub-clouds with larger stability are more likely to be matched. The most stable sub-clouds' features form the descriptor for the entire cloud. Figure 1 depicts the five most stable sub-cloud sequences in a down-sampled version of the Stanford Bunny from the Stanford 3D Scanning Repository[1]. We postulate that the extracted features are invariant to artifacts introduced by the acquisition. The extracted descriptor is both low-dimensional and robust. It is invariant to the resolution and orientation of the scan (as long as the scan line is preserved). Additionally, we define a distance measure for comparing point cloud objects using these sub-cloud features.

In the following section our technique is presented. Section 3 describes preliminary experiments showing our model's response to noise. Section 4 concludes the paper.

2 A Similarity Model Based on Stable Sub-clouds

Our method for the automatic recognition of 3D objects intends to be resistant to the influence of noise in the acquisition process. Toward this end it represents the original

[1] `http://graphics.stanford.edu/data/3Dscanrep`, retrieved on 2013-02-25.

```
**Require:** $pts$
 1: $i := 0$
 2: $d := inf$
 3: while $i < pts.length$ do
 4:     for $t \in 1 \ldots rest$ do
 5:         $s1, s2 := subclds(pts[i], t)$
 6:         if $diff(s1, s2) > d$ then
 7:             break
 8:         end if
 9:     end for
10: end while
11: $yield(s1)$
12: $i := i + 2 \cdot t$
```

Algorithm 1. Shape detection

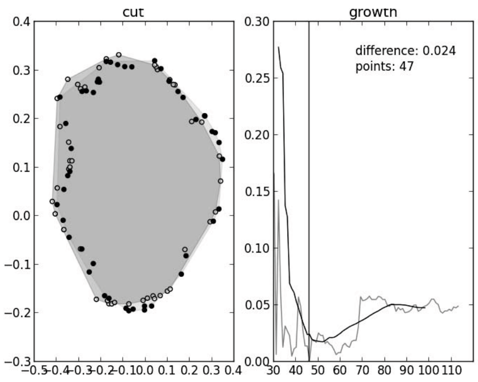


Fig. 2. Locally stable sub-cloud, stability graph

point cloud through a short descriptor and defines an accompanying distance measure to quantify differences between objects. This section considers finds regions of particular resilience to potential modifications during acquisition. In Section 2.1 we describe how our technique extracts subsets from a set of points to represent the original object. Section 2.2 explains the properties required of features that can be used to establish differences between these sub-clouds. Finally, Section 2.3 details how the individual distances between sub-clouds can be used to establish a global distance and how to reduce the magnitude of the descriptor to just the most relevant features.

2.1 Locally Stable Sub-clouds

In a first step, point sub-sets which correspond to the cross sections of the object at a given position need to be determined to serve as means to determine stability. To achieve this, we postulate that over a range of the scan axis only little change occurs to appropriately designed features. Point sets are then called *locally stable* if they are the smallest set that minimizes the difference to its immediate neighbor. This step is detailed in Algorithm 1. Stability is determined by progressively growing sub-clouds (from the point cloud pts) while comparing them with an immediate neighbor of the same magnitude until the determined difference value passes a minimum. This minimum can be expected to exist, because more points are at best redundant, at worst degrading to the shape descriptor, resulting in a larger difference value. To avoid random points influencing the difference and causing a premature stop of the sub-cloud detection, it is advisable to smooth the similarity values over an appropriate range. We take the smoothed stability values and detect the first local minimum to establish a magnitude where the sub-cloud is stable. Figure 2 shows a typical sub-cloud in an example object. The image on the left depicts points that are part of the current sub-cloud as empty and filled circles. The depicted sub-cloud is defined by the size indicated by the vertical line at position 47 in the graph on the right. On the right, the gray lines depict the difference value between sub-cloud candidates, the black line its smoothed counterpart.

```
Require: $clds$, $thresh$
 1: $i := 0$
 2: while $i < clds.length$ do
 3:     $j := i$
 4:     $d := diff(clds[i], clds[j])$
 5:     while $d < thresh$ do
 6:         $j := j + 1$
 7:     end while
 8:     $stability[i] := j - i$
 9:     $i := j$
10: end while
```

Algorithm 2. Determine sub-cloud stability.

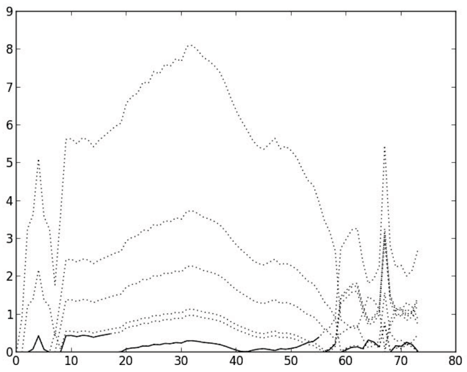


Fig. 3. Differences between sub-clouds. Without (dotted) and with (solid) reference updates.

2.2 Features and Sub-cloud Comparison

A feature that is suitable for the previously mentioned similarity computation must degrade continuously with progressively changing point positions. A simple example that satisfies this condition is the area of the sub-cloud's hull. Comparing these features requires a suitable distance function. To get a result that considers only the relative magnitude of the features, the function expresses the ratio between both values in terms of a relative change to the original value. Concretely, we calculate the ratio of the smaller feature to the absolute difference between the features: $d(f_1, f_2) = \frac{|f_1 - f_2|}{min(f_1, f_2)}$

2.3 Object Descriptor and Distance Calculation

The extracted sub-clouds are combined into a single object descriptor. We propose to use a list of sub-cloud features as the object's descriptor. The used sub-cloud descriptors should be invariant to resolution, rotation, and scale. Examples of features that can be used satisfying these requirements is (again) the area of the convex hull of the sub-cloud's points or a measure of the sub-cloud's shape. One possibility is to use the previously extracted feature (used to establish the sub-cloud's stability), which has already been computed. While it is possible to use all extracted features to establish a difference between objects, this requires a very complex object distance function. In contrast, robustly reducing the descriptor's dimensionality also reduces the number of candidates to erroneously establish matches with and to prune unreliable shapes. To achieve this only sub-clouds that were most stable are used. Algorithm 2 shows the process of determining sub-cloud stability in detail. The sequence of locally stable sub-clouds ($clds$) extracted in the previous step is examined for their relative similarity. Once they become more different than a threshold ($thresh$), the following cloud becomes the reference. This step does not backtrack to keep complexity low. While this potentially undervalues a regions stability, a single run over the point clouds is sufficient to determine it. We call the length of the range of similar features, a sequence's *stability*. Ranking the sequences by their stability allows us to extract only the k most stable features. Figure 3 shows the ranges generated in the example object. The dotted lines illustrate that it is acceptable to re-assign the reference value. The dotted lines depict how the solid line would have progressed had the reference cloud not been re-assigned. While a global view might have suggested other split points, the chosen positions are still valid features.

These features are similarly strongly represented in other depictions of the same objects, but differences in the features must be anticipated. To this end, we aggregate the feature differences into an object-wide value using *Dynamic Time Warping* (DTW) [1]. This technique allows additional sub-clouds in either of the compared objects, while preserving the order of matched elements. Under normal circumstance, the difference between varying representatives of the same sub-cloud sequence in different models changes only slightly and DTW will still find a match.

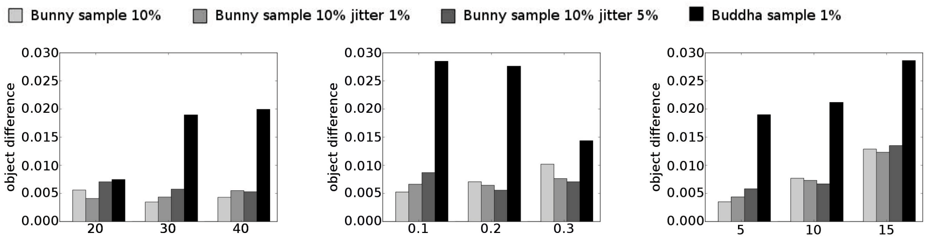
3 Experiments

In this section some preliminary results of experiments conducted using the presented technique are shown. We show the influence the algorithm's parameters have on its performance and give guidelines for choosing initial values. To be able to judge the effect on the noise tolerance we use three variations of the original model, to which we applied varying amount of noise. In addition the difference to another model is included, which has been scaled and sampled to have similar properties as the reference model.

We used the Stanford Bunny and Happy Buddha [4] models to test our method's descriptive capabilities. The graphs presented in this section all show the distance of the original Bunny model with – from left to right – three progressively worse representations of the original model as well as the Buddha model. The feature used to both determine and represent sub-clouds is the area of the convex hull of the point set. The modifications we applied to the Bunny are 1) to sample its points down to 10% of the original number of points, 2) an addition of normally distributed noise of 1%, then 5% of the distance to the model's center (limited to the two orthogonal axes to preserve the points' order). The Buddha model too has been downsampled by a factor of 100 to have a comparable number of points as the original Bunny model.

As our technique depends on a few parameters, we evaluate a number of settings for each. We expect progressively growing differences between the original model and modified versions. The difference to the other model should be significantly larger. Unless otherwise specified the smoothing range of the locally stable sub-cloud detector has been set to 40, the similarity threshold to 0.1, and the descriptor length to 5.

In a first experiment, we examined how many difference measures need to be combined to sufficiently dampen the influence of outliers when finding the first minimum in a series of differences in detecting locally stable sub-clouds. In this experiment we set this parameter to 20, 30, and 40 in turn and observed how the results (see Figure 4a) changed. Clearly it is necessary that this parameter is chosen sufficiently large. Beyond that, the performance degrades, but not particularly quickly. The threshold by which we judge whether a sequence of point clouds is still considered stable could potentially influence the result negatively. Smaller difference value force more stable sub-cloud candidates, which should lead to better results. In this experiment we set the maximum allowed difference value to 0.10, 0.20, and 0.30. The results can be seen in Figure 4b.



(a) Smoothing length. (b) Sequence similarity. (c) Descriptor length.

Fig. 4. Effect of changing the algorithm's parameters

As expected allowing a maximum difference of 0.1 is quite sufficient. At 0.3 the results begin to degrade strongly. Finally, the length of the extracted descriptor contributes to the runtime of the comparison step. The results can be seen in Figure 4c. A descriptor length of 5 produced the expected results. The reason that longer descriptors produce worse results than shorter ones is that a high maximum difference within a stable sub-cloud can subsume many sub-clouds into one. If the length of the descriptor exceeds the number of available significantly stable sub-clouds, its discriminative power suffers.

4 Summary

We have sketched a similarity model for 3D point cloud data designed to be robust against noise and, can be used within a pipeline for automatic annotation of 3D objects from scanners and sensors. Our method extracts subsets of point clouds that can be reliably detected under varying scanning conditions. First experiments on real data support this claim. For future work we plan to conduct more experiments on larger data sets like SHREC [5] and include models with real noise. With the results of these experiments, we plan to improve our descriptors and try other sub-cloud features.

References

1. Bellman, R., Kalaba, R.: On adaptive control processes. IRE Transactions on Automatic Control 4(2), 1–9 (1959)
2. Bernardini, F., Rushmeier, H.: The 3d model acquisition pipeline. Computer Graphics Forum 21, 149–172 (2002)
3. Castellani, U., Bartoli, A.: 3d shape registration. In: 3D Imaging, Analysis and Applications, pp. 221–264. Springer London (2012)
4. Curless, B., Levoy, M.: A volumetric method for building complex models from range images. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, pp. 303–312. ACM (1996)
5. Boyer, E., et al.: Shrec 2011: robust feature detection and description benchmark. In: Proceedings of the 4th Eurographics Conference on 3D Object Retrieval, EG 3DOR 2011, pp. 71–78. Eurographics Association, Aire-la-Ville (2011)
6. Matas, J., Chum, O., Urban, M., Pajdla, T.: Robust wide baseline stereo from maximally stable extremal regions. In: British Machine Vision Conference (BMVC), pp. 384–393 (2002)

Accurate and Efficient Search Prediction Using Fuzzy Matching and Outcome Feedback

Christopher Shaun Wagner, Sahra Sedigh, and Ali R. Hurson

Missouri University of Science and Technology, Rolla MO 65409, USA
{csw6g3,sedighs,hurson}@mst.edu

Abstract. While search engines have demonstrated improvement in both speed and accuracy, the response time to queries is prohibitively long for applications that require immediate and accurate responses to search queries. Examples include identification of multimedia resources related to the subject matter of a particular class, as it is in session. This paper begins with a survey of recommendation and prediction algorithms, each of which applies a different method to predict future search activity based on the search history of a user. To address the shortcomings identified in existing techniques, we draw inspiration from bioinformatics and latent semantic indexing to propose a novel predictive approach based on local alignment and feedback-based neighborhood refinement. We validate our proposed approach with tests on real-world search data. The results support our hypothesis that a majority of users exhibit search behavior that is predictable. Modeling this behavior enables our predictive search engine to bypass the common query-response model and proactively deliver a list of resources to the user.

1 Introduction

Search engine development began as a study on indexing techniques - a topic as old as information storage itself [1]. Early on, search engines used keywords to allow people to refer to an index in a collection of resources without actually understanding how the resources were indexed. Providing a search query of keywords and seeing a list of resources has become the standard query-response model of search engines [1]. Improvement of search engines focuses on two main aspects of performance: accuracy and speed [2–4]. Accuracy refers to how well the search engine ties a search query to a relevant information resource. It is subjective and determined by the user's perception of what is relevant and what is not. Speed refers to how quickly the search engine is able to process a query and return results and is commonly denoted as "response time."

Our research focuses on the use of recommendation (proactively returning responses to queries that the user is expected to make) to improve both accuracy and speed of search engines. Successful recommendation improves accuracy, as the resources recommended will be relevant to the user's interests. Eliminating the need for query entry improves speed. In a perfect system, resources in which a user is interested will be recommended and waiting on the user, rather than the user waiting on the search engine.

Amazon uses recommendation to direct users to products that they are anticipated to purchase. Netflix uses recommendation to direct users to movies they are anticipated to find of interest. Recommendation has proven successful in guiding a user to resources of interest to him or her; however it is of limited use in applications with time constraints as it generally ignores the context and presents a user with a “wholesale” list of items of potential interest. [5, 6].

Prediction is a specialized form of recommendation that recommends the resources a user will want in a particular context [6]. Using the previous examples, prediction would identify the item an Amazon user will want to purchase next Tuesday or the movie a Netflix user will want to watch tonight.

A predictive search engine does not need to identify the exact time that a resource will be requested. It is enough to identify the order in which resources will be requested. Given a user’s search history, the resources that are likely to be requested next are recommended. Accuracy and speed are improved.

The success of search prediction is entirely dependent on the ability to model user behavior. Typical behavioral models divide a population into groups of users whose past behavior has been similar. These groups are denoted as neighborhoods of similarity, or simply neighborhoods. Algorithms that carry out the grouping are known as k-nearest neighbor (k-NN) algorithms. All variants of these algorithms are based on forming a neighborhood around a target user [7].

The contribution of this paper is in proposing a novel k-NN algorithm, where the outcome of grouping the users is used to refine the classifier and improve accuracy. Group members who “do not belong” are replaced to increase the probability of successful prediction. Avoiding comparisons for members who “belong” reduces complexity and improves accuracy. Allowing for fuzzy, rather than perfect matches further improves the accuracy of our technique.

We elaborate on and justify these statements in the remainder of this paper. Section 2 provides a background, through a survey of common prediction and grouping algorithms. Our proposed grouping algorithm and general approach to search prediction is described in Section 3. The design of experiments we carried out for validating our work is presented in Section 4; where our specific hypothesis, test methods, real-world search engine data used, and scoring methods are described. Section 5 details the results of this validation. We conclude the paper with Section 6, which summarizes our findings on the viability of our search prediction approach.

2 Background

Electronic storage capacities have grown rapidly over the last decade. Interaction and communication between storage devices and data centers has similarly increased. The explosion of demand for information has resulted in a plethora of recommendation algorithms designed to limit the result set that a user must examine when attempting to find a resource of interest. In this section, we survey recommendation and grouping algorithms and discuss their use in search prediction.

General recommendation techniques are not necessarily acceptable for prediction. Consider an Amazon user who has just purchased volume five of the *Harry Potter* series. Recommendation would identify volumes one through four as being of interest to this target user, even though he or she is likely to have read them already. Prediction would correctly identify volume six as the next book likely to be of interest to the user. As such, prediction is context-aware recommendation.

In the Amazon example, it is obvious that the books are what the algorithm should recommend or predict. In a search engine setting, prediction could target either the query that a user will type or the information resource that he or she will deem useful. Google and Bing have produced algorithms for predicting what a user will type as a query. Their algorithms attempt to complete or revise a query. However, the user is actually searching for an information resource, not a query. Further, a query is not a discrete index to a specific resource. As an example, a search for "hedgehog" may refer to a small spiny mammal, a popular chocolate treat, or an old anti-submarine mine. Prediction of the query is not as effective as predicting the exact resource that the user will want to select.

2.1 Recommendation/Prediction Algorithms

While there are many specific implementations of recommendation, most are based on a few simple strategies, such as recommending items based on frequency or sequential order. The simplest form of recommendation is the *rank model.* Items are ranked by the frequency with which they are selected, and the highest ranking (most popular) items are suggested to the user. This strategy is likely to be effective, as the most popular items will be selected more than half of the time (Zipf's law) [8].

A refinement of the rank model has become popularly known as the *"people who bought X also bought Y"* algorithm from Amazon. Instead of identifying the items most popular among all users, this "also" model considers popularity among only those users whose purchase history includes a specific item - a neighborhood of (slightly) similar users. This similarity should cause the results to be more accurate than the rank model.

Neither model takes the time or sequence of purchases into account. Such consideration would make these algorithms useful for search prediction. Specifically, the resulting algorithm would make the more accurate statement of "people who bought X **then** bought Y."

It may be possible to improve the results further by considering the last n items selected by the target user. This sequence, denoted as an *n-gram*, is then used to constrain the neighborhood of users considered in determining the most popular items. The fact that these users accessed the same n items in the same order should increase accuracy.

Humans are not automatons, and as such, there is variance in human behavior - a human may carry out the same action slightly differently in each repetition. The n-gram model does not capture such variance. Instead of a seeking perfect match between the target user's most recent n-gram and an n-gram in another user's history, using a fuzzy match will allow for variance in search behavior and should be more accurate than a perfect n-gram match.

All of the aforementioned methods, except the simple rank model, are essentially creating a neighborhood of similar users for the target user. How these neighborhoods are created directly impacts the success of prediction.

2.2 Neighborhoods of Similarity

Neighborhoods of similarity are nearly synonymous with the k-nearest neighbor (k-NN) algorithm [9–11]. The task is to identify other users similar (or dissimilar) to a target user, and as such defining a suitable measure of similarity is critical. Measuring similarity between users, or more generally, objects; is not straightforward, especially when multiple attributes are to be considered. Some algorithms characterize each object as a vector, where each attribute is an element (dimension) in the vector space [12]. Two vectors that point in the same direction (regardless of magnitude) are considered identical. The angle between two vectors becomes the measure of similarity. Often, the cosine of the angle is used instead, producing -1 for opposite and 1 for identical vectors.

If all of the attributes are binary, the Jaccard similarity coefficient captures a set comparison between two entities [13]. The more attributes the two entities have in common, the more similar they are. This is very similar to Hamming distance, which identifies the number of bits in which two binary strings differ. If each bit of the binary string is considered an attribute, the Jaccard and Hamming measures may be used interchangeably to measure either similarity or difference.

Neither vector or set comparison algorithms require attributes to maintain a specific order. Consider comparing two users based on gender, age, and nationality. The result will be the same, regardless of whether the order of attributes is {gender, age, nationality} or {age, gender, nationality}. For prediction, order is important, and hence the attributes must be stored as ordered sequences of values (i.e., strings) and the comparison algorithm must be sensitive to the order.

String-based comparison measures are extensions of Hamming distance, which is not truly sensitive to order - if we swap the first and second bits of each of two binary strings, their Hamming distance will remain the same. The fact that the difference occurs in a different bit is not captured by the measure - a shortcoming addressed by string-based comparison measures. One such measure, Levenshtein distance, has become the foundation of most string-based comparison algorithms [14–16]. Levenshtein distance identifies how many changes are required to turn one string of values into another string of values. The Wagner-Fischer algorithm is a popular matrix solution for Levenshtein distance [17]. The Needleman-Wunsch and Smith-Waterman algorithms, are matrix solutions that can align one string to another to produce a global (full string) or local (substring) alignment; respectively, with minimum difference.

In developing a predictive search algorithm, we seek to model the selection of search results by the user and predict the result that will be selected next. This necessitates comparison of the search histories of users. It is possible to consider demographic attributes such as gender or age in addition to the search history of a user, but these demographic attributes rarely have a bearing on Internet search, and considering them may lead to inaccurate results.

Search histories are ordered sequences of items, implying that a string-based comparison is necessary in creating neighborhoods. Simply counting the results selected by both of two users is a simpler and less effective measure. In string-based comparison, the target user's search history is represented as an n-gram and compared to the respective n-grams representing the search history of other uses. The existence of long sequences in common between two n-grams indicates similar search behavior of the corresponding users. Computational complexity is the problem with this approach - the complexity of algorithms for identifying the longest common substring between two strings of data is $\mathcal{O}(n^2)$.

As noted in section 2.1, human behavior has natural variance. Defining similarity as a perfect behavioral match will omit users with slight variances from the resulting neighborhood. Using local alignment, it is possible to obtain a fuzzy match between parts of the respective search histories of different users. The longer the matching substring, and the less difference in the local alignment, the greater the similarity. Allowing for slight variance in the search histories has lower computational complexity than finding a perfect match, and serves as the basis of our search prediction technique. We present and validate our approach in the remaining sections of this paper.

3 Proposed Approach to Search Prediction

In this paper, we present a search prediction technique with lower computational complexity than existing alternatives. This improvement is achieved by using fuzzy matching, which allows for the slight variations typical of human behavior in determining similarity among search histories. Without allowing for this variance, the number of users that can be used for prediction drops dramatically, often to zero. A fuzzy match increases the size of the neighborhood of similar users.

The search prediction process centers around creating a neighborhood of up to k users whose search behavior is similar. In practice, a background process would continually scan the universal population of search engine users for users that are more similar to the target user than the most dissimilar user currently in the neighborhood, updating and refining the neighborhood. Similarity is measured by performing a local alignment of the target user's most recent history against the other user's complete history. The number of gaps and omissions present in the local alignment serve as the measure of distance.

Prediction is carried out by performing a local alignment of the target user's most recent search history against the complete history of each user in its neighborhood. As local alignment was performed to create the neighborhood, the bulk of this calculation has already been performed. For each user in the neighborhood, the item that follows the local alignment is a suggestion for the target user. The most popular suggestion, or list of most popular suggestions, is shown to the target user as the result of predictive search.

Local alignment is a costly operation to perform. To further reduce processing time, outcomes are used to refine the neighborhood of similar users. Users who supply the correct prediction are kept in the neighborhood without further calculation. Those who fail at prediction are replaced.

The novelty of our approach is in the use fuzzy matching in conjunction with outcome feedback. We anticipate that the proposed scheme will offer a higher performance in accuracy with limited increase in processing time. Experimental validation, as described in the remainder of this paper, confirmed this result.

4 Design of Validation Experiment

The goals of our experimental validation were to a) demonstrate the superiority of our approach over the existing alternatives described in Section 2 and b) shed light on the tradeoff between computational complexity and accuracy. To this end, we implemented each of the following methods. When used with a neighborhood, the prediction algorithm is carried out only for the members of the neighborhood. Otherwise, the algorithm uses the entire population of users for prediction. The algorithms implemented are:

1. *Random*: suggests s random resources and is used as a baseline comparison.
2. *Popular*: suggests the s most frequently selected resources.
3. *Also*: suggests the s most frequently selected resources from users who also selected the last n resources as the target user, regardless of selection order. Application of this algorithm in conjunction with certain neighborhood models yields no additional information.
4. *Next*: is a variant of the *Also* algorithm, with the added constraint that any resources to be suggested must have been selected *after* the common resource between the users was selected.
5. *n-gram*: suggests the s resources most frequently selected following (with a perfect ordered match) the last n resources in common with the target user.
6. *Fuzzy*: suggests the s resources most frequently selected following (with a fuzzy ordered match) the last n resources in common with the target user. Local alignment is used as the fuzzy matching algorithm.

Multiple methods exist for measuring similarity; hence, multiple neighborhood models were tested. The underlying methods of similarity comparison are the same as the similarity methods used to identify the resources to suggest. Each time a prediction is made, a neighborhood of similar users is also determined. The algorithms used are:

1. *None*: is the baseline - produces no neighborhood, forcing prediction to use the entire population.
2. *Common*: selects users who selected the target user's s most recently selected resources.
3. *n-gram*: selects users who selected, in the same order, the target user's s most recently selected resources.

4. *Fuzzy*: selects users who have a similar local alignment of the target user's s most recently selected resources.

As a separate test, we added outcome-based feedback to all of the neighborhood algorithms, with the exception of *None*. After the prediction is tested against the target user's actual selection, members of the neighborhood that correctly predicted the resource selected by the target user are kept in the neighborhood. Those who did not correctly predict the target user's selection are replaced when the following prediction is made.

4.1 Test Data

While it is possible to theorize how well recommendation or prediction may perform, tests on actual data sets from actual search engines demonstrate performance in a real-world setting. Data sets were collected from multiple forms of search engines. Many had little or no longitudinal data to develop user search history, because their use was limited to a very short time period or was heavily manipulated for a specific purpose other than complete analysis. The following six data sets were deemed acceptable for use in our validation experiments.

1. Set A, from AOL. In 2006, AOL Research released a data set of three months of Internet searches with anonymized users and identified resource information. The user, query time, and resource selection are considered in our validation experiment, without use of the actual query supplied by the user.
2. Set E, from one year of log data for Every Busy Woman, an online catalog of women-friendly businesses. The log files from the website's search engine store the id of the user making the query, the time of the query, and the business listing selected from the results.
3. Set L, a testing file of one year of data from MovieLens. As a recommender system, MovieLens is not truly a search engine. However, the use of data set N (described below) to predict the movie a user will watch made one aspect of the MovieLens data set interesting. The data set includes the user, the rating the user gave to a movie, the movie, and when the movie was watched. This can be translated into user, time seen, and movie selection, predicting which movie will be seen next.
4. Set M, from the Medical University of South Carolina, includes three months of log files from a search engine for employee training resources. Being small, it is used primarily as a very quick code test before the larger data sets are tested.
5. Set N, from Netflix. Many Netflix data sets are available, each heavily pruned to focus on a specific data mining task. Most are designed to predict the rating that a user will give a movie. The most complete data set contained the user, movie, and the time the movie was watched. In this research, it is not important to know how the user will rate a movie. We are attempting to identify the movie that will be selected for watching, based on the past history of movies watched.

6. Set Y, from Yandex, the largest search engine in Russia. Unlike American search engines, Yandex regularly produces anonymized data sets for research. Similar to data set A, this is a data set of real Internet searches performed by real people, but it is limited to one week of data.

All data sets were normalized such that users and resources were identified by a unique integer. The time of each resource selection was converted to a timestamp indicating the number of seconds since 1970-01-01 00:00:00. Each database was stored in the format {user, selection_time, resource_selected, previous_selection_time}. The previous selection time was necessary to easily link sequences of resource selections.

When the same user selected the same resource more than once in succession, it may be assumed that this is simply a reload or refresh of the resource and not a true selection of a new resource. Repeated selections of the same resource in succession were flagged as repeats and ignored. A selection sequence of {1, 2, 2, 3} would then become {1, 2, 3}. However, a selection sequence of {1, 2, 3, 2} would not be altered, because a different resource was selected in between the two selections of resource 2. Most sets had no repetitions.

Both recommendation and prediction require knowledge of user behavior. Some users in the data sets have only one recorded resource selection. There is no behavior based on which to make a recommendation or prediction. These selections were flagged as short and ignored in testing.

The resulting test sets are radically different from one another. As shown in Table 1, some are extremely large. Some are small. Some have many resources selected per user, creating long history sequences. Some have many users per resource, allowing for large neighborhoods of similarity.

Table 1. Statistics of data sets used in validation

Set	Records	Users	Resources	Users per Resource	Resources per User
A	114,494	18,526	57,018	0.32	3.08
E	168,387	12,857	10,458	1.23	0.81
L	1,000,209	68,404	3,708	18.36	0.05
M	3,168	45	255	0.18	5.67
N	23,168,232	463,616	17,755	26.11	0.04
Y	30,655	3,121	27,910	0.11	8.94

A legitimate concern regarding these data sets is whether they truly reflect true user behavior. The distribution of frequencies in a natural environment should be inversely proportional to the rank of the frequencies, according to Zipf's law [8]. If the distribution of frequencies is linear, it may be assumed that an intervention has altered the data in some way.

The respective frequencies of users per resource and resources per user were checked for natural distribution. Set A is capped on the maximum number of records allowed per user, which matches the description of how the data was generated. Set N has nearly the same number of users per resource, which matches the description of how the data was generated. Set L exhibits fixed user/resource and resource/user frequency, indicating that it is a very manipulated, very unnatural representation of search queries. Data that is manipulated may not allow for acceptable prediction, indicating that sets A, L, and N will likely have poor results.

If it is possible to predict one user's behavior based on the behavior of another user, it is expected that there must be common sequences of events between the two users. In other words, in a large population of search engine users, prediction is dependent on the existence of common sequences of resource selections. If so, it is possible to replace the common sequences with a new resource identifier that indicates the sequence was selected. This is similar to file compression, in which a common sequence of bits is replaced with a much shorter identifier or placeholder.

Andrey Kolmogorov, one of the founders of algorithmic complexity theory, described this form of compression as a measure of entropy [18]. The greater the number of common sequences, the less complexity there is in the data. A lack of common sequences indicates entropy. In this research, lack of entropy is referred to as *order*, indicating that there is a common (and predictable) order to the selection of search results.

Within each data set, common sequences of five resource selections were identified. Order is measured as the total number of records, divided by the number of unique sequences of five resource selections. Table 2 shows the order of the data sets. Sets E and M have higher order, indicating that prediction should work well on those sets. Set Y has nearly no order. With nearly complete entropy, it should not be possible to use prediction on set Y.

Table 2. Order (predictability) of the data sets

Set	Records	Sequences	Order
A	114494	14847	0.13
E	168387	189111	1.12
L	1000209	995253	1.00
M	3168	2484	0.78
N	23168232	12004826	0.52
Y	30655	1539	0.05

A combination of distribution and order is denoted as *convergence*. If the data set is naturally distributed and there is order, the sequences of data selections should converge on a subset of the total resources in the data set.

Measuring convergence begins with identifying the distribution of data. The resources selected are ranked from most to least selected. A best fit logarithmic trend line is calculated to match the frequency counts. It has the form $f(x) = -a\ln(x) + c$, where a is the natural logarithmic coefficient and c is a constant. The trend lines of the data sets cannot be compared directly, due to the radical differences in population sizes. Dividing a by c will produce an estimate of convergence that can yield a meaningful comparison. The greater the magnitude of the result, the more convergent the resource selections.

Sets A, E, and M were found to exhibit the greatest convergence, indicating that prediction should be successful. Set Y showed nearly no convergence, indicating that prediction will not be possible. Numerical details have been omitted for brevity.

4.2 Evaluation of Algorithms

With test algorithms and data sets, the evaluation method must demonstrate how effective the algorithms are in predicting the resource the target user will select. In a very simple form, the score is a percent of tests in which the algorithm produces a suggestion that is the resource the target user selects. The suggestion will be returned as a set of resources most to least likely to be selected, a weighted score will lower the success rate when the suggestion is not the most likely selection.

In recommendation systems, weighted testing uses the interest level in each suggestion when weighting the success rate. For prediction systems, interest is limited to absolute interest in the one suggestion that is selected and no interest in the suggestions that are not selected. Therefore, the weighting is merely the rank of the suggestion. For each test, $1/r$ is added to the success count, where r is the rank (in the suggestion list) of the resource selected . The final success rate is the success count, divided by the number of tests.

The success rate of interest is the total success rate over all tests on a data set. As the users do not have equally lengthy resource selection histories, some users will affect the overall success rate more than others. To determine the skew of the overall score, the average success rate per user is also calculated. The average success rate per user should be close to the overall success rate if a single user does not heavily skew the outcome.

The duration of complete testing of each algorithm over each complete data set was also recorded. The average number of tests per second and the average number of seconds per test can be calculated using this duration. The amount of time that the tests require is important in justifying the extra complexity for algorithms that should have a higher success rate.

5 Results of Experimental Validation

This section presents the results of the experiment described in Section 4. The results of testing prediction algorithms are presented first, followed by the results for neighborhood algorithms. For the latter, the effect of using outcome feedback is examined - we expected a decrease in prediction time and an increase in accuracy.

5.1 Prediction Results

The initial prediction results were measured without the benefit of neighborhoods. The results, shown in Table 3, were not far from expectations. Set M is very small and does not follow the normal trend. In general, each subsequent algorithm shows an improvement over preceding algorithms, with the exception of *n-gram*.

The *n-gram* algorithm is very restrictive, requiring an exact match on n consecutive resource selections (n=5 in this test). For sets A and Y, it was very rare for two users to have five identical sequences of resource selections. Allowing variance with the fuzzy match allowed near matches to be included, increasing accuracy.

The unweighted results imply that *fuzzy* yields little improvement over *next*. Examining the weighted results makes it obvious that *fuzzy* has placed the selected resource near the top of the list of suggestions, while *next* places it near the bottom. Therefore, if the result list was cut in half, the fuzzy algorithm would retain nearly the same accuracy, while the next algorithm would become drastically less accurate.

Table 3. Unweighted (weighted)success rate of prediction algorithms

Set	Random	Popular	Also	Next	N-gram	Fuzzy
A	1% (0%)	3% (0%)	4% (1%)	8% (3%)	1% (1%)	10% (7%)
E	10% (2%)	30% (5%)	54% (7%)	58% (6%)	51% (28%)	59% (42%)
L	0% (0%)	12% (2%)	14% (2%)	20% (5%)	19% (12%)	27% (16%)
M	56% (12%)	77% (10%)	73% (9%)	75% (7%)	39% (30%)	45% (30%)
N	0% (0%)	1% (0%)	5% (1%)	9% (3%)	6% (4%)	14% (8%)
Y	0% (0%)	1% (0%)	3% (1%)	4% (1%)	0% (0%)	3% (2%)

5.2 Neighborhood Results

Each neighborhood algorithm was tested separately, using the *next* prediction algorithm, chosen because *next* is not similar to any of the neighborhood algorithms being tested. As shown in Table 4, the success rate improves successively from the *none* to the *fuzzy algorithm.* Once again, sets A and Y did not improve much with the n-gram algorithm because of the rarity of users who share exact sequences of n resources selected.

Each prediction algorithm was tested a second time, with the use of outcome feedback to refine the neighborhood. Every prediction improved, as may be seen by comparing Tables 4 (results without outcome feedback) and 5 (results with outcome feedback).

5.3 Timing Results

Each algorithm was optimized to facilitate rapid testing on a single-processor machine. As expected, the number of tests per second decreases with algorithm

Table 4. Unweighted (weighted) success rate of neighborhood algorithms

Set	None	Common	N-gram	Fuzzy
A	4% (1%)	5% (2%)	9% (7%)	10% (7%)
E	54% (7%)	55% (14%)	58% (28%)	59% (42%)
L	14% (2%)	13% (6%)	19% (12%)	27% (16%)
M	73% (9%)	81% (80%)	39% (30%)	45% (30%)
N	5% (1%)	8% (2%)	11% (5%)	16% (8%)
Y	3% (1%)	4% (2%)	0% (0%)	6% (4%)

Table 5. Unweighted (weighted) success rate of neighborhood algorithms with feedback

Set	None	Common	N-gram	Fuzzy
A	4% (1%)	12% (6%)	14% (10%)	15% (12%)
E	54% (7%)	61% (27%)	69% (41%)	72% (51%)
L	14% (2%)	25% (14%)	31% (17%)	32% (20%)
M	73% (9%)	82% (80%)	52% (41%)	57% (41%)
N	5% (1%)	17% (10%)	21% (13%)	25% (16%)
Y	3% (1%)	6% (4%)	4% (1%)	11% (9%)

complexity. Overall, *popular* was the fastest, with 4 to 26 tests per second, depending on the size of the data set. The *n-gram* test was the slowest, with 0.5 to 8 tests per second. *Fuzzy*, with 1 to 9 tests per second, was not significantly faster than *n-gram*. It was expected that the *n-gram* algorithm would be significantly faster than *fuzzy*, as it eliminates more users from comparison. Customized implementation, tailored to these specific tests, allowed for a linear implementation of fuzzy string matching rather than a polynomial implementation.

Using outcome feedback provided measurable improvement for every neighborhood algorithm tested. The extent of improvement was directly related to the complexity of the neighborhood algorithm, with *n-gram* and *fuzzy* showing the greatest improvement. However, the improvement in runtime was not significant enough to suggest using outcome feedback to increase efficiency. Improvement in prediction accuracy is the primary reason to consider using outcome feedback in refining the neighborhood.

5.4 Correlation

While successful prediction is important, it is also important to identify a simple attribute (of the data set) that can be used to predict this success. This can help in identifying data sets for which prediction would be ineffective. All three of the measures introduced in Section 4.1; distribution, order, and convergence; exhibit correlation with the success rate. Distribution has the weakest correlation, -

0.22. Order and convergence have correlations of 0.84 and 0.6, respectively. As convergence is a measure of both distribution and order, it is reasonable to assume that convergence alone may be used to identify data sets for which prediction is likely to be successful.

6 Conclusion

In this paper, we demonstrated through extensive testing that search prediction is at least as effective as common recommendation algorithms that do not consider selection order in making recommendations. Prediction is often considerably more successful in identifying the resource that will be selected by the user. It is also faster, as it eliminates the time associated with entering a query. With a predictive search engine; the user a) accesses the search engine, b) sees a list of suggested resources, and selects a resource. Search prediction eliminates the need for query entry, query analysis, and index searching.

Apart from performance, the implementation of a predictive search engine causes a fundamental change in how electronic resources are perceived. Currently, each resource is a single entity. Prediction will suggest a sequence of resources, aggregating individual resources into a meaningful whole.

References

1. Levene, M.: An Introduction to Search Engines and Web Navigation, 2nd edn. John Wiley & Sons (2010)
2. Mostafa, J.: Seeking better web searches. Scientific American 292(2), 66–73 (2005)
3. Hawking, D., Craswell, N., Brailey, P., Griffihs, K.: Measuring search engine quality. Information Retrieval 4(1), 33–59 (2001)
4. Wickelgren, W.A.: Speed-accuracy tradeoff and information processing dynamics. Acta Psychologica 41(1), 67–85 (1977)
5. Konstan, J.A., Riedl, J.: Recommender systems: From algorithms to user experience. User Modeling and User-Adapted Interaction 22, 101–123 (2012)
6. Cleger-Tamayo, S., Fernández-Luna, J.M., Huete, J.F., Pérez-Vázquez, R., Rodríguez Cano, J.C.: A proposal for news recommendation based on clustering techniques. In: García-Pedrajas, N., Herrera, F., Fyfe, C., Benítez, J.M., Ali, M. (eds.) IEA/AIE 2010, Part III. LNCS, vol. 6098, pp. 478–487. Springer, Heidelberg (2010)
7. Tellez, E.S., Chavez, E., Navarro, G.: Succinct nearest neighbor search. In: Proceedings of the 4th International Conference on Similarity Search and Applications (SISAP), pp. 33–40 (June 2011)
8. Zipf, G.K.: The psycho-biology of language. Language 12, 196–210 (1936)
9. Cover, T.M., Hart, P.E.: Nearest neighbor pattern classification. IEEE Transactions on Information Theory 13(1), 21–27 (1967)
10. Xiong, L., Xiang, Y., Zhang, Q., Lin, L.: A novel nearest neighborhood algorithm for recommender systems. In: Proceedings of the Third Global Congress on Intelligent Systems(GCIS), pp. 156–159 (November 2012)
11. Dasarathy, B.V.: Nearest neighbor (NN) norms: NN pattern classification techniques. IEEE Computer Society Press, Los Alamitos (1991)

12. Tan, P.-N., Steinbach, M., Kumar, V.: Introduction to Data Mining, 1st edn. Pearson Addison-Wesley (2005)
13. Jaccard, P.: Étude comparative de la distribution florale dans une portion des alpes et des jura. Bulletin del la Société Vaudoise des Sciences Naturelles 37, 547–579 (1901)
14. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10, 707–710 (1966)
15. Bertsekas, D.P.: Dynamic Programming and Optimal Control, 3rd edn. Athena Scientific (2007)
16. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press (2009)
17. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. Journal of the ACM 21(1), 168–173 (1974)
18. Kolmogorov, A.N.: On tables of random numbers. Theoretical Computer Science 207, 387–395 (1963)

Beyond Bag of Words for Concept Detection and Search of Cultural Heritage Archives

Costantino Grana, Giuseppe Serra, Marco Manfredi, and Rita Cucchiara

Università degli Studi di Modena e Reggio Emilia, Modena MO 41125, Italy

Abstract. Several local features have become quite popular for concept detection and search, due to their ability to capture distinctive details. Typically a Bag of Words approach is followed, where a codebook is built by quantizing the local features. In this paper, we propose to represent SIFT local features extracted from an image as a multivariate Gaussian distribution, obtaining a mean vector and a covariance matrix. Differently from common techniques based on the Bag of Words model, our solution does not rely on the construction of a visual vocabulary, thus removing the dependence of the image descriptors on the specific dataset and allowing to immediately retargeting the features to different classification and search problems. Experimental results are conducted on two very different Cultural Heritage image archives, composed of illuminated manuscript miniatures, and architectural elements pictures collected from the web, on which the proposed approach outperforms the Bag of Words technique both in classification and retrieval.

Keywords: cultural heritage, bag of words, local descriptors, concept detection, image retrieval, similarity search.

1 Introduction

The creation of large digital archives of cultural heritage images, requires experts from different areas to explore the issues of digital collections [7], for the development of information systems and operating platforms able to support both the organization and the access to these repositories [12]. The success of text-based retrieval has raised user expectations about the possibilities of research on media collections, but search engines based only on textual keywords demonstrated their intrinsic limits: the entire content of an image cannot be easily summarized in few keywords. Cultural heritage repositories —incorporating by definition images, texts and often videos, 3D data etc.— should be considered as multimedia repositories so as to adopt all multimedia techniques for digging, understanding and handling such a heterogeneous amount of data.

To cope with multimedia collections, it is necessary to manage the content itself, which may require specific storage and presentation devices, and to manage the associated metadata that can be of different nature and be generated according to a variety of standards. In cultural heritage collections, in which objects are generally subject to some kind of expert analysis, an information system must meet two objectives: the first is to allow the association of elements with their descriptive metadata, and the second is to offer content based retrieval using state-of-the-art technologies. In the last five years most of the proposals for managing multimedia content assume the Bag of Words (BoW) paradigm as an effective approach to provide a compact representation, by clustering local features in a codebook and exploiting visual keyword data for search, concept detection and content understanding.

Recently a novel approach was proposed [9], which represents local features extracted form an image as a multivariate Gaussian distribution, obtaining a mean vector and a covariance matrix. In contrast with the BoW approaches this solution does not require the construction of a visual vocabulary, thus extracting the image descriptor independently from the specific dataset. Allowing the use of linear classifiers, the proposed representation exploits on-line solvers able to deal with large scale datasets that do not fit in memory.

In this paper, we propose to use this novel local features summarization technique for two different tasks, image retrieval and automatic annotation, on two archives of cultural heritage data: the first scenario targets image retrieval on pictures extracted from a Renaissance illuminated manuscript, the second scenario aims at automatically enrich a large dataset of images (automatically crawled from the web) with tags in a concept detection system. For the visual search task (content-based retrieval) we compare different metrics and show which distance is better suited for which descriptor configuration. For the semantic concept detection problem, Stochastic Gradient Descent on-line solver is used, which allows to deal with large scale datasets and high dimensional feature spaces. Differently from the previous use of the descriptor, we evaluate its applicability to coarsely annotated training datasets, showing its ability to deal with the semantic noise, that is the uncertainty of the automatically crawled labels.

2 Related Work

Recently, several local features such as SIFT, SURF, ORB, HOG have become quite popular in representing images due to their ability to capture distinctive details of the images [14]. As introduced, a common approach to integrate the local features into a global representation is to use the the BoW approach, given its simplicity and effectiveness. It consists in three steps: extract local features, generate a codebook and then encode the local features into codes; pool all the codes together to generate the global image representation. The histogram is then fed to a classifier to predict the category [4]. In this approach a key step is the codebook generation, because it is the base to define a high-dimensional BoW histogram. Typically a codebook is built by quantizing local feature descriptors extracted from training images. In recent years, there have been numerous vector quantization approaches to build visual codebooks, such as k-means clustering, or vocabulary trees [15]. However, generated codebooks are not sufficiently flexible to model heterogeneous kinds of new datasets. This is an underlying problem of the BoW approach, because every time the dataset (or more generally the context) changes, the feature vector of an image must be recomputed. Other elements that have attracted research efforts are the encoding and pooling. The simplest encoding in the literature assigns a local feature to the closest visual word and computes a histogram of visual word frequencies [5]. A recent approach replaces the hard quantization of features with soft-assignment in which each local feature is assigned to multiple visual words [6]. In spite of their simplicity, BoW approaches often introduce large quantization errors and limits in the classification performance. To alleviate these problems, several authors have proposed alternative encodings that retain more information about the original image features [21,16,10]. The Locality-constrained Linear Coding [21] applies locality constraint to select similar basis of local descriptors from a codebook, and learns a linear combination weight of these basis to reconstruct each descriptor. Fisher encoding [16], captures the average first and second order differences between the image descriptors and the centers of a Gaussian mixture model; while the Vector of Locally Aggregated descriptors [10] (VLAD) is a non probabilistic version of Fishers kernels. All of these techniques represent an image by exploiting different strategies to describe relationships between local descriptors and visual words of a codebook.

Instead, we propose to use a parametric distribution and compare its capabilities and proprieties to histogram based approaches. A reasonable first choice is to assume that our data follows a Gaussian distribution, because it has useful mathematical properties, it was extensively used and studied, and its representation requires few parameters [1]. In statistical learning a main aspect is to define function to measure similarity/dissimilarity between two distributions. Several measures in closed form expressions between two multivariate Gaussian densities have been proposed, such as the Bhattacharyya divergence and the symmetric Kullback-Leibler (KL) divergence [11]. Based on these dissimilarities, it is possible to build a non-linear kernel function, which can be used in the classification process. However, this would require an enormous computational effort and would soon become prohibitive when moving to large scale classification problem with high-dimensional feature vectors.

3 Multivariate Gaussian Descriptor

The proposed image signature represents local features extracted from an image (or a sub-region) by a multivariate Gaussian distribution. Let $F = \{\mathbf{f}_1 \ldots \mathbf{f}_\mathbf{N}\}$ be a set of local features (e.g. SIFT descriptors, where $d = 128$) extracted through densely sampling in a regular grid on an image W (or a sub-region of W, when Spatial Pyramid Matching is used), we describe them with a multivariate Gaussian distribution supposing that they are normally distributed. The multivariate Gaussian distribution of a set of d-dimensional vectors F is given by

$$\mathcal{N}(\mathbf{f}; \mathbf{m}, \mathbf{C}) = \frac{1}{|2\pi\mathbf{C}|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{f}-\mathbf{m})^T \mathbf{C}^{-1}(\mathbf{f}-\mathbf{m})}, \tag{1}$$

where $|\cdot|$ is the determinant, $\mathbf{m}$ is the mean vector and $\mathbf{C}$ is the covariance matrix ($\mathbf{f}, \mathbf{m} \in \mathbb{R}^d$ and $\mathbf{C} \in \mathbb{S}_{++}^{d\times d}$, with $\mathbb{S}_{++}^{d\times d}$ the space of real symmetric positive semi-definite matrices).

Although the covariance matrix encodes information about the variance of the features and their correlation, it does not lie in a vector space. In fact, the covariance space is a Riemannian manifold and is not closed under multiplication with a negative scalar. Since most of the common machine learning algorithms assume that the data lies in a vector space, we need to define a suitable transformation. The covariance matrix is symmetric positive definite therefore we can adopt the Log-Euclidean metric. The basic idea of the Log-Euclidean metric is to construct an equivalent relationship between the Riemannian manifold and the vector space of the symmetric matrix.

In [19] an approach to map from Riemannian manifolds to Euclidean spaces is described. The first step is the projection of the covariance matrices on an Euclidean space tangent to the Riemannian manifold, on a specific tangency matrix $\mathbf{P}$. The second step is the extraction of the orthonormal coordinates of the projected vector. In the following, matrices (points in the Riemannian manifold) will be denoted by bold uppercase letters, while vectors (points in the Euclidean space) by bold lowercase ones. The projection of $\mathbf{C}$ on the hyperplane tangent to $\mathbf{P}$ becomes:

$$\mathbf{c} = \mathrm{vec}_{\mathbf{I}}\left(\log\left(\mathbf{P}^{-\frac{1}{2}}\mathbf{C}\mathbf{P}^{-\frac{1}{2}}\right)\right), \tag{2}$$

where log is the matrix logarithm operator and $\mathbf{I}$ is the identity matrix, while the vector operator on the tanget space at identity of a symmetric matrix $\mathbf{Y}$ is defined as:

$$\mathrm{vec}_{\mathbf{I}}(\mathbf{Y}) = \left[y_{1,1}\ \sqrt{2}y_{1,2}\ \sqrt{2}y_{1,3}\dots y_{2,2}\ \sqrt{2}y_{2,3}\dots y_{d,d}\right]. \tag{3}$$

Thus, after selecting an appropriate projection origin, every covariance matrix is projected to an Euclidean space. Since $\mathbf{c}$ is a symmetric matrix of size $d \times d$ a $(d^2 + d)/2$-dimensional feature vector is obtained.

The projection point $\mathbf{P}$ is arbitrary and even if, as observed in [13], it could influence the performance (distortion) of the projection, from a computational point of view, the best choice is the identity matrix, which simply translates the mapping into a standard matrix logarithm.

The image descriptor is the concatenation of the mean vector and the projected covariance matrix on a Euclidean space obtaining, in the case of SIFT descriptor, a feature with 8384 dimensions. Finally, we empirically observe that most of the values in the concatenated descriptor are low, while few are high. In order to distribute the values more evenly, we adopt the power normalization method proposed by Perronnin et al. [16].

4 Image Similarity Search

The extracted features can be used for a visual search system. In particular we have considered three different metrics which allow to directly compare the similarity of two images: Cosine Similarity, Euclidean Distance, Kullback-Leibler (KL) divergence.

The cosine distance treats both vectors as unit vectors by normalizing them, giving a measure of the angle between the two vectors. It does provide an accurate measure of similarity but with no regard to magnitude, in contrast to the Euclidean distance which gives the magnitude of difference between the two feature vectors.

The third metric, KL divergence, is a measure of the dissimilarity between two completely determined probability distributions. It is based on Kullback's measure of discriminatory information:

$$I(P_1, P_2) = -\int_{\epsilon} p_1 log(p_1/p_2)\, dx. \tag{4}$$

Kullback realizes the asymmetry of $I(P_1, P_2)$ and describes it as the directed divergence. To achieve symmetry, Kullback defines the divergence as $I(P1, P2)+ I(P2, P1)$. The closed form expression for the symmetric KL (sKL) divergence between two multivariate Gaussian densities, $\mathcal{N}_1$ and $\mathcal{N}_2$, can be written as:

$$d_{KL}(\mathcal{N}_1, \mathcal{N}_2) = \frac{1}{2}\mathbf{u}^T(\mathbf{C}_1^{-1} + \mathbf{C}_2^{-1})\mathbf{u} + \frac{1}{2}tr(\mathbf{C}_1^{-1}\mathbf{C}_2 + \mathbf{C}_2^{-1}\mathbf{C}_1 - 2\mathbf{I}), \tag{5}$$

where tr is the matrix trace, $\mathbf{u} = (\mathbf{m}_1 - \mathbf{m}_2)$ and $\mathbf{I}$ is the identity matrix. Note that since the sKL divergence directly compares the Gaussian distributions is not necessary to project the covariance matrices in a vector space through matrix logarithm.

5 Large Scale Online Learning

Multivariate Gaussian Descriptors can be used to learn SVM for classication, in order to automatically enrich a large dataset with detection of semantic concept. Batch-type SVM solvers, such as LibSVM/LIBLINEAR are effective and well known solutions for train classifiers, however they are not feasible for training large digital archives of cultural heritage images. In fact, they are batch methods which require to go through all data to compute gradient in each iteration and most of them require to pre-load training data into memory, which is impossible when the size of the training data explodes.

To deal with large datasets, we propose to use the stochastic gradient descent (SGD) algorithm, recently introduced for SVM classifiers training, because it is an online method and can be easily parallelized to simultaneously train several classifiers. In fact it updates the learning system on the basis of the loss function measured for a single example.

We have training data that consists of N feature-label pairs, denoted as $\{\mathbf{x}_t, y_t\}_{t=1}^N$, where $\mathbf{x}_t$ is a $s \times 1$ feature vector representing an image and $y_t \in \{-1, +1\}$ is the label of the image. The selected cost function for binary SVM classification is the hinge loss, that can be written as:

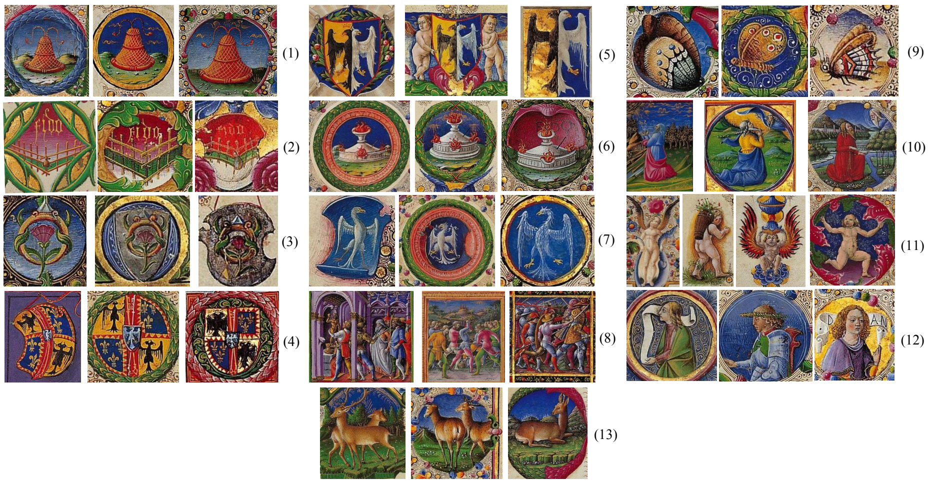


Fig. 1. Example of pictures grouped by class

$$L = \sum_{t=1}^{T} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \max\left[0, 1 - y_t(\mathbf{w}^T \mathbf{x}_t + b)\right], \tag{6}$$

where $\mathbf{w}$ is $s \times 1$ SVM weight vector, λ (nonnegative scalar) is a regularization parameter, and b (scalar) is a bias term. In the SGD algorithm, training data are fed to the system one by one, and the update rule for $\mathbf{w}$ and b respectively are:

$$\begin{aligned} \mathbf{w}_t &= (1 - \lambda\eta)\mathbf{w}_{t-1} + \eta y_t \mathbf{x}_t \\ b_t &= b_{t-1} + \eta y_t \end{aligned} \tag{7}$$

if margin $\Delta_t = y_t(\mathbf{w}^T \mathbf{x}_t + b)$ is less than 1; otherwise, $\mathbf{w}_t = (1 - \lambda\eta)\mathbf{w}_{t-1}$ and $b_t = b_{t-1}$. The parameter η is the step size. We set $\eta = (1 + \lambda t)^{-1}$, following the `vl_pegasos` implementation [20].

In order to parallelize the computation for training SVM classifiers, we randomize the data on disk and we load the data in chunks which fit in memory. We then train the classifiers on further randomizations of the chunks, so that different epochs (one training epoch is defined as providing all training samples to the classifier once) will get the chunks data with different orderings. This last step of randomization turns out to be essential to make the SGD algorithm work properly.

6 Experimental Results

6.1 Datasets Description

We perform the experiments on two different datasets: the first one is a set of pictures from an illuminated manuscript used for image retrieval purposes, the second one was created by querying GoogleImages and used for concept detection.

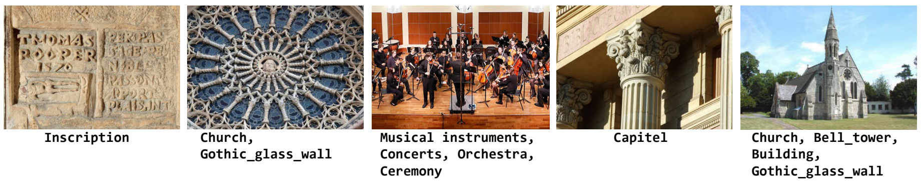

Inscription
Church, Gothic_glass_wall
Musical instruments, Concerts, Orchestra, Ceremony
Capitel
Church, Bell_tower, Building, Gothic_glass_wall

Fig. 2. Sample images extracted from the GoogleCH dataset, with the corresponding ground truth annotations

The first dataset ("Bible dataset") was created from digitalized pages of the Holy Bible of Borso d'Este, duke of Ferrara (Italy) from 1450 A.D. to 1471 A.D. It is one of the best Renaissance illuminated manuscripts in the world, whose original is held in the Biblioteca Estense Universitaria in Modena (Italy). It is composed by 640 pages, with two-column layered text in Gothic font, spaced out with some decorated drop caps, enclosing thousands of painted masterpieces surrounded by rich decorations. These pages have been digitized at 10 Mpixels. Then an automatic procedure [8] has been adopted to segment the miniature illustrations. The set of images obtained from the segmentation process has been manually refined to define the final dataset of 2281 pictures, publicly available for scientific purposes [2] [1]. In collaboration with a group of art experts, the authors performed a manual classification obtaining a subset of 13 classes, characterized by a clear semantic meaning and a significant search relevance (see Fig. 1). As a result, 41% of the original dataset (903 images) has been uniquely annotated into those classes, while the remaining pictures are considered as distractors, often with similar color, shape and texture distribution.

The second dataset ("GoogleCH dataset", see sample images in Fig. 2) was automatically crawled from GoogleImages, by searching for 20 semantic concepts related to cultural heritage (altar, archaeological sites, bell tower, bridge, building, capital, ceremony, church, city square, concerts, crown, Gothic glass wall, inscription, manuscript, mosaic, musical instruments, orchestra, rose window, statue, Tuscany food). For each concept, about 500 images were downloaded and, excluding some which were wrong links, resized to a fixed width of 640 pixels, with a proportional height scaling. The final dataset contains 9594 images, each annotated with the single concept used on the query. Another 1000 images was downloaded, and manually annotated selecting all the concepts present in the image. In this way the training set can be considered a noisy source of information, but definitely containing some useful information. Of course some of the images thus obtained suffer from the ambiguity of the concept terms or their different meaning in different languages (e.g. "inscription" is the French word for "subscription").

[1] Download the Bible dataset at
`http://imagelab.ing.unimo.it/files/bible_dataset.zip`

6.2 Content-Based Visual Similarity Retrieval

In order to propose a valuable comparison, a large variety of visual descriptors based on BoW has been tested in addition to the Multivariate Gaussian Model on the Bible dataset. In particular, we relied on the code and the implementation proposed by [17], employing the following descriptors: RGB Color Histogram (localCH), a combination of three 1D histograms based on the R, G, and B channels of the RGB color space; Transformed Color Histogram (localTCH), RGB histogram obtained by normalizing the pixel value distributions, achieving scale-invariance and shift-invariance with respect to light intensity; Color Moments (localCM), generalized color moments up to the second order, giving a 27-dimensional shift-invariant descriptor; SIFT descriptor (128-dimensional feature vector); RGB-SIFT descriptor (rgbSIFT), for a total 3×128-dimensional feature vector; RG-SIFT descriptor (rgSIFT), computed for R and G channels independently, for a total 2×128-dimensional feature vector; HSV-SIFT descriptor (hsvSIFT), computed converting the original image into the HSV color space, and considering each channel independently, for a total 3×128-dimensional feature vector; Opponent-SIFT descriptor (oppSIFT), describing all of the channels in the opponent color space [18] using SIFT descriptors; C-SIFT descriptor (cSIFT), as proposed by [3], using the C-invariant color space which eliminates the remaining intensity information from the opponent channels; SURF, a scale- and rotation-invariant interest point detector and descriptor which uses integral images and other optimization and approximations to reduce the computational time.

All these descriptors were extracted using the Harris-Laplace region detector. A codebook has been created for every descriptor through a k-means clustering over 10% of the annotated dataset, randomly selected among all the classes in order to ensure an equal amount of visual information for each of them. The employed distance function is the histogram intersection. The sizes k of the codebooks have been determined empirically. In fact, since the clustering is a process of data compression, too small k's (large compression ratio) will force diverse keypoints into the same visual word reducing the quality of the representation; instead too large k's (small compression ratio) might lead to a sparse representation with similar keypoints mapped into different visual words, increasing the computational requirements without any real benefit. Therefore in our experiments we tested values of k between 2^9 and 2^{14}.

Table 1 reports the detailed results obtained using the different features in terms of Mean Average Precision (MAP). For the BoW approaches every column reports the performance using several codebook sizes. The bottom part of the table reports the results obtained with the proposed descriptor, changing the similarity measure. When using the Cosine similarity or the Euclidean distance, the covariance matrix is projected on the tangent space with the matrix logarithm, while the symmetric KL divergence works directly on the covariance matrix. It is possible to observe that the best results are achieved with the Multivariate Gaussian Model of the rgbSIFT descriptors using the dot product, and the result is significantly better then the best result obtained with the BoW

Table 1. Detailed MAP results obtained using the different features. For the BoW approaches (top of the table) every column reports the performance using several codebook sizes. The distance used is always Histogram Intersection. The bottom part of the table reports the results obtained with the proposed descriptor, changing the similarity measure.

	512	**1024**	**2048**	**4096**	**8192**	**16384**
localCH	0.142	0.145	0.147	0.147	0.149	0.147
localTCH	0.129	0.135	0.139	0.141	0.145	0.147
localCM	0.135	0.141	0.146	0.150	0.152	**0.155**
SIFT	0.134	0.136	0.138	0.139	0.142	0.144
rgbSIFT	0.136	0.137	0.138	0.139	0.139	0.142
rgSIFT	0.144	0.147	0.149	0.152	0.152	0.150
hsvSIFT	0.137	0.137	0.138	0.140	0.141	0.139
oppSIFT	0.138	0.141	0.141	0.142	0.143	0.145
cSIFT	0.139	0.139	0.140	0.143	0.143	0.143
SURF	0.119	0.127	0.130	0.129	0.129	0.128

	Cosine Similarity	**Euclidean**	**sKL divergence**
mgm-SIFT	0.146	0.134	0.151
mgm-rgbSIFT	**0.191**	0.119	0.130

approaches (i.e. the local color moments). Fig. 3 shows the performance comparison of the best configuration for each feature summarization method.

6.3 Concept Detection for Image Enrichment

For the concept detection task, we employ the proposed descriptor and compare it with the state-of-the-art BoW approach setting the histogram size, i.e. the number of cluster centers for the k-means algorithm, to 4000. For this task, the rgbSIFT descriptors are extracted at four scales, defined by setting the width of the spatial bins to 4, 6, 8, and 10 pixels respectively, over a dense regular grid with a spacing of 3 pixels. We use the function `vl_phow` provided by the `vl_feat` library [20] and, apart from the spacing step, the defaults options are used. Since the rgbSIFT descriptor is a 384-dimensional feature, the multivariate Gaussian descriptor of an image (or a sub-region) would become an extremely large vector. For this reason, we obtain the image feature by concatenating the multivariate Gaussian descriptors computed for each color channel separately. Images are hierarchically partitioned into 1×1, 2×2 and 1×3 blocks on 3 levels respectively. The resulting descriptors are then concatenated for both methods. The Mean Average Precision (MAP) is used to evaluate the performance, because commonly adopted in concept annotation scenarios.

With this dataset, we apply SGD, which allows us to deal with the large number of images available. Loading the entire training set on memory (9594 samples) occupies about 8.0GB, requiring to split the data in chunks, each loaded

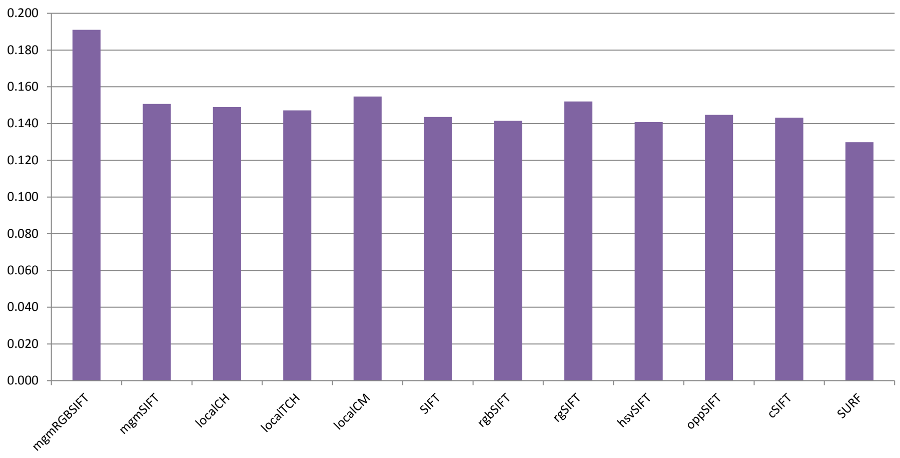


Fig. 3. Comparison of the best MAP values obtained using the different features

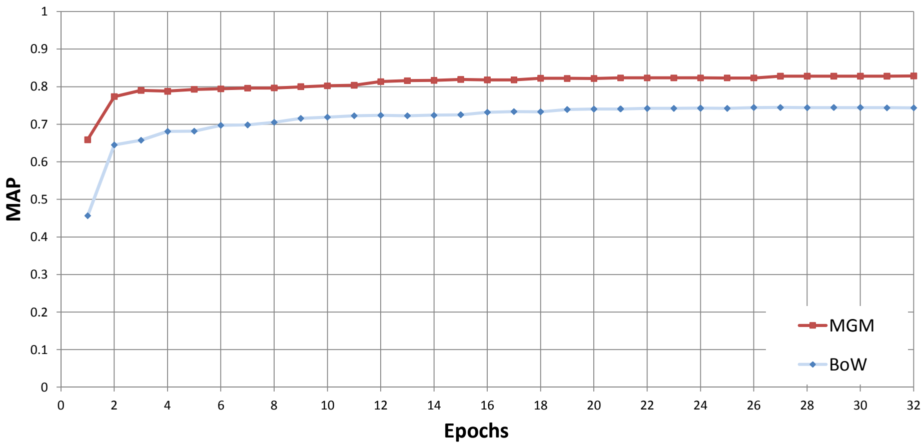


Fig. 4. Mean Average Precision values obtained on the GoogleCH dataset, using the proposed approach and BoW

in turns. To select an appropriate regularization parameter λ for the SGD solver, we randomly split the training set in two and run the SGD varying λ from 10^{-3} to 10^{-7} in power of 10 steps. Based on this preliminary experiments we fix $\lambda = 10^{-5}$. Fig. 4 reports the results of both the proposed approach and BoW in term of MAP at different number of training epochs. Note that the performance rapidly increases in the first 10 epochs, and later tends to remain quite constant. In addition, our method obtains a MAP of 0.83 compared to 0.74 of the BoW approach (at the 30th epoch) and presents better performance at all epochs.

7 Conclusions

In this paper we propose a novel approach for image retrieval and automatic annotation of cultural heritage images. For image retrieval scenario we analyzed three different metrics, while for the automatic annotation we explored the possibility to use noisy data in the training set. The experimental results, on the Bible and GoogleCH datasets, show interesting results both in classification and similarity search with respect to a large variety of visual signatures based on BoW.

References

1. Ali, S., Silvey, S.: A general class of coefficients of divergence of one distribution from another. J. of the Royal Stat. Soc (B) 28(1), 131–142 (1966)
2. Borghesani, D., Grana, C., Cucchiara, R.: Miniature illustrations retrieval and innovative interaction for digital illuminated manuscripts. In: Multimedia Systems (2013)
3. Burghouts, G.J., Geusebroek, J.M.: Performance evaluation of local colour invariants. Computer Vision and Image Understanding 113, 48–62 (2009)
4. Chatfield, K., Lempitsky, V., Vedaldi, A., Zisserman, A.: The devil is in the details: an evaluation of recent feature encoding methods. In: BMVC (2011)
5. Csurka, G., Dance, C.R., Fan, L., Willamowski, J., Bray, C.: Visual categorization with bags of keypoints. In: ECCV Workshop Stat. Learn. Comput. Vision (2004)
6. van Gemert, J.C., Geusebroek, J.-M., Veenman, C.J., Smeulders, A.W.M.: Kernel codebooks for scene categorization. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008, Part III. LNCS, vol. 5304, pp. 696–709. Springer, Heidelberg (2008)
7. Gonçalves, M.A., Fox, E.A., Watson, L.T., Kipp, N.A.: Streams, structures, spaces, scenarios, societies (5s): A formal model for digital libraries. ACM Trans. Inf. Syst. 22(2), 270–312 (2004)
8. Grana, C., Borghesani, D., Cucchiara, R.: Automatic segmentation of digitalized historical manuscripts. In: Multimedia Tools and Applications, pp. 1–24 (2010)
9. Grana, C., Serra, G., Manfredi, M., Cucchiara, R.: Image classification with multivariate gaussian descriptors. In: ICIAP (2013)
10. Jegou, H., Douze, M., Schmid, C., Perez, P.: Aggregating local descriptors into a compact image representation. In: 2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3304–3311 (2010)
11. Kailath, T.: The divergence and Bhattacharyya distance measures in signal selection. IEEE T. Commun. Techn. 15(1), 52–60 (1967)
12. Lagoze, C., Payette, S., Shin, E., Wilper, C.: Fedora: an architecture for complex objects and their relationships. Int. J. Digit. Libr. 6(2), 124–138 (2006)
13. Martelli, S., Tosato, D., Farenzena, M., Cristani, M., Murino, V.: An FPGA-based Classification Architecture on Riemannian Manifolds. In: DEXA Workshops (2010)
14. Mikolajczyk, K., Schmid, C.: A performance evaluation of local descriptors. IEEE T. Pattern Anal. 27(10), 1615–1630 (2005)
15. Nister, D., Stewenius, H.: Scalable recognition with a vocabulary tree. In: IEEE International Conference on Computer Vision and Pattern Recognition (2006)
16. Perronnin, F., Sánchez, J., Mensink, T.: Improving the fisher kernel for large-scale image classification. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010, Part IV. LNCS, vol. 6314, pp. 143–156. Springer, Heidelberg (2010)

17. van de Sande, K.E.A., Gevers, T., Snoek, C.G.M.: Evaluating color descriptors for object and scene recognition. IEEE T. Pattern Anal. 32(9), 1582–1596 (2010)
18. Tuytelaars, T., Mikolajczyk, K.: Local invariant feature detectors: A survey. Foundations and Trends in Computer Graphics and Vision 3(3), 177–280 (2007)
19. Tuzel, O., Porikli, F., Meer, P.: Pedestrian Detection via Classification on Riemannian Manifolds. IEEE T. Pattern Anal. 30(10), 1713–1727 (2008)
20. Vedaldi, A., Fulkerson, B.: VLFeat: An open and portable library of computer vision algorithms (2008), `http://www.vlfeat.org/`
21. Wang, J., Yang, J., Yu, K., Lv, F., Huang, T., Gong, Y.: Locality-constrained linear coding for image classification. In: CVPR (2010)

Large Scale Image Retrieval Using Vector of Locally Aggregated Descriptors

Giuseppe Amato, Paolo Bolettieri, Fabrizio Falchi, and Claudio Gennaro

ISTI - CNR, Pisa, Italy
{giuseppe.amato,paolo.bolettieri,
fabrizio.falchi,claudio.gennaro}@isti.cnr.it

Abstract. Vector of locally aggregated descriptors (VLAD) is a promising approach for addressing the problem of image search on a very large scale. This representation is proposed to overcome the quantization error problem faced in Bag-of-Words (BoW) representation. However, text search engines have not be used yet for indexing VLAD given that it is not a sparse vector of occurrence counts. For this reason BoW approach is still the most widely adopted method for finding images that represent the same object or location given an image as a query and a large set of images as dataset.

In this paper, we propose to enable inverted files of standard text search engines to exploit VLAD representation to deal with large-scale image search scenarios. We show that the use of inverted files with VLAD significantly outperforms BoW in terms of efficiency and effectiveness on the same hardware and software infrastructure.

Keywords: bag of features, bag of words, local features, compact codes, image retrieval.

1 Introduction

In the last few years, local features [16] extracted from selected regions [22] have emerged as a promising method of representing image content in such a way that tasks of object recognition, and other similar (e.g. landmark recognition, copy detection, etc.) can be effectively executed. A drawback of the use of local features is that a single image is represented by a large set (typically thousands) of (local) descriptors that should be individually matched and processed in order to compare the visual content of two images. In principle, a query image should be compared with each dataset object independently. In fact, each local feature of the query should be compared with all the local features of any dataset image in order to find a possible match. Moreover, candidate matches should be validated evaluating a geometric transformation (typically an Homography) able to map a region of the query to a region of the dataset image. Even though data structures as kd-tree [9] are used to efficiently search candidate matching pairs in any two images, still the approach is not scalable.

A very popular method to achieve scalability is the Bag-of-Words (BoW) [21] (or bag-of-features) approach that consists in replacing original local descriptors with the id of the most similar descriptor in a predefined vocabulary. Following the BoW approach, an image is described as a histogram of occurrence of visual words over the global vocabulary. Thus, the BoW approach used in computer vision is very similar to the traditional BoW approach in natural language processing and information retrieval [5]. However, as mentioned in [24], "a fundamental difference between an image query (e.g. 1500 visual terms) and a text query (e.g. 3 terms) is largely ignored in existing index design". From the very beginning [21] a words reduction technique was used (e.g. removing 10% of the more frequent images). In [2], removing query words with small *tf*idf* [20] revealed very good performance in improving efficiency of the BoW approach with a reduced lost in effectiveness. In this work, we make use of the parametric *tf*idf* approach for facilitating trade-offs between efficiency and effectiveness in the BoW approach.

Efficiency and memory constraints have been recently addressed by aggregating local descriptors into a fixed-size vector representation that describe the whole image. In particular, Fisher Vector (FV) [18] and VLAD [12] have shown better performance than BoW [15]. In this work we will focus on VLAD which has been proved to be a simplified non-probabilistic version of FV. Despite its simplicity, VLAD performance is comparable to that of FV [15].

Euclidean Locality-Sensitive Hashing [7] is, as far as we know, the only indexing technique tested with VLAD. While may other similarity search indexing techniques [23] could be applied to VLAD, in this work we decide to investigate the use of inverted files for allowing comparison of the VLAD and BoW approach on the same index. Permutation-Based Indexing [6,4,8] allows using inverted files to perform similarity search with an arbitrary similarity function. Moreover, in [10,1] a Surrogate Text Representation (STR) derivated from the MI-File has been proposed. The conversion of the image description in a textual form allows us to employ the search engine off-the-shelf indexing and searching abilities with a little implementation effort.

In this paper, we applied the STR technique to the VLAD method comparing both effectiveness and efficiency with the state-of-the-art BoW approach on the very same hardware and software infrastructure using the publicly available and widely adopted 1M photos dataset. Given that the STR combination gives approximate results with respect to a complete sequential scan, we also compare the effectiveness of VLAD-STR with the one of standard VLAD. Moreover, we considered balancing efficiency and effectiveness with both BoW and VLAD-STR approaches. For the VLAD-STR, a similar trade-off is obtained varying the number of results used for re-ordering. Thus, we do not only compare VLAD-STR and BoW on specific settings but we show efficiency vs effectiveness graphs for both. For the VLAD-STR, a trade-off is obtained varying the number of results used for re-ordering.

Results confirm the higher performance obtained by VLAD with respect to BoW already showed in [12,15] even when VLAD is combined with STR a off-the-shelf text search engine (i.e., Lucene) is used. Thus, our main contribution is proving that the proposed VLAD-STR approach, can be used, in place of BoW, in combination with traditional text search engines achieving good scalability and preserving the improvement in effectiveness already showed in [15]

The paper is organized as follows. Section 2 presents relevant previous works. In Section 3 we present the STR approach that is used for indexing VLAD with a text search engine. Results are presented in Section 4. Finally, in Section 5 we present our conclusions and describe future work.

2 Related Work

2.1 Local Features

Local features [16] describe the visual content of local interest regions computed for local interest regions [22]. Good local features should be distinctive and at the same time robust to changes in viewing conditions as well as to errors of the detector. Developed mainly in Computer Vision, their typical applications include finding locations and particular objects, detecting image near duplicates and deformed copies. A drawback of the use of local features is that a single image is represented by a large set (typically thousands) of descriptors that should be individually matched and processed in order to compare the visual content of two images.

2.2 Bag of Words (BoW)

State-of-the art techniques for performing large scale content based image retrieval using local features typically involve the BoW approach. BoW was initially proposed in [21] and has been studied in many other papers. The goal of the BoW approach is to substitute each local descriptor of an images with visual words obtained from a predefined vocabulary in order to apply traditional text retrieval techniques to CBIR.

The first step is selecting some visual words creating a vocabulary. The visual vocabulary is typically built clustering, using *k-means*, local descriptors of the dataset ad selecting the centroids. The second step assigns each local descriptor to the identifier of the first nearest word in the vocabulary. For speeding-up this second phase approximate *kd-tree* is often used at a small effectiveness price. At the end of the process, each image is described as a set of visual words. The retrieval phase is then performed using text retrieval techniques considering a query image as disjunctive text-query. Typically, the *cosine* similarity measure in conjunction with a term weighting scheme is adopted for evaluating the similarity between any two images.

Even though inverted files offer a significant improvement in efficiency, in many cases efficiency is not yet satisfactory. In fact, a query image is associated with thousands of visual words. Therefore, the search algorithm on inverted files has to access thousands of different posting lists. From the very beginning [21] words reduction techniques were used (e.g. removing 10% of the more frequent images). However, as far as we know, no experiments have been reported on the impact of the reduction on both efficiency and efficacy.

In [2], various techniques to reduce the number of words describing an image obtained with the BoW approach were evaluated. *tf*idf* [20] revealed very good performance in improving efficiency with a reduced lost in effectiveness. In this work, we make use of the parametric *tf*idf* approach to allow trade-offs between efficiency and effectiveness.

2.3 Fisher Vector

Fisher kernels [11] describe how the set of descriptors deviates from an average distribution, modeled by a parametric generative model. Fisher kernels have been applied in the context of image classification [17] and large scale image search [18]. In [15] it has been proved that Fisher vectors (FVs) extend the BoW. While the BoW approach counts the number of descriptors assigned to each region in the space, FV also encodes the proximate location of the descriptors in each region and has a normalization that can be interpreted as an IDF term. The FV image representation proposed by [17] assumes that the samples are distributed according to a Gaussian Mixture Model (GMM) estimated on a training set. Results reported in [15] reveal that FV indexed using LSH outperforms BoW.

2.4 VLAD

The VLAD representation was proposed in [12]. As for the BoW, a codebook $\{\mu_1, \ldots, \mu_K\}$ is first learned using a cluster algorithm (e.g. k-means). Each local descriptor x_t in each image is then associated to its nearest visual word $NN(x_t)$ in the codebook. For each codeword the differences between the vectors x_t assigned to μ_i are accumulated:

$$v_i = \sum_{x_t : NN(x_t)=i} x_t - \mu_i$$

VLAD is the concatenation of the accumulated vectors, i.e. $V = [v_1^T \ldots v_K^T]$. Please note that all v_i $(i = 1, ...K)$ have the same size which is equal to the size of the used local feature (e.g. 128 for SIFT). Given a codebook $\{\mu_1, \ldots, \mu_K\}$, K is fixed (typically $16 \leq K \leq 128$. Thus the dimensionality of the whole vector V describing any image is fixed too. In other words, VLAD evaluates a global descriptor statistically describing a set of local features with respect to a predefined codebook.

In order to improve the effectiveness of the VLAD approach, two normalization are performed: first, a power normalization with power 0.5; second, a L2 normalization. After this process two global descriptor V_1 and V_2 related to any two images can be compared using the inner product.

The observation that VLAD descriptor has high dimensionality but is relatively sparse and very structured suggests a principal component analysis (PCA) that is usually performed to reduce the size of the K-dimensional VLAD vectors. In this work, we decide not to use dimensionality reduction techniques because we will show that our space transformation approach is independent from the original dimensionality of the description. In fact, the STR approach that we propose, transforms the VLAD description in a set of words from a vocabulary that is independent from the original VLAD dimensionality. In our proposal, PCA could be used to increase efficiency of the STR trasformation.

In [15], it has been shown that VLAD is a simplified non-probabilistic version of FV: VLAD is to FV what k-means is to GMM clustering. The k-means clustering can be viewed as a non-probabilistic limit case of GMM clustering.

In [15] Euclidean Locality-Sensitive Hashing and its variant have been proposed to efficiently search VLAD descriptors.

3 Perspective Transformation and Surrogate Text Representation

In this paper, we propose to index the VLAD descriptors using a surrogate text representation. This allows using any text retrieval engine to perform image similarity search. As discussed later, for the experiments, we implemented these ideas on top of the Lucene text retrieval engine.

The approach to encode global features (as VLAD) used in this paper leverages on the perspective based space transformation developed in [4,10]. The idea at the basis of this technique is that when two descriptors are very similar, with respect to a given similarity function, they 'see' the 'world around' them in the same way. In the following, we will see that the 'world around' can be encoded as a *surrogate text representation* (STR), which can be managed with an inverted index by means of a standard text-based search. The conversion of the visual descriptor in a textual form allows us to employ the search engine off-the-shelf indexing and searching abilities with a little implementation effort.

3.1 STR Generation

Let $\mathcal{D}$ be the domain of the global descriptors o, and $d : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$ a distance function able to assess the dissimilarity between any two $o_1, o_2 \in \mathcal{D}$. Let $R \in \mathcal{D}^m$, be a vector of m distinct *reference descriptors* (or *pivots*) r_i, i.e., $R = (r_1, \ldots, r_m)$. We denote the vector of positions of the reference objects in R ranked by increasing distance with respect to an object $o \in \mathcal{D}$ as $P(o) = (p_1(o), \ldots, p_m(o))$. As an example, if $p_3(o) = 2$ then r_3 is the 2nd nearest object to o among those in R.

The objective is to define a function that transforms a global descriptor into a sequence of terms (ie, a textual document) that can be fed into a text search engine as for instance Lucene. Of course, the ultimate goal is to obtain that the distance between the documents and the query is an approximation of the original distance function of the global descriptors. To achieve this, we associate each element $r_i \in R$ with a unique alphanumeric keyword τ_i, and define a function $t^k(o)$ that returns a space-separated concatenation of zero or more repetitions of τ_i keywords, as follows:

$$t^k(o) = \bigcup_{i=1}^{m} \left(\bigcup_{j=1}^{(k+1)-p_i^k(o)} \tau_i \right)$$

where $p_i^k(o) = p_i(o)$ if $p_i(o) < k$ and $p_i^k(o) = k$ otherwise. By abuse of notation, we denote the space-separated concatenation of keywords with the union operator $\cup$. The inner $\cup$ simply repeat $(k+1)-p_i^k(o)$ times the alphanumeric keyword τ_i used for indicating the reference object $r_i \in R$. The outer $\cup$ concatenates the repeated occurrences, if any, of keywords τ_i for $i = 1...m$. The function $t^k(o)$ is used to generate the STR to be used for both indexing and querying purposes. k is used to consider only the k nearest reference object in R to o, and typically assumes two distinct values for the query q and for the objects in the dataset (k_x for indexing and k_q for querying). For instance, consider the case exemplified in Figure 1, and let us assume τ_1 =A, τ_2 =B, etc. The function t^k will generate the following outputs
$t^{k_x}(o_1)$ = "E E E B B A"
$t^{k_x}(o_2)$ = "D D D C C E"
$t^{k_q}(q)$ = "E E A"

As can be seen intuitively, strings corresponding to o_1 and q are more similar to those corresponding to o_2 e q, this approximate the original distance d. Without going to the mathematical details, we leverage on the fact that a text based search engine will generate a vector representation of STRs generated with $t^{k_x}(o)$ and $t^{k_q}(q)$ containing the number of occurrences of words in texts. This is the case of the simple term-frequency weighting scheme. This means that, if for instance keyword τ_i corresponding to the reference object $r_i \in R$ appears n times, the i-th element of the vector will contain the number n, and whenever τ_i does not appear it will contain 0. With simple mathematical manipulations, it is easy to see how applying the cosine similarity on the query vector and a vector in the database corresponding to $t^{k_x}(o)$ and $t^{k_q}(q)$ respectively, we get a degree of similarity that reflects the similarity order of reference descriptors (pivots) around descriptors in the original space.

For more information on how the technique works from the mathematical point of view, we remind the reader to [10,1]. The impact of k_x on the effectiveness of the search has been studied in [3].

3.2 Reordering Search Results

The idea described so far uses a textual representation of the descriptors and a matching measure based on a similarity offered by standard text search engines to order the descriptors in the dataset in decreasing similarity with respect to the query. The result set is more precise if we order it using the original distance function d.

Suppose we are searching for the k most similar (nearest neighbors) descriptors to the query. We can improve the quality of the approximation by re-ranking, using the original distance function d, the first c ($c \geq k$) descriptors from the

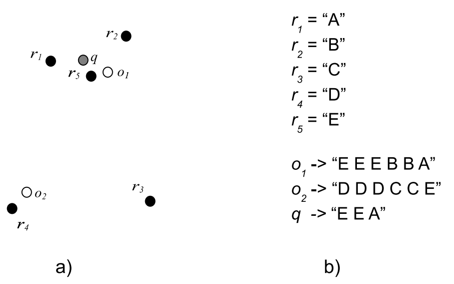


Fig. 1. Example of perspective based space transformation and Surrogate Text Representation. a) Black points are reference objects; white points are data objects; the gray point is a query. b) Encoding of the data objects in the STR.

approximate result set at the cost of more c distance computations. We will show that this technique significantly improves the accuracy, though only requiring a very low search cost. In fact, when c is much smaller than the size of the dataset, this extra cost can be considered negligible with respect to the cost of accessing the inverted file. For instance, when k is 10 and c =1,000, with a dataset size of 1,000,000 it means that we have to reorder a number of descriptors equivalent to just 0.1% of the entire dataset. Usually, this is not true for other access methods, for instance tree-based access methods, where the efficiency of the search algorithms strongly depends on the amount of descriptors retrieved.

4 Experiments

4.1 Setup

INRIA Holidays [14,15] is a collection of 1,491 holiday images. The authors selected 500 queries and, for each of them, a list of positive results. To evaluate the approaches on a large scale, we merged the Holidays dataset with the Flickr1M[1] collection as in [13,12,15]. The ground–truth is the one built on the INRIA Holidays dataset alone, but it is largely accepted that no relevant images can be found between the Flickr1M images. SIFT descriptors and various vocabulary were made publicly available by Jegou et al. for both the Holidays and the Flickr 1M datasets[2]. For the BoW approach we used the 20K vocabulary.

For representing the images using the VLAD approach, we selected 64 reference descriptors using *k-means* over a subset of the Flickr1M dataset. As explained Section 3, a drawback of the perspective based space transformation used for indexing the VLAD with a text search engine is that it is an approximate technique. However, to alleviate this problem, we reorder the best results using

[1] `http://press.liacs.nl/mirflickr/`
[2] `http://lear.inrialpes.fr/~jegou/data.php`

Table 1. Effectiveness (mAP) and efficiency (mSec) with respect to the average number of distinct words per query obtained with the BoW approach varying the query size

BoW		
avg#Words	**mAP**	**avg mSec**
7	0.03	525
16	0.07	555
37	0.11	932
90	0.14	1463
233	0.15	2343

Table 2. Effectiveness (mAP) and efficiency (mSec) obtained with the VLAD approach in combination with STR, with respect to the number of results used for reordering

VLAD		
#reordered	**mAP**	**avg mSec**
0	0.13	139
100	0.24	205
1000	0.29	800
2000	0.30	1461
4000	0.31	2784

the actual distance between the VLAD descriptors. For the STR we used 4,000 references (i.e., $m = 4,000$) randomly selected from the Flickr1M dataset.

During the experimentation also 256 references for VLAD and up to 10,000 references for the STR were selected but the results were only slightly better than the ones presented while efficiency significantly reduced.

All experiments were conducted on a Intel Core i7 CPU, 2.67 GHz with 12.0 GB of RAM a 2TB 7200 RPM HD for the Lucene index and a 250 GB SSD for the VLAD reordering. We used Lucene v3.6 running on Java 6 64 bit over Windows 7 Professional.

The quality of the retrieved images is typically evaluated by means of precision and recall measures. As in many other papers [19,13,18,15], we combined this information by means of the mean Average Precision (mAP), which represents the area below the precision and recall curve.

4.2 Results

In Table 1, we report the mAP obtained with the BoW approach varying the size of the query in terms of average number of distinct words. The query words have been filtered using the *tf*idf* approach as mentioned in Section 2.2. The average number of words per image, as extracted by the INRIA group, is 1,471 and they were all inserted in the index without any filtering. The filtering was used only for the queries and results are reported for average number of distinct words up to 250. In fact, bigger queries result in heavy load of the system. It is worth to mention that we were able to obtain 0.23 mAP performing a sequential scan of the dataset with the unfiltered queries.

The results show that while the BoW approach is in principle very effective (i.e. performing a sequential scan), the high number of query visual words needed for achieve good results significantly reduces his usability. As mentioned in [24], "a fundamental difference between an image query (e.g. 1,500 visual terms) and

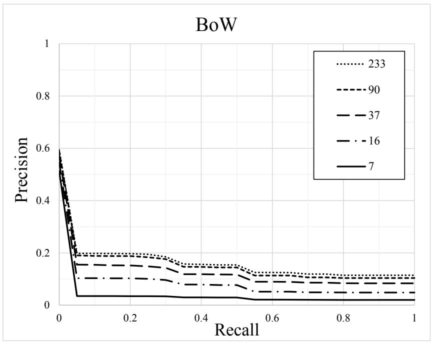


Fig. 2. Precision and recall curves obtained with the BoW approach in combination with STR for various query size

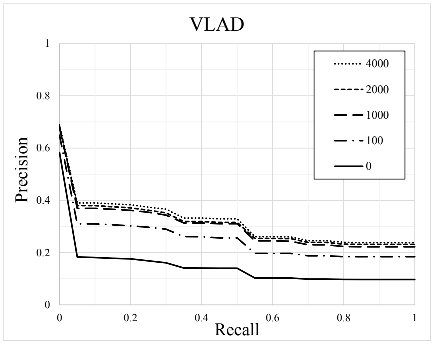


Fig. 3. Precision and recall curves obtained with the VLAD-STR for various number of results used for reordering

a text query (e.g. 3 terms) is largely ignored in existing index design. This difference makes the inverted list inappropriate to index images".

In Table 2, we report the results obtained using the VLAD approach in combination with the use of the STR illustrated in Section 3. As explained in 4.1, given that for indexing the images we used a STR, it is useful to reorder the better results obtained from the text search engine using the actual VLAD distance. Thus, we report mAP and avg mSec per query for the non–reordering case and for various values of results used for reordering. The reordering phase dominates the average query time but it significantly improves effectiveness especially if only 100 or 1,000 objects are considered for reordering. As mentioned before, we make use of SSD for speed-up reordering phase but even higher efficiency could be obtained using PCA as proposed in [15]. Please note that even though the reordering phase cost for VLAD can be reduced, the reported results already show that VLAD outperform BoW.

It is worth to mention that we also performed a sequential scan of the entire dataset obtaining a mAP of 0.34 for VLAD. In fact, as depicted in 3, the results obtained with the VLAD-STR approach are an approximation of the results obtained with a complete pair wise comparison between the query and the dataset object. The same is true when LSH indexing is used as in [15]. Results show that the approximation introduced byt STR does not impact significantly the effectiveness of the system when at least 1,000 objects are considered for reordering.

In Figures 2 and 3 we report the precision and recall curves for BoW and VLAD. The results essentially confirm the ones reported in Table 1 and 2. In fact, no significant differences can be found in the distribution of the precision with respect to the recall.

In Figure 4 we plot mAP with respect to the average query execution time for both BoW and VLAD as reported in Table 1 and Table 2. The graph underlines both the efficiency and effectiveness advantages of the VLAD with respect to the BoW approach.

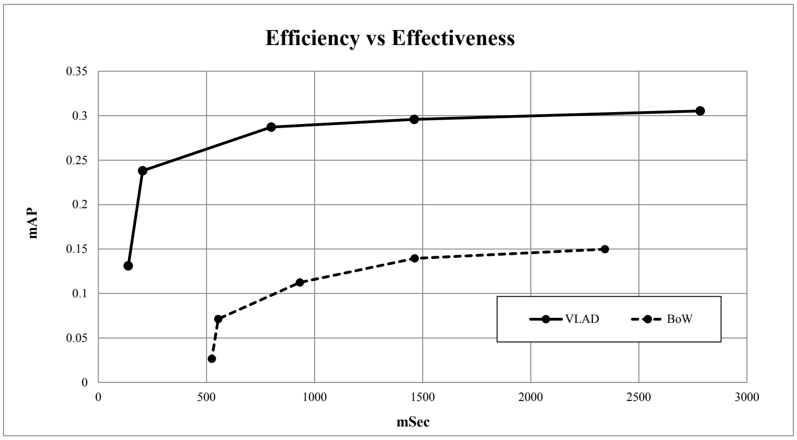


Fig. 4. Effectiveness (mAP) with respect to efficiency (mSec per query) obtained by VLAD and BoW for various settings

5 Conclusions and Future Work

In this work, we proposed the usage of STR in combination with VLAD descriptions in order to index VLAD with off-the-shelf text search engines. Using the very same hardware and text search engine (i.e., Lucene), we were able to compare with the state-of-the-art BoW. Results obtained for BoW confirm that the high number of visual terms in the query significantly reduces efficiency of inverted lists. Even though results showed that this can be mitigated reducing the number of visual terms in the query with a *tf*idf* weighting scheme, the VLAD-STR significantly outperforms BoW in terms of both efficiency and effectiveness. The efficiency vs effectiveness graph reveals that VLAD-STR is able to obtain the same values of mAP obtained with BoW for an order of magnitude less in response time. Moreover, for the same response time, VLAD-STR is able to obtain twice the mAP of BoW.

Future work includes VLAD-STR improving the reordering phase. With regards to efficiency, PCA could be used on VLAD as suggested in [15]. Moreover, in recognition scenarios (e.g., landmark recognition) the reordering phase typically involves geometric consistency checks performed using RANSAC. This could be also done with the VLAD description.

As mentioned in the paper, VLAD is essentially a non probabilistic version of the Fisher Kernels that typically results in almost the same performance. It would be interesting to test the STR approach also with Fisher Kernels comparing with both VLAD-STR and BoW.

References

1. Amato, G., Bolettieri, P., Falchi, F., Gennaro, C., Rabitti, F.: Combining local and global visual feature similarity using a text search engine. In: 2011 9th International Workshop on Content-Based Multimedia Indexing (CBMI), pp. 49–54 (June 2011)
2. Amato, G., Falchi, F., Gennaro, C.: On reducing the number of visualwords in the bag-of-features representation. In: Battiato, S., Braz, J. (eds.) VISAPP 2013 - Proceedings of the International Conference on Computer Vision Theory and Applications, Barcelona, Spain, February 21-24, vol. 1, pp. 657–662. SciTePress (2013) ISBN: 978-989-8565-47-1
3. Amato, G., Gennaro, C., Savino, P.: Mi-file: using inverted files for scalable approximate similarity search. In: Multimedia Tools and Applications, pp. 1–30 (2012)
4. Amato, G., Savino, P.: Approximate similarity search in metric spaces using inverted files. In: Proceedings of the 3rd International Conference on Scalable Information Systems, InfoScale 2008, pp. 28:1–28:10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels (2008)
5. Baeza-Yates, R.A., Ribeiro-Neto, B.A.: Modern Information Retrieval - the concepts and technology behind search, 2nd edn. Pearson Education Ltd., Harlow (2011)
6. Chávez, E., Figueroa, K., Navarro, G.: Effective proximity retrieval by ordering permutations. IEEE Trans. Pattern Anal. Mach. Intell. 30(9), 1647–1658 (2008)
7. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the Twentieth Annual Symposium on Computational Geometry, SCG 2004, pp. 253–262. ACM, New York (2004)
8. Esuli, A.: Mipai: Using the pp-index to build an efficient and scalable similarity search system. In: Proceedings of the 2009 Second International Workshop on Similarity Search and Applications, SISAP 2009, pp. 146–148. IEEE Computer Society, Washington, DC (2009)
9. Friedman, J.H., Bentley, J.L., Finkel, R.A.: An algorithm for finding best matches in logarithmic expected time. ACM Trans. Math. Softw. 3(3), 209–226 (1977)
10. Gennaro, C., Amato, G., Bolettieri, P., Savino, P.: An approach to content-based image retrieval based on the lucene search engine library. In: Lalmas, M., Jose, J., Rauber, A., Sebastiani, F., Frommholz, I. (eds.) ECDL 2010. LNCS, vol. 6273, pp. 55–66. Springer, Heidelberg (2010)
11. Jaakkola, T., Haussler, D.: Exploiting generative models in discriminative classifiers. In: Advances in Neural Information Processing Systems 11, pp. 487–493. MIT Press (1998)
12. Jégou, H., Douze, M., Sánchez, J., Pérez, P.: Aggregating local descriptors into a compact image representation. In: 2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3304–3311 (June 2010)
13. Jegou, H., Douze, M., Schmid, C.: Packing bag-of-features. In: 2009 IEEE 12th International Conference on Computer Vision, September 29 - October 2, pp. 2357–2364 (2009)
14. Jégou, H., Douze, M., Schmid, C., Pérez, P.: Aggregating local descriptors into a compact image representation. In: IEEE Conference on Computer Vision & Pattern Recognition, pp. 3304–3311 (June 2010)
15. Jégou, H., Perronnin, F., Douze, M., Sánchez, J., Pérez, P., Schmid, C.: Aggregating local image descriptors into compact codes. In: IEEE Transactions on Pattern Analysis and Machine Intelligence (September 2012) QUAERO

16. Mikolajczyk, K., Schmid, C.: A performance evaluation of local descriptors. IEEE Transactions on Pattern Analysis and Machine Intelligence 27(10), 1615–1630 (2005)
17. Perronnin, F., Dance, C.: Fisher kernels on visual vocabularies for image categorization. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2007, pp. 1–8 (June 2007)
18. Perronnin, F., Liu, Y., Sanchez, J., Poirier, H.: Large-scale image retrieval with compressed fisher vectors. In: 2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3384–3391 (June 2010)
19. Philbin, J., Chum, O., Isard, M., Sivic, J., Zisserman, A.: Object retrieval with large vocabularies and fast spatial matching. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2007)
20. Salton, G., McGill, M.J.: Introduction to Modern Information Retrieval. McGraw-Hill, Inc., New York (1986)
21. Sivic, J., Zisserman, A.: Video google: A text retrieval approach to object matching in videos. In: Proceedings of the Ninth IEEE International Conference on Computer Vision, ICCV 2003, vol. 2. IEEE Computer Society, Washington, DC (2003)
22. Tuytelaars, T., Mikolajczyk, K.: Local invariant feature detectors: a survey. Found. Trends. Comput. Graph. Vis. 3(3), 177–280 (2008)
23. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search - The Metric Space Approach. Advances in Database Systems, vol. 32. Kluwer (2006)
24. Zhang, X., Li, Z., Zhang, L., Ma, W.-Y., Shum, H.-Y.: Efficient indexing for large scale visual search. In: 2009 IEEE 12th International Conference on Computer Vision, September 29-October 2, vol. 2, pp. 1103–1110 (2009)

Longest Common Subsequence in k Length Substrings

Gary Benson[1,⋆], Avivit Levy[2], and B. Riva Shalom[2]

[1] Department of Computer Science, Boston University, Boston, MA 02215
gbenson@bu.edu
[2] Department of Software Engineering, Shenkar College, Ramat-Gan 52526, Israel
{avivitlevy,rivash}@shenkar.ac.il

Abstract. In this paper we define a new problem, motivated by computational biology, *LCSk* aiming at finding the maximal number of k length *substrings*, matching in both input string while preserving their order of appearance in the input strings. The traditional LCS definition is a spacial case of our problem, where $k = 1$. We provide an algorithm, solving the general case in $O(n^2)$ time, where n is the length of the input, equaling the time required for the special case of $k = 1$. The space requirement is $O(kn)$. In order to enable backtracking of the solution $O(n^2)$ space is needed.

Keywords: Longest Common Subsequence, Similarity of strings, Dynamic Programming.

1 Introduction

The *Longest Common Subsequence* problem, whose first famous dynamic programming solution appeared in 1974 [18], is one of the classical problems in Computer Science. The widely known string version appears in Definition 1.

Definition 1. The String Longest Common Subsequence (LCS) Problem*:*
Input: *Two sequences $A = a_1a_2 \ldots a_n$, $B = b_1b_2 \ldots b_n$ over alphabet Σ.*
Output: *The length of the longest subsequence common to both strings, where a subsequence is a sequence that can be derived from another sequence by deleting some elements without changing the order of the remaining elements.*

For example, for the sequences appearing in Figure 1. $LCS(A, B)$ is 5, where a possible such subsequence is $T\ T\ G\ T\ G$.

The LCS problem has been very well studied. For a survey, see [6]. The problem is mainly motivated by the need to measure similarity over the input sequences. The well known dynamic programming solution [16] requires running time of $O(n^2)$, for two input strings of length n.

⋆ This work was partially funded by NSF grant IIS-1017621.

The LCS problem had also been investigated on more general structures such as trees and matrices [3], run-length encoded strings [4], weighted sequences [2], [7] and more. Many variants of the LCS problem were studied as well, among which LCS alignment [15], [14], [13], constrained LCS [17], [8], restricted LCS [10] and LCS approximation [12].

Motivation. The LCS has been used as a measure of sequence similarity for biological sequence comparison. Its strength lies in its simplicity which has allowed development of an extremely fast, bit-parallel computation which uses the bits in a computer word to represent adjacent cells a row of the LCS scoring matrix and computer logic operations to calculate the scores from one row to the next [1], [9], [11]. For example, in a recent experiment, 25,000,000 bit-parallel LCS computations (sequence length = 63) took approximately 7 seconds on a typical desktop computer [5] or about 60 times faster than a standard algorithm. This speed makes the LCS attractive for sequence comparison performed on high-sequencing data. The disadvantage of the LCS is that it is a crude measure of similarity because consecutive matching letters in the LCS can have different spacings in the two sequences, i.e., there is no penalty for insertion and deletion. What is proposed here is a definition of LCS that makes the measure of similarity more accurate because it forces adjacent letters in the LCS to be adjacent in both sequences. In our problem, the common subsequence is required to consist of k length substrings. A formal definition appears in Definition 2.

Definition 2. The Longest Common Subsequence in k Length Substrings Problem (*LCSk*):
Input: *Two sequences* $A = a_1 a_2 \ldots a_n$, $B = b_1 b_2 \ldots b_n$ *over alphabet* Σ.
Output: *The maximal* ℓ *s.t. there are* ℓ *substrings,*
$a_{i_1} \ldots a_{i_{1+k-1}} \ldots a_{i_\ell} \ldots a_{i_\ell+k-1}$*, identical to* $b_{j_1} b_{\cdot} \ldots j_{1+k-1} \ldots b_{j_\ell} \ldots b_{j_\ell+k-1}$
where $\{a_{i_f}\}$ *and* $\{b_{j_f}\}$ *are in increasing order for* $1 \leq f \leq \ell$ *and where two k length substrings in the same sequence, do not overlap.*

We demonstrate *LCSk* considering the sequences appearing in Figure 1.

$$A = \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \mathrm{T} & \mathrm{G} & \mathrm{C} & \mathbf{G} & \mathbf{T} & \mathrm{G} & \mathbf{T} & \mathbf{G} \end{array}$$

$$B = \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \mathbf{G} & \mathbf{T} & \mathrm{T} & \mathrm{G} & \mathbf{T} & \mathbf{G} & \mathrm{C} & \mathrm{C} \end{array}$$

Fig. 1. An LCS2 example

A possible common subsequence in pairs ($k = 2$) is $GTTG$ obtained from a_4, a_5, a_7, a_8 and b_1, b_2, b_5, b_6. Though $a_6 = b_4$, and such a match preserves the order of the common subsequence, it cannot be added to the common subsequence in pairs, since it is a match of a single symbol. For $k = 3$, one of the possible solutions is TGC achieved by matching a_1, a_2, a_3, with b_5, b_6, b_7. For $k = 4$ a possible solution is $TGTG$ obtained from matching a_5, a_6, a_7, a_8 and b_3, b_4, b_5, b_6. Note that in the last two cases the solution contains a single triple, quadruplet.

The paper is organized as follows: Section 2 gives some preliminaries. The solution for the *LCSk* problem is detailed in Section 3. Section 4 concludes the paper.

2 Preliminaries

The LCSk problem is a generalization of the LCS problem. We might consider using the solution of the latter in order to solve the former. If we perform the LCS algorithm on the input sequences, we can backtrack the dynamic programming table and mark the symbols participating in the common subsequence. We can then check whether those symbols appear in consecutive k length substrings in both input sequences, and delete them if not. Such a procedure guarantees a common subsequence in k length substrings but not necessarily the optimal length of the common subsequence. For example consider $LCS2$ of the sequences appearing on Figure 1. Applying the LCS algorithm on these strings may yield $T\ T\ G\ T\ G$, containing a single non-overlapping pair matching while there exists LCS2 of $T\ G\ T\ G$ having two pair matchings. Hence, a special algorithm designed for *LCSk* is required.

As the LCSk problem considers matchings of k consecutive symbols, we call such a matching, throughout this paper, a *k matching* and define the following definitions:

Definition 3

$$kMatch(i,j) = \begin{cases} 1\ if\ a_{i+f} = b_{j+f},\ \ for\ every\ 0 \le f \le k-1 \\ 0\ Otherwise \end{cases}$$

If $kMatch(i,j) = 1$, the matching substring is denoted (i,j).

Definition 4. ***Predecessors.*** *Let $candidates(i,j)$ be the set of all longest common subsequences, consisting of k matchings, of prefix $A[1...i+k-1]$ and prefix $B[1...j+k-1]$. Then let $pred(i,j)$ be all the possible last k matchings in $candidates(i,j)$. That is, $pred(i,j) = \{(s,t) | \exists c \in candidates(i,j),\ where\ (s,t)\ is\ the\ last\ \ k\ matching\ in\ c\}$.*

We define the length of $p \in pred(i,j)$ derived from candidate c, to be the number of k matchings in c and denote it by $|p|$.

Example. Consider LCS2 of the sequences of Figure 1. Let $candidates(5,3)$ be the common subsequences in pairs of $B[1...4] = G\ T\ T\ G$ and of $A[1...6] = T\ G\ C\ G\ T\ G$, thus, $candidates(6,4)$ contains$\{TG, GT\}$. TG can be obtained in two ways: a_1a_2 matched to b_3b_4, or a_5a_6 matched to b_3b_4, and GT by a_4a_5 matched to b_1b_2 therefore, we have $pred(i,j) = \{(1,3),(5,3),(4,1)\}$. In this example all predecessors are of length 1. Keeping the predecessors enables backtracking to reveal the longest common subsequence in k length substrings itself.

The following Lemma is necessary for the correctness of the solution.

Lemma 1. *Let p_1,$p_2 \in pred(i,j)$, then if $|p_1| + 1 = |p_2|$, then any maximal common subsequence of k length substrings using the k matching p_2 has length greater or equal to that using the k matching p_1.*

Proof: Suppose $p_1 = (s, t)$ and $p_2 = (s', t')$. Several cases are possible for p_1, p_2:

1. If $s' < s$ and $t' < t$, then the candidate whose last k matching is p_2 might be further extended till $A[s]$ and $B[t]$, enlarging the difference between p_1 and p_2.
2. If $s' = s$ and $t' = t$ both predecessors have the same opportunities for extension.
3. If $s + k - 1 < s'$ and $t + k - 1 < t'$, the k matching (s', t') can be added to the candidate whose last k matching is (s, t), contradicting its maximality.
4. If there is an overlap between the k matchings represented by the predecessors, $s < s' < s + k$ or $t < t' < t + k$, starting from $a_{s'+k}$, every k matching can be used to extend the common subsequence in k length substring, represented by both predecessors. However, the subsequence using p_1 cannot have an additional k matching before $a_{s'+k}$, as overlaps are forbidden. Consequently, the difference between the length of p_1 and p_2 is preserved in the extended maximal common subsequences. ∎

3 Solving the LCSk Problem

As in other LCS variants, we solve the problem using a dynamic programming algorithm. We denote by $LCSk_{i,j}$ the longest common subsequence, consisting of k matchings in the prefixes $A[1...i + k - 1]$ and $B[1...j + k - 1]$. Lemma 2 below, formally describes the computation of $LCSk_{i,j}$.

Lemma 2. ***The Recursive Rule***

$$LCSK_{i,j} = max \begin{cases} LCSk_{i,j-1}, \\ LCSk_{i-1,j}, \\ LCSk_{i-k,j-k} + kMatch(i,j) \end{cases}$$

Proof: $LCSK_{i,j}$ contains the maximal number of common k length substrings, preserving their order in the input sequences. A possible subsequence can be constructed by matching the substrings $a_i, \ldots, a_{i+k-1}$ with $b_j, \ldots, b_{j+k-1}$, in case all a_{i+f} and all b_{j+f}, for $0 \leq f \leq k-1$, are not part of previous k *matching*s. This is guaranteed when considering the prefixes $A[1..i-k]$ and $B[1..j-k]$ while trying to extend by one the common subsequence for cell $LCSk_{i,j}$. Another option of extending the subsequence is by using the k matching (s, j) , for $s < i$. Similarly, we can use the k matching (i, t) for $t < j$. Note that the options of extending $LCSk_{i-f,j-f}$, for $1 \leq f \leq k - 1$ is included in both $LCSk_{i,j-1}$ and $LCSk_{i-1,j}$. These claims can be easily proven using induction. ∎

According to Lemma 2 we can solve the $LCSk$ problem using a dynamic programming algorithm working on a two dimensional table of size $(n - k + 1)^2$ where the rows represent the A sequence and the columns stand for sequence B. Cell $LCSk[i, j]$ contains the value $LCSk_{i,j}$ and the appropriate predecessors. Nevertheless, when we wish to attain the common subsequence itself, we encounter a complication.

In the original LCS algorithm, computing the common subsequence, requires maximizing three options of possible prefixes of the LCS. When some of these prefixes have the same length, there is no significance which of them is chosen, as a single common subsequence is sought and the selection has no effect on future matches. However, in the $LCSk$ problem the situation is different. For example, consider $LCS2$. Let $A = a\ c\ a\ b$ and $B = c\ a\ c\ a\ b$. LCS2[3,3] equals 1 due to the 2 matching $(1, 2)$ (matching a_1a_2 to b_2, b_3) ("ac"), or by the 2 matching $(2, 1)$ ("ca"). In spite of the fact that both common subsequences, share the same length, the former is part of the final solution as it enables a further 2 matching $(3, 4)$ while the latter cannot be extended due to the overlap restriction. It, therefore, seems that all possibilities of common subsequence in k length substrings, that is, all predecessors should be saved at every calculation in order to enable a correct backtracking of the optimal solution. As the dynamic programming proceeds, this information can exponentially increase. Nevertheless, we prove in Lemma 3 that in the $LCSk$ problem only one maximal previously computed subsequence is required.

The three options of forming $LCSk_{i,j}$, appearing in Lemma 2 form $candidates(i, j)$, hence $pred(i, j)$. Therefore, the $pred(i, j)$ set should be updated after computing $LCSk_{i,j}$.

Corollary 1. *If* $LCSk_{i,j-1} = LCSk_{i-1,j} = LCSk_{i-2,j-2}+1$*, and kMatch(i,j)=1 then* $pred(i, j) = pred(i, j-1) \bigcup pred(i-1, j) \bigcup (i, j)$.

If $LCSk_{i,j-1} = LCSk_{i-1,j}$ *and kMatch(i,j)=0 then* $pred(i, j) = pred(i, j-1) \bigcup pred(i-1, j)$.

In both cases, if one or more of the relevant $LCSk_{x,y}$, $x \leq i, y \leq j$ *has shorter length, its corresponding pred is not included in* $pred(i, j)$.

Proof: Note, that the length of a predecessor $p \in pred(i, j)$ equals the value of $LCSk_{i,j}$. Due to Lemma 1 there is no necessity to consider the shorter predecessors. Suppose all three sets contain predecessors representing common subsequences of the same length. Without further information, we cannot determine which common subsequence ending in $pred(i, j-1)$, $pred(i-1, j)$, or in k matching of (i, j), will be in the maximal output, therefore, all predecessors must be considered. ∎

3.1 The Backtrack Process

Using the recursive rule of Lemma 2, the value computed for $LCSk_{i-k+1,j-k+1}$ is the length of the common subsequence in k length substrings of sequences A and B. In order to obtain the common subsequence itself we perform the following procedure. Consider the value saved in cell $LCSk[i, j]$, where i and j are initialized by $n-k+1$. We suppose that each cell contains a single predecessor, as will be proven hereafter in Lemma 3. Let the predecessor saved in the current cell be (x, y). Two cases are regarded as long as $i, j > 0$.

1. if $x = i$ and $y = j$, then a k matching starts in these indices, therefore a_{i+f} *for every* $0 \leq f \leq k-1$ can be added to the constructed output, preserving the increase of the indices. In order to proceed we decrease both i, j by k to avoid previous k matchings overlapping (i, j).
2. Otherwise, no k matching occurs in the current indices. The predecessor (x, y) directs us to the cell containing a k matching which is part of an LSCk with the value $LCSk_{i,j}$. Therefore, we decrease the indices $i = x$ and $j = y$

3.2 Predecessors Elimination

We aim at minimizing the number of predecessors per $LCSk[i, j]$ and therefore define a process of predecessors elimination. Eliminating a predecessor p, that is, deleting it from $pred(i, j)$ can be safely done if a maximal common subsequence in k length substrings of the same length can be attained using another predecessor from $pred(i, j)$. Lemma 3 provides the elimination procedure and its correctness.

Lemma 3. ***Elimination Lemma*** *Let* p_1, $p_2 \in pred(i, j)$ *be* k *matchings, where* $|p_1| = |p_2|$, *then one of* p_1, p_2 *can be arbitrarily eliminated.*

Proof: Let $p_1 = (s, t)$, $p_2 = (s', t')$. In case $kMatch(i, j) = 0$ then, if the backtracking pass through table cell [i,j] it implies that the previously found k matching is $(i + k, j + k)$ due to the second case of the backtracking procedure. Moreover, according to Corollary 1, both $\{s, s'\} \leq i$ and $\{t, t'\} \leq j$. As a consequence, there is no preference to one of the equal length predecessors as both cannot overlap the previous k matching.

Suppose then that $kMatch(i, j) = 1$ and $p_2 = (i, j)$. According to the backtracking procedure, we get to cell $[i, j]$ either by the first case of the procedure where there is a k matching $(i+k, j+k)$ or by its second case where at cell $[i', j']$ there is no k matching but it contains a predecessor (i,j). The latter implies that the previously found k matching is $(i + k + f, j + k + h)$ for $f, h > 0$.

There are two cases to consider.

1. If no optimal solution uses the k matching (i, j) it implies that the optimal solution includes k matchings (i', j') and (i'', j'') where $i' < i < i' + k$ and $i'' - k < i < i''$ or $j' < j < j' + k$ and $j'' - k < j < j''$. If only one inequality holds for i or j then some optimal solution will include (i, j), contradicting the case definition. According to the first case of the backtrack procedure, when backtracking from cell [i", j"], including the k matching (i'', j''), we decrease both indices by k. Since $i'' - k < i$ and $j'' - k < j$ cell [i,j] will not be considered, therefore even if we saved p_2, that is we eliminated p_1, it has no consequence on the optimal solution.
2. If there exists an optimal solution including p_2 but we arbitrarily eliminated it. Since we proved that the previously found k matching is $(i+k+f, j+k+h)$ for $f, h \geq 0$ there is no preference to p_2 over p_1 as they are both of the same length and both do not overlap the previously found k matching according to Corollary 1. Apparently, p_1 is included in another optimal solution. ∎

Example. Figure 2 depicts an LCS2 table. We demonstrate the two cases in Lemma 3 where $kMatch(i, j) = 1$. For the first case, consider cell $LCS2[5, 6]$ including $pred_{5,6} = \{(4, 5), (5, 6)\}$. Suppose we arbitrarily eliminate $(4, 5)$. The LCS2 may contain the 2 matching $(5, 6)$ that overlaps with (4,1),(4,5) and on the same time overlaps also $(6, 7)$ what can decrease the length of the solution. Nevertheless, according to the backtracking procedure, after considering $LSC2[6, 7]$ we decrease the indices and go to $LCS2[4, 5]$ in which $(5, 6)$ cannot exist, due to Corollary 1.

For the second case consider cell $LCS2[2, 3]$ including $pred_{2,3} = \{(1, 1), (2, 3)\}$. (2,3) is included in one of the optimal solution. Suppose we eliminated it and retained $(1, 1)$. The backtracking path goes through cells $[7, 7]$ to $[6, 7]$ to $[4, 5]$ to $[2, 3]$ where it finds a non overlapping predecessor $(1, 1)$ with the same length as the deleted $(2, 3)$.

Theorem 1 *The $LCSK(A, B)$ problem can be solved in $O(n^2)$ time and $O(kn)$ space, where n is the length of the input sequences A, B. Backtracking the solution requires time of $O(\ell)$ where ℓ is the number of k matchings in the solution, and $O(n^2)$ space.*

Proof: The algorithm fills a table of size $(n-k+1)^2$. Each entry is filled according to Lemma 2 in constant time as we perform a constant number of comparisons. We assume that k is rather a small constant thus computing $kMatch(i, j)$ is done in constant time. In addition unifying three *pred* sets of size one each, does not increase the time requirements per entry. The Elimination procedure requires also constant time according to Lemma 3. All in all, constant time

	1	2	3	4	5	6	7	8
	C	T	T	G	C	T	T	T
1 C	1 (1,1)	1 (1,1)	1 (1,1)	1 (1,1)	1 (1,1)(1,5)	1 (1,5)	1 (1,5)	- -
2 T	1 (1,1)	1 (1,1)	1 (1,1)(2,3)	1 (1,1)	1 (1,1)(1,5)	1 (1,5)	1 (1,5)	- -
3 G	1 (1,1)	1 (1,1)	1 (1,1)	1 (1,1)	1 (1,1)(1,5)	1 (1,5)	1 (1,5)	- -
4 C	1 (4,1)	1 (4,1)	1 (1,1),(4,1)	1 (4,1)	2 (4,5)	2 (4,5)	2 (4,5)	- -
5 T	1 (4,1)	1 (4,1),(5,2)	1 (4,1)	1 (4,1)	2 (4,5)	2 (4,5),(5,6)	2 (4,5),(5,7)	- -
6 T	1 (4,1)	1 (4,1),(6,2)	1 (4,1)	1 (4,1)	2 (4,5)	2 (4,5),(6,6)	3 (6,7)	- -
7 T	1 (4,1)	1 (4,1)	2 (7,3)	2 (7,3)	2 (7,3),(4,5)	2 (7,3)(6,6)	3 (6,7)	- -
8 G	-	-	-	-	-	-	-	-

Fig. 2. An LCS2 Table. The numbers represent the length of the common subsequence. The pairs in parenthesis stand for the predecessors. Each cell contains all possible predecessors according to Corollary 1. Due to the Elimination Lemma only one predecessor is retained in the to following cells.

operations are performed for each of the table entries, concluding in $O(n^2)$ time requirement for computing the optimal length of the common subsequence in k length substrings.

During the backtracking process we go through the cells representing the k matchings of one optimal solution. If the difference between two such k matchings is more than k, we will go through an intermediate cell whose predecessor directs us to the next k matching. Hence finding the common subsequence in k length substrings requires $O(\ell)$ where ℓ is the number of k matchings in the solution.

Regarding space: Each of the n^2 entries contains, according to Corollary 1 three predecessors and the Eliminate function, due to Lemma 3, results in a single predecessor before considering further entries, implying $O(n^2)$ space requirement. Nevertheless, due to Lemma 2, during the computation of $LCSk[i,j]$ we need only row $i-k$ and column $j-k$. As a consequence, at each step we save only k rows and columns implying the space requirement is $O(kn)$. In order to backtrack the solution, the whole table is needed, implying $O(n^2)$ space requirement. ∎

4 Conclusion

In this paper we defined a generalization of the LCS problem, where each matching must consist of k consecutive symbols. We proved that using the known LCS algorithm does not always output an optimal solution. However, by thoroughly understanding the traits of the problem we proved a similar algorithm with the same time complexity can solve the problem. Due to the importance of the LCS problem as a measure of similarity between the inputs, more generalizations may be thought of.

References

1. Allison, L., Dix, T.I.: A bit-string Longest-Common-Subsequence. Information Processing Letters 23(5), 305–310 (1986)
2. Amir, A., Gotthilf, Z., Shalom, R.: Weighted LCS. J. Discrete Algorithms 8(3), 273–281 (2010)
3. Amir, A., Hartman, T., Kapah, O., Shalom, R., Tsur, D.: Generalized LCS. Theor. Comput. Sci. 409(3), 438–449 (2008)
4. Apostolico, A., Landau, G.M., Skiena, S.: Matching for run-length encoded strings. Journal of Complexity 15(1), 4–16 (1999)
5. Benson, G., Hernandez, Y., Loving, J.: A bit-parallel, general integer-scoring sequence alignment algorithm. In: Fischer, J., Sanders, P. (eds.) CPM 2013. LNCS, vol. 7922, pp. 50–61. Springer, Heidelberg (2013)
6. Bergroth, L., Hakonen, H., Raita, T.: A survey of longest common subsequence algorithms. In: Proc, 7th Symposium on String Processing and Information Retrieval (SPIRE), pp. 39–48 (2000)
7. Blin, G., Jiang, M., Vialette, S.: The Longest Common Subsequence Problem with Crossing-Free Arc-Annotated Sequences. In: Calderón-Benavides, L., González-Caro, C., Chávez, E., Ziviani, N. (eds.) SPIRE 2012. LNCS, vol. 7608, pp. 130–142. Springer, Heidelberg (2012)

8. Chen, Y.C., Chao: On the generalized constrained longest common subsequence problems. Journal of Combinatorial Optimization 21(3), 383–392 (2011)
9. Crochemore, M., Iliopoulos, C.S., Pinzon, Y.J., Reid, J.F.: A fast and practical bit-vector algorithm forthe longest common subsequence problem. Information Processing Letters 80(6), 279–285 (2001)
10. Gotthilf, Z., Hermelin, D., Landau, G.M., Lewenstein, M.: Restricted LCS. In: Chavez, E., Lonardi, S. (eds.) SPIRE 2010. LNCS, vol. 6393, pp. 250–257. Springer, Heidelberg (2010)
11. Hyyro, H.: Bit parallel LCS- length computation revisited. In: Proc. 15th Australasian Workshop on Combinatorial Algorithms, AWOCA (2004)
12. Landau, G.M., Levy, A., Newman, I.: LCS approximation via embedding into locally non-repetitive strings. Inf. Comput. 209(4), 705–716 (2011)
13. Landau, G.M., Myers, E.W., Ziv-Ukelson, M.: Two Algorithms for LCS Consecutive Suffix Alignment. In: Sahinalp, S.C., Muthukrishnan, S.M., Dogrusoz, U. (eds.) CPM 2004. LNCS, vol. 3109, pp. 173–193. Springer, Heidelberg (2004)
14. Landau, G.M., Schieber, B., Ziv-Ukelson, M.: Sparse LCS Common Substring Alignment. In: Baeza-Yates, R., Chávez, E., Crochemore, M. (eds.) CPM 2003. LNCS, vol. 2676, pp. 225–236. Springer, Heidelberg (2003)
15. Landau, G.M., Ziv-Ukelson, M.: On the Common Substring Alignment Problem. J. Algorithms 41(2), 338–359 (2001)
16. Hirschberg, D.S.: A Linear space algorithm for Computing Maximal Common Subsequences. Commun. ACM 18(6), 341–343 (1975)
17. Tsai, Y.T.: The constrained longest common subsequence problem. Information Processing Letters 88(4), 173–176 (2003)
18. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. J. ACM 21, 168–173 (1974)

Database Similarity Join for Metric Spaces

Yasin N. Silva, Spencer S. Pearson, and Jason A. Cheney

Arizona State University,
4701 W. Thunderbird Road, Glendale, AZ 85306, USA
{ysilva,sspearso,jcheney1}@asu.edu

Abstract. Similarity Joins are recognized among the most useful data processing and analysis operations. They retrieve all data pairs whose distances are smaller than a predefined threshold ε. While several standalone implementations have been proposed, very little work has addressed the implementation of Similarity Join as a physical database operator. In this paper, we focus on the study, design and implementation of a Similarity Join database operator for any dataset that lies in a metric space (DBSimJoin). We describe the changes in each query engine module to implement DBSimJoin and provide details of our implementation in PostgreSQL. The extensive performance evaluation shows that *DBSimJoin* significantly outperforms alternative approaches.

1 Introduction

Similarity Joins (SJs) have been studied and extensively used in multiple application domains, e.g., data cleaning and sensor networks. Several SJ algorithms have been previously proposed. Very little work, however, has addressed the implementation of SJ as a first-class database operator. This type of implementation would enable interesting similarity queries that combine SJ with other operators. In this paper, we present a SJ database operator for any dataset that lies in a metric space. The main contributions of this paper are:

– We present *DBSimJoin*, a SJ database operator that is fully integrated into the database engine and incorporates techniques to: (1) enable a non-blocking behavior, (2) prioritize the early generation of results, and (3) support the database iterator interface (functions *open*, *getNext*, and *close*).
– To the best knowledge of the authors, DBSimJoin is the first SJ database operator that can be used with any dataset that lies in a metric space. The operator can be used with various distance functions and data types.
– We present multiple guidelines to implement DBSimJoin as an integrated component of a database system. We also provide details of our implementation in PostgreSQL, a popular open-source database system.
– We thoroughly evaluate the performance of DBSimJoin with multiple data types and show that it significantly outperforms alternative approaches.
– We show that DBSimJoin can be combined with other operators in complex similarity queries and can be used in important query transformation rules that enable cost-based query optimization, e.g., pushing selection below join, and Eager and Lazy aggregation transformations.

The rest of this paper is organized as follows. Section 2 presents the related work. Section 3 describes DBSimJoin. The performance evaluation is presented in Section 4. Section 5 concludes the paper.

2 Related Work

Significant work has been carried out on the study of the SJ (retrieves all data pairs whose distances are smaller than a threshold ε). This work proposed techniques to implement them primarily as standalone algorithms outside of a database system. Some implementation techniques rely on the use of pre-built indices, e.g., eD-index [8], D-index [7], and List of Twin Clusters (LTC) [13]. They strive to partition the data while clustering together the similar objects. While these indexing techniques support the SJ operation they also have some shortcomings: D-index and eD-index may require rebuilding the index to support queries with different ε, eD-index is applicable only to the case of self-joins, and LTC requires indexing each pair of input sets jointly. Several non-index-based techniques have also been proposed, e.g., EGO, GESS, and QuickJoin. The Epsilon Grid Order (EGO) algorithm [3] imposes an ε-sized grid over the space and uses a schedule of reads of blocks that minimizes I/O. The Generic External Space Sweep (GESS) algorithm [6] creates hypersquares centered on each data point with epsilon length sides, and joins these hypersquares using a spatial join on rectangles. The Quickjoin algorithm [12] recursively partitions the data until the subsets are small enough to be efficiently processed using a nested loop join. Quickjoin has been shown to outperform EGO and GESS [12]. DBSimJoin, the operator presented in this paper, builds on Quickjoin's approach to partition the data. However, the focus of our work is the design and implementation of an efficient database operator. The differences with the work in [12] are: (1) DBSimJoin uses a different partitioning sequence that prioritizes early generation of results and minimizes query response time, (2) DBSimJoin uses a non-blocking implementation approach that fully supports the database iterator interface, (3) DBSimJoin assumes a limited number of memory buffers, (4) our experimental section evaluates the effect on performance of key parameters not evaluated in [12], e.g., dimensionality and number of pivots, and (5) we study how DBSimJoin can be combined with other operators and used in query transformation rules.

Also, of importance is the work on SJ techniques in the context of database systems. Some work has focused on the implementation of SJs using standard database operators [4,5,10]. These techniques are applicable only to string or set-based data. The general approach decomposes data and query strings into sets of grams (substrings of a string that are used as its signature), and stores the results on separate tables. Then, the result of the SJ can be obtained using standard SQL queries. DBSimJoin is experimentally compared with one such technique (SSJoin [4]) in Section 4.1. More recently, the work in [15,14] proposed a SJ database operator for 1D numerical data based on a plane-sweep algorithm. This approach, however, cannot be easily extended to other data types. DBSimJoin is more generic and can be used with any dataset that lies in a metric space.

DBSimJoin supports multiple data types and distance functions. In a recent demo paper [17], we showed how DBSimJoin can be used to identify similar images (feature vectors), and similar publications in a bibliographic database.

3 The DBSimJoin Operator

The Similarity Join (SJ) operation between two datasets R and S is defined as: $R \bowtie_{\theta_\varepsilon(r,s)} S = \{\langle r,s\rangle | \theta_\varepsilon(r,s), r \in R, s \in S\}$, where $\theta_\varepsilon(r,s)$ represents the Similarity Join predicate, i.e., $dist(r,s) \le \varepsilon$. Even though the tuples of relations R and S are combined by DBSimJoin, each tuple is assumed to have an attribute that identifies its relation. DBSimJoin iteratively partitions the input data into smaller partitions until each partition is small enough to be efficiently processed by an in-memory SJ routine. The overall process is divided into a sequence of rounds. The initial round partitions the input data while any subsequent round partitions the data of a previously generated partition. Each round produces: (1) result pairs (links) for the small partitions that can be processed by an in-memory SJ routine, and (2) intermediate data for the partitions that will require further partitioning. Intermediate data is stored on disk (hibernated). The DBSimJoin operator executes the required rounds until all the input and intermediate data is processed. While rounds other than the first one can be processed in any order, DBSimJoin uses a partitioning sequence that favors the early generation of result links.

3.1 DBSimJoin Rounds

A goal of the partitioning step in each round is to divide the round input data into a set of partitions such that all the result links in the input data are obtained by combining the links found in each partition independently. To accomplish this, the input data is partitioned into: (1) non-overlapping partitions (*base partitions*), and (2) partitions that contain the records in the boundary of each pair of base partitions (*window-pair partitions*). Partitioning is performed using a set of K pivots, i.e., a random subset of the records to be partitioned. Each base partition contains all the records that are closer to a given pivot than to any other pivot. Each window-pair partition contains the records in the boundary between two base partitions. The window-pair records should be a superset of the records whose distance to the hyperplane that separates the base partitions is at most ε. This hyperplane does not always explicitly exist in a metric space. Instead, it is implicit and known as a *generalized hyperplane*. Since the distance of a record t to the generalized hyperplane between two partitions with pivots P_0 and P_1 cannot always be computed exactly, a lower bound of the distance is used [11]: $gen\_hyperpln\_dist(t, P_0, P_1) = (dist(t, P_0) - dist(t, P_1))/2$. This distance is replaced with the exact distance if this can be computed, e.g., in Euclidean spaces. Processing the window-pair partitions guarantees the identification of the links between records that belong to different base partitions. A round that repartitions a base partition or the initial input data is referred to as a *base*

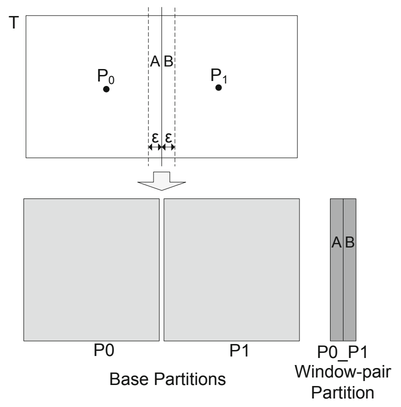


Fig. 1. Repartitioning a base partition

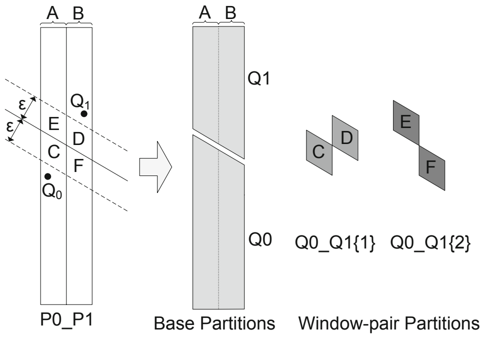
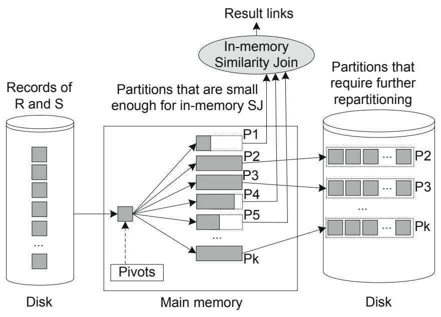


Fig. 2. Repartitioning a window-pair partition

partition round, a round that repartitions a window-pair partition is referred to as a *window-pair partition round*.

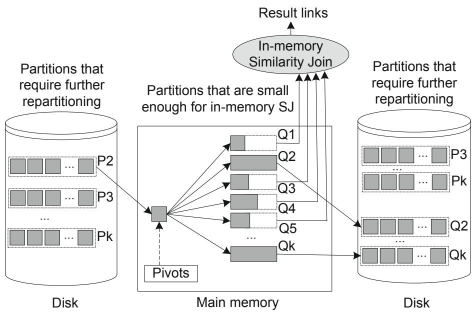
Fig. 1 shows the repartitioning of a base partition using pivots P_0 and P_1. In this case, the result of the SJ operation on the input dataset T is the union of the links in partitions $P0$ and $P1$, and the links in window-pair partition $P0\_P1$ where one element belongs to window A and the other one to window B. We refer to this last type of link as *window link*. Fig. 2 shows the repartitioning of the window-pair partition $P0\_P1$ of Fig. 1 using pivots Q_0 and Q_1. In this case, the set of window links in $P0\_P1$ is the union of the window links in $Q0$, $Q1$, $Q0\_Q1\{1\}$ and $Q0\_Q1\{2\}$. Note that windows C and F do not form a window-pair partition since their window links are a subset of the window links in $Q0$. Similarly, the window links between E and D are a subset of the ones in $Q1$.

Figures 3 and 4 represent the processing performed by DBSimJoin in round 0 and a generic round I, respectively. Round 0, shown in Fig. 3, partitions the original input data ($R \cup S$) into k partitions. Some generated partitions are small enough to be processed by the in-memory SJ routines, e.g., $P1$, $P4$, $P5$. Result links and window links are generated in these routines. The remaining partitions are stored on disk, e.g., $P2$, $P3$, Pk. Any other round further repartitions a previously generated partition. For instance the round represented in Fig. 4 repartitions partition $P2$. This round also generates some partitions that can be processed by the in-memory SJ routines, e.g., $Q1$, $Q3$, $Q4$, $Q5$, and partitions that need further processing, e.g., $Q2$, Qk. While rounds other than the first one can be processed in any order, DBSimJoin uses a partitioning sequence that favors the early generation of result links. The algorithmic details of this approach are presented in Section 3.3. The remaining part of this section presents the guidelines to implement the DBSimJoin operator inside the query engine of standard DBMSs. Although the presentation is intended to be applicable to any DBMS, some specific details refer to our implementation in PostgreSQL.



Result links
In-memory Similarity Join
Records of R and S
Partitions that are small enough for in-memory SJ
Partitions that require further repartitioning
P1
P2
P3
P4
P5
Pk
Pivots
Disk
Main memory
Disk

Fig. 3. Round 0



Result links
In-memory Similarity Join
Partitions that require further repartitioning
Partitions that are small enough for in-memory SJ
Partitions that require further repartitioning
P2
P3
Pk
Q1
Q2
Q3
Q4
Q5
Qk
Pivots
Disk
Main memory
Disk

Fig. 4. Round *I*

3.2 The Parser and Planner

To add support for Similarity Joins in the parser, the raw-parsing grammar rules, e.g., *yacc* rules in the case of PostgreSQL, are extended to recognize the syntax of the new SJ predicate. The parse-tree and query-tree data structures are extended to include the information of the new operator, i.e., type of join, value of ε and distance function. The routines in charge of transforming the parse tree into the query tree are updated accordingly to process the new fields in the parse tree. To add support for the operator in the planner, a new plan node is created to represent the SJ operator. This node is similar to the regular join node but also stores information about ε and the distance function. If a query has multiple SJ predicates, they are processed one at a time, i.e., multiple SJ nodes are pipelined. It is important to observe that key transformation rules to optimize queries with SJ [15,16], e.g., associativity of SJ operators and pushing selection below SJ, can be applied to plans with DBSimJoin operators. We evaluate the use of several transformation rules with DBSimJoin in Section 4.3.

3.3 The Executor

DBSimJoin Executor Routine. DBSimJoin's main routine is presented in Fig. 5. The routine first creates two lists to keep track of the base and window-pair partitions (line 1). Each partition is assigned a space in memory (*memT*) and if it needs to grow beyond this space, it is stored on disk and the memory space is used as a buffer. The routine partitions the initial input data ($R \cup S$) into base and window-pair partitions (line 2). The main loop will be executed while there is at least one base partition that needs to be processed (lines 3-16). In each iteration, the routine processes all the base partitions executing *InmemorySimJoin* (in-memory routine) to identify SJ links in small partitions (line 5) and hibernating larger partitions, i.e., transferring any in-memory data to disk (line 6). Then, the routine processes the window-pair partitions (and their sub-partitions) until all their SJ links have been produced (lines 7-13). When all the window-pair partitions have been fully processed, the routine gets the

```
DBSimJoin(R, S, eps, numPiv, memT)
Input: R and S (input data), eps, numPiv (No. of pivots), memT (memory threshold)
Output: the result of the Similarity Join between R and S
1.  create basePList and winPairPList
2.  PartitionBasePart(R ∪ S, basePList, winPairPList, eps, numPiv)
3.  while basePList.size > 0 do
4.    for each partition P of basePList do
5.       if P ≤ memT then InmemorySimJoin(P, eps)
6.       else HibernatePartition(P)
7.    while winPairPList.size > 0 do
8.       for each partition W of winPairPList do
9.         if W ≤ memT then InmemorySimJoinWin(W, eps)
10.        else HibernatePartition(W)
11.      if winPairPList.size > 0 then
12.        W ← winPairPList.getFirst()
13.        PartitionWinPairPart(W, winPairPList, eps, numPiv)
14.   if basePList.size > 0 then
15.      P ← basePList.getFirst()
16.      PartitionBasePart(P, basePList, winPairPList, eps, numPiv)
17. delete basePList and winPairPList
```

Fig. 5. DBSimJoin's main executor routine

first base partition that needs further processing and repartitions it calling *PartitionBasePart* (lines 14-16). The main DBSimJoin routine prioritizes the early generation of links. After any partitioning step, the algorithm will process first all the partitions that can be solved in-memory. The routine has the potential to produce links starting at the first round. This behavior enables the support of the iterator interface and its *getNext* function. The algorithm also prioritizes the processing of window-pair partitions before base partitions. This reduces the number of partitions that the routine needs to keep track of. Window-pair partitions are in general significantly smaller than base partitions. Thus, in general, it takes less time to reach the point where they can be processed in memory.

The main routine calls *PartitionBasePart* and *PartitionWinPairPart* to partition a base and a window-pair partition, respectively. *PartitionBasePart* randomly selects *numPiv* pivots and processes each tuple t adding it to the new base (*basePList*) and window-pair (*winPairPList*) partitions this tuple belongs to. This involves: (1) adding t to the base partition corresponding to its closest pivot p, and (2) adding t to all the window-pair partitions (corresponding to pivots p and i) where $gen\_hyperpln\_dist(t,p,i) \leq eps$. Fig. 6 shows an example of the partitions generated by *PartitionBasePart* with pivots P_0 and P_1. T is partitioned into $P0$, $P1$ and $P0\_P1$. Note that the tuples of the window-pair partition have an extra attribute that specifies their previous partition. This is used during the generation of window links and also to correctly repartition this partition. *PartitionWinPairPart* is similar to *PartitionBasePart* but distinguishes between the two window-pair partitions of any pair of pivots. Also, all the generated partitions are added to *winPairPList* since the links generated in a window-pair partition should always be window links (links between tuples of different previous partitions). Fig. 7 shows a partitioning generated by *PartitionWinPairPart*. $P0\_P1$ is partitioned into $Q0$, $Q1$, $Q0\_Q1\{1\}$, and $Q0\_Q1\{2\}$.

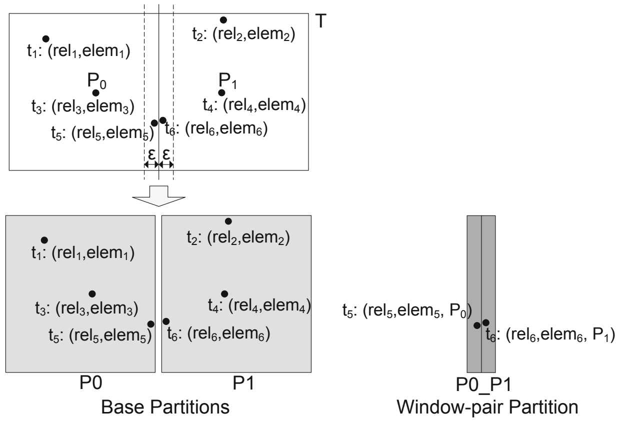


Fig. 6. Partitioning the tuples of a base partition

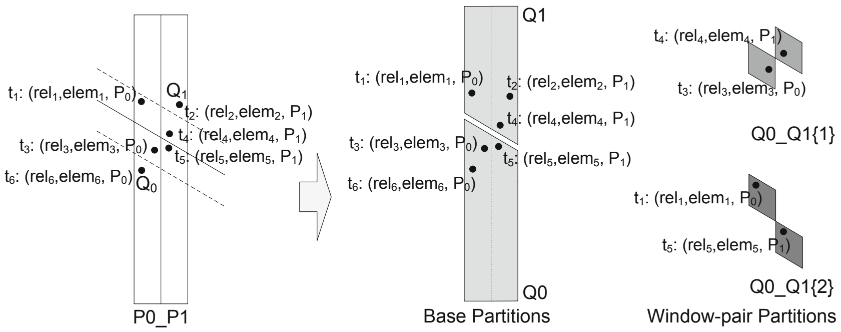


Fig. 7. Partitioning the tuples of a window-pair partition

Implementation Using the Iterator Interface. The DBSimJoin algorithms presented in the previous subsection are realized in a way that enables generating links one at a time, i.e., using the iterator interface and its function *getNext*. DBSimJoin is a non-blocking operator. That is, it does not require the full generation of results before it can start reporting results. Database queries are commonly composed of a tree of operators where tuples flow bottom-up. The process is initiated by calls to the *getNext* function at the root operator. Each call in turn calls the *getNext* function of its children nodes until it gets the required information to produce a result tuple. This process is propagated top-down. A non-blocking behavior reduces query response time. When *getNext* is called in a DBSimJoin node, the operator executes the described process only until the next result link is found. Since small partitions that can be solved using the in-memory routines can be generated starting at the first round, DBSimJoin will quickly find the next link. The *getNext* routine is implemented in the fashion of a state machine that uses the states and transitions presented in Fig. 8. When

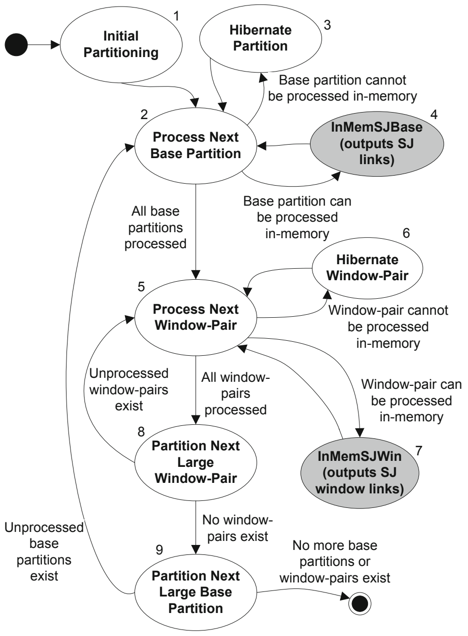

Initial Partitioning
1
Hibernate Partition
3
Base partition cannot be processed in-memory
2
Process Next Base Partition
4
InMemSJBase (outputs SJ links)
All base partitions processed
Base partition can be processed in-memory
6
Hibernate Window-Pair
5
Process Next Window-Pair
Window-pair cannot be processed in-memory
Unprocessed window-pairs exist
All window-pairs processed
Window-pair can be processed in-memory
8
Partition Next Large Window-Pair
InMemSJWin (outputs SJ window links)
7
Unprocessed base partitions exist
No window-pairs exist
9
No more base partitions or window-pairs exist
Partition Next Large Base Partition

Fig. 8. DBSimJoin's GetNext

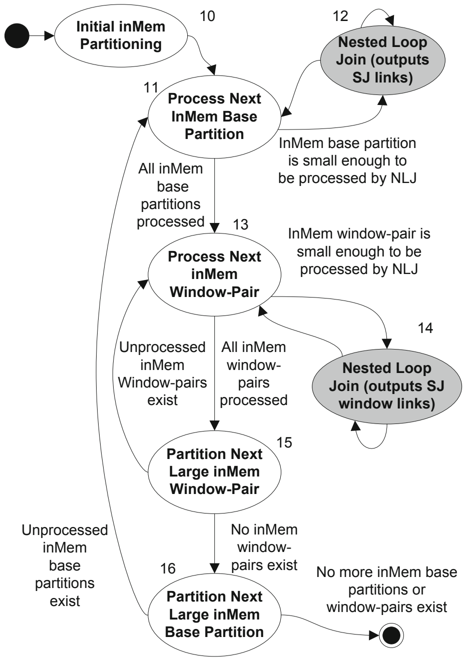

Initial inMem Partitioning
10
12
Nested Loop Join (outputs SJ links)
11
Process Next InMem Base Partition
InMem base partition is small enough to be processed by NLJ
All inMem base partitions processed
13
InMem window-pair is small enough to be processed by NLJ
Process Next inMem Window-Pair
14
Unprocessed inMem Window-pairs exist
All inMem window-pairs processed
Nested Loop Join (outputs SJ window links)
Partition Next Large inMem Window-Pair
15
Unprocessed inMem base partitions exist
No inMem window-pairs exist
16
No more inMem base partitions or window-pairs exist
Partition Next Large inMem Base Partition

Fig. 9. Details of InMemSJBase

getNext is called in the DBSimJoin operator, the routine transitions over the states until it produces the next tuple. The system keeps track of the current state and other required information to resume execution when the next *getNext* is invoked. The states InMemSJBase and InMemSJWin (4 and 7) represent the in-memory SJ routines. These two routines are also implemented using a state machine approach to further reduce the time to produce the next link. The states and transitions of InMemSJBase are presented in Fig. 9. Observe that states 12 and 14 produce the links using a Nested Loop Join approach. The states and transitions of InMemSJWin are very similar to the ones of InMemSJBase with the difference that InMemSJWin only produces window links.

3.4 Analysis of I/O Cost and Number of Pivots

The work in [12] showed that the average I/O cost of the external Quickjoin algorithm is $O(N(1+w)^{\lceil log(N/M)\rceil})$. N and M are the number of blocks of the input data and the number of tuples that fit in internal memory, respectively. w is the fraction of tuples that lie within ε of the partition boundary. DBSimJoin's analysis is similar to the one of Quickjoin but considers we have a limited number of buffer pages B to store the partitions. Moreover, each partition is assigned L buffer pages to store its data. If the partition grows beyond this space, it is stored on disk. The maximum number of new partitions generated by the algorithm in a round will be limited by $P_{max} = \frac{B}{L}$. Also, the value of M in our case is the number of tuples that fit in L buffer pages. That is $M = T \times L$, where T is the number of tuples that fit in a single page. Using these properties, we have that the average I/O cost of DBSimJoin is $O(N(1+w)^{\lceil log(\frac{N \times P_{max}}{B \times T})\rceil})$. This cost will be close to $O(N)$ for small ε and close to $O(N^2)$ for large ε.

P (No. of partitions in a round) is related to the number of pivots K. Given K pivots, DBSimJoin generates K base partitions and $K^2 - K$ window-pair partitions, that is $P = K^2$. Since $P \leq \frac{B}{L}$, we have that $K \leq \lfloor\sqrt{\frac{B}{L}}\rfloor$. We use $K = \lfloor\sqrt{\frac{B}{L}}\rfloor$ as an initial value of K. Also note that the number of partitions P is not affected by the number of dimensions.

4 Performance Evaluation

We implemented DBSimJoin in PostgreSQL 8.2.4. In this section we compare DBSimJoin with other approaches proposed for database systems. We do not compare DBSimJoin with standalone algorithms. These algorithms can outperform database implementations since they are not affected by database features like transaction processing, recovery, etc. The experiments used an Intel Core i5 2.27 GHz machine (4GB RAM, 500GB/5400RPM hard disk, Linux). We use the following datasets:

- **SynthData** This is a synthetic vector dataset. A version of this dataset was created for each evaluated number of dimensions, i.e., 4D, 6D and 8D. The components of each vector are randomly generated numbers in the range [0 - 100]. The dataset for scale factor 1 (SF1) contains 80,000 records.
- **ColorData** This dataset contains feature vectors extracted from a Corel image collection [9]. Each record is a 9D vector with components in the range [-4.8 - 4.4]. The SF1 dataset contains 68,040 records.
- **DBLPData** This dataset is a subset of the DBLP bibliographic dataset [1]. Each extracted record contains a unique identifier and the title. The SF1 dataset contains 2,500 records. The minimum, maximum and average lengths of the title attribute are 33, 281, and 57, respectively.

The datasets for SF greater than 1 were generated in such a way that the number of links of any SJ operation in SFN is N times the number of links in SF1. For vector data, the datasets for higher SF were obtained adding shifted copies of the SF1 dataset where the distance between copies were greater than the maximum value of ε. For string data, the datasets for higher SF were obtained adding a copy of the SF1 data where characters are shifted similarly to the process in [18]. The records of each dataset are equally divided between R and S. We used Euclidean and Levenshtein distance functions for vector and string data, respectively. The number of pivots ($numPiv$) in the experiments was 30 for SynthData and Colordata and 50 for DBLPData, the threshold to switch to in-memory SJ was 4KB and the threshold to switch to nested loop join in the in-memory SJ routines was 20 tuples.

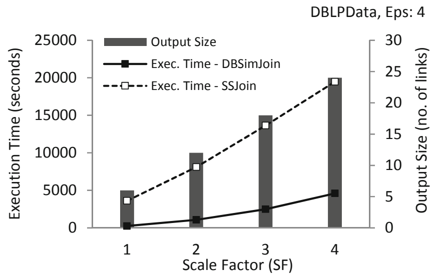


Fig. 10. Increasing SF - DBLPData

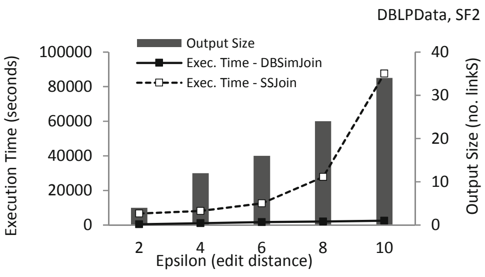


Fig. 11. Increasing Epsilon - DBLPData

4.1 Performance Evaluation with DBLP String Data

We compare DBSimJoin with an implementation of SSJoin (q=3), the q-gram based approach proposed in [4] and explained in Section 2. SSJoin's execution times do not include q-gram generation. Sample queries are presented next.

```
SSJoin query: SELECT R.pka, R.origstringa, S.pkb, S.origstringb
 FROM qgramsR R, qgramsS S WHERE R.qgrama = S.qgramb
 GROUP BY R.pka, R.origstringa, S.pkb, S.origstringb
 HAVING count(*) >= (char_length(R.origstringa) - 3 + 1 - 3 * 2)
    AND editdist(R.origstringa, S.origstringb) <= 2;
```

```
DBSimJoin query: SELECT R.pka, R.origstringa, S.pkb, S.origstringb FROM
  R, S WHERE R.origstringa WITHIN 2 OF S.origstringb USING EditDistance;
```

Increasing Scale Factor. Fig. 10 shows the performance when data size increases. DBSimJoin's execution time is between 7% (SF1) and 24% (SF4) of the one of SSJoin. DBSimJoin also uses significantly less space than SSJoin. In our experiments, for SF1, SSJoin's q-gram tables have about 55 times the number of rows of the original tables. DBSimJoin uses only the original tables.

Increasing Epsilon. Fig. 11 compares the performance when ε increases. DBSimJoin's execution time is 13% of that of SSJoin for ε=2, and only 3% for ε=10. While for very low values of ε the number of tuples returned by the join used in SSJoin is relatively small, this number grows quickly when ε increases affecting negatively its execution time. DBSimJoin's execution time increases moderately when ε increases since larger values of ε generate larger window-pair partitions.

4.2 Performance Evaluation with Vector Data

We run all the tests using both vector datasets and found the same performance trends. We present the results using one dataset (specified in each figure) due to space constraints. We compare DBSimJoin with queries that produce the same results using only regular (non-similarity) database operators (RegDBOps). To

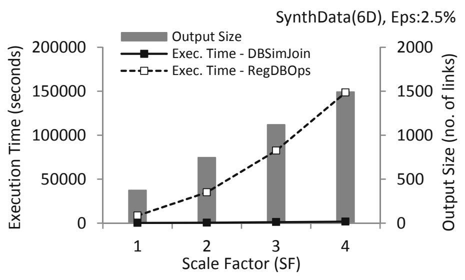


Fig. 12. Increasing SF - SynthData

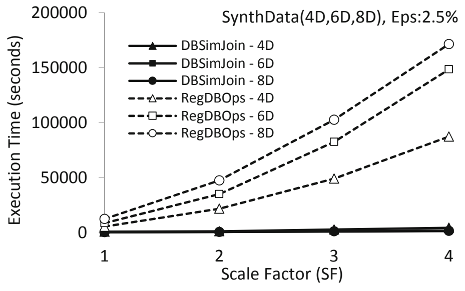


Fig. 13. Increasing SF and Number of Dimensions - SynthData

the best of our knowledge, no previous work has proposed an alternative approach to support SJ over vectors in a DBMS. PostGIS, a spatial database extender for PostgreSQL [2], is not considered since it only supports 2D/3D data. Also, PostGIS' spatial distance function (ST-Distance) does not use indexes and thus SJ will perform like RegDBOps. Some sample queries are presented below.

```
RegDBOps query: SELECT R.r1, R.r2, S.s1, S.s2 FROM R, S
 WHERE sqrt((R.r1-S.s1)^2 + (R.r2-S.s2)^2) <= 0.5;
DBSimJoin query: SELECT R.r1, R.r2, S.s1, S.s2 FROM R, S
 WHERE [R.r1, R.r2] WITHIN 0.5 OF [S.s1, S.s2] USING EuclideanDistance;
```

Increasing Scale Factor. Fig. 12 shows how DBSimJoin and RegDBOps scale when the data size increases. This experiment uses 6D vectors and a value of ε of 2.5% of the maximum possible distance. DBSimJoin performs significantly better than RegDBOps for all the values of SF. RegDBOps' execution time grows from being 33 times the one of DBSimJoin for SF1 to 87 times for SF5. RegDBOps' poor performance is due to a nested loop join between the joined relations.

Increasing SF and Number of Dimensions. DBSimJoin perform much better than RegDBOps also for different number of dimensions as shown in Fig. 13. In all cases, the execution time of DBSimJoin is a small fraction of that of RegDBOps. Moreover, when the number of dimensions increases, DBSimJoin takes a smaller fraction of the execution time of RegDBOps. Specifically, for 4D data the execution time of DBSimJoin is at most 15% of that of RegDBOps. The percentage is 3% for 6D data and only 2% for 8D data. Fig. 14 shows that the execution time of DBSimJoin decreases when the number of dimensions increases. This is mainly due to the large difference between the links reported in 4D data (46,109-184,436) and the ones reported in 6D and 8D data (< 1,500).

Increasing Epsilon. Fig. 15 shows that, for all the evaluated values of ε, DBSimJoin significantly outperforms RegDBOps. The execution time of DBSimJoin is only 0.42% of that of RegDBOps for ε=0.5% and 2.97% for ε=2.5%.

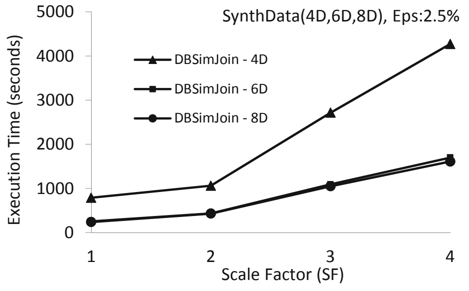


Fig. 14. Increasing SF and Number of Dimensions (DBSimJoin) - SynthData

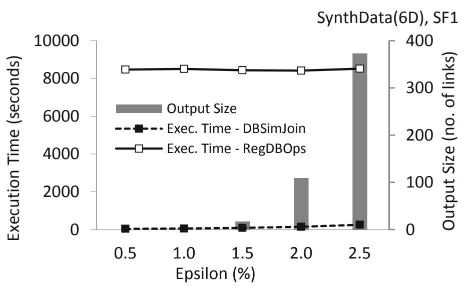


Fig. 15. Increasing Epsilon - SynthData

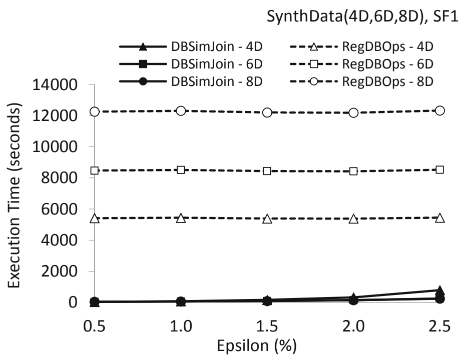


Fig. 16. Increasing Epsilon and Number of Dimensions - SynthData

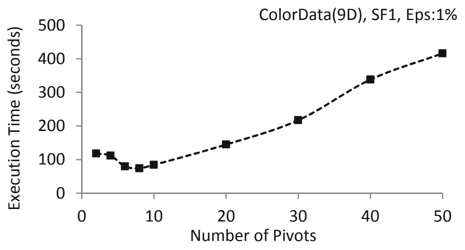


Fig. 17. Increasing No. of Pivots - ColorData

RegDBOps' execution time remains almost constant because it always executes a nested loop join checking the SJ predicate. DBSimJoin's execution time increases slightly with larger values of ε since they generate larger window-pair partitions.

Increasing Epsilon and Number of Dimensions. DBSimJoin performs significantly better than RegDB-Ops also for different numbers of dimensions as shown in Fig. 16. For 4D data the execution time of DBSimJoin is at most 14.5% of that of RegDBOps. The percentage is 3% for 6D and only 1.9% for 8D.

Varying Number of Pivots. Fig. 17 shows DBSimJoin's execution time when the number of pivots (K) increases from 2 to 50. As K increases, the execution time decreases at first due to fewer rounds required to reach the point where all partitions can be processed in memory. When K increases past the optimal value ($K = 8$), the execution time grows because the extra data duplication and I/O costs of the window-pair partitions outweigh the effect of decreased partition size and number of rounds.

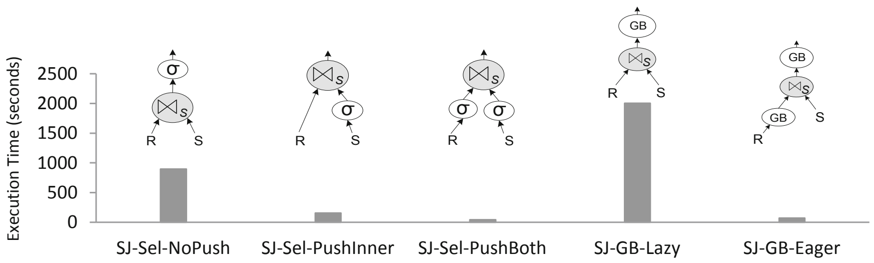


Fig. 18. Combining DBSimJoin with other Operators

4.3 Combining DBSimJoin with other Database Operators

This section shows that DBSimJoin can be combined with other database operators and used in important transformation rules. We use the following queries.

```
Combining SJ and Selection (SJ-Sel): SELECT * FROM R, S
 WHERE [R.r1 ... R.r9] WITHIN 0.28 OF [S.s1 ... S.s9] USING
 EuclideanDistance AND EucDist([S.s1 ... S.s9],[0.02 ... 0.02])<=1.38;
Combining SJ and Group-by (SJ-GB): SELECT count(*), S.s1 ... S.s9
 FROM R, S WHERE [R.r1 ... R.r9] WITHIN 0.28 OF [S.s1 ... S.s9]
 USING EuclideanDistance GROUP BY [S.s1 ... S.s9];
```

This section uses ColorData (SF1) with the following changes. Table S in SJ-GB has 1,000 randomly selected 9D vectors that are used as reference points around which the points of R are grouped. Also, in order to generate duplicate tuples, table R in SJ-GB is generated taking 20% of the original dataset and duplicating each tuple 5 times. The thresholds 0.28 and 1.38 correspond to ε=1.0% and ε=5.0%, respectively. SJ-Sel-NoPush, SJ-Sel-PushInner and SJ-Sel-PushBoth in Fig. 18 are different ways to execute SJ-Sel. SJ-Sel-NoPush executes the SJ first and then the selection operator. In SJ-Sel-PushInner, the selection operator ($\sigma_{EucDist(S,Const)\leq 1.38}$) is pushed to S. SJ-Sel-PushInner's execution time is 17% of that of SJ-Sel-NoPush. In SJ-Sel-PushBoth, the filtering benefit is further improved by pushing selection operations on both inputs of the join ($\sigma_{EucDist(S,Const)\leq 1.38}$ on S and $\sigma_{EucDist(R,Const)\leq (1.38+0.28)}$ on R). SJ-Sel-PushBoth's execution time is only 5% of the one of SJ-Sel-NoPush. SJ-GB-Lazy and SJ-GB-Eager correspond to the eager and lazy aggregation plans of SJ-GB. SJ-GB-Lazy executes SJ first and then group-by. Grouping is split into two parts in SJ-GB-Eager. The first part groups on R.r and calculates the count before the SJ. The second part groups on S.s and uses the intermediate data to calculate the final results (sum of the intermediate counts) after the SJ. The execution time of SJ-GB-Eager is only 4% of that of SJ-GB-Lazy.

5 Conclusions and Future Work

This paper presents DBSimJoin, an efficient and non-blocking SJ database operator. DBSimJoin supports the iterator interface and uses a sequence of rounds that prioritizes the quick generation of results. DBSimJoin can be used with multiple data types and distance functions. We present the implementation details of DBSimJoin and extensively evaluate its performance showing that DBSimJoin outperforms alternative approaches. We also present queries that combine DBSimJoin with other database operators and show that important transformation rules can be effectively applied to queries with DBSimJoin. Our paths for future work include the study of: (1) other similarity operations as database operators, (2) indexing techniques to improve the efficiency of similarity queries, and (3) database queries with multiple similarity operators.

References

1. Dblp bibliography, `http://www.informatik.uni-trier.de/~ley/db/`
2. PostGIS, `http://postgis.net/documentation`
3. Böhm, C., Braunmüller, B., Krebs, F., Kriegel, H.-P.: Epsilon grid order: an algorithm for the similarity join on massive high-dimensional data. In: SIGMOD 2001, pp. 379–388 (2001)
4. Chaudhuri, S., Ganti, V., Kaushik, R.: A primitive operator for similarity joins in data cleaning. In: ICDE 2006, p. 5 (2006)
5. Chaudhuri, S., Ganti, V., Kaushik, R.: Data debugger: An operator-centric approach for data quality solutions. IEEE Data Eng. Bull. 29(2), 60–66 (2006)
6. Dittrich, J.-P., Seeger, B.: Gess: a scalable similarity-join algorithm for mining large data sets in high dimensional spaces. In: KDD 2001, pp. 47–56 (2001)
7. Dohnal, V., Gennaro, C., Rabitti, F., Zezula, P.: Similarity join in metric spaces. In: Sebastiani, F. (ed.) ECIR 2003. LNCS, vol. 2633, pp. 452–467. Springer, Heidelberg (2003)
8. Dohnal, V., Gennaro, C., Zezula, P.: Similarity join in metric spaces using ed-index. In: Mařík, V., Štěpánková, O., Retschitzegger, W. (eds.) DEXA 2003. LNCS, vol. 2736, pp. 484–493. Springer, Heidelberg (2003)
9. Frank, A., Asuncion, A.: UCI machine learning repository (2010), `http://archive.ics.uci.edu/ml`
10. Gravano, L., Ipeirotis, P.G., Jagadish, H.V., Koudas, N., Muthukrishnan, S., Srivastava, D.: Approximate string joins in a database (almost) for free. In: VLDB 2001, pp. 491–500 (2001)
11. Hjaltason, G.R., Samet, H.: Index-driven similarity search in metric spaces (survey article). ACM Trans. Database Syst. 28(4), 517–580 (2003)
12. Jacox, E.H., Samet, H.: Metric space similarity joins. ACM Trans. Database Syst., 33(2), 7:1–7:38 (2008)
13. Paredes, R., Reyes, N.: Solving similarity joins and range queries in metric spaces with the list of twin clusters. J. of Discrete Algorithms 7(1), 18–35 (2009)
14. Silva, Y.N., Aly, A.M., Aref, W.G., Larson, P.-A.: SimDB: a similarity-aware database system. In: SIGMOD 2010, pp. 1243–1246 (2010)
15. Silva, Y.N., Aref, W.G., Ali, M.H.: The similarity join database operator. In: ICDE 2010, pp. 892–903 (2010)
16. Silva, Y.N., Aref, W.G., Larson, P.-A., Pearson, S., Ali, M.H.: Similarity queries: their conceptual evaluation, transformations, and processing. VLDB Journal 22(3), 395–420 (2013)
17. Silva, Y.N., Pearson, S.: Exploiting database similarity joins for metric spaces. Proc. VLDB Endow. 5(12), 1922–1925 (2012)
18. Vernica, R., Carey, M.J., Li, C.: Efficient parallel set-similarity joins using mapreduce. In: SIGMOD 2010, pp. 495–506 (2010)

Engineering Efficient and Effective Non-metric Space Library

Leonid Boytsov[1] and Bilegsaikhan Naidan[2]

[1] Language Technologies Institute,
Carnegie Mellon University,
Pittsburgh, PA, USA
leo@boytsov.info

[2] Department of Computer and Information Science,
Norwegian University of Science and Technology,
Trondheim, Norway
bileg@idi.ntnu.no

Abstract. We present a new similarity search library and discuss a variety of design and performance issues related to its development. We adopt a position that engineering is equally important to design of the algorithms and pursue a goal of producing realistic benchmarks. To this end, we pay attention to various performance aspects and utilize modern hardware, which provides a high degree of parallelization. Since we focus on realistic measurements, performance of the methods should not be measured using merely the number of distance computations performed, because other costs, such as computation of a cheaper distance function, which approximates the original one, are oftentimes substantial. The paper includes preliminary experimental results, which support this point of view. Rather than looking for the best method, we want to ensure that the library implements competitive baselines, which can be useful for future work.

Keywords: benchmarks, (non)-metric spaces, Bregman divergences.

1 Introduction

A lot of domains, including content-based retrieval of multimedia, computational biology, and statistical machine learning, rely on similarity search methods. Given a finite database of objects $\{o_i\}$, a search query q and a dissimilarity measure (which is typically represented by a distance function $d(o_i, q)$), the goal is to find a subset of database objects sufficiently similar to the query q.

Two major retrieval tasks are typically considered: a nearest neighbor and a range search. The nearest neighbor search aims to find the least dissimilar object, i.e., the object at the smallest distance from the query. Its direct generalization is the k-nearest neighbor (or the k-NN) search, which looks for the k most closest objects. Given a radius r, the range query retrieves all objects within a query ball (centered at the query object q) with the radius r, or, formally, all the objects $\{o_i\}$ such that $d(o_i, q) \leq r$.

The queries can be answered either exactly, i.e., by returning a complete result, or, approximately, e.g., by finding only some nearest neighbors. The exact versions of near neighbor and range search received a lot of attention. Yet, in many applications exact searching is not essential, because the notion of similarity, e.g., between two images, is not specified rigorously. Applying an exact retrieval method does not necessarily mean that we will find the image that is most similar to a query from a human perspective. Likewise, a k-NN classifier may perform well even if the search method does not produce a precise and/or a complete result [4,32].

Search methods for non-metric spaces are especially interesting. This domain does not provide sufficiently generic *exact* search methods. We may know very little about analytical properties of the distance or the analytical representation may not be available at all (e.g., if the distance is computed by a black-box device [36]). Hence, employing an approximate approach is virtually unavoidable.

Approximate search methods are typically more efficient than exact ones. Yet, it is harder to evaluate them, because we need to measure retrieval speed at different levels of recall (or any other effectiveness metric). To the best of our knowledge, there is no publicly available software suit that (1) includes state-of-the-art approximate search methods for both non-metric and metric spaces and (2) provides capabilities to measure search quality. Thus, we developed our own test framework and presented it in this paper.

1.1 Related Work

There is large body of literature devoted to exact search methods in metric spaces, which are thoroughly surveyed in the books by Faloutsos [12], Samet [33], and Zezula et al. [41] (see also a survey by Chávez et al. [5]). Exact methods have a limited value in high-dimensional spaces, which exhibit phenomena of the empty space [34] and measure concentration [5]. Experiments show that, as the dimensionality increases, every nearest neighbor search method degrades to sequential searching [40]. This is commonly known as the "curse of dimensionality". In that, methods, which are allowed to return inexact answers, are less affected by the curse [31]. For a discussion of these phenomena, we address the reader to the papers of Indyk [21] and Pestov [31].

To answer the approximate nearest neighbor queries, Indyk and Motwani [22] as well as Kushilevitz et al. [26] proposed to use random projections. The locality sensitivity hashing (LSH) is one of the most well-known implementations of this idea [22,21]. The LSH indexing uses several hash functions, such that a probability of a collision (hashing to the same value) is sufficiently high for close objects, but is small for distant ones.

The LSH works best in L_p spaces where $p \in (0, 2]$. There exists an extension of the LSH for an arbitrary metric space [29] as well as for *symmetric* non-metric distances [28]. Performance of the LSH depends on the choice of parameters, which can be tuned to fit the distribution of a data set [7].

Most exact search methods can be transformed into approximate ones by applying an *early termination strategy.* In particular, Zezula et al. [42] demonstrated that this approach works well for M-trees. One of the most efficient strategies relies on density estimates for a distribution of distances [42,1]. The density-based approach to space pruning was also discussed by Chávez and Navarro [6] (in the context of pivoting methods), who called it "stretching" of the triangle inequality.

Let us consider a metric space, where we selected a single reference point π, known as *pivot*. The pivot is used to prune space during searching. Imagine that we computed a distance from the pivot π to every other data point. Then, points are sorted in the order of increasing distances from π. The median distance is m and points are divided into two buckets. If the distance from a point to π is smaller than m, the point is put into the first bucket. Points with distances larger than (or equal to) m are placed into the second bucket.

Let q be a query point and r be a radius of the range query. If $r < m - d(\pi, q)$, an answer can be only in the first bucket. If $r \leq d(\pi, q) - m$, an answer can be only in the second bucket. Otherwise, the answer can be in both buckets and no pruning is possible (without risking to miss an answer). In the "stretched" triangle inequality, we choose constants $\alpha_1, \alpha_2 \geq 1$.[1] If $r < \alpha_1(m - d(\pi, q))$, we check only the left bucket. If $r < \alpha_2(d(\pi, q) - m)$, we check only the right bucket. This is an example of an *oracle* procedure that defines a pruning algorithm of a pivoting method. Note that (1) it is possible to learn the oracle in both metric and *non-metric* spaces, (2) we can learn a pivot-specific oracle, instead of the global one, (3) most existing methods designed for metric spaces can be converted into non-metric search methods by simply replacing the triangle-inequality based pruning method with a search oracle. We plan to present these learning approaches in detail elsewhere.

In a recent survey [37], Skopal and Bustos discussed several types of non-metric access methods, which we divide into the following categories: (1) projective and lower/upper bounding approaches, (2) methods that prune the space using properties other than the triangle inequality (e.g., the Ptolemaic inequality [27]), and (3) domain-specific methods. Inverted files are a classic domain-specific algorithm applicable to high-dimensional, but sparse, vector spaces, where the distance function is the cosine similarity (or a similar distance).

Jacobs et al. [23] review various projection methods and argue that a projection is not always feasible, for instance, when the similarity cannot be expressed by a numeric distance function, or the distance function is not symmetric. In the case of symmetric, non-negative, and reflexive distance, one can use the TriGen algorithm [36], which applies a monotonic transformation to the distance function. Consider, e.g., the squared Euclidean distance, which is a (non-metric) Bregman divergence. By taking the square root, we obtain the metric function. Similarly, the TriGen algorithm allows one to convert a distance into a function that satisfies the triangle inequality only approximately. In addition, it provides control over the degree of approximation.

Chávez et al. [18] proposed a projective method, which is applicable to both metric and *non-metric spaces*. The method, called the permutation index, selects

[1] Chávez and Navarro [6] employed only one stretching constant.

k pivots $\{\pi_i\}$ and for every data point o it creates a permutation of pivots: a list where pivots are sorted in the order of increasing distances $d(\pi_i, o)$. Independently, this method was invented by Amato and Savino [2], who additionally proposed to index permutations using an inverted file.

To answer the query, the correlation is computed between the permutation of the vector and the permutation of every data point. Then, all data points are sorted in the order of ascending correlation values and a given fraction of objects are compared directly with the query (by computing the distance in the original space). Performance of the permutation index can be improved by using incremental sorting [17] or by indexing permutations using an inverted file [2], a permutation prefix tree [11], or a metric space index [14].

Bregman divergences is a class of non-metric distance functions. This divergences include the squared Euclidean distance, the KL-divergence:

$$d(x, y) = \sum x_i \log(x_i/y_i) \tag{1}$$

and the Itakura-Saito distance:

$$d(x, y) = \sum x_i/y_i - \log(x_i/y_i) - 1. \tag{2}$$

For the Bregman divergences, there exist two exact search methods. The Bregman ball tree (bbtree) [4], which recursively divides the space using two covering Bregman balls at each recursion step, and a mapping method due to Zhang et al. [43]. Both approaches use properties of Bregman divergences to lower/upper bound distance values.

2 Methodology

2.1 Evaluation Approach

Performance of approximate methods is typically represented by a curve that plots efficiency against effectiveness. Two most common efficiency metrics are retrieval time and a number of distance computations. Additionally, we use the *improvement in efficiency* (with respect to the single-thread sequential search algorithm) and the *improvement in the number of distance computations.*

Recall is a commonly used effectiveness metric. It is equal to the fraction of all correct answers retrieved. The *relative error* [42] is defined for a pair of points o and $\widetilde{o}$, such that o is an exact and $\widetilde{o}$ is an approximate answer. It is simply a ratio of the distances $d(\widetilde{o}, q)$ and $d(o, q)$. The relative error can be misleading, especially in high dimensional spaces. Due to high concentration of measure, an increase in relative error can be very small, but the method can return the 1,000th nearest-neighbor instead of the most closest one. This concern was also expressed by Cayton [4]. Similarly, recall does not account for position information and has the same issue [1].

Let $\text{pos}(o_i)$ represent a positional distance from o_i to the query, i.e., the number of objects closer to the query than o_i plus one. In the case of ties, we assume that the object with a smaller index is closer to the query. Note that $\mathrm{pos}(o_i) \geq i$. A *relative position* error is equal to $\mathrm{pos}(o_i)/i$ and is more informative than a relative distance error and/or recall. We average relative position errors using the geometric mean [24].

Zezula et al. [42] proposed to use the average value of the inverse relative position error (called the precision of approximation) as a performance metric (m is the number of found objects):

$$\frac{1}{m}\sum_{i=1}^{m}\frac{i}{pos(o_i)} \tag{3}$$

Amato et al. [1] suggested the metric that measures the absolute position error. It is equal to:

$$\frac{1}{m}\sum_{i=1}^{m}\frac{pos(o_i)-i}{\#\text{of indexed points}} \tag{4}$$

Unfortunately, this metric produces results that are not comparable across collections and result sets of different sizes. Consider an example of the result set, where $pos(o_i) = 2i$. The absolute position error is equal to:

$$\frac{1}{m}\sum_{i=1}^{m}\frac{2i-i}{\#\text{of indexed points}} = \frac{0.5(m+1)}{\#\text{of indexed points}}$$

We have no good explanation why the position error should grow with m, while the relative position error and the degree of approximation remain constant (in this case). Even worse, due to the large factor in the denominator of Eq. 4, the computed error is generally very small. It is easy to make a wrong conclusion that the algorithm works almost ideally, whereas, in truth, it provides a poor approximation.

If we have a separate test set, testing is straightforward. Otherwise, we need to randomly divide the original data set into indexable data and testing data. This method is based on the assumption that distributions of test queries and indexed data objects are similar. The random division should be repeated several times (an approach known as bootstrapping), and performance metrics computed for each split should be aggregated.

One may be tempted to select queries among indexed data objects, or, alternatively, to create test vectors by randomly perturbing the indexed data. Both approaches are not ideal and can lead to overly optimistic or pessimistic results, especially, in the case of the nearest neighbor searching. We experimented with the `Colors` data set[13], indexed using the Vantage Point tree (VP-tree) [38]. If we selected queries from the vectors that were already indexed, it took on average only 20 distance computations to find the query's nearest neighbor. Since the query and the found vector were identical, the pruning algorithm was unrealistically efficient. For the randomly selected held-out test data, it took about 6,000 distance computations to answer the nearest neighbor query! If we used a query obtained by random additive (and uniform) perturbations of vector elements, the results depended on the amount of noise. In our experiment, we got 800 distance computations in one case and 10^5 distance computations (i.e., the algorithm degraded to the linear scan) in another.

We speculate that queries obtained by random perturbations can be useful if the model of random perturbations fits data well. This assumption is apparently reasonable for the Euclidean data, but additive transformations may significantly change the histogram of distances in the case of the KL-divergence. In one example, the application of the additive noise led to a 2x decrease in the median distance value between two randomly selected vectors. The *multiplicative log-normal* noise seemed to produce more realistic results, yet, additional experimentation is needed to understand applicability of this approach.

2.2 Choice of Programming Language

C,C++, and Java are the three most popular general-purpose programming languages[25]. [2] The authors are familiar with all three and considered them as implementation languages. According to "The Computer Language Benchmarks Game", C and C++ have comparable performance.[3] Major C/C++ compilers (GNU C++ and Microsoft Visual C) support Single Instruction Multiple Data (SIMD) commands, which allows one to compute distances more efficiently.

Yet, only C++ supports run-time and compile-time polymorphism. The new C++ specifications standardize multi-threading and simplify the use of STL containers (threads are not standardized in the pure C). [4] There is evidence, including anecdotal experience of authors, that C++ allows programmers to be more productive than does C [3]

Even though performance of Java sometimes matches performance of C++ [39,35], Java is generally 2-3 times slower than C or C++ [20,16]. Unlike C/C++, there is no built-in support for the SIMD instructions [30]. Java objects are heavy and programmers have to use parallel arrays as well as manual memory management (e.g., reusing small objects) to work around this problem [10]. Thus, writing "algorithmic-intensive" applications in Java may sometimes be harder than in C++.

Because C++ is largely a superset of C, reusing the code already implemented in the Metric Spaces Library would be straightforward. Yet, it is harder to port C-code to Java. There are tools for seamless integration of C++ and R. In particular, one can call R scripts directly from a C++ program [9]. All in all, using the latest C++ compiler that implements the new standard is the most appealing choice for us.

[2] See, also `http://www.langpop.com/` and `http://spectrum.ieee.org/at-work/tech-careers/the-top-10-programming-languages`

[3] According to at least this page: `http://benchmarksgame.alioth.debian.org/u64/benchmark.php`

[4] See `http://www.open-std.org/jtc1/sc22/wg21/`

2.3 Design

Our software was designed in the spirit of the Metric Spaces Library [13], but there are multiple important differences. We have classes that represent an `Object`, a `Space`, and an `Index`. The `Space` abstraction is necessary to encapsulate the computation of the distance. We can have multiple `Space` sub-classes implementing different distance functions. In addition, the `Space` class provides functionality to read objects from a file.

A distance can be integer-valued, real-valued, or represented by an arbitrarily complex object (if, e.g., we compare objects using multiple criteria). Similarly to the Metric Spaces Library, the same implementation can work with different distance types (e.g., the VP-tree can be used with both the integer-valued edit distance and with the real-valued L_1 metric). This is effectively supported by the compile-time polymorphism of C++ (templates). All implementations (including indices for real-valued and integer-valued distances) co-exist in the same binary and there is no need to update makefiles when a new method or a distance is implemented.

The `Object` has an identifier and can store arbitrary data (of any type). When necessary objects are transformed: One may need to reduce the dimensionality or precompute the logarithms to accelerate evaluation of the KL-divergence (see Section 2.4). Unlike the Metric Spaces Library, a distance function accepts pointers to the objects rather than object ids.

A `Query` object proxies distance evaluations during search time, which allows us to get the average number of computations carried out by a search method as well as to compute confidence intervals (even in the *multi-threading* testing mode). It is still possible to access the distance function through a `Space` object, but this should be done only at indexing time. If (due to programmer's error) an instance of the `Index` tried to access distance through the `Space` object, the program would terminate. [5]

There are two types of query classes and both classes have the `Radius` function. For the range queries, this function returns the constant value specified by the user. For the k-NN queries, the value returned by `Radius` changes during the search, because it represents the distance from the query to the k-th closest object found so far. Because of this abstraction, it is often sufficient to implement a single (template) function that handles both the range and the k-NN search.

The distance function can be non-symmetric, thus two types of queries (left and right are possible). Currently, the framework directly supports only the left queries (q is the second argument of the distance function). For some methods, e.g., permutation-based approaches, right queries can be implemented by simply swapping arguments of the distance function. Yet, a different distance function as well as a transformation of original data may be necessary for Bregman divergences [4]. We plan to address this issue in the future.

[5] We actually have two versions of the `Space` distance function. The public, restricted, version "knows" the current phase (indexing or searching). It terminates the program if called during the search phase. The unrestricted, but private, function is accessible by a `Query` object at search time, because the latter is a friend of `Space`.

As explained in Section 1.1, one can use the concept of the search oracle to convert metric access methods into non-metric ones. We implement two oracle classes (one is based on sampling and another on "stretching" of the triangle inequality). Currently, only the VP-tree can use the generic search oracle, but we plan to embed the search oracle into other metric indices. In addition, most tree-based methods in our library implement a simple early termination strategy, where the search stops after visiting a given number of buckets.

There is a special function that governs the test process. It creates a `Space` object, which loads a data set into memory, divides the data into testing and training sets or loads a separate test set (see Section 2.1). Then, the `Factory` creates instances of specific methods. Parsing of method-specific command line parameters, though, is delegated to the `Index`. Search methods explicitly return pointers/ids of found objects. Thus, we can verify methods' correctness, as well compute recall and other effectiveness metrics discussed in Section 2.1.

Because there are no exact search methods for generic non-metric spaces, evaluation involves comparing a query object against every object in the database. This expensive procedure can be optimized: When we test several different methods in a single session, we compute exact answers only once for each query object. This is reasonably fast on our current data sets, but in the future we may memorize answers, so that they can be re-used when we run multiple tests (using the same data set).

The testing module saves evaluation results to a CSV-file and produces a human readable report. Note that the plots in Section 3 are produced by a Python script that read and processed such CSV-files.

We decided to focus on memory-resident indices. On one hand, modern servers have plenty of memory and a typical high-performance search application would keep most of its index in memory. On the other hand, we do not have to implement serialization/de-serialization or, the code that searches data stored on disk. This simplification allows us to be more productive coders. Our implementations create essentially static indices from scratch. In the future, we plan to consider incremental indexing approaches as well.

One purpose of serialization is to estimate space requirements. Yet, it is possible to obtain an approximate size of the index by measuring the amount of memory used by the program before and after the index is created (one should also include memory to store data objects). There is an opinion that better estimates can be achieved, if we compute the size of allocated memory ourselves, by writing the special code that traverses the index and measures the size of atomic index elements (such as vectors). Yet, we believe that this approach is error prone.

2.4 Efficiency Issues

Even though some distance functions are expensive, it can be quite cheap to compare two vectors using an L_p norm. Furthermore, it can be done even faster using special SIMD instructions [15]. Currently, most x86 CPUs support operations with 128-bit registers containing, e.g., 4 single-precision or 2 double-precision

Table 1. The number of computations per second for optimized and unoptimized distance functions

	128 elements				1024 elements			
Distance	L_1	L_2	Itakura-Saito	KL-div.	L_1	L_2	Itakura-Saito	KL-div.
C++ (no logs)	$9.6 \cdot 10^6$	$9.1 \cdot 10^6$	$1.9 \cdot 10^5$	$5.3 \cdot 10^5$	$1.2 \cdot 10^6$	$1.2 \cdot 10^6$	$2.4 \cdot 10^4$	$6.7 \cdot 10^4$
SIMD (precomp. logs)	$2.7 \cdot 10^7$	$3.3 \cdot 10^7$	$8.3 \cdot 10^6$	$2.8 \cdot 10^7$	$3.4 \cdot 10^6$	$4.5 \cdot 10^6$	$1.04 \cdot 10^6$	$2.4 \cdot 10^6$

Note: vector elements are randomly, uniformly, and independently drawn from $(0, 1]$

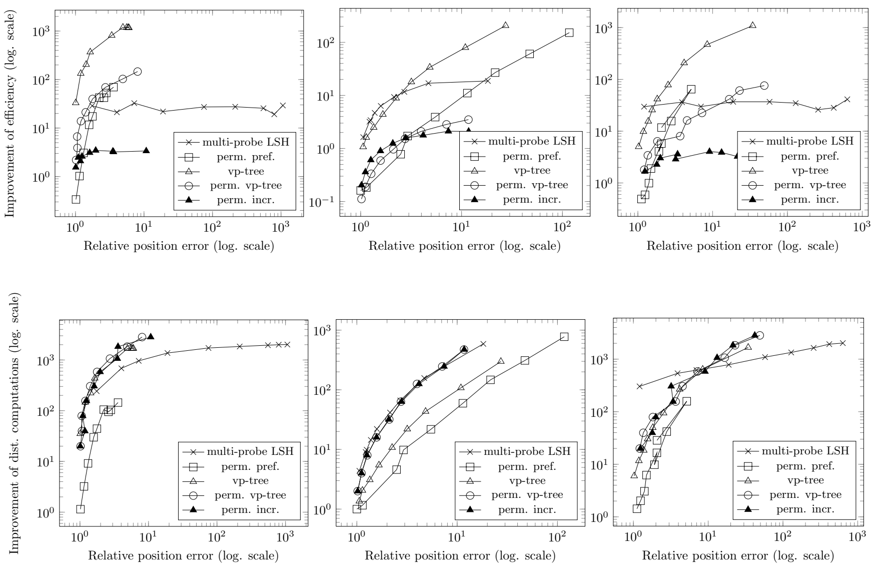
numbers. Some CPUs already support operations with 256-bit registers, which can process 8-element vectors of single-precision numbers.[6] This fact is rather well known, but it appears to be underappreciated. In addition, evaluation of some distance functions can be accelerated at the expense of higher storage requirements (or by dimensionality reduction). In the case of the KL-divergence and the Itakura-Saito distance we can precompute and memorize logarithms of vector elements.

According to Table 1, a single CPU core can carry out more than 30 million computations of the Euclidean distance between two 128-element vectors and more than 4 million distance computations between two 1024-element vectors. In that, the efficient SIMD version spends about one CPU cycle per vector element. [7] The optimized versions of the L_1, L_2 and distances, which use SIMD, are 3 times faster than pure C++ versions. The optimized versions of the KL-divergence and of the Itakura-Saito distance are about 30-50 times as fast as the original ones. In comparison, for a data set of dimensionality 128, the bbtree, which is designed to search using the KL-divergence, is only 5 times faster than sequential scan [4]. It should now be clear that (1) distance computations are not necessarily expensive and (2) optimizing computation of the distance function can be more important than designing data structures.

It has been claimed that a random memory access may take hundreds of CPU cycles [8]. Yet, our experiments showed the cost of a random access on our server to be only 60 cycles. Thus, reducing memory fragmentation may not necessarily lead to substantial improvements in performance. In particular, storing vectors of a VP-tree bucket in adjacent memory regions did not allow us to get more than a 2x speedup. Perhaps, more importantly, the SIMD-based algorithms of distance computations are so fast that communications with RAM can become a major bottleneck in a multi-threading environment. Indeed, to sustain the processing speed of one vector element per CPU cycle (see Table 1) we need to read from memory at the speed of $\approx$ 12 GB/sec (one element is a 4-byte single-precision number). Our server's memory bandwidth of 20 GB/sec can be exhausted with just two threads.

[6] See, e.g., `http://software.intel.com/en-us/avx`

[7] Reading unaligned data does not apparently hurt performance, even for SIMD operations.



(d) `Colors`, L_2 (e) `Unif-64`, L_2 (f) `RCV-128`, L_2

Fig. 1. Improvement in efficiency and in the number of distance computations for 1-NN search in L_2

3 Experiments

Experiments were carried out on a Linux server equipped with Intel Core i7 2600 (3.40 GHz, 8192 KB of L3 CPU cache) and 16 GB of DDR3 RAM (transfer rate 20GB/sec). The code was compiled using GNU C++ 4.7 (optimization flag -Ofast) and tested in a single-thread (using 1,000 queries). The library can be downloaded from GitHub.[8]

The following collections were used:

1. `Colors`: 112-dimensional data set from the Metric Spaces Library [13];
2. `Unif64`: 64-dimensional vectors with elements generated randomly, independently, and uniformly;
3. `RCV-16` and `RCV-128`: 16- and 128-dimensional topic histograms [4];
4. `SIFT`: the normalized 1111-dimensional SIFT signatures [4].

We extracted the first 10^5 vectors from collections (1)-(3) and used the whole collection (4), which contained only 10^4 vectors.

We carried out two series of experiments (both involving 1-NN search). In the first series (see Fig. 1), we used collections `Colors`, `Unif-64`, and `RCV-128`. The distance was Euclidean. We measured both the improvement in efficiency and in the number of distance computations. The values of efficiency metrics were

[8] `https://github.com/searchivarius/NonMetricSpaceLib`

plotted against the relative position error. In the second experimental series (see Fig. 2), we measured how the improvement in efficiency corresponded to the relative position error. Two Bregman divergences were used: the KL-divergence (see Eq. 1) and the Itakura-Saito distance (see Eq. 2). Implemented methods included domain specific and permutation-based approaches as well the VP-tree.

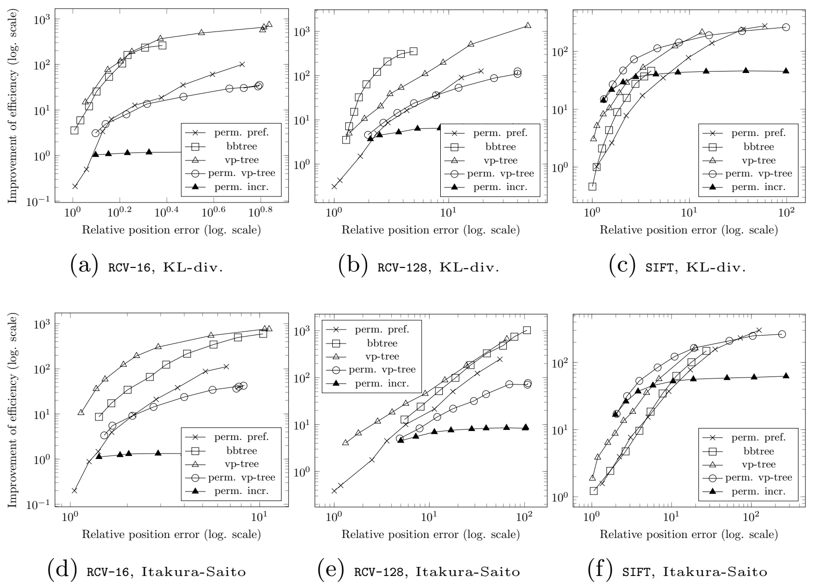
– The VP-tree employed the search oracle that "stretched" the triangle inequality (see Section 1.1). Optimal stretching coefficients were found using a simple grid search. We indexed a small database sample ($\approx$ 1,000 vectors), executed the 1-NN search for various values of stretching coefficients and measured performance. Then, we selected coefficients resulting in the fastest search at given recall values.
– Permutation-based approaches were: an improved permutation index with incremental sorting [17], a permutation prefix tree [11], and the method where permutations were indexed using a metric space index, as proposed by Figueroa and Fredriksson [14]. Unlike Figueroa and Fredriksson, we used an approximate method (the VP-tree that stretched the triangle inequality using $\alpha_1 = \alpha_2 = 2$). In all cases, we used 16 pivots and the prefix length was 4. The maximum fraction of the objects exhaustively compared against the query depended on the data set and varied from 0.01 to 0.05. The minimum fraction of the database objects to be scanned was 0.0002. The number of candidate objects in the permutation prefix index varied from 1 to 24,000.
– The bbtree [4] is the exact indexing method for Bregman divergences. It was extended by the early termination strategy, where the search stopped after visiting a certain number of buckets (the number varied from one to 1,000).
– The multi-probe LSH is designed only for L_2. We used the LSHKit implementation with the following parameters: $H = 1017881$, $T = 10$, $L = 50$.[9]

All methods, including the multi-probe LSH, relied on optimized distance functions. The correlation function (Spearman's rho) was also optimized and implemented using SIMD instructions. The vectors in the buckets of the VP-tree and bbtree were stored in contiguous chunks of memory (the bucket size was 50).

From Fig. 1 we learn that both the classic permutation method (without the index over permutations) and the multi-probe LSH carried out fewer distance computations than most other methods. Yet, they were generally outperformed by the VP-tree and the methods that index permutations (using either the prefix tree or the VP-tree). The reason is that exhaustive comparison of data-object permutations against the permutation of the query vector is costly. In that, the permutation index worked better for high-dimensional data (see Fig. 2f and 2c). Again, we see that the number of distance computations is not necessarily a good predictor of method's performance. Yet, it may give insights into scalability of methods with respect to the size of the data set and data dimensionality.

As can be seen from Fig. 2, we implemented strong baselines that worked well in *non-metric* spaces with *non-symmetric* distance functions. Note that the bbtree, which was tailored to spaces with Bregman divergences, was outperformed

[9] Remaining parameters were automatically computed by the LSHKit [7].



(a) RCV-16, KL-div. (b) RCV-128, KL-div. (c) SIFT, KL-div.

(d) RCV-16, Itakura-Saito (e) RCV-128, Itakura-Saito (f) SIFT, Itakura-Saito

Fig. 2. Improvement in efficiency of 1-NN search for the KL-divergences and Itakura-Saito distance

by the VP-tree (which is a generic method) in most cases. These are encouraging results, but more work needs to be done. We plan to employ new complex domains and implement additional search methods.

Acknowledgments. We would like to thank Lawrence Cayton for providing the data sets, Vladimir Pestov for the discussion on the curse of dimensionality, and anonymous reviewers for helpful suggestions.

References

1. Amato, G., Rabitti, F., Savino, P., Zezula, P.: Region proximity in metric spaces and its use for approximate similarity search. ACM Trans. Inf. Syst. 21(2), 192–227 (2003)
2. Amato, G., Savino, P.: Approximate similarity search in metric spaces using inverted files. In: Proceedings of the 3rd International Conference on Scalable Information Systems, InfoScale 2008, pp. 28:1–28:10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels (2008)
3. Bhattacharya, P., Neamtiu, I.: Assessing programming language impact on development and maintenance: a study on C and C++. In: 33rd International Conference on Software Engineering (ICSE), pp. 171–180 (2011)
4. Cayton, L.: Fast nearest neighbor retrieval for bregman divergences. In: Proceedings of the 25th International Conference on Machine Learning, ICML 2008, pp. 112–119. ACM, New York (2008)

5. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquin, J.L.: Searching in metric spaces. ACM Computing Surveys 33(3), 273–321 (2001)
6. Chávez, E., Navarro, G.: Probabilistic proximity search: Fighting the curse of dimensionality in metric spaces. Information Processing Letters 85(1), 39–46 (2003)
7. Dong, W., Wang, Z., Josephson, W., Charikar, M., Li, K.: Modeling lsh for performance tuning. In: Proceedings of the 17th ACM Conference on Information and Knowledge Management, CIKM 2008, pp. 669–678. ACM, New York (2008)
8. Drepper, U.: What every programmer should know about memory (2007), `http://www.akkadia.org/drepper/cpumemory.pdf` (last checked August 2012)
9. Eddelbuettel, D., Francois, R.: Rcpp: Seamless R and C++ integration. Journal of Statistical Software 40(8), 1–18 (2011)
10. Elizarov, R.: Millions quotes per second in pure Java (2013), `http://blog.devexperts.com/millions-quotes-per-second-in-pure-java/` (last accessed on May 14, 2013)
11. Esuli, A.: Use of permutation prefixes for efficient and scalable approximate similarity search. Inf. Process. Manage. 48(5), 889–902 (2012)
12. Faloutsos, C.: Searching Multimedia Databases by Content. Kluwer Academic Publisher (1996)
13. Figueroa, K., Navarro, G., Chávez, E.: Metric Spaces Library (2007), `http://www.sisap.org/Metric_Space_Library.html`
14. Figueroa, K., Fredriksson, K.: Speeding up permutation based indexing with indexing. In: Proceedings of the 2009 Second International Workshop on Similarity Search and Applications, SISAP 2009, pp. 107–114. IEEE Computer Society, Washington, DC (2009)
15. Fredriksson, K.: Engineering efficient metric indexes. Pattern Recognition Letters 28(1), 75–84 (2007)
16. Fulgham, B.: The computer language benchmarks game (2013), `http://benchmarksgame.alioth.debian.org/` (last accessed on May 14, 2013)
17. Gonzalez, E., Figueroa, K., Navarro, G.: Effective proximity retrieval by ordering permutations. IEEE Transactions on Pattern Analysis and Machine Intelligence 30(9), 1647–1658 (2008)
18. Gonzalez, E.C., Figueroa, K., Navarro, G.: Effective proximity retrieval by ordering permutations. IEEE Transactions on Pattern Analysis and Machine Intelligence 30(9), 1647–1658 (2008)
19. Hedges, L.V., Vevea, J.L.: Fixed-and random-effects models in meta-analysis. Psychological Methods 3(4), 486–504 (1998)
20. Hundt, R.: Loop recognition in C++/Java/Go/Scala. In: Proceedings of Scala Days 2011 (2011)
21. Indyk, P.: Nearest neighbors in high-dimensional spaces. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, pp. 877–892. Chapman and Hall/CRC (2004)
22. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, STOC 1998, pp. 604–613. ACM, New York (1998)
23. Jacobs, D., Weinshall, D., Gdalyahu, Y.: Classification with nonmetric distances: Image retrieval and class representation. IEEE Transactions on Pattern Analysis and Machine Intelligence 22(6), 583–600 (2000)
24. King, G.: How not to lie with statistics: Avoiding common mistakes in quantitative political science. American Journal of Political Science, 666–687 (1986)
25. King, R.S.: The top 10 programming languages (the data). IEEE Spectrum 48(10), 84–84 (2011)

26. Kushilevitz, E., Ostrovsky, R., Rabani, Y.: Efficient search for approximate nearest neighbor in high dimensional spaces. In: Proceedings of the 30th Annual ACM Symposium on Theory of Computing, STOC 1998, pp. 614–623. ACM, New York (1998)
27. Lokoč, J., Hetland, M.L., Skopal, T., Beecks, C.: Ptolemaic indexing of the signature quadratic form distance. In: Proceedings of the Fourth International Conference on SImilarity Search and APplications, SISAP 2011, pp. 9–16. ACM, New York (2011)
28. Mu, Y., Yan, S.: Non-metric locality-sensitive hashing. In: AAAI (2010)
29. Novak, D., Kyselak, M., Zezula, P.: On locality-sensitive indexing in generic metric spaces. In: Proceedings of the Third International Conference on SImilarity Search and APplications, SISAP 2010, pp. 59–66. ACM, New York (2010)
30. Parri, J., Shapiro, D., Bolic, M., Groza, V.: Returning control to the programmer: Simd intrinsics for virtual machines. Commun. ACM 54(4), 38–43 (2011)
31. Pestov, V.: Indexability, concentration, and VC theory. Journal of Discrete Algorithms 13, 2–18 (2012); Best Papers from the 3rd International Conference on Similarity Search and Applications (SISAP 2010)
32. Pestov, V.: Is the k-NN classifier in high dimensions affected by the curse of dimensionality? Computers & Mathematics with Applications (2012)
33. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann Publishers Inc. (2005)
34. Scott, D.W., Thompson, J.R.: Probability density estimation in higher dimensions. Technical Report, Rice University, Texas Huston (1983)
35. Shafi, A., Carpenter, B., Baker, M., Hussain, A.: A comparative study of java and c performance in two large-scale parallel applications. Concurrency and Computation: Practice and Experience 21(15), 1882–1906 (2009)
36. Skopal, T.: Unified framework for fast exact and approximate search in dissimilarity spaces. ACM Trans. Database Syst. 32(4) (November 2007)
37. Skopal, T., Bustos, B.: On nonmetric similarity search problems in complex domains. ACM Comput. Surv. 43(4), 34:1–34:50 (October 2011)
38. Uhlmann, J.: Satisfying general proximity similarity queries with metric trees. Information Processing Letters 40, 175–179 (1991)
39. Vivanco, R.A., Pizzi, N.J.: Scientific computing with Java and C++: a case study using functional magnetic resonance neuroimages. Software: Practice and Experience 35(3), 237–254 (2005)
40. Weber, R., Schek, H.J., Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: Proceedings of the 24th International Conference on Very Large Data Bases, pp. 194–205. Morgan Kaufmann (August 1998)
41. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach (Advances in Database Systems). Springer-Verlag New York, Inc., Secaucus (2005)
42. Zezula, P., Savino, P., Amato, G., Rabitti, F.: Approximate similarity retrieval with m-trees. The VLDB Journal 7(4), 275–293 (1998)
43. Zhang, Z., Ooi, B.C., Parthasarathy, S., Tung, A.K.H.: Similarity search on bregman divergence: towards non-metric indexing. Proc. VLDB Endow. 2(1), 13–24 (2009)

Designing Similarity Indexes with Parallel Genetic Programming

Tomáš Bartoš and Tomáš Skopal

Charles University in Prague, Faculty of Mathematics and Physics
Department of Software Engineering, SIRET Research Group
Malostranské nám. 25, 118 00 Prague 1, Czech Republic
{bartos,skopal}@ksi.mff.cuni.cz

Abstract. The increasing diversity of unstructured databases leads to the development of advanced indexing techniques as the metric indexing model does not fit to the general similarity models. Once the most critical postulate, namely the triangle inequality, does not hold, the metric model produces notable errors during the query evaluation. To overcome this situation and to obtain more qualitative results, we want to discover better indexing models for databases using arbitrary similarity measures. However, each database is unique in a specific way, so we outline the automatic way of exploring the best indexing method. We introduce the exploration approach using parallel genetic programming principles in a multi-threaded environment built upon recently introduced SIMDEX Framework. Furthermore, we introduce *smart pivot table* which is an intelligent indexing method capable of incorporating obtained results. We supplement the theoretical background with experiments showing the achieved improvements in comparison to the single-threaded evaluations.

1 Introduction

The current trends such as Big Data leads us to the challenge of finding information in large-scale databases of unstructured data. We know how to model data, how to store it, and which similarity model provides the best results for specific databases when searching for the most similar objects to the given query. However, we still struggle with the speed of query evaluations and we try to find optimizations in terms of query efficiency with respect to the accuracy of results.

In the similarity search domain, which affects our daily lives to a considerable extent (social network data, multimedia or biometric databases, etc.), a big challenge is how to handle distinct similarity models. The metric model already has its ideal solution – using *metric access methods* [17] for efficient database indexing. But for *nonmetric* similarity models [16] which better correspond to demanding user requirements, the situation is different. In most cases, each query degrades to slow sequential scanning which is applicable only to small-sized data.

There are various ways of how to deal with the situation when the metric space model is not applicable. We can tune the data [13,14], modify the indexing model [9,1], reveal the applicability of a specific method [12], or we can explore all potential indexing expression to find the best fit [15,2,3].

One common thing is to address the problematic *triangle inequality* and the corresponding triangle lowerbounding $LB_{\triangle}$ [17] which applies to the metric spaces using a single reference point (pivot p):

$$LB_{\triangle}(\delta(q,o)) = |\delta(q,p) - \delta(p,o)| \leq \delta(q,o) \tag{1}$$

It is a cheap and effective lowerbound method but for nonmetric distances it usually leads to false dismissals [14]. In this paper, we focus on this issue in a consistent way and demonstrate how to solve challenges with nonmetric similarity models by introducing the *smart pivot table* index (Section 4) which leverages the outcomes of parallelized SIMDEX Framework (Section 3.1).

2 Related Work

One of the most commonly used method dealing with the nonmetric models is TriGen algorithm [13,14] which converts nonmetric similarity spaces to semi-metric or metric ones. It uses a database sample and tunes the modified distance $g(\delta)$ with respect to the rate of non-triangular triplets (T-error) while minimizing the intrinsic dimensionality ρ [5]. The resulting modified distance $g(\delta)$ is directly employed into the pivot table [17] and it is used for exact but slower or approximate but fast similarity search. However, in some cases the distance modifications lead to large retrieval error or low indexability [16].

A similar concept based on the Lambda Tuning Algorithm has been studied for the nonmetric databases in which the triangle inequality does not hold [1]. Authors focus on tuning nine fuzzy T-norm operators also with the connection to the pivot table indexing. As they state, these alternative methods provide increased efficiency but together with higher error rate.

The introduction of Ptolemaic indexing [9,12] shows a way of replacing the problematic triangle lowerbound (Equation 1) with *Ptolemy's inequality* which for any database objects x, y, u, and v defines the following relation

$$\delta(x,v) \cdot \delta(y,u) \leq \delta(x,y) \cdot \delta(u,v) + \delta(x,u) \cdot \delta(y,v) \tag{2}$$

If the distance function δ holds the properties of *identity*, *positivity*, *symmetry*, and satisfies *Ptolemy's inequality*, we can use *ptolemaic lowerbound* ($\mathrm{LB}_{\mathtt{ptol}}$). To do so, we first define the candidate bound δ_{C} using two pivots p and s:

$$\delta_{\mathrm{C}}(q,o,p,s) = \frac{|\delta(q,p) \cdot \delta(o,s) - \delta(q,s) \cdot \delta(o,p)|}{\delta(p,s)} \tag{3}$$

For simplicity, we let $\delta_{\mathrm{C}}(q,o,p,s) = 0$ if $\delta(p,s) = 0$. Considering a set of pivots $\mathbb{P}$, we maximize the candidate bound δ_{C} over all pairs of distinct pivots:

$$\mathrm{LB}_{\mathtt{ptol}}(\delta(q,o)) = \max_{p,s \,\in\, \mathbb{P}} \delta_{\mathrm{C}}(q,o,p,s) \leq \delta(q,o) \tag{4}$$

Ptolemaic method is valid for *signature quadratic form distance* [12] applied to image signatures [4], however it is difficult to determine which preconditions must hold, so it performs better than any other approach.

Algorithm 1. Parallel GP-SIMDEX ($M_{\delta,S}$, P)

```
Require: Distance matrix $M_{\delta,S}$, Number of parallel instances $P$
1: $coordinator \leftarrow$ new PGPCoordinator($instances$)
2: for $i = 1$ to $P$ do {$P$ denotes the level of parallelism}
3:     $instances$[i] $\leftarrow$ new GP-SIMDEX()   {[A] Init}
4:     $instances$[i].SetRandomInitialPopulation()
5:     $instances$[i].RunExplorationInBackground()   {[B] Map}
6: end for
7: $coordinator$.WaitForInstancesToFinish()   {[C] Process in the background}
8: return $coordinator$.BestResults;   {[D] Reduce and consolidate results}
```

More consolidated approach of dealing with nonmetric similarity spaces uses the recently introduced SIMDEX Framework [15,2,3] which explores the potentially infinite space of all lowerbound expressions and tries to find the best fit for indexing a given database. Our previous work outlines two approaches, one based on iterative searching [15] and another utilizing genetic algorithms [2,3].

3 Exploration Method

In this section, we introduce the parallel enhancements of the recently proposed GP-SIMDEX Framework [2] which applies genetic programming [10,11] for discovering suitable lowerbounds. The main contribution is in the revalidation of GP-SIMDEX in multi-threaded environment while further improving the qualitative results by employing nontrivial parallel genetic programming methods.

3.1 Parallel Genetic Programming

The idea of driving the exploration of new indexing methods by genetic programming (GP) principles is not completely new. Our recent work [2] reveals the potential of GP for lowerbound discovery. However, we admit that greater benefits will come with the parallel implementation of the proposed (single-threaded) GP-SIMDEX Framework. As we strongly believe its potential, we build a prototype that applies the framework to the multi-threaded environment. We implement intelligent Parallel Genetic Programming (PGP) exploration method using *island-population model* [7,8] together with the *map-reduce* approach [6]. Algorithm 1 outlines the high-level overview of the individual stages (A-D).

3.2 Parallel GP-SIMDEX Algorithm

PGP-SIMDEX uses more sophisticated evaluation than just running multiple independent evaluations of GP-SIMDEX at the same time. During the initial phase (**A**), we create a selected number of GP-SIMDEX instances (so called *islands*), each built with a random initial population of expressions. Every instance then repeatedly apply genetic operations to build further populations

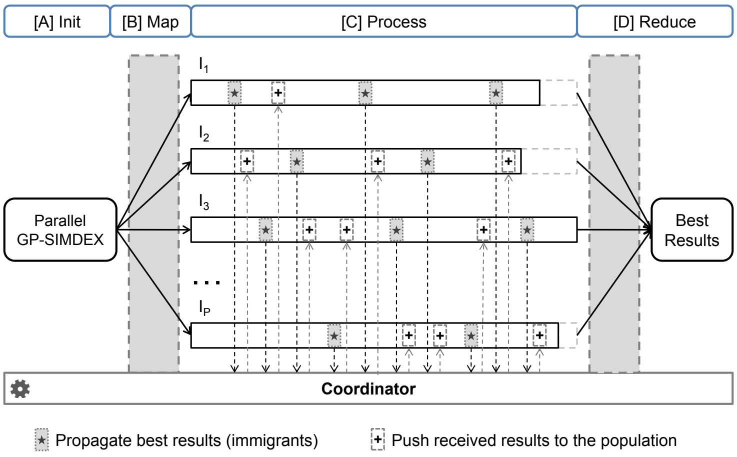


Fig. 1. PGP-SIMDEX – High-level overview of the parallel approach

(**B,C**) and it also regularly propagates the best results (*immigrants*) to another instance through the *coordinator* (**C**). These expressions are appended to the next population of the target instance and provide "fresh food" to drive the exploration efficiency. This approach is recommended to obtain better results more quickly [7,8]. The redistribution of expressions between running instances is completely random, so one instance might receive more immigrants than the others. Figure 1 outlines a sample visualization of this process.

3.3 Comparing GP-SIMDEX vs. PGP-SIMDEX

To validate our concept and to compare it with GP-SIMDEX, we generated eight smaller datasets of 11,000 random objects (represented as points in $4D$ space) and applied several nonmetric similarity models[1] mostly with nonzero triangle inequality errors (see *Triangle Error* in Table 1). For the exploration, we used database samples consisting of 1,000 objects and sampled 10,000 N-tuples for evaluating each candidate expression. We run all experiments multiple times and provide only the best results.

The initial population includes 1,000 expressions which evolved in 1,000 generations. PGP-SIMDEX uses 10 islands and only 100 generations.

During evaluations, we studied the comparison of best fitness values (PGP-SIMDEX slightly outperforms GP-SIMDEX), the number of immigrants (additional expressions that "helped" PGP-SIMDEX) and how they relate to the overall improvement (see Table 1). With initial evaluations, we validated the applicability of parallel processing and its power to improve results, however till now we have obtained only small efficiency increase. To overcome this, we need to find the relation between the improvements and ideal PGP-SIMDEX settings.

[1] We selected L_p distances to demonstrate the variability of similarity models.

Table 1. Dataset overview

#	Distance	Triangle Error	Best Fitness		PGP Improvement	Total PGP immigrants
			GP	PGP		
1	Jeffrey Divergence	14.0 %	38.08	38.40	0.86 %	7,198
2	$L_{0.2}$	7.6 %	23.82	24.44	2.61 %	9,452
3	$L_{0.4}$	5.5 %	22.53	23.13	2.65 %	6,984
4	$L_{0.6}$	3.8 %	21.18	22.09	4.32 %	8,150
5	$L_{0.8}$	2.5 %	20.75	21.19	2.15 %	7,671
6	L_1	0.3 %	19.94	20.39	2.25 %	7,278
7	L_2	0.0 %	16.82	17.40	3.46 %	7,343
8	L_∞	0.3 %	12.27	12.50	1.85 %	6,923

4 Smart Pivot Table

The previous section shows the first steps towards building the *smart pivot table* indexing scheme that will be suitable for any data combined with arbitrary similarity measures. Based on the pivot table index [17], we extend the standard functionality by deriving the lowerbounding mechanism that will fit for the data.

The concept behind the *intelligent indexing* is based on the *double-pivot constraint* [17] and it follows the approach introduced with ptolemaic filtering [12]. During the query evaluation, we compute the lowerbound value

$$\texttt{LB\_Value} = \max_{S_k \subseteq \mathbb{P}} \Big(LB(q, o, S_k) \Big) \tag{5}$$

where S_k is a set of pivots from the pivot set $\mathbb{P}$, q is the query object and o is a database object. The number of evaluated sets S_k corresponds to all N-tuples or to a constant C while the cardinality of S_k depends on the number of independent variables N in the lowerbound expression LB, precisely $|S_k| = N$.

This approach allows us to use any valid lowerbound expression and filter out objects o_i without computing the generally expensive distance $\delta(q, o_i)$ for which the lowerbound value is greater than the given radius or the distance to the k-th nearest neighbor, i.e. $\texttt{LB\_Value} > maxRadius$.

Currently, we still use a small database sample with sampling N-tuples for evaluating candidates [2,3] in order to discover the best indexing method. As the next step, we will focus only on the data stored within/inserted to the pivot table to provide the indexing mechanism based purely on the indexed data – this is feasible as we modify the filtering options, not the data.

As the next step, we intend to extensively evaluate the *smart pivot table* and compare our concept with existing methods such as LAESA [17], the combination of LAESA and TriGen, or fuzzy approaches [1] on larger real-world databases.

5 Conclusions

In this paper, we introduce a new way of discovering lowerbound expressions for indexing databases. Compared to previous methods, it improves the efficiency within the multi-threaded environment using the parallel genetic programming combined with map-reduce principles. We perceive the proposed PGP-SIMDEX as the shift towards an intelligent indexing method (*smart pivot table*) applicable to any similarity model. To achieve this, we need to make two additional steps – dismiss the required database sampling and run the experiments in the extremely large scale which remain as the future work.

Acknowledgments. This research has been supported by Grant Agency of Charles University (GAUK) project 567312 and by Czech Science Foundation (GAČR) project 202/11/0968.

References

1. Bartoš, T., Eckhardt, A., Skopal, T.: Fuzzy Approach to Non-metric Similarity Indexing. In: SISAP 2011, pp. 115–116. ACM (2011)
2. Bartoš, T., Skopal, T., Moško, J.: Efficient Indexing of Similarity Models with Inequality Symbolic Regression. In: GECCO 2013. ACM (2013)
3. Bartoš, T., Skopal, T., Moško, J.: Towards Efficient Indexing of Arbitrary Similarity. SIGMOD Record 42(2), 5–10 (2013)
4. Beecks, C., Uysal, M.S., Seidl, T.: Signature quadratic form distance. In: Proc. ACM International Conference on Image and Video Retrieval, pp. 438–445 (2010)
5. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Comp. Surveys 33(3), 273–321 (2001)
6. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM 51(1), 107–113 (2008)
7. Fernandez, F., Spezzano, G., Tomassini, M., Vanneschi, L.: Parallel genetic programming. In: Parallel Metaheuristics, pp. 127–153. Wiley Interscience (2005)
8. Gagné, C., Parizeau, M., Dubreuil, M.: The Master-Slave Architecture for Evolutionary Computations Revisited. In: Cantú-Paz, E., et al. (eds.) GECCO 2003. LNCS, vol. 2724, pp. 1578–1579. Springer, Heidelberg (2003)
9. Hetland, M.L.: Ptolemaic indexing. arXiv:0911.4384 [cs.DS] (2009)
10. Koza, J.R.: Genetic programming. MIT Press, Cambridge (1992)
11. Koza, J.R., Poli, R.: Genetic programming. In: Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques. Springer (2005)
12. Lokoč, J., Hetland, M., Skopal, T., Beecks, C.: Ptolemaic indexing of the signature quadratic form distance. In: SISAP 2011, pp. 9–16. ACM (2011)
13. Skopal, T.: On fast non-metric similarity search by metric access methods. In: Ioannidis, Y., et al. (eds.) EDBT 2006. LNCS, vol. 3896, pp. 718–736. Springer, Heidelberg (2006)
14. Skopal, T.: Unified framework for fast exact and approximate search in dissimilarity spaces. ACM Transactions on Database Systems 32(4), 1–46 (2007)
15. Skopal, T., Bartoš, T.: Algorithmic Exploration of Axiom Spaces for Efficient Similarity Search at Large Scale. In: Navarro, G., Pestov, V. (eds.) SISAP 2012. LNCS, vol. 7404, pp. 40–53. Springer, Heidelberg (2012)
16. Skopal, T., Bustos, B.: On nonmetric similarity search problems in complex domains. ACM Comp. Surv. 43, 1–50 (2011)
17. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. Advances in Database Systems. Springer, USA (2005)

(Very) Fast (All) *k*-Nearest Neighbors in Metric and Non Metric Spaces without Indexing

Natalia Miranda[1], Edgar Chávez[2], María Fabiana Piccoli[1], and Nora Reyes[1]

[1] Universidad Nacional de San Luis, Argentina
[2] Universidad Nacional Autónoma de México

Abstract. Proximity queries consists in retrieving objects near a given query. To avoid a brute force scan over a large database, an index can be used. However, for some problems, indexes are mostly useless (their running times are worst than sequential scan).

On the other hand, researchers have tried massively parallel hardware (as GPGPU) in the quest of faster query times. The results have been modest because current algorithms are cumbersome, while GPGPU architectures favor simple kernels, have a clear memory hierarchy and need close to zero cross-talk between processing units. We have engineered very fast algorithms for proximity queries taking into account this principles, all of them are presented in this paper.

In our approach no index is built, the cross-talk between threads is eliminated, and the higher (faster) levels of memory hierarchy are consistently used. The absence of data structures allows to use all the available memory for the database, and furthermore makes possible to do stream processing on very large data collections.

1 Introduction

Due to an increasing interest in manipulating and retrieving complex data, the problem of proximity searching has received a lot of attention, while simultaneously give hard times to practitioners. Obtaining the k-Nearest Neighbors (k-NN) of a query object is central to many complex data operations such as query by content, copy detection, object tracking, and a large set of other applications. The problem is pervasive and it is found in many areas of research, from statistics, pattern recognition, computer vision to multimedia databases, and much more applications. For this and other proximity searching problems a sequential scan over the database can solve the problem, but for large instances it does not scale. This is a very active topic of research well documented in the literature[1–3]. The goal of indexing is to avoid a sequential scan to answer proximity queries. Indexing is a two stage problem, firstly the database is preprocessed and then it is ready to be queried. Indexing can be very time consuming, some indexes use quadratic time and/or quadratic space. Moreover, there are situations where the need to query is immediate. In those cases building a index is not even an option. An example is *object tracking*. As soon as an object is marked for tracking it needs to be reported in real time in every frame; there is no time to index.

Another example is duplicate detection in video and/or audio streams. Indexing is very slow and furthermore, in some applications the database will be queried only a few times making the amortized cost higher than a few sequential scans.

The above scenario is very restrictive and the usual (sequential and massively parallel) techniques documented in the literature are of no use for the problem. In this paper we present algorithms to query on the fly databases for proximity, using massive parallel hardware, more specifically, we are interested in the General Purpose Graphics Processing Unit (GPGPU). Current solutions for the GPU architecture are translated from the sequential counterparts, the net result is a very modest speedup due to auxiliary data structures, and/or the need to precompute an index. In this paper, by using a very strict adherence to the design principles and recommendations for the GPU architecture we report a solution orders of magnitude faster than sequential indexes and faster than other massively parallel approaches reported in the literature. Since we use no index, the data is readily available for querying on load, leveraging streaming applications where massive data is seeing only once and there is no time to preprocess or index.

1.1 Metric Space and Proximity Searching

A database for proximity searching is usually modeled as a finite subset of a metric space. A metric load is (X, S, d), with X a set (possibly infinite), a finite subset $S \subseteq X$ the database, and a distance $d : X \times X \to R^+$. Distances are preferred to plain (dis)similarity functions because they can be indexed, in principle, exploiting the triangle inequality. Non metric spaces have been also indexed making a reduction to metric spaces. We will not make use of triangle inequality, hence our model is more general. Queries of interest include Range Searching and the k Nearest Neighbors (k-NN). A range search is defined as $(q, r)_d = \{x \in U / d(q, x) \leq r\})$. In a k-NN query the goal is to retrieve the k closest elements in S to a query q, namely $|k\text{-NN}(q)| = k$ and $\{\forall x \in k\text{-NN}(q), v \in S \wedge v \notin k\text{-NN}(q), d(q, x) \leq d(q, v)\}$. Other problem of interest is the *All-k*-NN, solving the above problem for all x in S.

The goal of indexing is to avoid a sequential scan, but there is a well documented phenomenon known as *the curse of dimensionality* [1–3]. The measurable effect is that using an index can be slower than a sequential scan over the data. To avoid this odd behavior, approximate algorithms have been proposed, they usually have a threshold ϵ as parameter, so that the retrieved elements are guaranteed to have a distance to the query q at most $(1 + \epsilon)$ times of what was asked for, or they have probabilistic guarantees[4].

1.2 GPGPU

The GPU is a dedicated graphic card for personal computers, workstations or video game consoles. It is an interesting architecture for high performance computing. The GPU was developed with a highly parallel structure, high memory bandwidth and more chip surface dedicated to data processing than to data caching and flow control. It offers, in principle, a speedup to any standard graphics application[5]. Mapping general-purpose computation onto GPU implies the use of the graphics hardware to solve any applications, not necessarily of graphic nature. This is called GPGPU (General-Purpose GPU), GPU computational power is used to solve general-purpose problems[5, 6]. The parallel programming over GPUs has many differences from parallel programming in typical parallel computer, the most relevant are: *The number of processing units*, the *CPU-GPU memory structure* and the *Number of parallel threads*.

Every GPGPU program have some basic steps. Firstly the input data should be transferred to the graphics card from the CPU(host). Once the data is in place, a massive amount of threads can be started (with little overhead). Each thread works over its data and, at the end of the computation, the results should be copied back to the host main memory. Not every class of problem can be solved with the GPU architecture, the most suitable problems are those implementable with stream processing and using limited memory size, i.e. applications with abundant data parallelism without cross-talking among processes. The programming model is Single Instruction Multiple Data (SIMD).

The CUDA, supported since the NVIDIA Geforce 8 Series, enables to use GPU as a highly parallel computer for non-graphics applications [5, 7]. CUDA provides an essential high-Level development environment with standard C/C++ language. It defines the GPU architecture as a programmable graphic unit which acts as a coprocessor for the CPU. It has multiple streaming multiprocessors (SMs), each of them contains several (eight, thirty-two or forty-eight) scalar processors (SPs).

The CUDA programming model has two main characteristics: the parallel work through concurrent threads and the memory hierarchy. The user supplies a single source program encompassing both host (CPU) and *kernel* (GPU) code. Each CUDA program consists of multiple phases that are executed on either CPU or GPU. All phases that exhibit little or no data parallelism are implemented in CPU. In opposition, if the phases present much data parallelism, they are implemented as *kernel* functions in the GPU. A *kernel* function defines the code to be executed by all the threads launched in a parallel phase. The GPU resources are much more efficiently used if the kernel do not make branching (represented as `if` instructions), in other words, if all the threads follow the same execution path.

GPU computation considers a hierarchy of abstraction layers: *grid*, *blocks* and *threads*. The *threads*, basic execution unit that executes *kernel* funtion, in the CUDA model are grouped into *blocks*. All threads in a block are executed on one SM and can communicate among them through the *shared memory*. Threads in different blocks can communicate through *global memory*. Besides shared and global memory, the threads have their own local data space for variables. All $Thread - blocks$ form a *grid*. The number of grids, blocks per grid and threads per block are parameters fixed by the programmer, they can be adjusted to improve the performance.

With respect to the memory hierarchy, CUDA threads may access data from multiple memory spaces during their execution. Each thread has private local memory and each block has shared memory visible to all its threads. These memories have the same lifetime than the kernel. All threads have access to the global memory and two additional read-only memory spaces: the constant and texture memory spaces. The constant and texture memory spaces are optimized for different memory usages. The global, constant and texture memory spaces are persistent all the application life time. Each kind of memory has its own access cost, in order of speed it will be local, shared and global memory which is the most expensive to access. Please notice that local and shared memory have higher throughput and smaller latency than standard RAM in the CPU. Please bear that in mind, because this contributes to the very large speedup of our algorithms.

1.3 Related Work on GPU Proximity Searching

There are many massively parallel algorithms for metric indexes implemented in a GPU. Querying for k-NN has obtained most of the attention of researchers in the area. In [8–14] improve explicity the brute force algorithm (or sequential scan) to find the k-NN. They differ in the parts parallelized or the methodology applied. Other works [8, 15, 16] implemented some well known sequential metric indices, such as the List of Clusters (LC) and the SSS-Index. For the case of vector data authors in [17, 17–19] use Kd-trees for finding the k-NN and [18] apply a variant of the Kd-tree for the all k-NN problem.

All algorithms in the literature [8–13, 15, 16, 19] for k-NN using GPU, solutions have high complexity in the data structures. Furthermore, they have a high granularity. Kernels are not uniform and have a lot of branching. This implies synchronization and serialization of the threads, which means all of them have to wait to be in the same path again to resume. In a nutshell, they use conditionals and do diverse tasks depending on comparisons. On the other hand the algorithms demand a lot of memory resources for the data structures and intermediate data, e.g. distances to pivots, and allocate only very small instances of the metric databases. For example in [16] they use only one thread block for the actual k-NN search, this implies overloading all the threads in the block and consequently suboptimal GPU resources usage, most of the threads are not used. In [10–12] they propose to solve several queries at a time, but they use just the same amount of threads than for a single query. This again implies thread overload, memory starvation and idle processing units in the GPU. In [13] is also suboptimal in resource usage, to the point of letting a single thread to finish the searching process, implying all other threads are idle.

We have learned from all the above examples and in our proposal, detailed below, we have closely tailored a solution which is uniform, with a single branch which maximize the GPU usage. We have carefully selected the number of threads, have coalescent memory access, using sharing memory and with data independence which implies zero cross talk between threads. Additionally we have zero overload in the data structures, which implies all the available memory can be used for the database. Other researchers in High Performance Computing arrived to similar conclusions parallelizing matrix to matrix multiplication[20]. Shared memory is faster than GPU memory, and by avoiding branching they have obtained super linear speedup for their problem.

2 Our Approach

In the GPU we have enough juice to compute the distances from a query to all the database elements in one step. Even more, there will be much more threads available than the number of database elements allocated in the global GPU memory. After all the distances are computed, the objects with the smallest distances have to be identified, this is the real challenge. Those objects have to be selected avoiding crosstalk. In this context crosstalk is simply comparing the result of two threads. We will exemplify the problem with an unplugged setup. Please also notice that for a range query, all threads having distances smaller than the query radius can report themselves without the need to compare with anyone else, the problem is relevant only for k-NN queries.

The above problem is equivalent to the following: In a room with one thousand (say N) people everybody gets a number in a piece of paper. How can you know if you are among the k people with the smallest number in the piece of paper? A sorting algorithm is overkill, and looking at every other number is also very slow. Please notice that we have dropped the condition of a central process selecting the k smallest, the person have to claim the place without talking to others. We have devised a mechanism to obtain those k smallest numbers without exchanging information between everyone in the room. The kernel of a GPU/CPU CUDA program works better if the code is simple. Our contribution relies precisely in this step. We essentially need an algorithm to find the Top-k smallest elements in an array without a central supervisor.

Following the people in the room example, what we do is to arrange people in small groups and all of them show their paper among each other. If all of the numbers are bigger than mine I have the smaller one. I know I will be among the finalists and silently move to the finalist stage without talking to anyone else. The process can be repeated until only the smaller one remains. Extending the algorithm to k elements is straightforward.

We have implemented three different solutions for obtaining the $Top-k$ smallest elements in the array. Please notice that it is not necessary to sort all the objects in the stream, since we need only the k smallest. Figure 1 illustrates this stage.

The process iterates while the size of k-NN$_P$ list is grater than k. The *Partition* and *Join-K* stages are implicitly computed. Each launched thread determines the sub-list of work (*Partition*). Each thread determines over which element it has to work based on the local information brought from *IdThread* in the block and *IdBlock* that thread belongs. The *Select_k* stage is a computation unit that is responsible to choose k-NN objects of list L_i ($\cup L_i = L \forall i = 0 \cdots x-1$ and $x = \#B$) and store in partial list k-NN$_P$ ($Join-K$). The first time L is equal

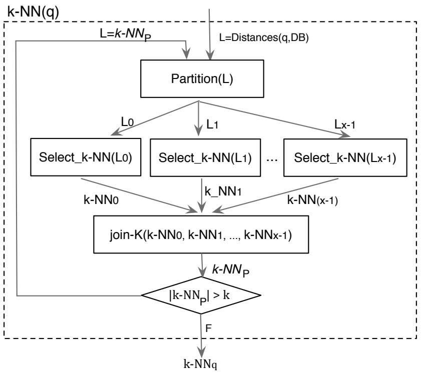


Fig. 1. Generic GPU Process to obtain the k nearest neighbor of a query q

to distances list. The next iteration, L has only all k-NN objects of each block. In the next sections, we explain different implementations of the *Select_k* stage.

2.1 *TopK_AA*

This is the implementation of the algorithm we designed to solve the problem, we named it *TopK_AA* (*AA=All-to-All*). The *Select_k* stage selects k-NN objects from the local list. Each thread in its block determines if its *distance* is among k-NN of its block. The comparison is *one-to-all*, each thread in the block contrasts its *distance* against all *distances* in the block. There are as many threads as distances in the local list. Once each thread establishes which is its order, it analyses if its distance is one of k-NN. If true, it reports its object to k-NN$_P$, otherwise it finishes. In the next iteration, the k-NN$_P$ size is equal to $k \times \#B$, where $\#B$ is the number of blocks.

TopK_AA has been designed taking into account the GPU characteristics such as shared memory, coalesced access to global memory and number of threads by block. In the code we avoid central decisions. For example each thread will compute where in the shared memory she can write and when she is among the top k results. Using atomic functions over the shared memory is faster than serialization over the global memory operations. Even more, the shared memory of the GPU is faster than the RAM memory in the CPU.

2.2 QuickSelect

The well known QuickSelect algorithm finds the x smallest elements of an unsorted array of z elements. In this case, *Select_k* stage applies the *QuickSelect* (*QSeP*) algorithm to choose the k-NN objects of a local list (in shared memory) by block. This process is applied iteratively while the pivot position is not equal to k. For each iteration, we select the pivot (it is a mean of three values in the local list), and partition the local list. If pivot position is equal to k, the partition with smaller elements than pivot is the k-NN of local list. Otherwise, two cases are possible: the position pivot is greater than k, or it is lower than k. In the first case, we run the *QuickSelect* over the partition with smaller elements. Otherwise, we work only over the second partition of local list (it has elements larger than the pivot).

2.3 TopK_AA + QuickSelect

This implementation of Select k stage is a combination of the two routines above. We called as QSeH (H for Hybrid). In this case, the first time we select the k-NN elements of local list applying QuickSelect algorithm (The local list is equal to distances list calculated in Distances stage). In the next steps, we determine the local k-NN using TopK_AA algorithm. We tried this implementation to see if the combination of the two approaches can boost the performance.

2.4 Thrust Library

There are many Sort libraries for CUDA, one of them is *Thrust*. It is part of the CUDA repositories. In this solution, we used the *Thrust* library to obtain the k-NN. Using the *Thrust* library is just a global sort, the k-NN stage is replaced by a call to *sort* algorithm of *Thrust* library, its output is the k-NN of each q. *Sort* algorithm is a black box, its implementation details are hidden to the user.

3 Solving Many k-NN Queries in Parallel

In large-scale systems it is not enough to speedup one query at a time, but it is necessary to leverage the capabilities of the GPU to answer in parallel several queries. Thereby we have to show how to achieve efficient and scalable performance in this context. We need to devise algorithms and optimizations specially tailored to support high-performance parallel query processing in the GPU.

In order to answer many queries in parallel, the GPU receives a query set and solves all of them at once. Each query, in parallel, applies a k-NN routing as explained in the previous sections, therefore the number of resources for this is equal to the resource amount to compute one query multiplied the number of queries solved in parallel. The number of queries to solve in parallel is determined according to the GPU resources, mainly the available memory.

Solving many queries in parallel involves carefully managing the blocks and their threads. At the same time, blocks of different queries are accessed in parallel. Hence, it is important a good administration of threads: which query is solved and which database element correspond to the query outcome. This can be easily accomplished by establishing a relationship among *Thread Id*, *Block Id*, *Query Id*, and *Database Element*.

4 The *All-k*-NN Problem

We need to determine the k-NN for each object in the database. Since we have solved one instance, it is enough to iterate through all the elements in the database considering them as queries. One just need to be careful to codify the results appropriately before sending the results back to the CPU. Considering these characteristic and the possibility of solving many queries in parallel, the solution is simple. If X queries are answered in parallel, we can compute *All-k*-NN in $\frac{N}{X}$ iterations, the value of X depends of GPU characteristics and the size of database.

5 Experimental Results

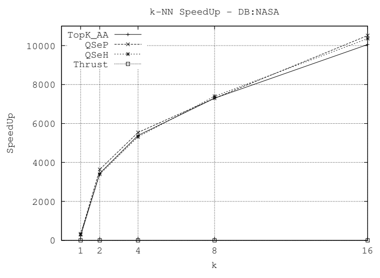
In this section we show and analyze the experimental results for the solutions of k-NN and *All-k*-NN problems. Please notice that range queries are trivial since it naturally avoids crosstalk between threads. We have not included this experiments, but they are faster than 1-NN queries. We selected two widely used benchmark databases from the SISAP Metric Library (`www.sisap.org`).

– *NASA images:* a set of 40,700 20-dimensional feature vectors, generated from images downloaded from NASA. The distance is euclidean.
– *COLORS histograms:* a set of 112,682 feature vectors of dimension 112, obtained from color histograms from an image database. Any quadratic form can be used as a distance, so we chose Euclidean distance as the simplest meaningful alternative.

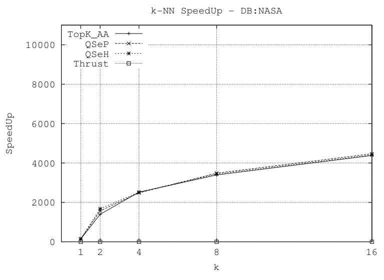
Each reported value is the average of many executions of the corresponding algorithm. We use for both databases k values of: 1, 2, 4, 8, and 16. The analysis was made for two generations of GeForce GPU whose characteristics (Global Memory, SM, SP, Clock rate, Compute Capability) are GTX330: (512MB,6,8,1.04GHz,1.2), GTX470: (1280MB, 14, 32, 1.2GHz, 2.0) and GTX550Ti: (1024MB, 4, 48, 1.96GHz,2.1). The CPU is an Intel core i3, 2.13 GHz and 3 GB of memory. The results are expressed in Speed up ($Sp = \frac{Time_{Sec}}{Time_{Par}}$). For lack of space, the shown results are only on architectures GTX330 and GTX470.

To evaluate the behavior of our solutions against a good sequential solution, we did choose the SAT^{+} metric index. It is a new version of *Spatial Approximation Tree* (*SAT*) [21], improved by a new selection order of the neighbors in the tree (distal nodes). SAT^{+} has shown to be one of the most competitive exact proximity searching index in the literature, and it is very scalable and resistant to the curse of dimensionality [22]. It is implemented in the *C* language.

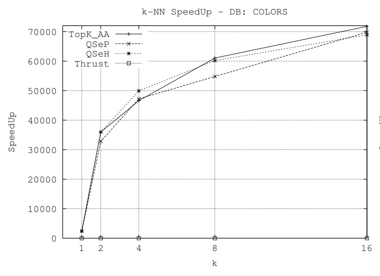
Figure 2 shows the obtained speed up by k-NN queries for two before mentioned metric spaces. For all GPUs, we achieve very good results. The best speed up is obtained by our implementation, it can noticed that the *Thrust* solution reaches a significantly lower speedup compared to the others three proposed solutions.



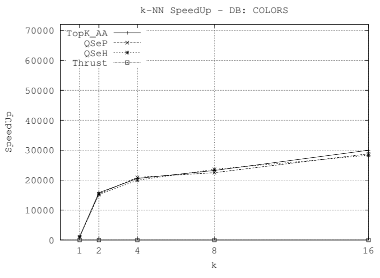
(a) GTX 470



(b) GTX 330M

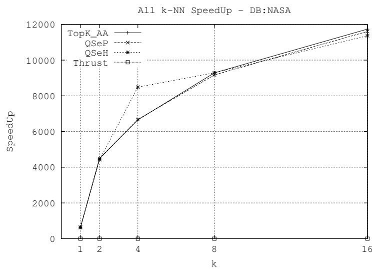


(c) GTX 470

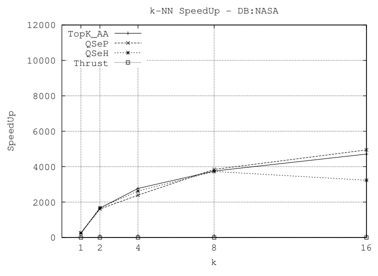


(d) GTX 330M

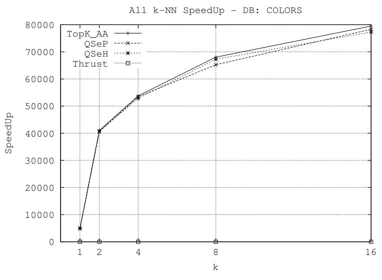
Fig. 2. Speedup of k-NN query implementations for NASA and COLORS databases, on two different GPUs



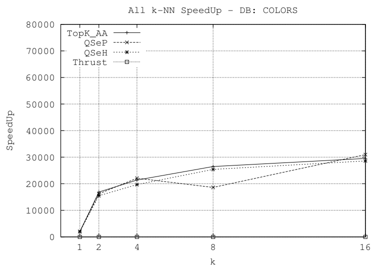
(a) GTX 470



(b) GTX 330M



(c) GTX 470



(d) GTX 330M

Fig. 3. Speedup of All-k-NN query implementations for NASA and COLORS databases, on two different GPUs

The accelerations are different because they depend of GPU resources, but in all cases they are significant. Figure 3 resumes the behavior of four proposed solutions of *All-k*-NN query. Like k-NN solutions, the accomplished speedups by our three implementation are much higher than the one obtained using *Thrust*.

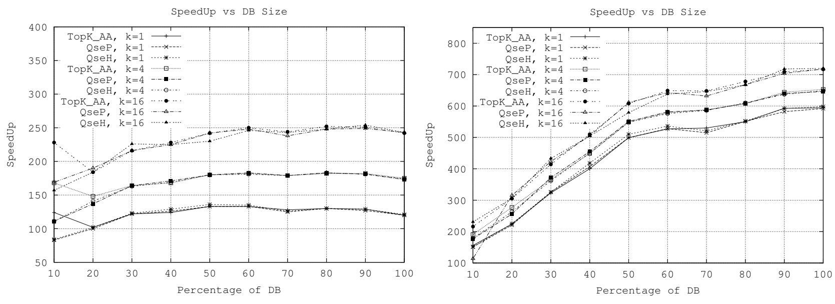
The speedup increase as both the database size and k, the number of neighbors, increase. Tables 1 and 2 show the obtained throughput (number of queries by second) by all implementations evaluated. Table 1 illustrates the throughput obtained for the sequential index used SAT^+. For lack of space, we only report the results obtained for $k = 1$ and $k = 16$. The results clearly show the benefits for all GPU architectures. In every case and k value, the number of queries by second is impressively high.

Table 1. Throughput of k-NN query for SAT^+

DB	k	PC
NASA	1	35160,14
	16	1219,08
COLORS	1	4608,80
	16	163,49

Table 2. Throughput of k-NN query, for four algorithms in two GPUs

Algorithm	k	NASA		COLORS	
		GTX 470	GTX 330M	GTX 470	GTX 330M
TopK_AA	1	6779661,02	3883495,15	6106870,23	3827751,20
	16	6851612,90	3919909,50	6666666,67	3903703,70
QSeP	1	9913258,98	4938271,60	9720534,63	4040404,04
	16	11428571,43	4993449,78	10624169,99	4444444,44
QSeH	1	7272727,27	3960396,04	6153846,15	3418803,42
	16	7666666,67	3982439,02	7272727,27	3448275,86
Thrust	1	94,35	41,31	94,35	41,31
	16	94,45	41,35	94,45	41,35



(a) SP without construction cost (b) SP with construction cost

Fig. 4. Speedup of k-NN query implementations for COLORS databases, considering increasing database sizes

6 Conclusions, Remarks and Future Work

We have shown a very fast algorithm for the k-Nearest Neighbors and the All k-Nearest Neighbors problems. Our proposal is several orders of magnitude faster than state of the art sequential and massively parallel approaches. We have not included other GPU implementations in our benchmark because they used different hardware, we can only deduct from the published results that the speedup obtained is very modest. Our dramatic speedup is due to a combination of faster memory, non blocking parallel processing and adherence to the design principles. A more detailed experimental evaluation is needed to know if the speedup can be considered super linear on the number of processing units.

We do not need to build an index beforehand, widening the applications of our algorithms. Also notice that our approach can easily solve range queries, much faster than nearest neighbor queries because Top-k selection is skipped. We are working on a randomized (exact) version of the k-NN algorithm using repeated calls to a range query routine. Others properties of our approach is the absence of the curse of dimensionality, and the possibility to query non-metric (dis)similarity databases.

References

1. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. Advances in Database Systems, vol. 32. Springer (2006)
2. Samet, H.: Foundations of Multidimensional and Metric Data Structures. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Morgan Kaufmann Publishers Inc., San Francisco (2005)
3. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.: Searching in metric spaces. ACM Comput. Surv. 33(3), 273–321 (2001)
4. Benjamin, B., Navarro, G.: Probabilistic proximity searching algorithms based on compact partitions. Discrete Algorithms 2(1), 115–134 (2004)
5. Kirk, D., Hwu, W.: Programming Massively Parallel Processors, A Hands on Approach. Elsevier, Morgan Kaufmann (2010)
6. Owens, J., Houston, M., Luebke, D., Green, S., Stone, J., Phillips, J.: GPU Computing, pp. 879–899. IEEE (2008)
7. NVIDIA. Nvidia cuda compute unified device architecture, programming guide version 4.2. In: NVIDIA (2012)
8. Barrientos, R., Gomez, J., Tenllado, C., Prieto, M.: Heap based k-nearest neighbor search on gpus. In: XXI Jornadas de Paralelismo, pp. 559–566 (September 2010)
9. Garcia, V., Debreuve, E., Barlaud, M.: Fast k nearest neighbor search using GPU. In: CVPR Workshop on Computer Vision on GPU (CVGPU), Anchorage, Alaska, USA (June 2008)
10. Garcia, V., Debreuve, E., Nielsen, F., Barlaud, M.: k-nearest neighbor search: fast GPU-based implementations and application to high-dimensional feature matching. In: IEEE International Conference on Image Processing, Hong Kong (September 2010)
11. Kato, K., Hosino, T.: Solving k-nearest neighbor problem on multiple graphics processors. In: ACM (ed.) 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, CCGRID, pp. 769–773 (2010)

12. Kuang, Q., Zhao, L.: A practical gpu based knn algorithm. In: International Symposium on Computer Science and Computational Technology (ISC-SCT), pp. 151–155 (2009)
13. Liang, S., Liu, Y., Wang, C., Jian, L.: Design and evaluation of a parallel k-nearest neighbor algorithm on CUDA-enabled GPU. In: IEEE 2nd Symposium on Web Society (SWS), p. 53 (2010)
14. Rozen, T., Boryczko, K., Alda, W.: Gpu bucket sort algorithm with applications to nearest-neighbour search. Journal of WSCG 16(1-3), 161–167 (2008)
15. Barrientos, R., Gomez, J., Tenllado, C., Prieto, M.: Query processing in metric spaces using gpus. In: XXII Jornadas de Paralelismo (2011)
16. Barrientos, R.J., Gómez, J.I., Tenllado, C., Matias, M.P., Marin, M.: kNN Query Processing in Metric Spaces Using GPUs. In: Jeannot, E., Namyst, R., Roman, J. (eds.) Euro-Par 2011, Part I. LNCS, vol. 6852, pp. 380–392. Springer, Heidelberg (2011)
17. Zhou, K., Hou, Q., Wang, R., Guo, B.: Real-time kd-tree construction on graphics hardware. In: ACM SIGGRAPH Asia, Papers, SIGGRAPH Asia 2008, pp. 126:1–126:11. ACM, New York (2008)
18. Brown, S., Snoeyink, J.: Gpu nearest neighbors using a minimal kd-tree. In: Second Workshop on Massive Data Algorithmics (MASSIVE 2010), Snowbird, Utah, USA (June 2010)
19. Qiu, D., May, S., Nüchter, A.: Gpu-accelerated nearest neighbor search for 3d registration. In: Fritz, M., Schiele, B., Piater, J.H. (eds.) ICVS 2009. LNCS, vol. 5815, pp. 194–203. Springer, Heidelberg (2009)
20. Djinevski, S.R.L., Gusev, M.: Superlinear speedup for matrix multiplication in gpu devices. In: ICT Innovations 2012, pp. 285–294 (2013)
21. Navarro, G.: Searching in metric spaces by spatial approximation. The Very Large Databases Journal (VLDBJ) 11(1), 28–46 (2002)
22. Chavez, E., Ludueña, V., Reyes, N., Roggero, P.: Faster proximity searching with the distal sat (2012) (submitted, draft)

On Scalable Approximate Search with the Signature Quadratic Form Distance

Jakub Lokoč, Tomáš Grošup, and Tomáš Skopal

SIRET Research Group, Department of Software Engineering,
Faculty of Mathematics and Physics, Charles University in Prague
{lokoc,skopal}@ksi.mff.cuni.cz, tomasgrosup@gmail.com

Abstract. The signature quadratic form distance and feature signatures have become a respected similarity space for effective content-based retrieval. Furthermore, the similarity space is configurable by a parameter alpha affecting both retrieval precision and intrinsic dimensionality, and thus interesting trade-offs can be achieved when a metric index is used for exact search. In this paper we combine such configurable model with state of the art approximate search techniques developed for the M-Index. In the experiments, we show that employing a configuration resulting in the best effectiveness of the measure leads also to very competitive approximate search effectiveness when using the M-Index, regardless the high intrinsic dimensionality of the corresponding similarity space.

Keywords: Similarity Search, Approximate Search, Content-based Retrieval, Signature Quadratic Form Distance, M-Index.

1 Introduction and Related Work

The metric space approach [6,15] has become a widely accepted model for efficient similarity search in huge collections of unstructured data. However, the efficiency of the metric space approach remains still a big challenge for distance spaces and data suffering from high intrinsic dimensionality [6]. Furthermore, there are emerging more effective retrieval models based on metric spaces comprising more complex descriptors and more expensive (combinations of) distance metrics [3,9]. Such new complex metric spaces are designed for maximal effectiveness and thus often suffer from high intrinsic dimensionality. As a consequence, the database experts developing efficient metric indexing techniques face more often the problem of the "indexability" of the metric spaces.

Recently, several papers have investigated a novel approach which tries to face the problem of high intrinsic dimensionality by synergistic modeling. The approach tries to synergistically combine the domain specific knowledge to find optimal compromises between effectiveness and indexability already in the distance modeling phase (i.e., model indexable spaces). Since the indexability can be simply indicated by the intrinsic dimensionality, the distance modeling process is not overloaded too much. The signature quadratic form distance [3] (SQFD) designed to compare two feature signatures is a typical candidate for such synergistic modeling, because it provides an inner parameter α (used in the Gaussian similarity function) that can be utilized to fine-tune the retrieval performance. As has been recently shown [1], the α parameter affects both precision and intrinsic dimensionality and so the domain experts can provide various configurations of modeled SQFD distance spaces. Such domain specific configurations provided to database experts can result in interesting trade-offs, where at least two orders of magnitude speedup can be achieved for the price of only slightly decreased quality of the measure. Let us denote, low intrinsic dimensionality is crucial especially in cases where fast exact search is required.

However, the modern retrieval modalities (e.g., an interactive multimedia exploration [2]) do not require exact search and thus can benefit also from the approximate search techniques [10,14,5,13,11] that are orthogonal to the previously mentioned synergistic modeling approach. Hence in this paper, we focus on the combination of the synergistic modeling of the signature quadratic form distance and the approximate search techniques represented by the state of the art metric access method the M-Index [11,12]. More specifically, we investigate the effectiveness of the approximate nearest neighbor search using M-Index for various configurations of SQFD distance spaces. In the following section, we focus on the configurations resulting in the best quality of the measure but resulting also in the worst intrinsic dimensionality. We experimentally demonstrate, the best quality of the measure is sufficient criterion, while the high intrinsic dimensionality is not an obstacle, when the approximate nearest neighbor search using M-Index is utilized.

2 Experimental Evaluation

In this section, we describe the test settings and then we compare approximate search performance of SQFD spaces for various values of α. Finally, we discuss the results and point on the future work.

The Testbed

For the experiments, we make use of three different datasets, each with different source of ground truth. Specifically, we use a subset of the ALOI dataset [7] comprising 12,000 images divided into 1,000 classes, each class contains 12 images of a 3D object rotated by 30 degrees; a subset of the Profimedia dataset [4] comprising 21,993 images divided into 100 classes, where the ground truth was collected semi-automatically and verified by users; the TWIC dataset [9] comprising 11,555 images forming 197 classes, where each class represents images obtained by a keyword query to the google images search engine. Each TWIC class was further manually filtered by users. The feature signatures were extracted using a GPU extractor tool [8]. As the query objects, one representative from each class was selected for all three datasets[1], resulting in 1000 query

[1] Profimedia dataset is already provided with a set of query objects.

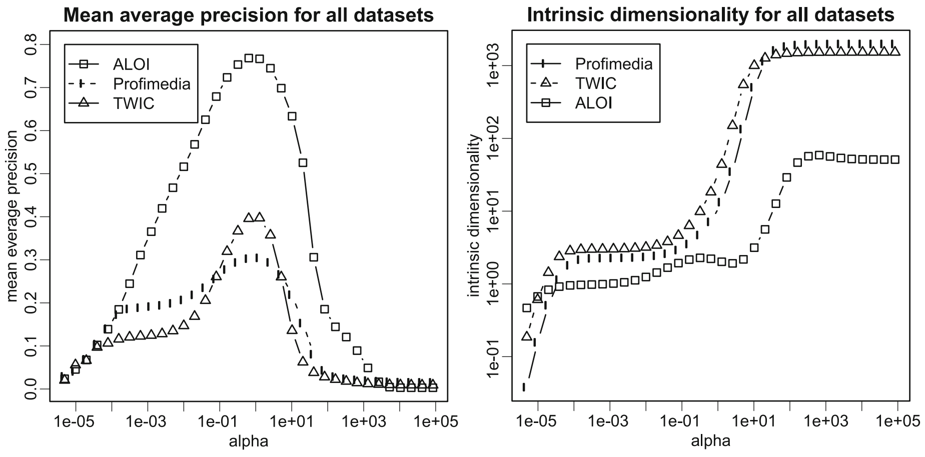


Fig. 1. Mean average precision and intrinsic dimensionality

objects for ALOI, 100 query objects for Profimedia and 197 query objects for TWIC. To create the PivotTable and the M-Index, we have used only 10 randomly selected pivots. In order to obtain more robust results, we have run all the experiments for each dataset four times and averaged the results. For each run, we have randomly selected one group of ten pivots from the corresponding dataset.

In Figure 1, the mean average precision evaluated for the query objects and the intrinsic dimensionality are depicted for all three datasets for varying α. We may observe there is an optimal value of α (≈ 1) for the mean average precision for all three datasets, which is probably caused by similar feature extraction settings for all the datasets. We may also observe the intrinsic dimensionality is almost always increasing function of the α parameter until some point is reached and then the function becomes constant. Let us also denote the x axis is logarithmic in both graphs, while the y axis is logarithmic only in the second graph. In Figure 2, we may observe the practical impact of low α on the efficiency of the exact metric search. We may also observe the interesting trade-off between precision and efficiency. For example, considering $\alpha = 0.32$ for the Profimedia dataset, the query processing according to $\alpha = 1.28$ is almost three times faster while the precision decreases just by 5.5%. However, as we show in the next section, the benefits of small α are neglected when using approximate search methods designed for the M-Index.

Approximate Search Results

As an approximate search method we have used a three level static variant of the M-Index with the same ordering heuristic for the kNN search as in section 3.5 of [11]. We use an early termination strategy controlled by the fixed number of allowed distance computations (10, 25, 50, 100, 200, 400 and 800). We also reorder

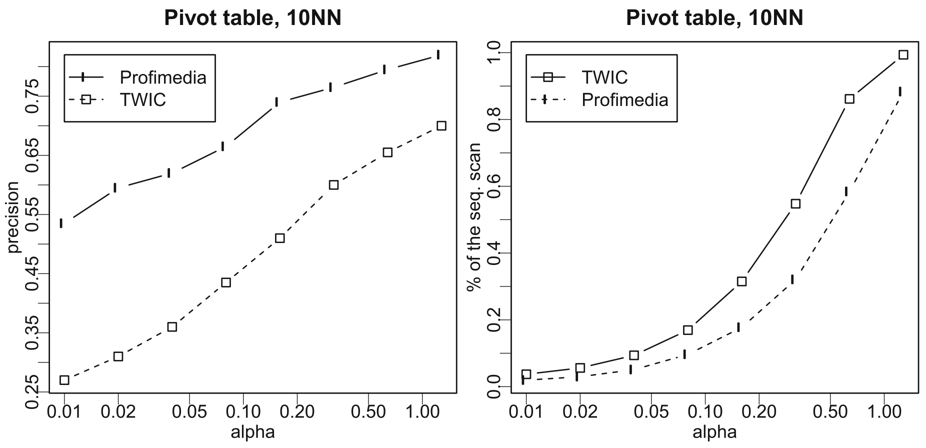


Fig. 2. Precision and query costs of the exact search using pivot table

the objects in each retrieved bucket according to the lower bound estimation of the distance to the query object (using L_{max} on vectors in the pivot space). In almost all the experiments, we focus on the precision defined by the percentage of relevant objects (the same class as of the query object) in the query result. For the lack of the space, we present mostly graphs for 10NN queries, however, the similar behavior was observed also for higher query cardinalities. Let us also denote fixing the number of allowed distance computations to 10 is equivalent to just reading of the first ten objects from the first bucket. We use this option to demonstrate the quality of the default ordering implied by the pivots.

Let us start with the number of visited buckets for particular configurations of the SQFD space for the ALOI dataset depicted in the upper left graph of Figure 3. Since for lower values of $\alpha \leq 0.04$ the intrinsic dimensionality is low, the filtering power of the pivots is high which enables nearly all buckets to be visited when the number of enable distance evaluations is fixed to 800. For higher values of α, the percentage of visited buckets is much lower because all the enabled distances are spent in the first visited buckets. As expected, the similar behavior was observed for all three datasets.

The most important results concerning the retrieval effectiveness are depicted in the last three graphs of Figure 3. In the upper right graph, the approximate search precision for the ALOI dataset is depicted. We may observe a slightly better performance of the configurations resulting in lower intrinsic dimensionality when high level of the approximation is required. On the other side, if the query can spend more distance evaluations, the precision can reach better values for $\alpha = 0.64$ because of the better quality of the measure. In the bottom graphs of Figure 3, a slightly different behavior can be detected for Profimedia and TWIC datasets, where configurations with $\alpha \in \{0.32, 0.64, 1.28\}$ clearly outperforms the configurations with lower α. Let us emphasize, although the intrinsic dimensionality of the signature quadratic form distance configured by $\alpha = 1.28$ is

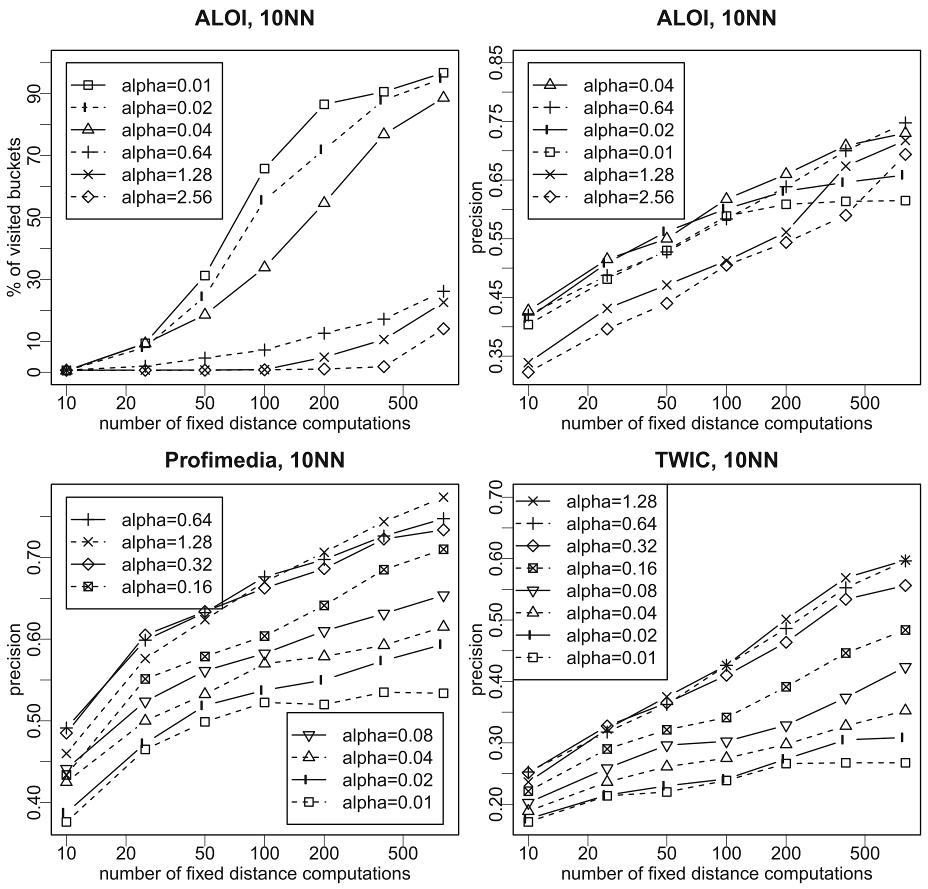


Fig. 3. Results of the approximate search controlled by the fixed number of allowed distance computations

almost 45 for the TWIC dataset, the configuration takes almost always the top precision. In the last set of experiments depicted in Figure 4, we may observe very similar behavior also for the changing size of the query result, where the configuration with highest α results in the best precision and visits on average just two buckets. To sum up the results, the configurations resulting in the best mean average precision but worst intrinsic dimensionality are highly competitive in approximate search.

Discussion and Future Work

The experiments show high intrinsic dimensionality is not an obstacle for the signature quadratic form distance when a set of k relevant objects is retrieved approximately using the M-Index. Furthermore, the high intrinsic dimensionality results also in a lower number of visited buckets for early termination strategies

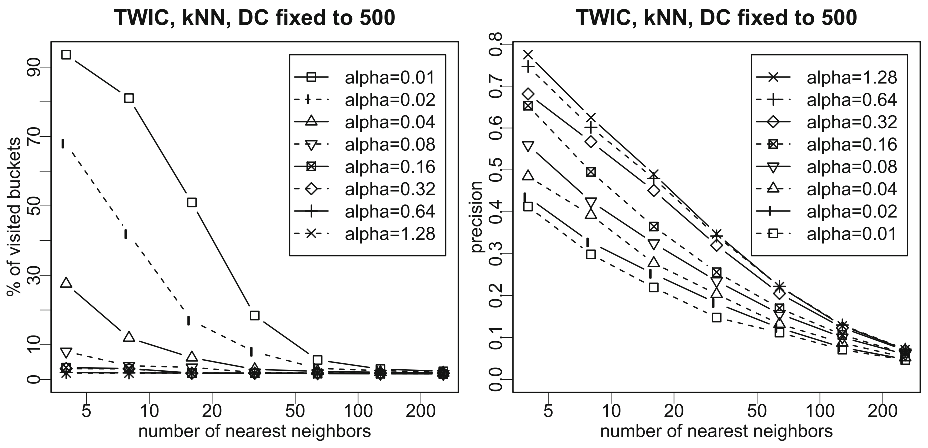


Fig. 4. Precision and number of visited buckets for growing number of nearest neighbors

that use fixed number of distance evaluations. Such performance can be beneficial for scenarios where the precision can be relaxed (exploration, browsing). However, in cases the real nearest neighbors are desired, the intrinsic dimensionality can become a serious problem. In the future, we plan some experiments focusing on the real nearest neighbor retrieval, however, such experiments are much more complicated because they require query objects with annotated ordering of the result. Furthermore, we plan to perform experiments on larger and more noisy datasets. Although the synergistic modeling can be beneficial in some cases, the approach has one main drawback – it requires an index for each utilized configuration. Therefore, we would also like to concentrate on such distance spaces, where the effectiveness of the approximate search techniques deteriorates rapidly and so the synergistic modeling could be the only clue to index the data.

3 Conclusions

In this paper we have combined various configurations of the signature quadratic form distance with the state of the art approximate search techniques developed for the M-Index. In the experiments, we show that employing a configuration resulting in the best effectiveness of the measure leads also to very competitive approximate search effectiveness when using the M-Index, regardless the high intrinsic dimensionality of the corresponding similarity space. We conclude the signature quadratic form distance trained for the best effectiveness can be also a good choice for effective and efficient approximate search using the M-Index.

Acknowledgments. This research has been supported in part by the grant SVV-2013-267312 and by Czech Science Foundation projects P202/11/0968 and P202/12/P297.

References

1. Beecks, C., Lokoč, J., Seidl, T., Skopal, T.: Indexing the signature quadratic form distance for efficient content-based multimedia retrieval. In: Proc. ACM Int. Conf. on Multimedia Retrieval, pp. 24:1–24:8 (2011)
2. Beecks, C., Skopal, T., Schöffmann, K., Seidl, T.: Towards large-scale multimedia exploration. In: Proc. 5th International Workshop on Ranking in Databases (DBRank 2011), Seattle, WA, USA, pp. 31–33 (2011)
3. Beecks, C., Uysal, M.S., Seidl, T.: Signature quadratic form distance. In: Proceedings of the ACM International Conference on Image and Video Retrieval, CIVR 2010, pp. 438–445. ACM, New York (2010)
4. Budikova, P., Batko, M., Zezula, P.: Evaluation platform for content-based image retrieval systems. In: Gradmann, S., Borri, F., Meghini, C., Schuldt, H. (eds.) TPDL 2011. LNCS, vol. 6966, pp. 130–142. Springer, Heidelberg (2011)
5. Chávez, E., Figueroa, K., Navarro, G.: Effective proximity retrieval by ordering permutations. IEEE Trans. Pattern Anal. Mach. Intell. 30(9), 1647–1658 (2008)
6. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. 33(3), 273–321 (2001)
7. Geusebroek, J.-M., Burghouts, G.J., Smeulders, A.W.M.: The Amsterdam Library of Object Images. IJCV 61(1), 103–112 (2005)
8. Kruliš, M., Lokoč, J., Skopal, T.: Efficient extraction of feature signatures using multi-GPU architecture. In: Li, S., El Saddik, A., Wang, M., Mei, T., Sebe, N., Yan, S., Hong, R., Gurrin, C. (eds.) MMM 2013, Part II. LNCS, vol. 7733, pp. 446–456. Springer, Heidelberg (2013)
9. Lokoč, J., Novák, D., Batko, M., Skopal, T.: Visual image search: Feature signatures or/and global descriptors. In: Navarro, G., Pestov, V. (eds.) SISAP 2012. LNCS, vol. 7404, pp. 177–191. Springer, Heidelberg (2012)
10. Navarro, G.: Searching in metric spaces by spatial approximation. The VLDB Journal 11(1), 28–46 (2002)
11. Novak, D., Batko, M., Zezula, P.: Metric index: An efficient and scalable solution for precise and approximate similarity search. Inf. Syst. 36(4), 721–733 (2011)
12. Novak, D., Batko, M., Zezula, P.: Large-scale similarity data management with distributed metric index. Inf. Process. Manage. 48(5), 855–872 (2012)
13. Patella, M., Ciaccia, P.: Approximate similarity search: A multi-faceted problem. J. Discrete Algorithms 7(1), 36–48 (2009)
14. Skopal, T.: Unified framework for fast exact and approximate search in dissimilarity spaces. ACM Trans. Database Syst. 32(4) (2007)
15. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach (Advances in Database Systems). Springer-Verlag New York, Inc, Secaucus (2005)

PFMFind: A System for Discovery of Peptide Homology and Function

Aleksandar Stojmirović[1,⋆], Peter Andreae[2], Mike Boland[3], Thomas William Jordan[4], and Vladimir G. Pestov[5]

[1] National Center for Biotechnology Information, National Library of Medicine, National Institutes of Health, Bethesda, MD 20894, United States
[2] School of Engineering and Computer Science, Victoria University of Wellington, PO Box 600, Wellington 6140, New Zealand
[3] Riddet Institute, Massey University, PB 11 222, Palmerston North 4442, New Zealand
[4] School of Biological Sciences, Victoria University of Wellington, PO Box 600, Wellington 6140, New Zealand
[5] Department of Mathematics and Statistics, University of Ottawa, 585 King Edward Ave., Ottawa, ON K1N 6N5, Canada

Abstract. Protein Fragment Motif Finder (PFMFind) is a system that enables efficient discovery of relationships between short fragments of protein sequences using similarity search. It supports queries based on amino acid similarity matrices and position specific score matrices (PSSMs) obtained through an iterative procedure. PSSM construction is customisable through plugins written in Python. PFMFind consists of a GUI client, an index for fast similarity search and a relational database for storing search results and sequence annotations. It is written mostly in Python. The components of PFMFind communicate through TCP/IP sockets and can be located on different physical machines. PFMFind is freely available for download (under a GPL licence) from `http://pfmfind.stojmirovic.org`.

Keywords: similarity search, indexing, protein fragments.

1 Introduction

The biological functions of proteins are as much a function of particular motifs of peptide sequence as they are of the overall protein structure. It is of interest to the biologist to search for examples of convergent motifs as they are likely to indicate a functional role. While many approaches exist for finding longer sequence motifs (50 amino acids or more), finding relationships between short fragments (3–18 amino acids long) of full protein sequences also promises great rewards in understanding novel aspects of protein structure and function. These

⋆ To whom correspondence should be addressed. Current affiliation: Janssen Research & Development LLC, 1400 McKean Rd., Spring House, PA 19477, United States.

relationships might be evolutionary in origin or might arise by convergence, that is, by acquisition of the same biological function in evolutionarily distant species.

Finding short motifs presents significant challenges because many of the apparent relationships between short fragments could have arisen by chance and thus have no functional significance. Furthermore, most widely available tools for sequence database search and motif finding were designed with longer motifs in mind. For example, Watt and Doyle [11] observed that the NCBI BLAST [1] family of programs, the best known set of tools for searching biological sequence datasets, is not suitable for identifying shorter sequences with particular constraints and proposed a pattern search tool to find DNA or protein fragments that match a given sequence or a pattern exactly. This paper outlines the Protein Fragment Motif Finder (PMFind), a tool that uses database search to identify the conserved short peptide motifs of a query sequence and associates them with the available functional annotations.

2 Overview

The PFMFind system consists of three major components: a search engine for fast similarity search of datasets of short peptide fragments called FSIndex, a relational database, and the PFMFind graphical user interface (GUI) client (Fig. 1). PFMFind client takes user input, and communicates with FSIndex and the database through its components. It passes search parameters in batches to FSIndex and receives the results of searches that are then stored in the database. It also retrieves the results from the database and displays them, together with available annotations, to the user. The annotations are stored in a separate (BioSQL) schema in the database.

Most of PFMFind was written in the Python programming language, and uses both the standard Python library and additional modules such as Biopython (`http://www.biopython.org`). The components communicate using the standard TCP/IP socket interface and can therefore be located on different machines. Since PFMFind is highly modular, the GUI client can be replaced by a Python script for non-interactive use.

2.1 Similarity Search

PFMFind supports searches of datasets of short peptide fragments of fixed length using an ungapped similarity score obtained by summing similarity scores at each position of the fragments being compared. The positional similarity scores can be defined by standard score matrices such as PAM [3] or BLOSUM [6], or by position specific score matrices (PSSMs) [4]. A fragment dataset consists of all fragments of a specified length from a given protein sequence dataset (where the fragments may overlap).

Iterative construction of PSSM, similar to that used by PSI-BLAST [1], is supported through plugins – Python routines that take the results of a previous search and construct a PSSM. The default plugin uses the weighting procedure of

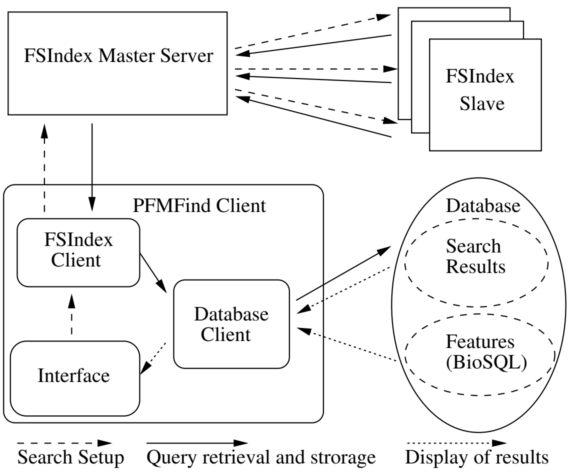


Fig. 1. Structure of PFMFind system

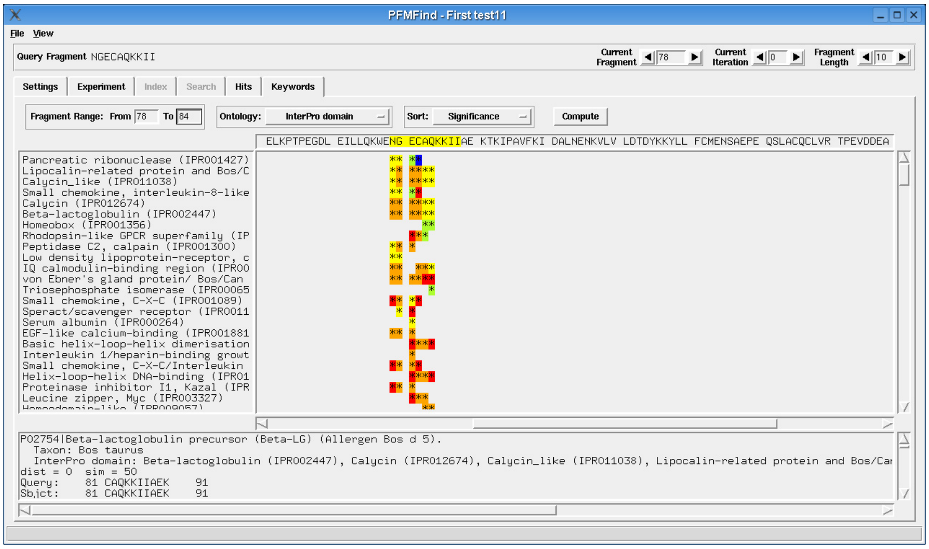
Henikoff and Henikoff [5] to assign weights to fragments and Dirichlet mixtures [9] for regularising the amino acid frequency counts at each position. Users with some knowledge of Python can create their own plugins and use them for searches by placing then in the appropriate directory.

Search criteria can be specified according to threshold raw similarity scores, distances, p-values and E-values (range search), as well as the number of closest data points to retrieve (kNN search). The probability model for calculation of p-values assumes that the score of each fragment is the sum of independent random variables corresponding to the score at each position and the score distribution is calculated using discrete Fast Fourier Transform.

2.2 FSIndex

The heart of PFMFind is FSIndex, an efficient indexing scheme for similarity search of very large datasets of short protein fragments of fixed length [8,10]. FSIndex is based on two principles: reduction of the amino acid alphabet to clusters largely based on their biochemical properties (hydrophobic, polar, charged, aromatic ...) and combinatorial generation of neighbours. The design of FSIndex means that a typical search involves scanning less than 1% of the fragment dataset, but ensures that no neighbours satisfying search criteria are ever missed.

FSIndex is implemented in the C programming language and embedded into Python, with the whole data structure as well as the indexed sequences stored in primary memory. For even greater efficiency, computation of searches can be distributed among several machines using a master/slave model: the master handles p-value computations, distributes queries to slaves, each of which is indexing a different part of the dataset, and communicates with the client.



```
Query Fragment NGECAQKKII          Current Fragment 78   Current Iteration 0   Fragment Length 10

Fragment Range: From 78 To 84    Ontology: InterPro domain    Sort: Significance

ELKPTPEGDL EILLQKWENG ECAQKKIIAE KTKIPAVFKI DALNENKVLV LDTDYKKYLL FCMENSAEPE QSLACQCLVR TPEVDDEA

Pancreatic ribonuclease (IPR001427)
Lipocalin-related protein and Bos/C
Calycin_like (IPR011038)
Small chemokine, interleukin-8-like
Calycin (IPR012674)
Beta-lactoglobulin (IPR002447)
Homeobox (IPR001356)
Rhodopsin-like GPCR superfamily (IP
Peptidase C2, calpain (IPR001300)
Low density lipoprotein-receptor, c
IQ calmodulin-binding region (IPR00
von Ebner's gland protein/ Bos/Can
Triosephosphate isomerase (IPR00065
Small chemokine, C-X-C (IPR001089)
Speract/scavenger receptor (IPR0011
Serum albumin (IPR000264)
EGF-like calcium-binding (IPR001881
Basic helix-loop-helix dimerisation
Interleukin 1/heparin-binding growt
Small chemokine, C-X-C/Interleukin
Helix-loop-helix DNA-binding (IPR01
Proteinase inhibitor I1, Kazal (IPR
Leucine zipper, Myc (IPR003327)

P02754|Beta-lactoglobulin precursor (Beta-LG) (Allergen Bos d 5).
  Taxon: Bos taurus
  InterPro domain: Beta-lactoglobulin (IPR002447), Calycin (IPR012674), Calycin_like (IPR011038), Lipocalin-related protein and Bos/Car
dist = 0   sim = 50
Query:    81 CAQKKIIAEK    91
Sbjct:    81 CAQKKIIAEK    91
```

Fig. 2. A screenshot of PFMFind GUI showing search results associated with their annotations

2.3 Database

The second major component of PFMFind is a relational database, used both for storage of search results and the sequence annotation. We use PostgreSQL, a freely available modern database management system.

Each user of the system has their own schema for storing search results. The database also stores all search parameters, including PSSMs and the results of each iteration, facilitating reversion to a previous iteration without repeating the whole procedure.

The database stores sequence annotations in a standard BioSQL schema available to all users. PFMFind also contains scripts for loading four types of information beyond the protein sequence: Uniprot [2] keywords and features, Uniref clusters [2] and InterPro [7] domains. When retrieved for display, annotations are joined to search results through accession numbers.

2.4 GUI Client

The final PFMFind component is a GUI client that connects to both the FSIndex master and the database component. To perform fragment searches, the user specifies a query sequence, usually a long sequence that is broken into overlapping fragments of fixed length, and chooses the fragment lengths, threshold parameters and the actual fragments in the query sequences that will be used for the search.

The GUI client can display search results both as lists of hits associated with a particular location in the query sequence and as a feature vs location dot plot –

each location matching a particular feature is marked by a coloured dot (Fig. 2). Dots are colour coded by the number of hits matching the feature to distinguish frequently represented features from those that appear only a few times in the hit list. The GUI client also performs all computations for constructing PSSMs.

3 Conclusion

PFMFind is an efficient, flexible, and extensible framework for similarity search of datasets of short peptide fragments. It supports fast similarity search with selectivity and sensitivity specified by PSSMs and associates search results with biological function by using sequence features and annotations.

Acknowledgements. We wish to thank Pavle Mogin and Danyl McLauchlan for their help with PostgreSQL and testing the software, respectively. A.S. was supported by a Bright Future PhD scholarship awarded by the NZ Tertiary Education Commission jointly with the Fonterra Research Centre, by a Fields Institute/University of Ottawa postdoctoral fellowship, and by the Intramural Research Program of the National Library of Medicine at the National Institutes of Health. V.G.P. and A.S. acknowledge support from NSERC discovery grant program and University of Ottawa internal grants.

References

1. Altschul, S.F., Madden, T.L., Schaffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J.: Gapped BLAST and PSI–BLAST: a new generation of protein database search programs. Nucleic Acids Res. 25, 3389–3402 (1997)
2. Bairoch, A., Apweiler, R., Wu, C.H., Barker, W.C., Boeckmann, B., Ferro, S., Gasteiger, E., Huang, H., Lopez, R., Magrane, M., Martin, M.J., Natale, D.A., O'Donovan, C., Redaschi, N., Yeh, L.S.L.: The Universal Protein Resource (UniProt). Nucleic Acids Res 33 Database Issue, 154–159 (2005)
3. Dayhoff, M.O., Schwartz, R.M., Orcutt, B.C.: A model of evolutionary change in proteins. In: Dayhoff, M.O. (ed.) Atlas of Protein Sequence and Structure, vol. 5, ch.22, pp. 345–352. National Biomedical Research Foundation (1978)
4. Gribskov, M., McLachlan, A.D., Eisenberg, D.: Profile analysis: detection of distantly related proteins. Proc Natl. Acad. Sci. USA 84, 4355–4358 (1987)
5. Henikoff, S., Henikoff, J.G.: Position-based sequence weights. J. Mol. Biol. 243(4), 574–578 (1994)
6. Henikoff, S., Henikoff, J.G.: Amino acid substitution matrices from protein blocks. Proc. Natl. Acad. Sci. USA 89, 10915–10919 (1992)
7. Hunter, S., Jones, P., Mitchell, A., Apweiler, R., Attwood, T.K., Bateman, A., Bernard, T., Binns, D., Bork, P., Burge, S., de Castro, E., Coggill, P., Corbett, M., Das, U., Daugherty, L., Duquenne, L., Finn, R.D., Fraser, M., Gough, J., Haft, D., Hulo, N., Kahn, D., Kelly, E., Letunic, I., Lonsdale, D., Lopez, R., Madera, M., Maslen, J., McAnulla, C., McDowall, J., McMenamin, C., Mi, H., Mutowo-Muellenet, P., Mulder, N., Natale, D., Orengo, C., Pesseat, S., Punta, M., Quinn, A.F., Rivoire, C., Sangrador-Vegas, A., Selengut, J.D., Sigrist, C.J.A., Scheremetjew, M., Tate, J., Thimmajanarthanan, M., Thomas, P.D., Wu, C.H., Yeats, C., Yong, S.Y.: Interpro in 2011: New developments in the family and domain prediction database. Nucleic Acids Res. 40(Database issue), D306–D312 (2012)

8. Pestov, V., Stojmirović, A.: Indexing schemes for similarity search: an illustrated paradigm. Fundam. Inform. 70(4), 367–385 (2006)
9. Sjölander, K., Karplus, K., Brown, M., Hughey, R., Krogh, A., Mian, I., Haussler, D.: Dirichlet mixtures: A method for improving detection of weak but significant protein sequence homology. Comput. Appl. Biosci. 12(4), 327–345 (1996)
10. Stojmirović, A., Pestov, V.: Indexing schemes for similarity search in datasets of short protein fragments. Inf. Syst. 32(8), 1145–1165 (2007)
11. Watt, T.J., Doyle, D.F.: ESPSearch: a program for finding exact sequences and patterns in DNA, RNA, or protein. Biotechniques 38(1), 109–115 (2005)

Retrieving Similar Movements in Motion Capture Data

Jan Sedmidubsky and Jakub Valcik

Masaryk University, Botanicka 68a, 602 00 Brno, Czech Republic

Abstract. Efficient and effective similarity retrieval in large human motion databases is valuable in many fields such as computer animation, security research, sports, and medicine. This demonstration paper presents a content-based retrieval system that is able to examine database motions and locate all their sub-motions similar to a query motion example. The proposed system does not require any textual annotations nor explicit knowledge of the data and can deal with spatio-temporal variances of individual human motions. The system is presented as an online web application indexing a real-life 68-minute human motion database.

1 Introduction

The development of motion capturing technologies (e.g., Microsoft Kinect) has caused an explosion in the usage of human motion data in different fields. For example, motion data are analyzed in sports to compare performance aspects of athletes, in security research to identify special-interest persons, in health care to determine the success of rehabilitative treatments, and in computer animation to synthesize realistic motions. In particular, production of high-quality computer games and animations requires an expensive and time-consuming synthesis of motions performed by specialized actors. Therefore, there is a rising need to reuse the recorded data, which makes animation and game production more efficient. One way is to manually or automatically annotate the motion data by textual descriptions [4]. Although it is very efficient in text retrieval, textual descriptions cannot always sufficiently express desired movements and limit users to search for only certain classes of movements. To overcome this limitation, content-based search techniques are used to retrieve motions that are similar to a query motion example. Although some existing content-based retrieval systems [1] focus on matching of the whole motions only, we especially concentrate on a much harder task of *sub-motion* retrieval. A lot of research on subsequence retrieval of general time series has been already done [2].

The main contribution of this paper is to demonstrate capabilities of our content-based subsequence retrieval algorithm proposed in [8]. This algorithm (see Section 3) examines database motions and identifies all their sub-motions that are similar to a query. To retrieve sub-motions effectively and efficiently, motion features in form of joint-angle rotations (see Section 2) are extracted and indexed. Indexing this kind of features does not require any user intervention in contrast to the approach in [5] that needs to manually tune each query in order to select a subset of features describing important aspects of the given query activity. In contrast to [4], queries in our system are specified by a single motion only without necessity of having several positive examples. In contrast to [7,9], we present the retrieval effectiveness and efficiency and make the whole system available as an online web application (see Section 4).

2 Pose Feature Extraction

Motion capture data can be seen as a sequence of frames that consist of 3D coordinates describing the positions of specific joints of human skeletons at a given time. We process the 3D joint coordinates to extract motion features in form of *joint-angle rotations*. A joint-angle rotation constitutes a discrete time series expressing how the angle related to the specific joint changes in time. Individual angles are computed independently of the observer view plane – they are computed in planes defined relatively to the examined joint and relevant joints in its surrounding (see [8] for more information).

We adopt the skeletal model in Figure 1a to extract joint-angle rotations of 18 important joints that significantly influence the human motion. In particular, we process each frame to extract (1) a single angle for 8 joints (left and right wrist, left and right elbow, left and right knee, and left and right ankle) moving in one plane and (2) two angles for 10 joints (left and right shoulder, head, upperneck, lowerneck, thorax, upperback, lowerback, and left and right hip) that move in two planes (see Figure 1b,c). In total, 28 angles are computed in each frame. Then we can define a database motion D of K frames as a sequence $D = (P_1^D, P_2^D, \ldots, P_K^D)$ of *poses* $P_i^D (i \in [1, K])$, where each pose P_i^D constitutes a 28-dimensional vector of specific angles captured in the i-th frame.

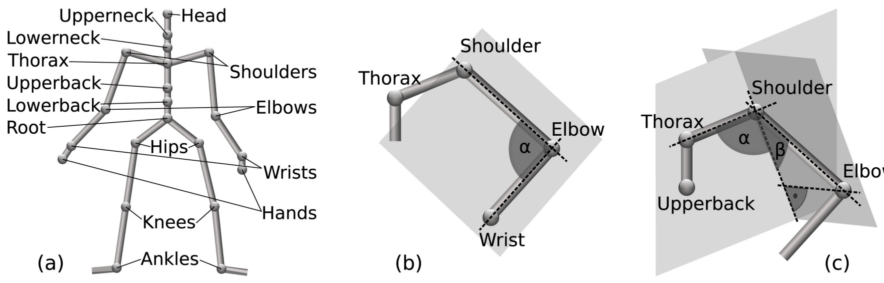


Fig. 1. (a) Skeletal model. (b) Elbow angle (α). (c) Shoulder angles (α, β).

3 Key-Pose Similarity Retrieval Algorithm

Let $D = (P_1^D, P_2^D, \ldots, P_K^D)$ and $Q = (P_1^Q, \ldots, P_M^Q)$ be sequences of poses representing a *database* motion and *query* motion, respectively. The objective of the key-pose algorithm is to explore each database motion D and retrieve all its sub-motions $(P_s^D, \ldots, P_t^D), 1 \leq s < t \leq K$, that are similar to the query motion Q. The retrieval process (Section 3.2) is based on efficient location of database poses that are similar to specific query poses (Section 3.1).

3.1 Indexing

We index database motions to be able to efficiently retrieve the set of database poses that are similar to a query pose. We define similarity $sim(P_i^Q, P_j^D)$ between the query pose $P_i^Q (i \in [1, M])$ and database pose $P_j^D (j \in [1, K])$ as the Manhattan distance: $sim(P_i^Q, P_j^D) = \sum_{k=1}^{28}(|P_i^Q[k] - P_j^D[k]|)$, where $P_i^Q[k]$ and $P_j^D[k]$ stand for the k-th angle extracted in the i-th query and j-th database frame, respectively. To enable spatial variances, two poses P_i^Q and P_j^D are considered as *similar* if their similarity is up to a fixed threshold τ, i.e., $sim(P_i^Q, P_j^D) \leq \tau$.

There are a lot of existing metric-based and vector-space-based index structures that can be utilized to index database poses. We chose the *Metric-Index* (M-Index) [6] that supports execution of precise as well as approximate range and k-nearest-neighbor queries. In particular, we used the range query to retrieve query-similar database poses up to the threshold $\tau = 150$. We have experimentally observed that M-Index needed to access roughly 6 % of indexed 28-dimensional poses to answer a single range query on average. In the future, we plan to utilize approximate queries to be able to index large database motions.

3.2 Retrieval

The retrieval algorithm locates all sub-motion *candidates* (within each database motion D) that could be similar to the query motion Q. The detailed algorithm description is available in [8]. The algorithm principally works as follows:

1. The query is processed to determine *query key poses* $P_{q_1}^Q, P_{q_2}^Q, \ldots, P_{q_N}^Q$. These key poses are simply selected as each δ-th query pose, starting from the first one. The parameter $\delta \in \mathbb{N}$ determines the fixed spacing between two consecutive query key poses, where $N = \lfloor \frac{M}{\delta} \rfloor$ and $q_i = (i-1) \cdot \delta + 1$;
2. For each query key pose $P_{q_i}^Q$, M-Index is used to retrieve the set of similar database poses $\{P_j^D \mid sim(P_{q_i}^Q, P_j^D) \leq 150, j \in [1, K]\}$. To be further independent of the τ threshold, we could retrieve only k-nearest neighbors;
3. A single sub-motion candidate is constructed for each pose retrieved in the previous step by picking the retrieved pose and its carefully-selected neighboring poses. In particular, the neighboring poses are scanned to identify the *candidate key poses* $P_{r_1}^D, P_{r_2}^D, \ldots, P_{r_N}^D$ that are similar to the query key poses. To enable temporal variances of retrieved sub-motions, the distance (in frames) of two consecutive candidate key poses $P_{r_i}^D$ and $P_{r_{i+1}}^D (i \in [1, N-1])$ is not fixed to δ frames but is kept within the tolerated time bound $\left[\delta \cdot \frac{1}{\sigma}, \delta \cdot \sigma\right]$, where the *stiffness* parameter $\sigma \in (0, 1]$ controls the degree of temporal deformations. The sub-motion candidate is then represented by database poses occurring between the first $P_{r_1}^D$ and last $P_{r_N}^D$ identified key pose;

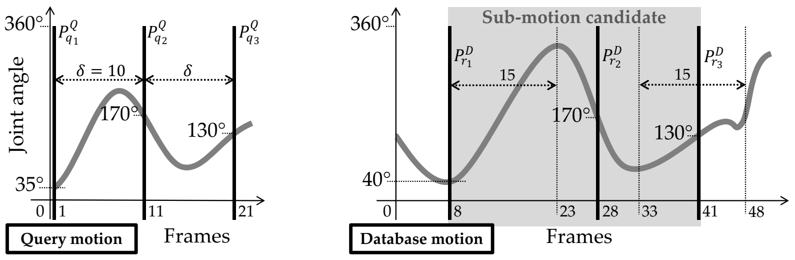


Fig. 2. Schematic process of retrieval of a sub-motion candidate that is similar to the query motion represented by three query key poses $P^Q_{q_1}, P^Q_{q_2}, P^Q_{q_3}$ (for better clarity, only a single joint-angle rotation is shown, instead of illustrating all 28 dimensions).

4. The similarity between the query motion and each constructed candidate is computed as the average similarity value of corresponding pairs of key poses: $\frac{1}{N} \cdot \sum_{i=1}^{N} sim(P^Q_{q_i}, P^D_{r_i})$. Since we want to ignore motion variances between consecutive key poses, we don't use sophisticated similarity measures such as DTW or EDR. Finally, all the candidates are ranked according to their similarity and less-similar overlapping ones are ignored.

On the graphically illustrated example in Figure 2, we explain how to retrieve a sub-motion that is similar to the query Q of length 25 ($M = 25$). By fixing the parameter $\delta = 10$, the three query key poses $P^Q_{q_1}, P^Q_{q_2}$, and $P^Q_{q_3}$ are determined, where $q_1 = 1, q_2 = 11$, and $q_3 = 21$. For each database pose P^D_j that M-Index retrieved as similar to the i-th query key pose $P^Q_{q_i}$, a single sub-motion candidate is constructed by locating its key poses $P^D_{r_1}, P^D_{r_2}$, and $P^D_{r_3}$. The i-th candidate key pose $P^D_{r_i}$ is directly represented by the j-th retrieved database pose, i.e., $r_i = j$. For instance, assume that the 28-th database pose P^D_{28} was retrieved as similar to the 2-nd query key pose $P^Q_{q_2} \Rightarrow r_2 = 28$. Further, we set the stiffness parameter $\sigma = 0.5$ to allow sub-motion candidates to be maximally two times shorter or longer with respect to the query length. Both the parameters $\sigma = 0.5$ and $\delta = 10$ force the consecutive candidate key poses to be distant from 5 to 20 frames. To determine the first key pose $P^D_{r_1}$, the database poses $P^D_{r_2-20}, \dots, P^D_{r_2-5}$ (i.e., $P^D_8, \dots, P^D_{23}$) are sequentially scanned to find the most similar database pose to the query key pose $P^Q_{q_1}$. The most similar pose then corresponds to the desired pose $P^D_{r_1}$. The third key pose $P^D_{r_3}$ is determined as the most similar pose to the query key pose $P^Q_{q_3}$ within $P^D_{r_2+5}, \dots, P^D_{r_2+20}$ (i.e., $P^D_{33}, \dots, P^D_{48}$).

4 Online Demonstration

The functionality of the retrieval system is demonstrated by an online web application (see Section 4.1). The system indexes 491,847 poses extracted from the 68-minute motion capture database HDM05 [3] (the database is logically divided into 102 motion parts). We used the same ground truth as the authors of [4] to evaluate retrieval effectiveness and efficiency (see Section 4.2).

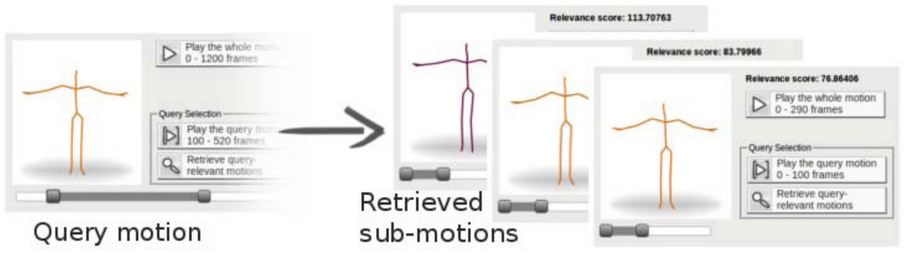


Fig. 3. Screenshot of the web application illustrating retrieved sub-motions

4.1 Web Application

The system is available at `http://mufin.fi.muni.cz/motion-retrieval/`. It allows users to specify a query motion from randomly chosen database motions and display the retrieved query results. This functionality is described in the following two paragraphs.

Random Motions & Query Selection By clicking the *Random selection* button, five randomly chosen sub-motions are selected from the motion database and presented to the user. Each such motion is represented by a sequence of poses that are drawn on the canvas in form of human stick figures. Stick figures originating from the same database motion have the same color. The displayed stick figure is drawn for a current frame that can be changed by moving a range slider located under the canvas (the slider length corresponds to the motion length). The slider also contains two handlers that can be used to define a query-motion example. By moving both the handlers, the start and end query frame is set. In addition, each motion is associated with two play buttons – *Play the whole motion* and *Play the query motion* – that automatically draw stick figures of consecutive poses on the canvas with a fixed sampling rate in order to show the human motion in real time. To obtain a new batch of random motions, the user can reload the page or click the *Random selection* button.

Query Results After selecting a query motion and clicking the *Retrieve query-relevant motions* button, the key-pose retrieval algorithm is executed. The retrieved query-similar sub-motions are then displayed on a web page and ordered from the left to right according to their similarity score. The number of the retrieved sub-motions is limited to the 50 most similar ones. A new query can also be defined by selecting a query motion from the retrieved results. The screenshot in Figure 3 illustrates the query with three retrieved sub-motions.

4.2 Effectiveness and Efficiency Evaluation

To evaluate search effectiveness objectively, we compared retrieved results of our system against predefined ground truth in order to measure *precision* and *recall* on the frame-level approach [4]. The recall and precision were evaluated for 1476 ground-truth queries divided to 15 classes of motions. The setting $\delta = 10, \sigma = 0.5$

and $\tau = 150$ achieved the 57 % precision and 75 % recall per query on average. When $\tau = 100$, the average 80 % precision and 46 % recall was obtained. These results approached the annotation standard stated in [4]. Remark that it is much harder to achieve very high values of recall and precision on the whole database in contrast to the top-k results, which was considered in [1].

Efficiency was evaluated by analyzing the number of similarity comparisons (the $sim()$ function calls) needed to answer a query. The system needed a sub-linear number of $432,352$ function calls to evaluate a single ground-truth query on average with respect to the total number of $491,847$ frames of all database motions. The average query took about 1.5 s to be evaluated. A more detailed evaluation of both the effectiveness and efficiency is available in [8].

5 Conclusions

We introduced a content-based subsequence retrieval system that is able to locate similar sub-motions with spatio-temporal variances. The retrieval effectiveness achieved the annotation standard stated in [4]. From the efficiency point of view, the system needed about 1.5 s to evaluate a single query on average.

Acknowledgements. This research was supported by the national project GAP103/12/G084.

References

1. Choensawat, W., Choi, W., Hachimura, K.: Similarity Retrieval of Motion Capture Data Based on Derivative Features. Journal of Advanced Computational Intelligence and Intelligent Informatics 16(1), 13–23 (2012)
2. Esling, P., Agon, C.: Time-Series Data Mining. ACM Computing Surveys 45(1), 12:1–12:34 (2012)
3. Müller, M., Röder, T., Clausen, M., Eberhardt, B., Krüger, B., Weber, A.: Documentation Mocap Database HDM05. Tech. Rep. CG-2007-2, Universität Bonn (2007)
4. Müller, M., Baak, A., Seidel, H.-P.: Efficient and Robust Annotation of Motion Capture Data. In: ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA 2009), p. 10. ACM Press, USA (2009)
5. Müller, M., Röder, T., Clausen, M.: Efficient Content-based Retrieval of Motion Capture Data. In: SIGGRAPH, pp. 677–685. ACM Press, USA (2005)
6. Novak, D., Batko, M., Zezula, P.: Metric Index: An Efficient and Scalable Solution for Precise and Approximate Similarity Search. Inf. Sys. 36(4), 721–733 (2011)
7. Ren, C., Lei, X., Zhang, G.: Motion Data Retrieval from Very Large Motion Databases. In: International Conference on Virtual Reality and Visualization (ICVRV 2011), pp. 70–77 (2011)
8. Sedmidubsky, J., Valcik, J., Zezula, P.: A Key-Pose Similarity Algorithm for Motion Data Retrieval. In: 12th International Conference on Advanced Concepts for Intelligent Vision Systems (ACIVS 2013), p. 12. Springer, Heidelberg (2013)
9. Wang, P., Lau, R.W., Zhang, M., Wang, J., Song, H., Pan, Z.: A real-time database architecture for motion capture data. In: 19th International Conference on Multimedia (MM 2011), pp. 1337–1340. ACM, USA (2011)

Author Index

Alegre, Enrique 141, 151
Amato, Giuseppe 91, 245
Andreae, Peter 319
Angiulli, Fabrizio 85
Arimura, Hiroki 73

Baeza-Yates, Ricardo 1
Bartoš, Tomáš 294
Benson, Gary 257
Boland, Mike 319
Bolettieri, Paolo 245
Boytsov, Leonid 280
Braithwaite, Billy 127

Cardillo, Franco Alberto 163
Chávez, Edgar 115, 300
Chen, Lei 43
Cheney, Jason A. 266
Chester, Uzi 59
Chum, Ondřej 3
Connor, Richard 163, 169
Cordeiro, Robson L.F. 30
Cucchiara, Rita 233

Duan, Hubert Haoyang 182

Emrich, Tobias 43, 175
Esuli, Andrea 91

Falchi, Fabrizio 91, 245
Fassetti, Fabio 85
Fernández-Robles, Laura 151
Figueroa, Karina 50, 115
Fredriksson, Kimmo 127

García-Olalla, Oscar 141, 151
García-Ordás, Diego 141
García-Ordás, María Teresa 141, 151
Gennaro, Claudio 245
Grana, Costantino 233
Grošup, Tomáš 312

Hirata, Kouichi 73
Houle, Michael E. 16
Hurson, Ali R. 219

Jordan, Thomas William 319
Jossé, Gregor 175

Kriegel, Hans-Peter 43, 175
Kröger, Peer 213

Leksin, Vasily A. 206
Levy, Avivit 257
Lokoč, Jakub 312

Mamoulis, Nikos 43
Manfredi, Marco 233
Marchand-Maillet, Stéphane 103
Matas, Jiří 3
Mauder, Markus 175, 213
Mikulik, Andrej 3
Miranda, Natalia 300
Mohamed, Hisham 103
Moss, Robert 163, 169

Naidan, Bilegsaikhan 280
Nett, Michael 16
Niedermayer, Johannes 43, 175
Nikolenko, Sergey I. 206

Paredes, Rodrigo 50
Pearson, Spencer S. 266
Pestov, Vladimir G. 182, 319
Piccoli, María Fabiana 300
Pola, Ives R.V. 30

Rabitti, Fausto 163
Ratsaby, Joel 59
Renz, Matthias 43, 175
Reyes, Nora 300
Ruiz, Guillermo 115

Santini, Simone 194
Santoyo, Francisco 115
Schinner, Karl-Ludwig 213
Schubert, Matthias 175
Sedigh, Sahra 219
Sedmidubsky, Jan 325
Serra, Giuseppe 233
Shalom, B. Riva 257
Silva, Yasin N. 266
Singla, Varun 182

Skopal, Tomáš 294, 312
Stojmirović, Aleksandar 319

Tellez, Eric Sadit 115
Traina, Agma J.M. 30
Traina Jr., Caetano 30

Uno, Takeaki 73

Valcik, Jakub 325

Wagner, Christopher Shaun 219
Wasa, Kunihiro 73

Züfle, Andreas 43, 175