Towards Artificial Priority Queues for Similarity Query Execution

Matej Antol #1, Vlastislav Dohnal *2

Faculty of Informatics, Masaryk University
Botanicka 68a, Brno, Czech republic
1 xantol@fi.muni.cz
2 dohnal@fi.muni.cz

***Abstract*—Content-based retrieval in large collections of unstructured data is challenging not only from the difficulty of the defining similarity between data images where the phenomenon of semantic gap appears, but also the efficiency of execution of similarity queries. Search engines providing similarity search typically organize various multimedia data, e.g. images of a photo stock, and support k-nearest neighbor query. Users accessing such systems then look for data items similar to their specific query object and refine results by re-running the search with an object from the previous query results. This paper is motivated by unsatisfactory query execution performance of indexing structures that use metric space as a convenient data model. We present performance behavior of two state-of-the-art representatives and propose a new universal technique for ordering priority queue of data partitions to be accessed during kNN query evaluation. We verify it in experiments on real-life data-sets.**

I. Introduction

Content-based retrieval systems have become often applied to complement traditional retrieval systems. For example, photo stocks then provide a user with visually similar images to a given one. If he or she is not satisfied with the result, they may browse the database by issuing new query by clicking on a previously returned image. This procedure exhibits the property that a non-negligible amount of queries processed by the system are alike. Query evaluation is generally supported by an indexing structure, so the number of disk I/Os needed to answer a query is greatly reduced. Handling complex and unstructured data requires extracting descriptors from data objects which are then organized and queried. The descriptors typically form high-dimensional spaces or even distance spaces where no implicit coordinate system is defined [1]. The problem of *dimensionality curse* then often appears [2]. This typically leads to visiting many data partitions by an index due to frequent overlaps among them, whereas the useful information is obtained from few of them. Efficiency is then improved by further filtering constraints and optimized node-splitting strategies in the indexing structures [3], [4] or by sacrificing precision in query results (approximate querying) [5], [6], [7].

In this paper, we survey possibilities of describing data partitions that can be used to prioritize their accesses. We respect the structure of the original index, so it does not introduce any space overhead. We combine multiple descriptions into a priority value that is used to organize a queue of data partitions to be accessed. This forms the main contribution of the paper.

The paper is structured as follows. In the next two sections, related work is surveyed and indexing and querying principles are concisely presented. Analysis of accessing data by current indexes is presented in Section IV. The proposal of organizing priority queues is described in Section V and its performance is presented in Section VI. Contributions of this paper and possible future extensions are summarized the last section.

II. Related work

There is a vast of indexing structures that use metric space as data model and they are surveyed in [8], [9]. Processing large data-sets needs disk-oriented indexing structures. In general, principles of organizing data in an index are based on (i) hierarchical clustering (e.g. M-tree [4]), where each subtree is covered by a preselected data object (pivot) and a covering radius; (ii) Voronoi partitioning (e.g. M-index [10], PPP-Codes [11]), where subtrees are formed by assigning objects to the closest pivot recursively; and (iii) precomputed distances (e.g. Linear AESA [12]), where no explicit structure is built, but rather distances among data objects are stored in a matrix. To improve efficiency, these concepts are often combined in one system.

Optimizations of query-evaluation algorithms are based on either extending a hierarchical structure with additional precomputed distances to tighten filtering, e.g. M*-tree [13], cutting local pivots [14]; or exploiting a large number of pivots in a very compact and reusable way, e.g. permutation prefix index [15]. The order of accessing data partitions is defined as an estimate of lower bound on distance to the data partition. The primal aim is to constrain the data partitions as much as possible.

An alternative solution to performance insufficiencies is to trade precision, i.e. approximate searching. Early-termination or relaxed-branching are the strategies to implement approximate searching. A recent approach called spatial approximation sample hierarchy [6] builds an approximated near-neighbor graph that does not rely on the triangle inequality, so it allows organizing semi-metric spaces. We note that the triangle inequality as a property of metric space is largely used to filter out irrelevant data partitions/objects. It was later improved and combined with cover trees to design Rank Cover Tree [7].

A special technique for storing precomputed distances dynamically is Distance-Cache [16]. It is a main-memory structure that collects information from previous querying and caches some distances to form tighter lower- and upper-bounds on distances between newly arriving queries and database objects. In this respect, it is applicable to any metric indexing structure.

Processing a bunch of queries simultaneously is another research topic. Snake table [17] is a dynamically-built structure for optimizing all queries corresponding to one user session. It constructs a temporary linear AESA to index all queries processed so far. Evaluation of next queries is more efficient then. A pure caching strategy of similarity queries was proposed in [18]. On the other hand, a combination of query cache and index structure was proposed in [19]. Its idea lies in reusing as much work spent in scanning the cache as possible in traversing an index structure. This is certainly advantageous if the query is not present in the cache. In principle, the list of clusters technique [20] is built using the query objects only, which is analogous to the Snake table. Recently, an approximation technique that combines the probability of an indexed object to be part of a query result and the time needed to examine the object, was proposed [21]. The time to examine an object depends on its presence in regular OS memory cache. Last, the Inverted Cache Index that was proposed in [22]. It takes into account previously processed queries. However, such information is exploited to increment so-called "popularity" of data partitions.

In the following two sections, we introduce the reader to the principles and experience with metric indexing structures.

III. Indexing and Querying Metric Spaces

We assume data is modeled in a metric space and indexing techniques are applied to make the search efficient. To make it possible, properties of metric space are exploited.

A. Metric space and Similarity Queries

A metric space $\mathcal{M}$ is defined as a pair $(\mathcal{D}, d)$ of a domain $\mathcal{D}$ representing data objects and a pair-wise distance function $d : \mathcal{D} \times \mathcal{D} \mapsto \mathbb{R}$ that satisfies:

$$\begin{array}{ll} \forall x, y \in \mathcal{D}, d(x,y) \geq 0 & \text{non-negavity,} \\ \forall x, y \in \mathcal{D}, d(x,y) = d(y,x) & \text{symmetry,} \\ \forall x, y \in \mathcal{D}, x = y \Leftrightarrow d(x,y) = 0 & \text{identity, and} \\ \forall x, y, z \in \mathcal{D}, d(x,z) \leq d(x,y) + d(y,z) & \text{triangle ineq.} \end{array}$$

The distance function measures similarity between two objects. The shorter the distance is, the more similar the objects are and vice versa. Consequently, a similarity query can be defined. There are many types of similarity queries [23], e.g. range query or join, but the k-nearest-neighbors query is prevalently used. The merit of k-nearest-neighbors query is in retrieving k most similar objects to the query object q without any particular knowledge about the distance function d, so such a query is very convenient for users. Formally, $kNN(q) = A$, where $|A| = k \wedge \forall o \in A, p \in X - A, d(q,o) \leq d(q,p)$, where $X \subset \mathcal{D}$ is the database we search in. In this paper, we primarily focus on k-nearest-neighbors query.

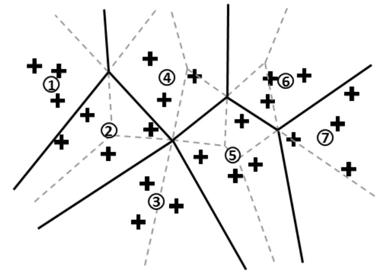


Fig. 1: Partitioning principle of M-index – 2 levels and 20 resulting leaf nodes.

B. Indexing and Query Evaluation

To organize a database to answer similarity queries efficiently, many indexing structures have been proposed [8]. Their principles are twofold: (i) recursive application of data partitioning/clustering defined by preselected data objects called *pivots* and respective distance thresholds, and (ii) an effective object filtering using lower-bounds on distance between a database object and a query object or a kind of approximation of such bounds. These principles have been firstly surveyed in [9].

In this paper, we use the Pivoting M-tree [24], a variant of the traditional index M-tree [4] and a recent technique M-index [10] with some improvements of PPP-Codes [11]. Both these structures create an internal hierarchy of nodes that partition the data space into buckets – an elementary object storage. Pivoting M-tree organizes data objects in compact clusters created in the bottom-up fashion, where each cluster is represented by a pair (p, r^c) – a pivot and a covering radius, i.e. distance from the pivot to the farthest object in the cluster. It extends M-tree by applying further filtering using a fixed independent set of pivots. On the other hand, M-index applies Voronoi-like partitioning using a predefined set of pivots in the top-down way. On the first level, clusters are formed by objects that have the cluster's pivot as the closest one. On subsequent levels, the objects are reclustered using the other pivots, i.e. eliminating the pivot(s) that formed the current cluster. To make the structure compact, only a prefix of closest pivots is used to denote a leaf node. Buckets of both the structures store objects of leaf nodes, as is exampled in the illustration in Fig. 1. So we use the terms *leaf node* and *bucket* interchangeably. Apart this, we refer to Pivoting M-tree as M-tree.

A kNN-query evaluation algorithm constructs a *priority queue* of nodes to access, and gradually updates a set of candidate objects forming the final query answer when the algorithm finishes. The priority is defined in terms of a lower bound on distance between the node and the query object. So a probability of node to contain relevant data objects is estimated this way. In detail, the algorithm starts with inserting the root node of hierarchy. Then it repeatedly pulls the head of priority queue until the queue is empty. The algorithm terminates immediately, when the pulled head's lower bound is greater than the distance of current k^{th} neighbor to the query object. If the pulled element represents a leaf node, its corresponding bucket is accessed and all data objects stored there are checked against the query, so query's answer is updated, i.e. the distance to k^{th} neigbor is shrunk. If it is a non-leaf node, all its children are inserted into the queue with estimates of lower-bounds. M-tree defines the lower bound for a node (p, r^c) and a query object q as the distance $d(q,p) - r^c$, where r^c is the covering radius forming a ball around the pivot p. For further details, we refer the reader to the cited papers where additional M-tree's node filtering principles as well as the M-index's approach to evaluate the query are described.

IV. Analysis of Query Evaluation Performance

Interactivity of similarity queries is the main driving force to make content-based information retrieval widely used [25]. Near real-time execution of similarity queries over massive data collections is even more important, because it allows various analytic tasks to be implemented [26]. In this section, we present arguments based on experience with a real-life content-based image retrieval system.

A. Query Statistics

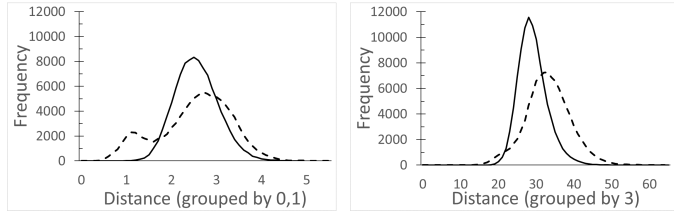
We gathered statistics of querying from two demonstration applications[1]. The first one organizes the CoPhIR data-set [27] consisting of 100 million images characterized by global MPEG-7 visual descriptors. Whereas the second application organizes the Profiset collection [28] that consists of 20 million high-quality images with rich and systematic annotations using descriptors from deep convolutional neural networks (Caffe descriptors) [29].

Fig. 2 shows density of distances among the individual top-1000 queries that were executed during the applications' life time. Reader may observe there are very similar query objects as well as distinct ones in CoPhIR. This proves that the users were also browsing the data collection as was described above. This phenomenon is almost negligible in Profiset, i.e. some similar queries are present but not many. For reference, we also include distance density of the whole data-sets depicted as solid curves in Fig. 2. To sum up, these two query sets form different conditions for our proposal to cope with.

B. Query Evaluation Performance

The major drawback of indexing structures in metric spaces is the high amount of overlaps among index substructures (data partitions), which is also supported by not very tight estimation of lower bounds on distances between data objects and a query object. So the kNN-query evaluation algorithm often accesses a very large portion of buckets to obtain the precise answer to a query.

[1] http://disa.fi.muni.cz/prototype-applications/image-search/



(a) CoPhIR (b) Profiset

Fig. 2: Density of distances among top-1000 query objects (dashed curve) and overall density of the data-set (solid curve).

Data-set	bucket cap. (objs)	total bckts	avg bckt occup.	hier. height	int. node cap.	avg. bckts visited by 30NN
Indexing structure: M-tree						
CoPhIR	200	11,571	43.2%	4	50	1775
CoPhIR	2,000	1,124	44.5%	3	100	227
Profiset	2,000	2,634	19.0%	3	100	1472
Indexing structure: M-index						
CoPhIR	200	62,049	8.1%	8	N/D	2795
CoPhIR	2,000	10,943	4.6%	8	N/D	782
Profiset	2,000	20,222	2.5%	6	N/D	5948

TABLE I: Details on indexing structures and statistics about accessed buckets during querying for 30NN with top-1000 query objects.

The selected indexing structure representatives were populated with 1 million data objects from the CoPhIR data-set and 30NN queries for the top-1000 query objects were evaluated. We have tested two configurations for both M-tree and M-index. In particular, the capacity of buckets was limited to 200 and 2,000 objects with intention of having bushier and more compact structures. Table I summarizes information about them. To this end, M-index's building algorithm was initialized with 128 and 512 pivots picked at random from the data-set and the maximum depth of M-index's internal hierarchy was limited to 8 and 6 for CoPhIR and Profiset, respectively. From the statistics, we can see that M-tree can adapt to data distribution better than M-index and does not create low occupied buckets, so M-tree is a more compact data structure. However, M-index terminates the search on average earlier with respect to the ratio of accessed data partitions. Nonetheless, there is still a high space for improvement since there are 30 buckets at maximum that contain the precise answer to any 30NN query. More details on this are available in [30].

V. Artificial Priority Queues

Answering a similarity query is a process of identifying promising data partitions that contain objects closest to the query object. Thus, a priority queue of data partitions to inspect is constructed and ordered by distances from the query object to the nodes with possible respect to node's extent. We present existing approaches to ordering priority queues and propose a technique to combine multiple aspects to catch more information about data object distribution in the nodes.

A. Current Approaches to Ordering in Priority Queue

The methods to define ordering in priority queues were briefly stated in Section IV-B. Here, we survey more possibilities for M-tree and M-index. The following three methods are naturaly possible in M-tree, where each node is described by a pivot p and covering radius r^c. For convenience, they are depicted in Fig. 3a.

1) **Query-pivot (QP) ordering** – $d(q, p)$. Although this method describes the "objective" position of objects within the node, it does not take into consideration the node's extent. In other words, the same data distribution of objects within nodes is assummed, which is not true since node radii may vary largely.
2) **Lower bound (LB) ordering** – $d(q, p) - r^c$. This method gives the preference to the closest possible object within the given node, which is advantageous over QP. This also constitutes the termination condition of search algorithms. However, it also favours large sparse nodes over small dense ones.
3) **Upper bound (UB) ordering** – $d(q, p) + r^c$. This has opposite properties to LB. It prefers compact close nodes, which makes it more efficient in well-clustered data-sets. On the other hand, larger nodes are visited later, where the pivot's distance to the query influences the ordering a little.

We also recognize a number of approaches for creating priority queues in M-index. Position of every object in M-index is determined by its closest pivots ordered by their distances $ppp_o = \langle p_1, ..., p_n \rangle$, denoted as the object's *PPP-code* [11]. An illustration is provided in Fig. 3b. Since there may not be any pivot centered in data partition inherently, it is more challenging to state good estimates of priority p of a data partition $area$. The following methods were proposed [11].

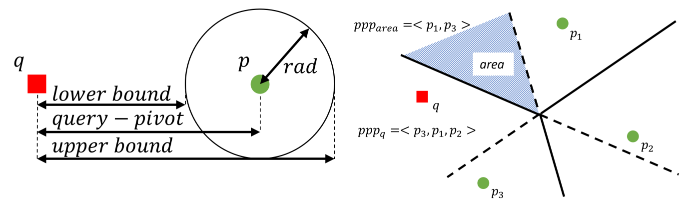
1) **Weighted Sum of Pivot Distances (WSPD).** It computes a weighted sum of distances between the query object and pivots of area's *PPP-code* as follows:

$$p_{area} = \sum_{i=0}^{|ppp_{area}|} d(q, ppp_{area}[i]) * 0.75^i.$$

2) **Sum of Differences between Pivot Distances (SDPD).** It sums distances of pivots in query object's *PPP-code* subtracted from the distances of pivots in the area's *PPP-code*. The formula is as follows:

$$p_{area} = \sum_{i=0}^{|ppp_{area}|} max(0,\ d(q, ppp_{area}[i]) - d(q, ppp_q[i])).$$

For detailed insight into the performance of these techniques, we refer the reader to [11].



(a) M-tree (b) M-index

Fig. 3: Methods for determining priority in M-tree (a). Denotation of data partitions in Voronoi partitioning in M-index with *PPP-codes* (b).

Each of the approaches presented above possess a number of advantages and drawbacks that make achieving of the best performance on any data-set impossible. In case of M-tree, usage of the distance from query object to pivot and covering radius is biased towards precisely computed lower and upper bound, which blindly respects covering radius, or towards ignoring the partition extent at all. The methods presented for M-index omit the information about the cell's area entirely, and typically rely on the weights of different levels of the structure to represent browsed data bulks. This leads to problems concerning outlier areas, areas with disproportionate data distribution and many other characteristics.

B. Ordering in Artificial Priority Queues

We propose an approach for priority queue management by basic supervised learning methods to make it optimized to the given dataset and indexing structure. We define an ordering of priority queue as a weighted compound of more measures of data and indexing structure characteristics – inter-object distances, covering radii, data partition usability [22], intrinsic dimensionality [31], and/or different approaches for priority queue ordering, as presented in the previous section.

Let us consider a query q and a pair of properties assigned to every node of the dataset np_1 and np_2. Every data partition can be consequently represented by its coordinates $[np_1, np_2]$, which makes possible to compute the linear regression over such data. Learning phase of the algorithm for naïve artificial priority queues firstly computes linear regressions for selected queries. Next, the average slope value k of these linear regressions is stated, serving as the only parameter for the execution phase (line shift coming from the regression can be omitted). Finally, the priority queue is computed as a projection of every candidate node on the line, formally:

$$order_{object} = k * np_1 + np_2.$$

This can be generalized to combining more than two properties $np_1, ..., np_n$ and by applying Principle Component Analysis (PCA) algorithm. Additionally, any ordering formed for artificial priority queues can be regarded as a property as well. The generalized form is computed as:

$$order_{object} = PCA(np_1, np_2, \ldots np_n).$$

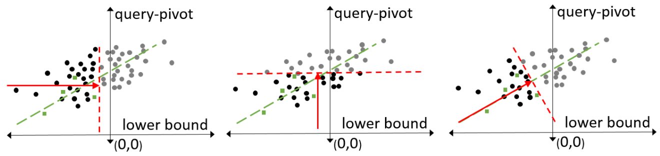


Fig. 4: Examples of priority queue ordering: LB (left), QP (middle) and APQ (right). Red arrow denotes direction of data access, perpendicular dashed line depicts boundary of already visited data and long dashed line represents linear regression of the positive (square) nodes.

In the next section, we provide a detailed analysis supported by experiments of performance gain of the naïve form of this approach. We selected the best performing pair of properties for priority queue for given indexing structure.[2] Fig. 4 provides an illustration of ordering leaf nodes by LB, QP and naïve APQ.

VI. Experiments

In this section, we provide an extensive experimental comparison of artificial priority queue and original priority queues applied in a kNN-query evaluation algorithm. Two different data-sets are used through the experiments. First, a one-million subset of CoPhIR data-set, where every object is formed by five MPEG-7 global visual descriptors in a 282-dimensional vector and the distance function is a weighted sum of L_1 and L_2 metrics [27]. Second, a one-million subset of Profimedia data-set [28], where Caffe visual descriptors as 4,096-dimensional vectors are extracted [29] and the Euclidean distance is used. Performance is measured on approximate evaluation of 30 nearest-neighbors queries with 1000 top used query objects, as is referred in Section IV-A.

Results of every experiment are computed as average values of performance measures over 1000 queries and are represented in two graphs. First, left-hand graphs depict precision for varying stop condition in the approximate kNN query evaluation. The stop condition is defined as the number of visited leaf nodes. Second, right-hand graphs illustrate the distribution of change in answer precision – x-axis shows the increase in precision (absolute change in precision), and y-axis characterizes the number of queries with the given precision shift. For legibility, we present only three select stop condition values in the right-hand graphs – the lowest, the greatest and one of the middle thresholds.

A. Experiments on M-tree

Results of experiments on M-tree presented in Fig. 5 depict four different methods for priority queue creation, all described in Section V: query-pivot, lower bound, upper bound and APQ that was computed over two best performing methods (LB and QP).

[2] In case of M-tree, there is next to nothing difference in optimizing between QP and node covering radii from optimizing between QP and LB, as they are mutually dependable.

APQ performs consistent 50% gain in precision for the same stop condition threshold. From the costs point of view, APQ requires to visit by 50% fewer nodes to provide the same precision as LB, the state-of-the-art technique in M-tree.

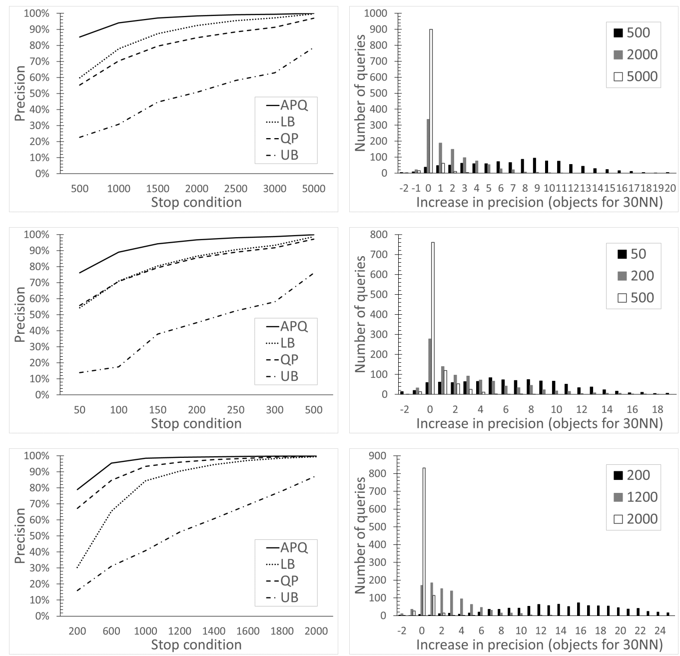


Fig. 5: Performance of M-tree, from top to bottom: CoPhIR 200, CoPhIR 2000 and Profiset 2000.

B. Experiments on M-index

The second group of experiments was conducted on M-index and compares the corresponding orderings of priority queues: WSPD (the state of the art) and SDPD (Sum of Differences between Pivot Distances), and APQ based on these approaches. The results in Fig. 6 show approximately 20% saves of resources to achieve the same precision. This is better depicted in the right-hand graphs, where the distribution of absolute precision change is in positive numbers, especially for small values of stop condition threshold.

VII. Conclusions and Future Work

We have presented a method for optimizing priority queues that is widely applicable to any hierarchical index structure. The method is implemented as a linear regression over two distinct properties of buckets that form query result. In this respect, it is an unsupervised technique that adapts to specific indexing structure characteristics as well as to data characteristics.

The proposal was verified in experiments on M-tree and M-index, showing significant gains in performance over the methods for priority queue ordering standard in the respective structure.

In the future, we intend to further analyze other properties, e.g. extent of internal nodes and occupation of nodes. We also plan to test the generalized version of APQ.

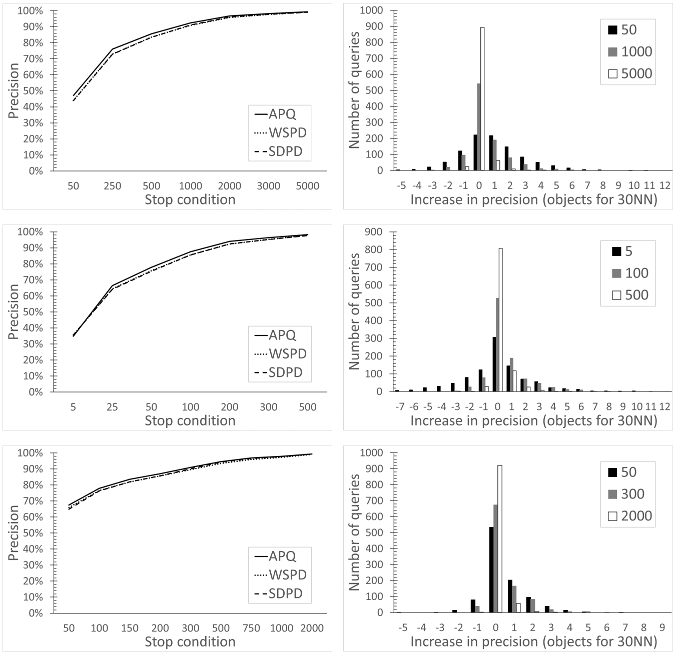


Fig. 6: M-index experiments, from top to bottom: CoPhIR 200, CoPhIR 2000 and Profiset 2000

References

[1] H. Samet, *Foundations of Multidimensional And Metric Data Structures*, ser. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 2006.

[2] C. Böhm, S. Berchtold, and D. A. Keim, "Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases," *ACM Computing Surveys*, vol. 33, no. 3, pp. 322–373, Sep. 2001.

[3] T. Skopal, J. Pokorný, and V. Snášel, "Nearest neighbours search using the PM-Tree." in *Proceedings of the 10th International Conference on Database Systems for Advanced Applications (DASFAA)*, ser. Lecture Notes in Computer Science, vol. 3453. Springer, 2005, pp. 803–815.

[4] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *Proceedings of VLDB*. Morgan Kaufmann, 1997, pp. 426–435.

[5] G. Amato, F. Rabitti, P. Savino, and P. Zezula, "Region proximity in metric spaces and its use for approximate similarity search," *ACM Transactions on Information Systems (TOIS 2003)*, vol. 21, no. 2, pp. 192–227, April 2003.

[6] M. E. Houle and J. Sakuma, "Fast approximate similarity search in extremely high-dimensional data sets," in *Proceedins of 21st International Conference on Data Engineering (ICDE)*, April 2005, pp. 619–630.

[7] M. E. Houle and M. Nett, "Rank-based similarity search: Reducing the dimensional dependence," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 1, pp. 136–150, Jan 2015.

[8] P. Zezula, G. Amato, V. Dohnal, and M. Batko, *Similarity Search: The Metric Space Approach*, ser. Advances in Database Systems. Springer, 2005, vol. 32.

[9] E. Chávez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroquín, "Searching in metric spaces," *ACM Computing Surveys*, vol. 33, no. 3, pp. 273–321, September 2001.

[10] D. Novak, M. Batko, and P. Zezula, "Metric index: an efficient and scalable solution for precise and approximate similarity search," *Information Systems*, vol. 36, 2011.

[11] D. Novak and P. Zezula, "Rank aggregation of candidate sets for efficient similarity search," in *Proceedings of the 25th International Conference on Database and Expert Systems Applications (DEXA)*. Springer International Publishing, 2014, pp. 42–58.

[12] J. M. Vilar, "Reducing the overhead of the AESA metric-space nearest neighbour searching algorithm," *Information Processing Letters*, vol. 56, no. 5, pp. 265–271, 1995.

[13] T. Skopal and D. Hoksza, "Improving the performance of m-tree family by nearest-neighbor graphs," in *Proceedings of 11th East European Conference on Advances in Databases and Information Systems (ADBIS)*. Springer Berlin Heidelberg, 2007, pp. 172–188.

[14] P. H. Oliveira, C. Traina, and D. S. Kaster, "Improving the pruning ability of dynamic metric access methods with local additional pivots and anticipation of information," in *Proceedings of 19th East European Conference on Advances in Databases and Information Systems (ADBIS),*. Springer International Publishing, 2015, pp. 18–31.

[15] A. Esuli, "Use of permutation prefixes for efficient and scalable approximate similarity search," *Inf. Process. Manage.*, vol. 48, no. 5, pp. 889–902, Sep. 2012.

[16] T. Skopal, J. Lokoc, and B. Bustos, "D-cache: Universal distance cache for metric access methods," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 5, pp. 868–881, May 2012.

[17] J. M. Barrios, B. Bustos, and T. Skopal, "Analyzing and dynamically indexing the query set," *Information Systems*, vol. 45, pp. 37 – 47, 2014.

[18] F. Falchi, C. Lucchese, S. Orlando, R. Perego, and F. Rabitti, "Caching content-based queries for robust and efficient image retrieval," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, ser. EDBT '09. New York, NY, USA: ACM, 2009, pp. 780–790.

[19] N. R. Brisaboa, A. Cerdeira-Pena, V. Gil-Costa, M. Marin, and O. Pedreira, "Efficient similarity search by combining indexing and caching strategies," in *Proceedings of the 41st International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 486–497.

[20] E. Sadit Tellez and E. Chávez, "The list of clusters revisited," in *Pattern Recognition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 187–196.

[21] F. Nalepa, M. Batko, and P. Zezula, "Cache and priority queue based approximation technique for a stream of similarity search queries," in *Similarity Search and Applications*. Cham: Springer International Publishing, 2017, pp. 17–33.

[22] M. Antol and V. Dohnal, "Optimizing query performance with inverted cache in metric spaces," in *Proceedings of the 20th East-European Conference on Advances in Databases and Information Systems (ADBIS)*, L. N. in Computer Science, Ed. Cham: Springer International Publishing, 2016, pp. 60–73.

[23] P. Deepak and M. D. Prasad, *Operators for Similarity Search: Semantics, Techniques and Usage Scenarios*. Springer, 2015.

[24] T. Skopal, "Pivoting M-tree: A metric access method for efficient similarity search," in *Proceedings of the Annual International Workshop on DAtabases, TExts, Specifications and Objects (DATESO 2004)*, ser. CEUR Workshop Proceedings, vol. 98. Technical University of Aachen (RWTH), 2004.

[25] M. S. Lew, N. Sebe, C. Djeraba, and R. Jain, "Content-based multimedia information retrieval: State of the art and challenges," *ACM Transactions on Multimedia Comput. Commun. Appl.*, vol. 2, no. 1, pp. 1–19, Feb. 2006.

[26] C. Beecks, T. Skopal, K. Schöffmann, and T. Seidl, "Towards large-scale multimedia exploration," in *Proceedings of the 5th International Workshop on Ranking in Databases (DBRank)*, 2011, pp. 31–33.

[27] M. Batko, F. Falchi, C. Lucchese, D. Novak, R. Perego, F. Rabitti, J. Sedmidubsky, and P. Zezula, "Building a web-scale image similarity search system," *Multimedia Tools and Applications*, vol. 47, no. 3, pp. 599–629, 2009.

[28] P. Budikova, M. Batko, and P. Zezula, "Evaluation platform for content-based image retrieval systems," in *Proceedings of the 15th International Conference on Theory and Practice of Digital Libraries: Research and Advanced Technology for Digital Libraries*, ser. TPDL'11. Springer-Verlag, 2011, pp. 130–142.

[29] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, ser. MM'14. New York, NY, USA: ACM, 2014, pp. 675–678.

[30] M. Antol and V. Dohnal, "Popularity-based ranking for fast approximate knn search," *Informatica*, vol. 28, no. 1, pp. 1–21, 2017.

[31] V. Pestov, "Intrinsic dimension of a dataset: what properties does one expect?" in *2007 International Joint Conference on Neural Networks*, Aug 2007, pp. 2959–2964.