INFORMATICA, 2017, Vol. 28, No. 1, 1–21

DOI: http://dx.doi.org/10.15388/Informatica.2017.118

Popularity-Based Ranking for Fast Approximate kNN Search

Matej ANTOL*, Vlastislav DOHNAL
Faculty of Informatics, Masaryk University
Botanicka 68a, Brno, Czech Republic
e-mail: xantol@fi.muni.cz, dohnal@fi.muni.cz

Received: October 2016; accepted: February 2017

Abstract. Similarity searching has become widely available in many on-line archives of multimedia data. Users accessing such systems look for data items similar to their specific query object and typically refine results by re-running the search with a query from the results. We study this issue and propose a mechanism of approximate kNN query evaluation that incorporates statistics of accessing index data partitions. Apart from the distance between database objects, it also considers the prior query answers to prioritize index partitions containing frequently retrieved data, so evaluating repetitive similar queries more efficiently. We verify this concept in a number of experiments.

Key words: kNN query, approximate search, query popularity, index structure, metric space.

1. 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 a new query by clicking on a previously returned image. This procedure exhibits the property that many queries processed by the system are alike, so search algorithms may optimize such repeated queries to save computational resources.

In general, the query efficiency is typically supported by various indexing structures, storage layouts and disk caching/buffering techniques. So the number of disk I/Os needed to answer a query is greatly reduced. However, handling more complex and unstructured data requires extracting so-called descriptors as surrogate information used to organize and query the data. The descriptors typically form high-dimensional spaces or even distance spaces where no implicit coordinate system is defined (Samet, 2006). The problem of *dimensionality curse* then often appears (Böhm *et al*., 2001). This typically leads to visiting many data partitions by an indexing mechanism due to high 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 (Skopal *et al*., 2005; Ciaccia *et al*., 1997) or by sacrificing precision

*Corresponding author.

in query results (approximate querying) (Amato *et al*., 2003; Houle and Sakuma, 2005; Houle and Nett, 2015).

In this paper, we further study the issue of evaluating repeated queries and apply a general technique to prioritize data partitions during query evaluation. This technique exploits queries executed previously to gather necessary statistics, so the so-called popularity of data partitions can be established and applied to prioritize access to the partitions. We have proposed this technique in Antol and Dohnal (2016) and applied it to precise kNN queries. Here, we elaborate on this concept more and reveal a connection between algorithm parameters and data-set properties. Our findings are confirmed by experiments on two different data-sets (CoPhIR, Batko *et al*., 2009 and Profiset, Budikova *et al*., 2011). Additionally, we applied our concept to evaluation of approximate kNN queries. The concept of popularity is commonly used in information retrieval for post-processing (ranking) the candidate set retrieved from an index. We rather make popularity inherent part of the search algorithm by warping the original distances, so it is applied during the searching itself.

The paper is structured as follows. In the next section, we summarize related work. Indexing and querying principles are concisely given in Section 3. Inefficiency in performance of current indexes is presented in Section 4. The application of Inverted Cache Index to approximate kNN query evaluation is described in Section 5 and its performance in Section 6. Contributions of this paper and possible future extensions are summarized in Section 7.

2. Related Work

The wide area of indexing structures for metric space is surveyed in Zezula *et al*. (2005), Chávez *et al*. (2001). To process large data-sets, indexing structures are designed as disk-oriented. Their data partitioning principles are typically based on (i) hierarchical clustering (e.g. M-tree, Ciaccia *et al*., 1997), where each subtree is covered by a preselected data object (pivot) and a covering radius; (ii) Voronoi partitioning (e.g. M-index, Novak *et al*., 2011, PPP-Codes, Novak and Zezula, 2014), where subtrees are formed by assigning objects to the closest pivot recursively; and (iii) precomputed distances (e.g. Linear AESA, Vilar, 1995), where no explicit structure is built, but rather distances among data objects are stored in a matrix.

Optimizations of query-evaluation algorithms are based on extending a hierarchical structure with additional precomputed distances to strengthen filtering capabilities, e.g. M*-tree (Skopal and Hoksza, 2007), cutting local pivots (Oliveira *et al*., 2015); or on exploiting large number of pivots in a very compact and reusable way, e.g. permutation prefix index (Esuli, 2012). These techniques, however, do not analyse the stored data and accesses to data partitions, but rather constrain the data partitions as much as possible.

Another way to make query evaluation much cheaper is to trade accuracy, i.e. to apply approximate searching. Existing approaches use early-termination or relaxed-branching strategies to stop searching prematurely, e.g. when the query result does improve marginally. A recent approach called spatial approximation sample hierarchy (Houle

and Sakuma, 2005) builds an approximated near-neighbour graph and does not exploit triangle inequality at all. We remind the reader that 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 (Houle and Nett, 2015).

Distance-Cache (Skopal *et al*., 2012) is a main-memory structure that collects information from previous querying. It caches some distances computed between metric objects to help forming 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, which is a resemblance with our approach.

With the advance of content-based retrieval, there are approaches to optimize a stream of kNN queries. Snake table (Barrios *et al*., 2014) is a dynamically-built structure for optimizing all queries corresponding to one user session. It remembers results of all queries processed so far and constructs a linear AESA to evaluate next queries more efficiently. There are also approaches that intercept a collection of queries in regular intervals and by grouping the queries by their similarity, the evaluation of some of them can be done simultaneously. This is also partly done in the Snake table. A pure caching strategy of similarity queries was proposed in Falchi *et al*. (2009). On the other hand, a combination of query cache and index structure was proposed in Brisaboa *et al*. (2015). 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 (Sadit Tellez and Chávez, 2012) is built using the query objects only, which is analogous to the Snake table.

The Inverted Cache Index was recently proposed in Antol and Dohnal (2016). It takes into account previously processed queries too, which is similar to some approaches presented above. However, such information is exploited to increment so-called "popularity" of data partitions only. In this respect, no query object or answer is stored/cached there. This paper builds on this idea, proposes a parameter setting tight closely to the data-set indexed, and verifies it in approximate kNN query evaluation.

3. Indexing and Querying Metric Spaces

We assume data is modelled in a metric space, so indexing techniques exploit properties of metric space to create a structure able to evaluate similarity queries efficiently.

3.1. *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:

$\forall x, y \in \mathcal{D}, \quad d(x, y) \geqslant 0$	non-negativity,
$\forall x, y \in \mathcal{D}, \quad d(x, y) = d(y, x)$	symmetry,
$\forall x, y \in \mathcal{D}, \quad x = y \Leftrightarrow d(x, y) = 0$	identity, and
$\forall x, y, z \in \mathcal{D}, \quad d(x, z) \leqslant d(x, y) + d(y, z)$	triangle inequality.

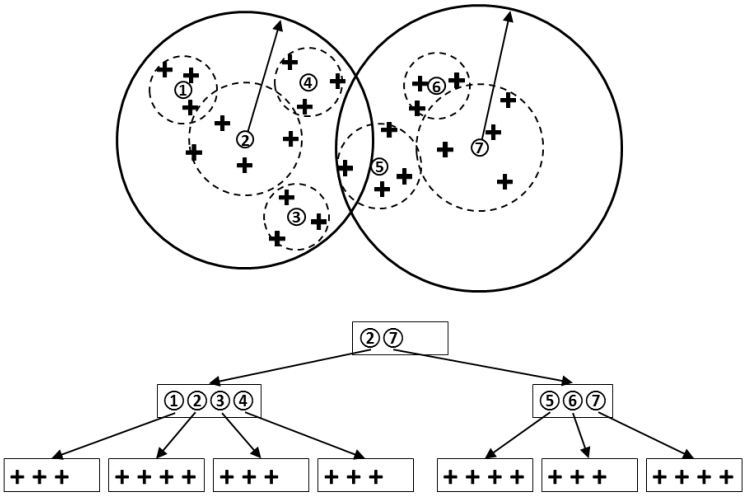


Fig. 1. Partitioning principle of M-tree.

The distance function is used to measure similarity between two objects. The shorter the distance is, the more similar the objects are. Consequently, a similarity query can be defined. There are many types of similarity queries (Deepak and Prasad, 2015) but the range and k-nearest neighbour queries are the most important ones. The range query $R(q, r)$ specifies all database objects within the distance of r from the query object $q \in \mathcal{D}$. In particular, $R(q, r) = \{o | o \in X,\ d(q, o) \leqslant r\}$, where $X \subset \mathcal{D}$ is the database to search in. In this paper, we primarily focus on k-nearest neighbour query since it is more convenient for users. The user wants to retrieve k most similar objects to q without the need to know details about the distance function d. Formally, $kNN(q) = A$, where $|A| = k \wedge \forall o \in A,\ p \in X - A,\ d(q, o) \leqslant d(q, p)$.

3.2. *Indexing and Query Evaluation*

To organize a database to answer similarity queries efficiently, many indexing structures have been proposed (Zezula *et al*., 2005). Their principles are twofold: (i) recursively applied data partitioning/clustering defined by a preselected data object called *pivot* and a distance threshold, and (ii) effective object filtering using lower-bounds on distance between a database object and a query object. These principles have been firstly surveyed in Chávez *et al*. (2001).

In this paper, we use the traditional index M-tree (Ciaccia *et al*., 1997) and a recent technique M-index (Novak *et al*., 2011). Both of these structures create an internal hierarchy of nodes that partition the data space into many buckets – an elementary object storage. Please refer to Figs. 1 and 2 for principles of their organization. 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. On the other hand, M-index applies Voronoi-like partitioning using a predefined set of pivots in the top-down way. In this case, clusters are formed

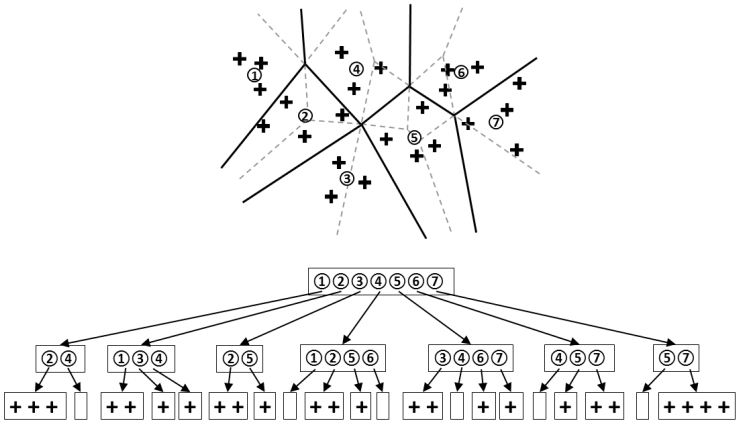


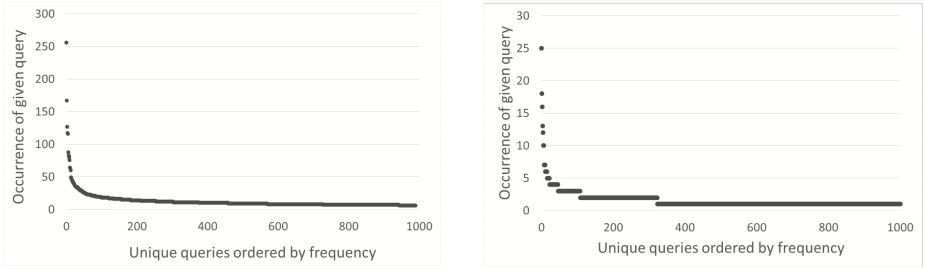
Fig. 2. Partitioning principle of M-index.

by objects that have the cluster's pivot as the closest one. On next levels, the objects are reclustered using the other pivots, i.e. eliminating the pivot that formed the current cluster. Buckets of both structures store objects in leaf nodes, as is exampled in the illustration. So we use the terms *leaf node* and *bucket* interchangeably.

A kNN-query evaluation algorithm constructs a *priority queue* of nodes to access and gradually improves 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 kth neighbour 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. If it is a non-leaf node, all its children are inserted into the queue with correct lower bounds estimated. 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, which is elaborate too, are described.

4. Effectiveness of Indexing and Approximate Searching

Interactivity of similarity queries is the main driving force to make content-based information retrieval widely used (Lew *et al.*, 2006). In the era of Big Data, near real-time execution of similarity queries over massive data collections is even more important, because it allows various analytic tasks to be implemented (Beecks *et al.*, 2011). In this section, we present motivating arguments based on experience with a real-life content-based retrieval system.



(a) CoPhIR (b) Profiset

Fig. 3. Distribution of top-1000 unique queries ordered by their appearances.

4.1. *Query Statistics*

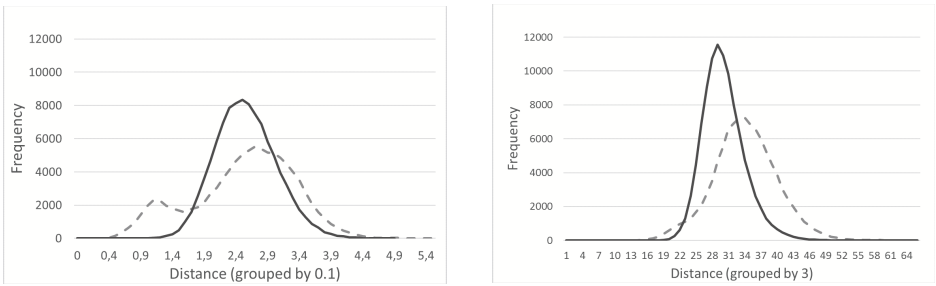
We gathered statistics of querying from two demonstration applications.[2] The first one organizes the well-known CoPhIR data-set (Batko *et al.*, 2009) consisting of 100 million images using global MPEG-7 descriptors. Whereas the second application searches in the Profiset collection (Budikova *et al.*, 2011) that consists of 20 million high-quality images with rich and systematic annotations using descriptors from deep convolutional neural networks (Caffe descriptors) (Jia *et al.*, 2014).

Figure 3 shows absolute frequencies of individual top-1000 queries that were executed during the applications' life time. The demo on CoPhIR was launched in Nov. 2008, while the demo on Profiset was made public in June 2015. These power-law like distributions are attributed to the way of presenting an initial search to a new website visitor. In CoPhIR demo, the users are initially provided with a similarity search to a query image randomly picked from a set of 105 preselected images, so there is a high frequency of such queries. In Profiset demo, the initial page contains 30 query images randomly selected from the whole data-set, so any repetition is caused by bookmarking or sharing popular images. There are few such queries only. Figure 4 depicts density of distances among the top-1000 queries, so the 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. 4. To sum up, these two query sets form different conditions for our proposal to cope with.

4.2. *Indexing Structure Performance*

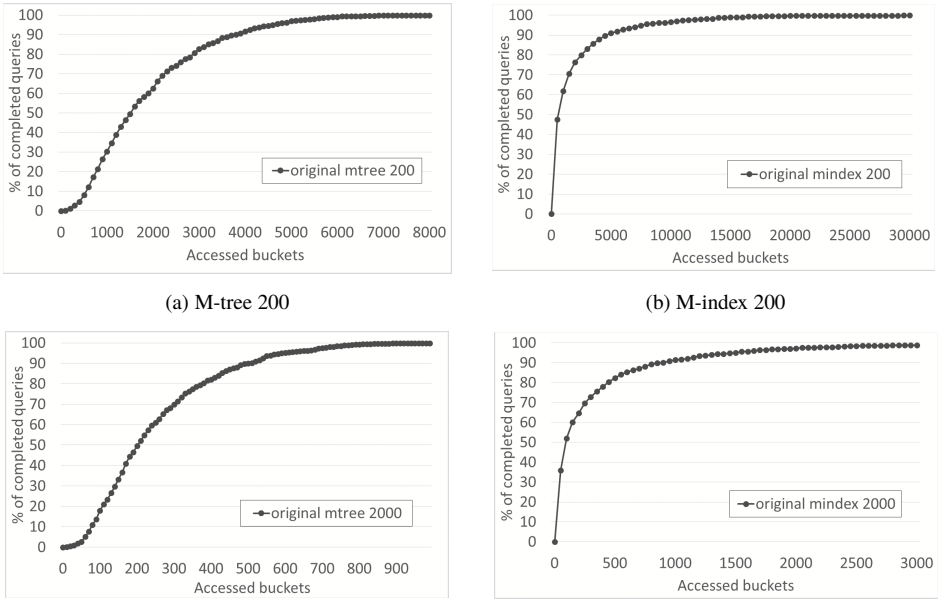
The major drawback of indexing structures in metric spaces is the high amount of overlaps among index substructures (data partitions), which is also supported by not very precise estimation of lower bounds on distances between data objects and a query object. So the

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



(a) CoPhIR (b) Profiset

Fig. 4. Density of distances among top-1000 query objects (dashed curve) and overall density of the data-set (solid curve).



(a) M-tree 200 (b) M-index 200

(c) M-tree 2000 (d) M-index 2000

Fig. 5. Percentage of fully completed queries of 30NN for increasing number of accessed buckets on CoPhIR.

kNN-query evaluation algorithm often accesses a large portion of indexing structure's buckets to obtain precise answer to a query.

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. In Fig. 5, we present the percentage of fully completed queries while constraining the number of accessed buckets. We have tested two configurations for both M-tree and M-index – the capacity of buckets was constrained to 200 and 2,000 objects to have bushier and more compact structures. Table 1 summarizes information about them. To

Table 1
Structure details of tested indexing techniques.

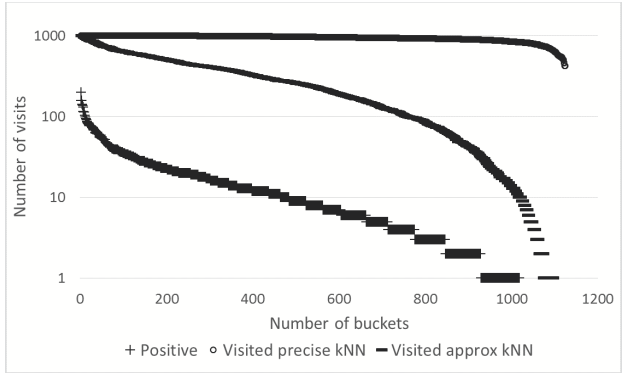
Indexing structure	Data-set	bckt cap. (objs)	Total bckts	Avg bckt occup.	Hierarchy height	Internal node cap.
M-tree 200	CoPhIR	200	11 571	43.2%	4	50
M-tree 2000	CoPhIR	2 000	1 124	44.5%	3	100
M-tree 2000	Profiset	2 000	2 634	19.0%	3	100
M-index 200	CoPhIR	200	62 049	8.1%	8	N/D
M-index 2000	CoPhIR	2 000	10 943	4.6%	8	N/D
M-index 2000	Profiset	2 000	20 222	2.5%	6	N/D

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 very low occupied buckets, so M-tree is a more compact data structure. However, the kNN evaluation algorithm of M-index can access promising data partitions early, so curve depicting percentage of fully completed queries is increasing more steeply.

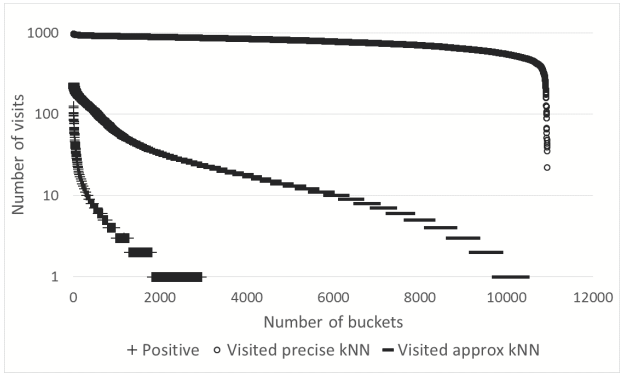
The other aspect of indexing structures to study is efficiency, i.e. how many data partitions are accessed to complete a 30NN query. In Fig. 6, we present the number of visits of individual buckets for M-tree and M-index structures after evaluating all top-1000 queries on CoPhIR and Profiset. In particular, we depict visited buckets for precise and approximate 30NN query evaluation with "O" and "X", respectively. The curve denoted with "+" corresponds to the buckets that contained at least one data object returned in the precise answer. Thus, it forms the minimal requirements to evaluate the queries. The approximate evaluation was terminated after visiting 303 buckets of M-tree (27.5%) and 254 buckets of M-index (2.3%) for CoPhIR, and 1 714 buckets of M-tree (65.9%) and 779 buckets of M-index (3.8%) for Profiset data-set. These thresholds were selected to correspond to the number of visited buckets that contains complete answer of 70% queries. The average precision of approximate queries is 97.5%, for both M-tree and M-index. From the figure, we can see very large inefficiency in navigating to buckets containing relevant data and terminating the search when precise answer is found, i.e. very loose estimated lower bounds on distance between the query object and a data partition. Even though approximate evaluation provides large performance boost, there is still a large number of buckets that do not need to be accessed at all (more than 90%). Once again, the advantage of M-index over M-tree is in navigating to relevant buckets earlier.

5. Popularity-Based Approximate Query Evaluation

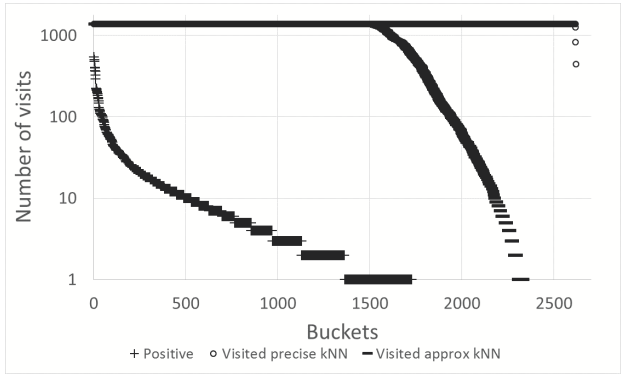
In this section, we describe a technique for prioritizing nodes of indexing hierarchies to locate relevant data objects earlier during kNN query evaluation. This technique is based on accumulating frequencies of accessing individual data partitions to re-order its priority queue during a query evaluation. We call this technique *Inverted Cache Index* (ICI).



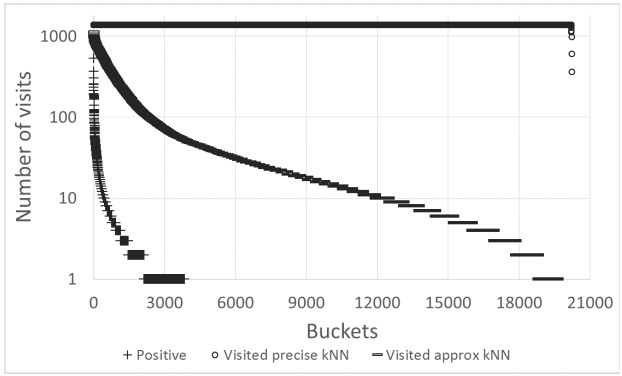
(a) M-tree 2000, CoPhIR



(b) M-index 2000, CoPhIR



(c) M-tree 2000, Profiset



(d) M-index 2000, Profiset

Fig. 6. Absolute frequency of visiting a bucket after evaluation of all queries on CoPhIR and Profiset data-sets. The x-axis displays buckets sorted by frequency descendingly, so different curves imply different ordering. The y-axis is in log scale.

It does not record the queries processed so far, but rather the number of times a given partition/bucket (or even data object) contributed to the final result of such queries. This technique was originally proposed in Antol and Dohnal (2016). In this article, we further study its properties and parameter settings and, more importantly, apply it to approximate kNN queries.

ICI requires any underlaying indexing structure to be extended with one counter per bucket. This counter accumulates the number of times the bucket contributed to the final query answer. If an indexing structure is hierarchical, this is applied also in internal "nodes". In particular, we apply the variant called *object ratio* (Antol and Dohnal, 2016). This ICI counter is then used to stretch/extend the original metric distance between the bucket's or node's representative (p – pivot) and a query (q) as follows:

$$d_{ICI} = \frac{d(q,\, p) \cdot \left(\left(\frac{d(q,p)}{d_{norm}}\right)^{pwr} + 1\right)}{\log_{base}(ICI + base)} \tag{1}$$

where d_{ICI} denotes the modified distance and d_{norm} is the normalization distance. The parameter *base* is fixed to 10 and the parameter *pwr* is set to 2 for M-tree and 5 for M-index, which is recommended in Antol and Dohnal (2016). The normalization distance controls the point where the "attraction" force turns into the "repulsive" force to push irrelevant data partitions away. It may be set to any value up to the maximum distance of the data-set, which is 10 in CoPhIR and 200 in Profiset.

Since approximate query evaluation allows some imprecision in the final answer, we do not take the whole answer to update ICI counters. In particular, we focus on a portion of the closest objects only, corresponding to the expected result precision. The amount of them is a parameter of Algorithm 1 as well as the maximum number of visited buckets.

6. Experiments

In this section, we provide an extensive experimental comparison of ICI-optimized precise and approximate kNN query evaluation with standard (non-ICI) algorithms. Two different data-sets are used through the experiments. First, a 1-million-object subset of CoPhIR data-set is used. Each object is formed by five MPEG-7 global descriptors (282 dimensional vector in total) and the distance function is a weighted sum of L_1 and L_2 metrics (Batko *et al.*, 2009). Second, a 1-million-object subset of Profimedia data-set is picked (Budikova *et al.*, 2011). Here, Caffe descriptors (4 096 dimensional vectors) are used (Jia *et al.*, 2014). The metric is Euclidean distance.

6.1. *Different Query Ordering Strategies*

The first group of experiments focuses on determining the best setting of d_{ICI} distance measure. We used M-tree with leaf node capacity fixed to 200 only and the other parameters fixed to log base 10 and to power of 2. We studied the progress of fully completed queries at particular number of accessed nodes (buckets). The results are depicted in Fig. 7, where the following approaches were compared:

Algorithm 1 Approximate kNN query evaluation with ICI.

Input: a query $Q = k\text{-NN}(q)$; an indexing structure hierarchy *root*; maximum visited buckets $bkt_{\max}$; expected query precision $prec_{\exp} \in [0; 1)$.
Output: List of objects satisfying the query in $Q.res$.

```
Q.res ← ∅ {init query result}
PQ ← {(root, 0)} {init priority queue with root and zero as the lower bound}
bkt_read ← 0 {clear counter of visited buckets}
while PQ is not empty and bkt_read < bkt_max do
  e ← PQ.poll {get the first element from the priority queue}
  if Q.res[k].distance > e.lowerBound then
    break {terminate if e cannot contain objects closer than kth neighbour}
  end if
  for all a ∈ e.children {check all children nodes} do
    if a is leaf then
      update Q with a.objects; increment bkt_read
      break if bkt_read ⩾ bkt_max {early termination}
    else
      n.lowerBound ← get estimate of lower-bound on distance between a and Q
      {e.g. M-tree's original alg. uses (d(Q.q, a.pivot) − a.radius) here}
      n.distICI ← apply d_ICI on original distance between node's pivot and Q.q
      n.children ← set of children of a
      insert n into PQ
    end if
  end for
  sort PQ by d_ICI of each PQ's element
end while
for all o ∈ Q.res {increment ICI of bucket and its all parental nodes} do
  increment ICI of o.leaf and its parents
  break if objects checked = |Q.res| · prec_exp
end for
return Q.res
```

original – M-tree's algorithm for precise kNN evaluation (search queue ordered by lower-bound distance $= (d(q, pivot) - r_{covering})$;

qdg – the proposed ICI-ordered queue, where ICI counter is updated for unique queries only;

qdg-freq – the same as "qdg", but incrementing ICI counters for all processed queries (including repeated queries).

The results show that the concept of ICI is valid as the percentage of completed queries rises faster than the original queue ordering based on lower bounds on distance. So the ordering that exploits the real distance between the query object and a pivot (node's representative) describes distribution of data objects in a node better, which is in compliance

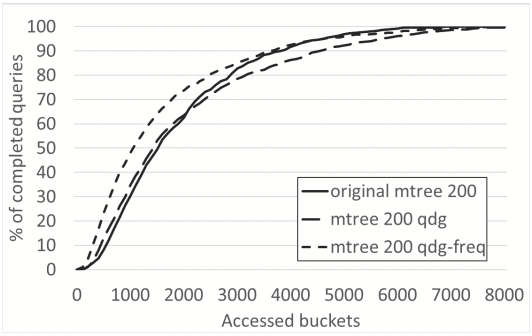


Fig. 7. Percentage of fully completed queries for standard M-tree's priority queue ordering and ICI-based ordering.

with the angle property defined by Pramanik *et al*. for node filtering (Pramanik *et al*., 1999). The best results are exhibited by the ICI strategy with counters advanced for every query executed, i.e. including repeated queries. We will examine this strategy thoroughly in the following sections.

6.2. *Precise kNN Evaluation*

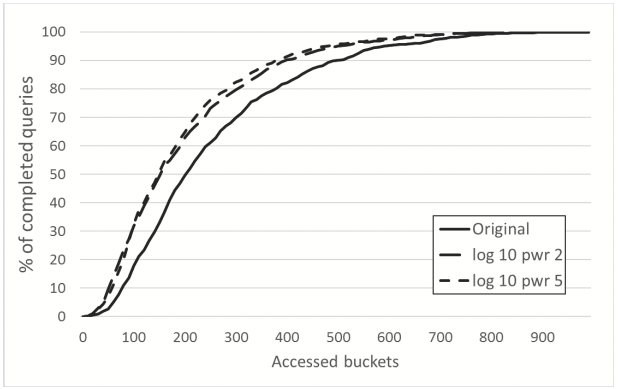
The second group of experiments focuses on precise kNN evaluation and impact of ICI in M-tree's and M-index's algorithms. We tested M-tree and M-index with capacity of buckets set to 2000 on both data collections. The test protocol is to take all queries processed by a demo application in a longer period of time and to separate it into training (adapting) and testing sub-sets.

For CoPhIR, we used traffic in the year of 2009 for counting the popularity and queries processed in January, 2010 for performance verification. They consisted of 993 and 1000 query objects, respectively. About 10% queries are in both the sub-sets in common and the remaining 90% are unique. For Profiset, the training set contained traffic from June, 2015 to February, 2016, whereas the testing one consisted of 6-week traffic following the training period. All tests were performed to evaluate precise 30NN queries.

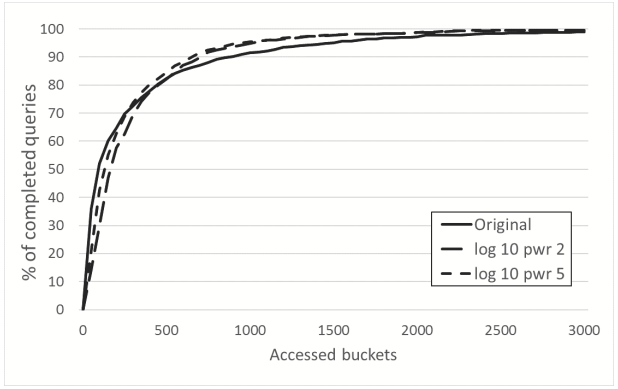
Figure 8 depicts the total percentage of completed queries while the evaluation progresses in terms of accessed buckets. We can observe that ICI can optimize performance substantially for M-tree, because M-tree visits large portion of all buckets to complete all queries (up to 80%). M-index is much more efficient in the lookup of relevant data partitions (about 30% of all buckets) and ICI-base ordering becomes better when at least 85% queries get completed on CoPhIR. Tables 2 and 3 list exact numbers of performance. We can conclude that ICI-optimized queue ordering is able to make more than 20% performance gain for 95% queries completed.

6.3. *Evolution of Performance on Monthly Basis*

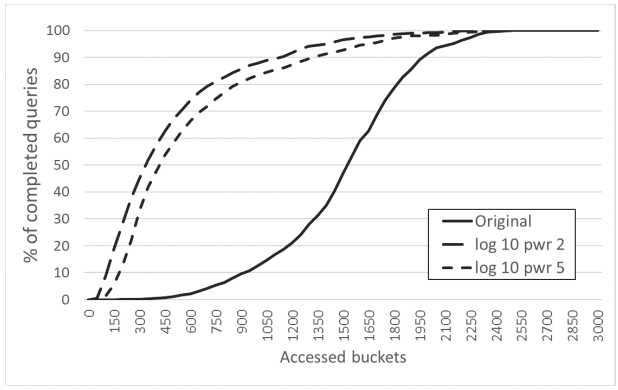
In the previous trials, the period of gathering ICI statistics was long and we focused on viability of it only. Here, we study the performance while sliding the training and testing



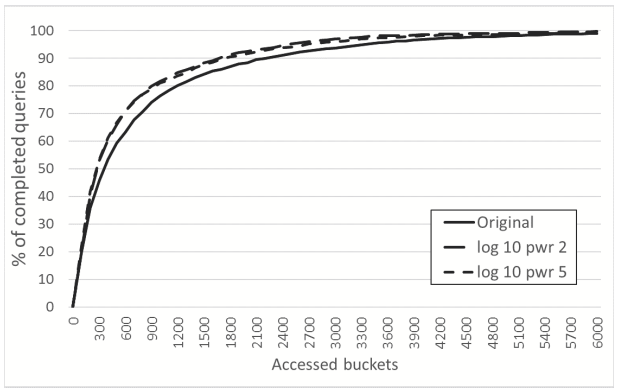
(a) M-tree 2000, CoPhIR



(b) M-index 2000, CoPhIR



(c) M-tree 2000, Profiset



(d) M-index 2000, Profiset

Fig. 8. Influence of ICI on precise 30NN query evaluation.

Table 2
Improvement in query costs of precise 30NN for CoPhIR.

CoPhIR		50% queries completed			95% queries completed		
Indexing structure	Log-pwr setup	Orig. nodes needed	Nodes needed	Total improvement	Orig. nodes needed	Nodes needed	Total improvement
M-tree	10-2	201	153	23.9%	588	495	15.8%
M-tree	10-5	201	149	25.9%	588	476	19%
M-index	10-2	89	125	−40.45%	1495	1009	32.5%
M-index	10-5	89	130	−20.2%	1495	948	36.6%

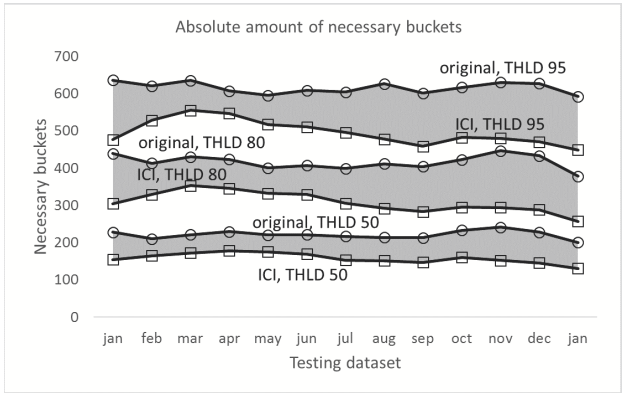
Table 3
Improvement in query costs of precise 30NN for Profiset.

Profiset		50% queries completed			95% queries completed		
Indexing structure	Log-pwr setup	Orig. nodes needed	Nodes needed	Total improvement	Orig. nodes needed	Nodes needed	Total improvement
M-tree	10-2	1521	339	77.7%	2145	1401	34.7%
M-tree	10-5	1521	416	72.6%	2145	1652	23%
M-index	10-2	349	275	21.2%	3346	2460	26.5%
M-index	10-5	349	253	27.5%	3346	2656	20.6%

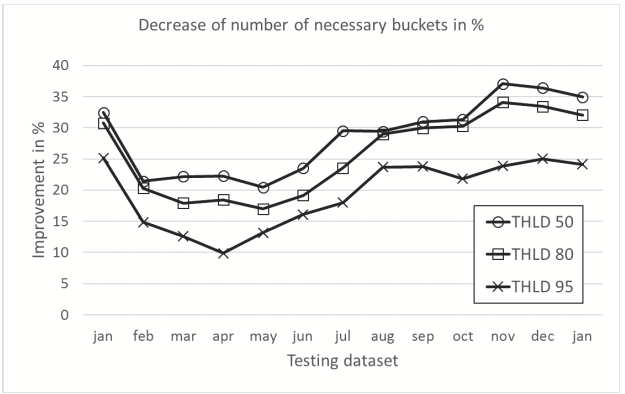
periods month by month. Since the Profiset demo application has been set up recently, we conduct this experiment on CoPhIR data-set only. Precise kNN queries are tested again, so the consistency of ICI's impact through series of consequent time frames can be compared. This also corresponds to the scenario closer to possible future applications, where the statistics should be rolled over periodically.

For every run of experiments depicted in Fig. 9, three-month traffic was used as a training data-set. The testing phase on the following-month traffic is executed on two versions of query evaluation algorithm: the original precise kNN and precise kNN with ICI-based ordering. Graphs in the figure depict results of experiments for the thresholds of 50, 80 and 95% queries completed, i.e. the average number of buckets needed to answer 50, 80 and 95% queries completely. The experiments are repeated through 13 consecutive time frames, covering more than a year of traffic. Every two consecutive months shared only 7% queries on average. In Table 4, there are numbers of training and testing queries in individual time periods and their overlap.

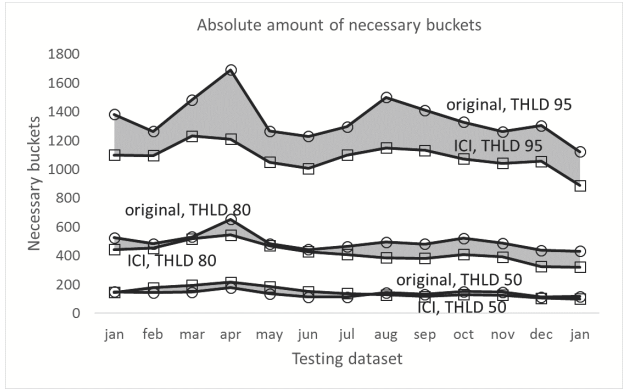
Conducted experiments show us considerably uniform improvement achieved by ICI over the original algorithm for precise kNN query. All experiments on M-tree display improvement in query answering greater than 10%. Average improvement for threshold of 95% completed queries is 19.4%. As for M-index, similar results are obtained considering the same threshold: average improvement of 19.2%. M-index's original kNN algorithm performs better for highly concentrated queries (answer is distributed in few buckets) – M-index completed 50% queries by visiting up to 89 buckets, which is less than 1% of all buckets. However, the other queries can be optimized by ICI ordering, where the overall improvement varies around 10–15%.



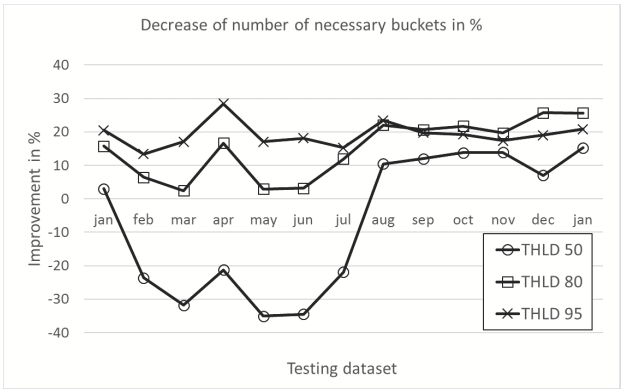
(a) M-tree 2000, absolute values



(b) M-tree 2000, relative improvement



(c) M-index 2000, absolute values



(d) M-index 2000, relative improvement

Fig. 9. Results of experiments on 13 consecutive months on CoPhIR. The green stripe emphasizes the improvement of ICI.

Table 4
Amount of queries for every training and testing period with the percentage of mutually same queries.

Testing time span											
Jan.	Feb.	Mar.	Apr.	May	Jun.	Jul.	Aug.	Sep.	Oct.	Nov.	Dec.
Number of testing queries											
12,144	8,279	12,887	7,392	6,173	4,592	2,359	4,346	4,423	4,444	3,065	8,520
Number of training queries											
62,994	75,138	80,650	33,310	28,558	26,452	18,157	13,124	11,297	11,128	13,213	11,932
Percentage of testing queries contained in the training ones											
26.68	18.28	14.57	14.09	9.32	12.46	16.08	7.21	17.32	7.52	8.67	10.12

6.4. *Approximate kNN Evaluation*

One of the challenges associated with the usage of ICI is tuning its parameters as presented in Eq. (1) in Section 5, namely: d_{norm}, *pwr* and *base*.

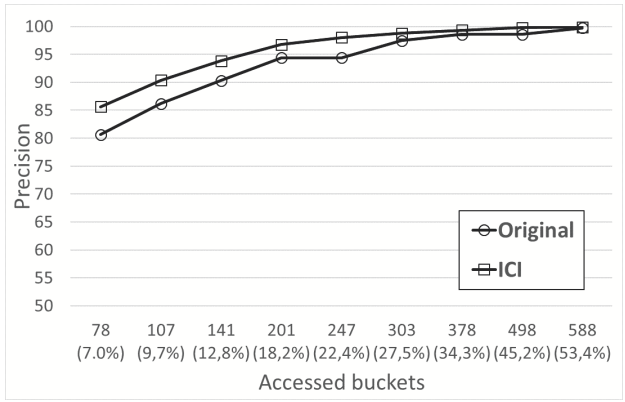
To make our method widely applicable, experiments in this section use values of these parameters derived from statistics of queries, indexing structures and data-sets. As was already mentioned, the parameter d_{norm} plays the role of a vanishing point, where two data items should become irrelevant or dissimilar. Here, we set its value to the average distance between a pair of objects in the data-set, which is 2.526 for CoPhIR and 85.84 for Profiset.

The parameter *pwr* specifies the relative impact of ICI in comparison with the original distance. In particular, it must be interpreted in two ways: (i) if a data object or partition is closer than d_{norm}, the higher the value of *pwr* is, the *greater* the influence of ICI is; (ii) if a data object is further than d_{norm}, the higher the value of *pwr* is, the *smaller* the influence of ICI is. The strength of *pwr* can be correlated to the data-set dimensionality – emphasis on ICI or distance should be and is deduced from the intrinsic dimensionality (iDim) of data (Chávez *et al*., 2001). For CoPhIR and Profiset data-sets, the value of iDim is 15 and 27, respectively.

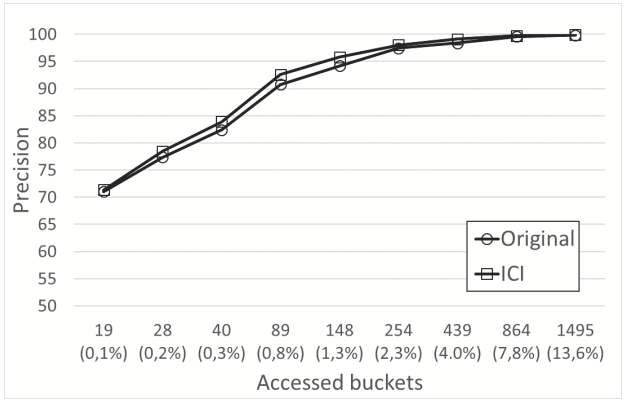
Similarly to d_{norm}, *base* serves mainly a normalization purpose: to reduce logarithmically the impact of nodes and objects with overly-high values of ICI. As the value of ICI corresponds to the number of queries processed, we deduce the value of *base* from the average number of queries (or average value of ICI) per bucket. Due to the behaviour of logarithmic function, some values are impractical or disallowed, so we should be constrained: $base \in [5; 15]$. The values of *base* used here are:

- 15 for M-tree 2000 on CoPhIR (130 000 queries/1124 buckets = 115);
- 12 for M-index 2000 on CoPhIR (130 000 queries/10 943 buckets = 12);
- 5 for M-tree 2000 on Profiset (4059 queries/2634 buckets = 1.54);
- 5 for M-index 2000 on Profiset (4059 queries/20 222 buckets = 0.20).

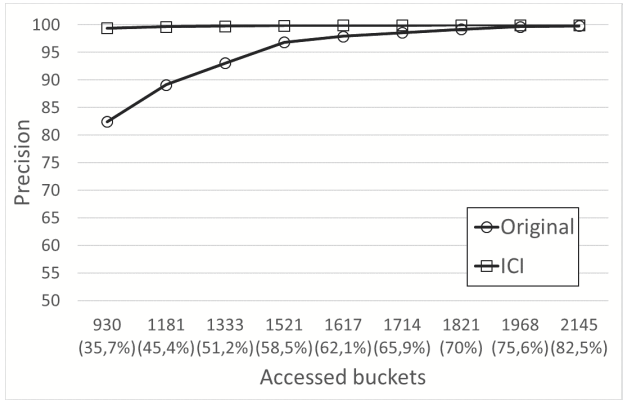
To verify such setting, we conducted a group of experiments on CoPhIR and Profiset for approximate kNN query evaluation. The results are presented in Fig. 10. We picked nine thresholds on the number of accessed buckets that correspond to the requirements of



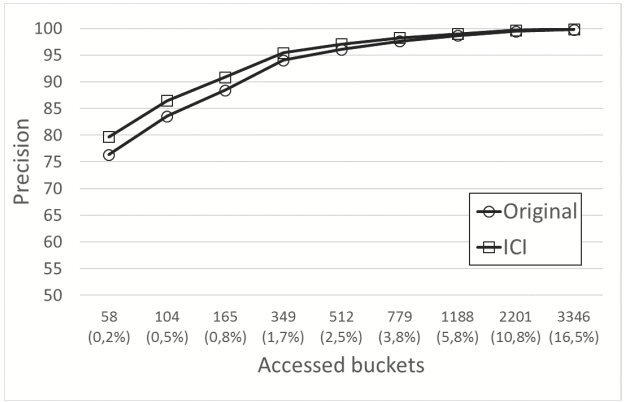
(a) M-tree 2000, CoPhIR



(b)M-index 2000, CoPhIR



(c) M-tree 2000, Profiset



(d) M-index 2000, Profiset

Fig. 10. Approximate kNN for varying limit on accessed buckets.

Table 5
Improvement in query costs for kNN precision 90% and 95%.

Structure, data-set	First precision $\geqslant 90\%$			First precision $\geqslant 95\%$		
	Original buckets needed	ICI buckets needed	Speedup	Original buckets needed	ICI buckets needed	Speedup
M-tree, CoPhIR	141	107	24%	250	170	32%
M-index, CoPhIR	89	80	10%	180	140	22%
M-tree, Profiset	1200	<800	>30%	1400	<800	>40%
M-index, Profiset	200	160	20%	420	350	17%

Table 6
Improvement in precision for different amount of visited buckets.

Structure, data-set	First threshold $\geqslant 90\%$			First threshold $\geqslant 95\%$		
	Original precision	ICI precision	Improvement	Original precision	ICI precision	Improvement
M-tree, CoPhIR	90.3%	93.8%	3.5%	97.5%	98.7%	1.2%
M-index, CoPhIR	90.7%	92.6%	1.9%	97.5%	98.0%	0.5%
M-tree, Profiset	93.1%	99.8%	6.7%	96.8%	99.9%	3.1%
M-index, Profiset	94.1%	95.4%	1.3%	96.0%	97.0%	1.0%

index structures to complete 10, 20, 30, 50, 60, 70, 80, 90 and 95% queries precisely. The exact limits on the number of accessed buckets are given as labels of x-axis.

Interpretation of the results obtained is twofold.

First, the difference between the approaches can be read horizontally, i.e. comparing the number of accessed buckets within the same precision. It indicates the possibility to lower the approximation threshold, which results into saving time to evaluate a kNN query. Table 5 gives approximate results of such optimization for kNN precision 90% and 95%. We can see that ICI is very competitive for all setups.

Second, it follows the vertical comparison of results, i.e. the change in precision within the same amount of accessed buckets. Table 6 summarizes the values of precision obtained for both the variants when the original approximation algorithm exceeded 90% and 95% precision, respectively. This demonstrates improvement in precision when ICI is applied.

Finally, the reader shall remember that we increment ICI counters only for first few objects in answer when queries are evaluated approximately. In Algorithm 1, ICI is increased for buckets and their parents covering the portion corresponding to the expected precision. The smaller the required precision is, the fewer buckets get their ICI increased. For example, an estimated precision of 90% on 30NN leads to updating buckets containing the first 27 objects. This certainly requires more analysis, which we plan as the future work.

7. Conclusion and Future Work

Building on the proposal called Inverted Cache Index (ICI), we presented further analysis of its parameters and additional experimental trials on CoPhIR and Profiset data collec-

tions. The results for precise kNN evaluation proved that recording previous accesses to data partitions gives performance improvement. ICI is generally applicable to any data structure and provides gains in tens of per cent in query response time, which directly corresponds to the number of accessed data partitions.

The next contribution lies in approximate kNN query evaluation with ICI. We made a proposal of setting the values of ICI parameters that is based on intrinsic dimensionality of data, the number of data partitions created by an indexing structure to organize a data-set, and amount of queries processed within a desired period of time. Though such a proposal is initial, it provides consistently better results.

The outcome of this article motivates our future work that will focus on proposing an automatic ICI parameter selection and its deep theoretical and experimental evaluation. We will also investigate the possibility to adaptively swap between the original priority query ordering and ICI-based ordering. Such a procedure might lead to improvement of all queries, i.e. also the queries that are not similar to any previously executed.

Acknowledgements. This work was supported by Czech Science Foundation project GA16-18889S.

References

Amato, G., Rabitti, F., Savino, P., Zezula, P. (2003). Region proximity in metric spaces and its use for approximate similarity search. *ACM Transactions on Information Systems (TOIS 2003)*, 21(2), 192–227.

Antol, M., Dohnal, V. (2016). Optimizing query performance with inverted cache in metric spaces. In: *Proceedings of ADBIS 2016: 20th East-European Conference on Advances in Databases and Information Systems*, pp. 1–14. Accepted for publication. For the reviewers, it is available at https://www.fi.muni.cz/~xdohnal/adbis2016.pdf.

Barrios, J.M., Bustos, B., Skopal, T. (2014). Analyzing and dynamically indexing the query set. *Information Systems*, 45, 37–47.

Batko, M., Falchi, F., Lucchese, C., Novak, D., Perego, R., Rabitti, F., Sedmidubsky, J., Zezula, P. (2009). Building a web-scale image similarity search system. *Multimedia Tools and Applications*, 47(3), 599–629.

Beecks, C., Skopal, T., Schöffmann, K., Seidl, T. (2011). Towards large-scale multimedia exploration. In: *Proceedings of 5th International Workshop on Ranking in Databases (DBRank 2011), Seattle, WA, USA*, pp. 31–33.

Böhm, C., Berchtold, S., Keim, D.A. (2001). Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33(3), 322–373.

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

Budikova, P., Batko, M., Zezula, P. (2011). Evaluation platform for content-based image retrieval systems. In: Gradmann, S., Borri, F., Meghini, C., Schuldt, H. (Eds.), *Proceedings of International Conference on Theory and Practice of Digital Libraries, Research and Advanced Technology for Digital Libraries (TPDL)*. Springer, Berlin, pp. 130–142.

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

Ciaccia, P., Patella, M., Zezula, P. (1997). M-tree: An efficient access method for similarity search in metric spaces. In: Jarke, M., Carey, M.J., Dittrich, K.R., Lochovsky, F.H., Loucopoulos, P., Jeusfeld, M.A. (Eds.), *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB 1997), Athens, Greece, August 25–29, 1997*. Morgan Kaufmann, pp. 426–435.

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

Esuli, A. (2012). Use of permutation prefixes for efficient and scalable approximate similarity search. *Information Processing & Management*, 48(5), 889–902.

Falchi, F., Lucchese, C., Orlando, S., Perego, R., Rabitti, F. (2009). 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, EDBT '09*. ACM, New York, pp. 780–790.

Houle, M.E., Nett, M. (2015). Rank-based similarity search: reducing the dimensional dependence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(1), 136–150.

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

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T. (2014). Caffe: convolutional architecture for fast feature embedding. In: *Proceedings of the 22Nd ACM International Conference on Multimedia, MM '14*. ACM, New York, pp. 675–678.

Lew, M.S., Sebe, N., Djeraba, C., Jain, R. (2006). Content-based multimedia information retrieval: State of the art and challenges. *The ACM Transactions on Multimedia Computing, Communications, and Applications*, 2(1), 1–19.

Novak, D., Batko, M., Zezula, P. (2011). Metric index: an efficient and scalable solution for precise and approximate similarity search. *Information Systems*, 36.

Novak, D., Zezula, P. (2014). Rank aggregation of candidate sets for efficient similarity search. In: Decker, H., Lhotská, L., Link, S., Spies, M., Wagner, R.R. (Eds.), *Proceedings of 25th International Conference on Database and Expert Systems Applications (DEXA)*. Springer International Publishing, Cham, pp. 42–58.

Oliveira, P.H., Traina, C., Kaster, D.S. (2015). 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), Poitiers, France, September 8–11, 2015*. Springer International Publishing, Cham, pp. 18–31.

Pramanik, S., Li, J., Ruan, J., Bhattacharjee, S.K. (1999). Efficient search scheme for very large image databases. In: Beretta, G.B., Schettini, R. (Eds.), *Proceedings of the International Society for Optical Engineering (SPIE) on Internet Imaging, San Jose, California, USA, January 26, 2000*, Vol. 3964, The International Society for Optical Engineering, pp. 79–90.

Sadit Tellez, E., Chávez, E. (2012). The list of clusters revisited. In: Carrasco-Ochoa, J.A., Martínez-Trinidad, J.F., Olvera López, J.A., Boyer, K.L. (Eds.), *Proceedings of 4th Mexican Conference on Pattern Recognition (MCPR)*. Springer, Berlin, pp. 187–196.

Samet, H. (2006). *Foundations of Multidimensional And Metric Data Structures*. *The Morgan Kaufmann Series in Data Management Systems*. Morgan Kaufmann.

Skopal, T., Hoksza, D. (2007). Improving the performance of m-tree family by nearest-neighbor graphs. In: Ioannidis, Y., Novikov, B., Rachev, B. (Eds.), *Proceedings of 11th East European Conference on Advances in Databases and Information Systems (ADBIS), Varna, Bulgaria, September 29–October 3, 2007*. Springer, Berlin, pp. 172–188.

Skopal, T., Lokoc, J., Bustos, B. (2012). D-cache: universal distance cache for metric access methods. *IEEE Transactions on Knowledge and Data Engineering*, 24(5), 868–881.

Skopal, T., Pokorný, J., Snášel, V. (2005). Nearest neighbours search using the PM-Tree. In: *Proceedings of the 10th International Conference on Database Systems for Advanced Applications (DASFAA), Beijing, China, April 17–20, 2005*, *Lecture Notes in Computer Science*, Vol. 3453, Springer, pp. 803–815.

Vilar, J.M. (1995). Reducing the overhead of the AESA metric-space nearest neighbour searching algorithm. *Information Processing Letters*, 56(5), 265–271.

Zezula, P., Amato, G., Dohnal, V., Batko, M. (2005). *Similarity Search: The Metric Space Approach*. *Advances in Database Systems*, Vol. 32. Springer.

M. Antol is a PhD student at the Faculty of Informatics, Masaryk University in Brno, Czech Republic. His area of interest is data analytics, namely optimization of hierarchical structures and their performance in metric spaces.

V. Dohnal is an associate professor at the Faculty of Informatics, Masaryk University. He has been interested in database systems and information retrieval. In particular, indexing principles and mechanisms for unstructured data modelled as a metric space.