Information Systems 45 (2014) 37–47

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Analyzing and dynamically indexing the query set

Juan Manuel Barrios [a,b,*,1], Benjamin Bustos [b,2], Tomáš Skopal [c,3]

[a] ORAND S.A., Chile
[b] PRISMA, Department of Computer Science, University of Chile, Chile
[c] SIRET Research Group, Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic

ARTICLE INFO

Available online 13 June 2013

Keywords:
Similarity search
Metric indexing
Multimedia information retrieval
Content-based multimedia retrieval

ABSTRACT

Most of the current metric indexes focus on indexing the collection of reference. In this work we study the problem of indexing the query set by exploiting some property that query objects may have. Thereafter, we present the Snake Table, which is an index structure designed for supporting streams of *k*-NN searches within a content-based similarity search framework. The index is created and updated in the online phase while resolving the queries, thus it does not need a preprocessing step. This index is intended to be used when the stream of query objects fits a snake distribution, that is, when the distance between two consecutive query objects is small. In particular, this kind of distribution is present in content-based video retrieval systems, image classification based on local descriptors, rotation-invariant shape matching, and others. We show that the Snake Table improves the efficiency of *k*-NN searches in these systems, avoiding the building of a static index in the offline phase.

1. Introduction

In the metric approach for similarity search, the most common search methods require an example object as query. As result, the search system returns the similar objects to the query. For example, in a *k*-NN query, the search returns the *k* closest objects to the query in the data collection. For improving the efficiency of the search, the standard approach is to preprocess the data collection for building a metric access method. While the preprocessing cost for building the index may be high, this cost is amortized during the processing of the queries. Normally, each query is processed in an isolated way with respect to previous or future queries. However in some cases, it is possible to improve the efficiency by taking advantage of inherent properties of the stream of queries.

Among the possible properties of a stream of queries, an interesting one is when consecutive queries are similar to each other. This property naturally arises in applications like similarity search for videos [1], where consecutive frames of a video may be used as query object. If the query frames are taken from the same shot, it follows that consecutive queries are similar to each other. Another application is time series for shape retrieval [2], where consecutive queries correspond to the same time series but shifted according to the temporal dimension. This results in consecutive queries that are also similar. Therefore, researching techniques that exploit the similarity between query objects to increase the efficiency becomes relevant, as it may have a large impact in the aforementioned applications.

In this paper, we study the approach of preprocessing the query set and indexing it dynamically. Specifically, we present the Snake Table, which is a dynamic indexing

* Corresponding author at: ORAND S.A. Tel.: +56 222474691.
E-mail addresses: juan.barrios@orand.cl, jbarrios@dcc.uchile.cl (J.M. Barrios), bebustos@dcc.uchile.cl (B. Bustos), skopal@ksi.mff.cuni.cz (T. Skopal).

[1] This research has been supported by CONICYT Project PAI-78120426.
[2] This research has been supported by FONDEF Project D09I1185.
[3] This research has been supported in part by Czech Science Foundation project GA CR 202/11/0968.

0306-4379/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.is.2013.05.010

structure designed for supporting streams of k-NN searches. Unlike most of the metric access methods, the Snake Table is short-lived and query-object oriented. The index is intended to be used when the stream of query objects fits a "snake distribution", which we define formally in this work. We show that the snake distribution for query objects arises naturally in some problems (like in the example for video similarity search) and also in other problems can be "artificially" forced by reordering the query set. We show experimentally that processing a stream of queries with snake distribution using the Snake Table can outperform a static metric access method.

An existing indexing structure with similar objectives and properties is called the D-file [3]. In this work, we show that the D-file suffers from high internal realtime complexity making it unviable to use it in metric spaces with computationally inexpensive (i.e., fast) distance functions (like Manhattan distance or Euclidean distance). We compare the Snake Table with D-file and LAESA index, showing that Snake Table achieves the best performance when the data follows a snake distribution.

The structure of the paper is as follows. Section 2 gives a background of metric spaces and efficiency issues. Section 3 analyses the properties of a query set and presents a taxonomy for query sets. Section 4 reviews the related work, focusing on the techniques that preprocess and index the query set. Section 5 defines a snake distribution and presents the Snake Table. Section 6 evaluates the performance achieved by indexing the query sets using different scenarios. Finally, Section 7 summarizes the contributions of this work.

2. Background

Let $\mathcal{M}=(\mathcal{D},d)$ be a metric space [4]. Given a collection $\mathcal{R}\subseteq\mathcal{D}$, and a query object $q\in\mathcal{D}$, a range search returns all the objects in $\mathcal{R}$ that are closer than a distance threshold ϵ to q, and a nearest neighbor search (k-NN) returns the k closest objects to q in $\mathcal{R}$.

For improving efficiency in metric spaces, Metric Access Methods (MAMs) [5] are index structures designed to efficiently perform similarity search queries. MAMs avoid a linear scan over the whole database by using the metric properties to save distance evaluations. Given the metric space $\mathcal{M}$, the object-pivot distance constraint [4] guarantees that $\forall a,b,p\in\mathcal{D}$:

$$|d(a,p)-d(p,b)|\le d(a,b)\le d(a,p)+d(p,b) \tag{1}$$

One index structure that uses pivots for indexing is the Approximating and Eliminating Search Algorithm (AESA) [6]. It first computes a matrix of distances between every pair of objects $x,y\in\mathcal{R}$. The structure is simply an $|\mathcal{R}|\times|\mathcal{R}|$ distance matrix. In fact, only a half of the matrix needs to be stored, due to symmetry of d. The main drawback of the AESA approach is the quadratic space of the matrix. Linear AESA (LAESA) [7] gets around this problem by selecting a set of pivots $\mathcal{P}\subseteq\mathcal{R}$. The distance between each pivot to every object is calculated and stored in a $|\mathcal{R}|\times|\mathcal{P}|$ distance matrix, also known as the *pivot table*. LAESA reduces the required space compared to AESA, however an algorithm for selecting a good set of pivots is required [8].

Given a query object q (not necessarily in $\mathcal{R}$), the similarity search algorithm first evaluates the distance $d(q,p)$ for each pivot $p\in\mathcal{P}$, then scans $\mathcal{R}$, and for each $r\in\mathcal{R}$ it evaluates the lower bound function $\mathrm{LB}_{\mathcal{P}}$:

$$\mathrm{LB}_{\mathcal{P}}(q,r)=\max_{p\in\mathcal{P}}\{|d(q,p)-d(r,p)|\} \tag{2}$$

Note that $\mathrm{LB}_{\mathcal{P}}$ can be evaluated efficiently because $d(q,p)$ is already calculated and $d(r,p)$ resides in the pivot table. In the case of range searches, if $\mathrm{LB}_{\mathcal{P}}(q,r)>\epsilon$ then r can be safely discarded because r cannot be part of the search result. In the case of k-NN searches, if $\mathrm{LB}_{\mathcal{P}}(q,r)\ge d(q,o^k)$ then r can be safely discarded, where o^k is the current kth nearest neighbor candidate to q. If r is not discarded, the actual distance $d(q,r)$ must be evaluated to decide whether or not r is part of the search result.

The efficiency of some MAM in $\mathcal{M}$ is related to: (1) the number of distance evaluations that are discarded when it performs a similarity search; and (2) the internal cost for deciding whether some distance can be discarded or not. A similarity search using any MAM will be faster than a linear scan when the time saved due to the discarded distances is greater than the time spent due to the internal cost. For example, in the case of LAESA, the internal cost for a similarity search comprises the evaluation of $d(q,p)$ for each pivot p in $\mathcal{P}$, and the evaluation of $\mathrm{LB}_{\mathcal{P}}(q,r)$ for each object r in $\mathcal{R}$, thus it increases linearly with $|\mathcal{P}|$. The amount of distances discarded by LAESA depends on the metric space itself and on the size and quality of $\mathcal{P}$.

In order to analyze the efficiency that any MAM can achieve in a collection $\mathcal{R}$, Chávez et al. [5] propose to analyze the histogram of distances of d. A histogram of distances is constructed by evaluating $d(a,b)$ for a random sample of objects $a,b\in\mathcal{R}$. The histogram of distances reveals information about the distribution of objects in $\mathcal{M}$. Given a histogram of distances for $\mathcal{M}$, the intrinsic dimensionality ρ is defined as $\rho(\mathcal{M})=\mu^2/2\sigma^2$, where μ and σ^2 are the mean and the variance of histogram of distances for $\mathcal{M}$. The intrinsic dimensionality estimates the efficiency that any MAM can achieve in $\mathcal{M}$, therefore it tries to quantify the difficulty in indexing a metric space. A histogram of distances with small variance (i.e., a high value of ρ) means that the distance between any two objects $d(a,b)$ with high probability will be near μ, thus the difference between any two distances with high probability will be a small value. In that case, for most of the pivots the lower bound from Eq. (1) will become ineffective at discarding objects. Increasing the number of pivots will improve the value of the lower bounds, however the internal cost of the MAM will also increase.

3. Streams of k-NN searches

MAMs can be classified as static or dynamic depending on how they manage the insertion or deletion of objects in $\mathcal{R}$ during the online phase. A dynamic MAM can update its structures to add or remove any object, hence it can remain online even for growing collections. Usually, the tree-based MAMs are dynamic, like the M-Tree [9]. A static MAM cannot manage large updates in its structures, thus after many modifications of $\mathcal{R}$ the whole indexing

structure must be rebuilt. LAESA is able to manage the insertion or deletion of objects and pivots by adding or removing rows or columns from the pivot table [10]. Nevertheless, LAESA is usually a static index, mainly because the actual implementation of the table might not support dynamic updates to its structure. Additionally, after many modifications in $\mathcal{R}$ the set of pivots can begin to perform poorly and a new set of pivots should be selected.

In the traditional use case, MAMs are created during the offline phase, i.e., the index structure is created by a time-expensive process prior to resolving any search. This indexing process has access to $\mathcal{R}$ (or a subset of it) and usually does not have any prior information about the query objects that will be resolved afterwards. It is expected that the MAM will amortize its construction time by resolving efficiently subsequent searches. During the online phase, the MAM receives query objects potentially from different sources and users. All the searches share the same MAM, and the MAM should achieve good performance at every search.

Depending on the domain, the query set $\mathcal{Q}$ may have some special properties that can be exploited to improve the performance of the MAM. In particular, we analyze two independent criteria. First, regarding the way in which the objects in $\mathcal{Q}$ are discovered or received from the user, we differentiate three types:

1. Full: $\mathcal{Q}$ is fully determined during the offline phase. In the online phase $\mathcal{Q}$ is static.
2. Groups: $\mathcal{Q}$ is unknown during the offline phase. In the online phase the query objects are received or defined incrementally in groups. The groups do not have to be the same length.
3. ByOne: $\mathcal{Q}$ is unknown during the offline phase. In the online phase the query objects are defined incrementally one by one. The value of the ith query object may depend on the result of the $(i-1)$th search.

Second, regarding the existence of some kind of similarity between objects in $\mathcal{Q}$, we differentiate the positive and the negative case:

1. Similar: The objects in $\mathcal{Q}$ have some degree of similarity or some correlation between them. The kind of similarity is usually inherent to the nature of the domain. For example, the query objects may correspond to variations or evolution of some source model.
2. Random: The objects in $\mathcal{Q}$ are indistinguishable from a random sample of $\mathcal{R}$.

Table 1 summarizes the proposed taxonomy. The traditional use case assumes a query set type Groups/$*$ or ByOne/$*$, hence the indexing process cannot access $\mathcal{Q}$. However, if the query set is type Full/$*$ the indexing process is able to access both $\mathcal{Q}$ and $\mathcal{R}$, therefore it can create a static MAM optimized to resolve $\mathcal{Q}$. For example, LAESA could select pivots either from $\mathcal{Q}$ or $\mathcal{R}$ and evaluate the resulting lower bounds considering the query objects that will be resolved.

Table 1
Taxonomy for query sets. Columns correspond to the similarity between query objects. Rows correspond to the way the query set is discovered.

Query set taxonomy	Similar	Random
Full	e	
Groups	a,c,d	
ByOne	b	

In the following we present some domains and scenarios identifying the type of query set according to the previous taxonomy:

a In the domain of video similarity, the content-based video retrieval systems and the video copy detection systems usually divide a query video into shots, extract one or more keyframes from each shot, and perform a similarity search for each keyframe [1,11]. In this scenario, because many consecutive keyframes are extracted from a single video and consecutive keyframes frequently are similar, the query set is type Groups/Similar.

b In the case of interactive content-based multimedia retrieval systems [12–14], a user starts a search (either with some visual example, some text or tags), a k-NN search is performed and the answers are shown. Then, iteratively the user selects a new query object among those shown, and a new search is performed refining the results. Because the new queries are selected from the answers of a previous search, it can be expected that two consecutive query objects will be similar. In this case the query set is type ByOne/Similar.

c In image classification and in image retrieval based on local descriptions [15], a set of local descriptors is extracted from the query image, and for each local descriptor a similarity search is performed in the whole collection of local descriptors. Each local descriptor represents the content of a small independent patch in the image, hence it might be expected that the query set would be type Groups/Random. However, as we shown in the experimental section, because the local patches proceed from the same image they have a higher degree of similarity than patches from random images, therefore in practice the query set is type Groups/Similar.

d The time series similarity can be used to address the problem of rotation-invariant shape matching [2]. A shape can be represented by a time series (e.g., as the distance of the shape contour from the shape centroid, sampled clock-wise). In order to obtain rotation-invariant matching with a query shape (rotated arbitrarily), a sequence of time series is created by using a sliding window on the query series. As consecutive time series coming from the sliding window vary only slightly, they are very similar, hence the query set is type Groups/Similar.

e In image forgery detection a typical approach consists in detecting the “copy-move” forgery [16]. This technique focuses on determining whether some patch of the image has been duplicated and pasted into another part

of the same image. This kind of forgery can be detected by computing descriptors using an overlapping sliding window on the image and then searching for duplicated windows. The query set produced by this technique can be classified as type Full/Similar.

The existence of these scenarios shows the relevance of considering the query set beyond the traditional use case, we specifically focus on query sets with some degree of internal similarity (type ∗/Similar). In these query sets, it can be expected that two consecutive query objects are similar, i.e., the distance between the ith and $(i+1)$th objects in $\mathcal{Q}$ is small compared with the median distance between objects in $\mathcal{R}$. In order to profit from this internal similarity we propose to resolve $\mathcal{Q}$ sequentially using a dynamic MAM, hence the distances evaluated while resolving the first i searches can potentially be used to save computation during the $(i+1)$th search. In particular, we propose a dynamic pivot table that stores a subset of the distances computed during previous searches to improve the performance of the current search.

4. Related work

In this section we review a few existing approaches that consider batches or streams of similarity queries.

4.1. Batch of queries

A general approach for improving the efficiency of MAMs for streams of k-NN searches is the simultaneous processing of multiple queries [17]. Instead of issuing many single queries, the idea is to process a batch of similarity queries aiming at reducing I/O cost and computation cost. The proposed technique reduces the I/O cost by reading each disk page only once per batch of similarity queries, and it reduces the CPU cost by avoiding distance computations. An avoidable distance computation is detected by computing the distances between query objects and then using these distances, together with the triangle inequality, to compute lower bounds of the distances between queries and database objects. If the lower bound distance is greater than a given tolerance radius for the similarity search, then the distance calculation is avoidable. The proposed technique is general, and it can be implemented based on a MAM or using a sequential file. However, besides the requirement to know all the queries beforehand (i.e., type Full), it also requires computing the distances between each pair of query objects to reduce the CPU cost, and it does not take advantage of evaluated distances between queries and database objects.

Paredes et al. [18] propose an approach to compute the k nearest neighbor graph in metric spaces which is equivalent to compute n k-NN queries, in (empirical) subquadratic time. However, the set of query objects was restricted to the database objects which is only marginally meaningful for this work.

4.2. Query result caching

In order to speed up the similarity searches, another approach provides a mechanism of caching query results [19,20]. Basically, the *metric cache* stores a history of similarity queries and their answers (identifiers and descriptors of database objects returned by the query). When a subsequent query is to be processed, the metric cache either returns the exact answer in case the same query was processed in the past and its result still sits in the cache. Or, in case of a new query, such old queries are determined from the metric cache, that spatially contain the new query object inside their query balls. If the new query is entirely bounded by a cached query ball, a subset of the cached query result is returned as an exact answer of the new query. If not, the metric cache is used to combine the query results of spatially close cached queries to form an approximate answer. In case the approximate answer is likely to exhibit a large retrieval error, the metric cache gives up and forwards the query processing to the underlying MAM (updating the metric cache by the query answer afterwards).

4.3. D-file and D-cache

To take advantage of the online indexing process and a stream of correlated queries, there is a recently proposed structure called D-file [3]. The D-file is the database file itself accompanied by a main-memory structure, called the D-cache. The D-cache stores a sample of every evaluated distance between query and database objects. When the q_n object in $\mathcal{Q}$ is processed, the D-cache evaluates the distance between q_n and a previous q_i in $\mathcal{Q}$, treats q_i as a pivot and calculates a lower-bound distance for $d(q_n, o_j)$. Hence, if the lower bound is large enough, o_j can be discarded without evaluating the actual distance to q_n. D-cache content is modeled as a sparse dynamic pivot table, where each table row is constructed with the stored distances. If there are not enough distances stored in the D-cache, some rows are incomplete, resulting in zeros on some cells. Using the reconstructed rows, the D-file tries to filter out each database object using the same approach as a regular pivot table. The D-file does not need an offline indexing step, as the D-cache is being built during query processing. As the D-cache uses the previously processed queries as dynamic pivots, the authors recommend that previous queries should be as close to the current query as possible.

The D-cache consists of: (1) a fixed-size hash table that stores triplets $(q_i, o_j, d(q_i, o_j))$; (2) a hash function $h(q_i, o_j)$ for accessing the bucket where a triplet is stored; (3) a collision interval, for searching a near available bucket when some triplet is mapped into an already used bucket; and (4) a replacement policy, that decides whether or not a new triplet should replace an old triplet when a collision occurs and there is not an available bucket in the collision interval.

The D-file is a dynamic structure that can be used for indexing queries type Groups/Similar and ByOne/Similar. However, as we show in the experimental section, the D-file suffers from high internal complexity. The main

problem arises when the distance function is not time-expensive. In that case, the internal complexity associated with the hash function and collision resolution dominates the search times rendering it unviable to use in many scenarios. In order to solve this problem, we introduce the Snake Table that preserves the idea and advantages of D-file and D-cache, but exhibits lower internal complexity.

5. Snake Table

In this work we propose a new dynamic indexing structure called Snake Table, which is designed to: (1) improve the search time for streams of queries where consecutive objects in the query set are similar (i.e., for query sets ∗/Similar); and (2) have low internal complexity in order to worth indexing metric spaces based on fast distance functions. We should stress that by *stream of queries* we mean a query set with unknown length whose query objects are incrementally discovered (either one by one or in groups) during the online phase.

The life cycle of the Snake Table is as follows: First, when a new session starts, an empty Snake Table is created and associated with it. When a query object q_1 is received, a k-NN search is performed. The distances between q_1 and the objects in the collection are added to the Snake Table, and the result is returned. When a new query object q_i is received, a k-NN is performed using the previous query objects $q_{i-p}, \ldots, q_{i-1}$ as pivots to accelerate the search. Finally, when the session ends, the Snake Table is discarded. Therefore, like D-file and unlike most of MAMs, the Snake Table is a session-oriented and short-lived MAM, see Fig. 1.

The Snake Table is implemented with a fixed-size $|\mathcal{R}| \times p$ matrix used as a dynamic pivot table. As in LAESA, the jth row in the dynamic pivot table represents the object o_j in $\mathcal{R}$ and contains the distances between o_j and up to p previously processed query objects. Each cell in the jth row of the table contains a pair $(q, d(q, o_j))$ for some query object q (not necessarily in order). When processing a new query object q_i, the lower bound $\mathrm{LB}_{\mathcal{P}}(q_i, o_j)$ is calculated (see Eq. (2)), with $\mathcal{P}$ dynamically determined by the query objects and distances in the jth row. As in the traditional exact search, the object o_j is discarded when $\mathrm{LB}_{\mathcal{P}}(q_i, o_j)$ is greater than the distance between q_i and the current kth nearest neighbor candidate (obtained between o_1 and o_{j-1}). If o_j is not discarded, the actual distance $d(q_i, o_j)$ is computed, added to some cell in the jth row, and the NN candidates are updated if necessary.

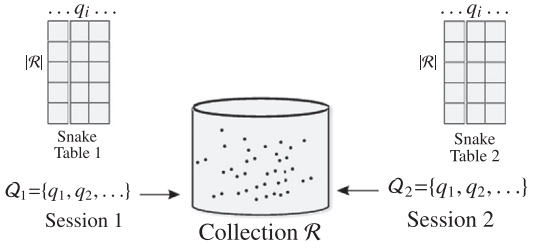

Fig. 1. Snake tables created for stream of queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$.

We present three different replacement strategies to assign a distance $d(q_i, o_j)$ to one of the p cells in the jth row:

1. FIFO/Sparse: Each query q_i picks a column in round-robin mode, i.e., the distance $d(q_i, o_j)$ is stored in the (i mod p) column of jth row, eventually replacing the stored distance $d(q_{i-p}, o_j)$. If the distance was not evaluated because it was discarded by $\mathrm{LB}_{\mathcal{P}}(q_i, o_j)$ then the corresponding cell is not used. This behavior can be implemented following two options: (1) the cell is updated with an ∞ distance; or (2) the cell is left unmodified, but before any read operation the query stored in the cell is matched with the last query for that column (the experimental section uses the latter). This strategy produces sparse rows containing at most p distances between $d(q_{i-p}, o_j)$ and $d(q_i, o_j)$. As a consequence, if $p=1$ and most of the distances for q_{i-1} were discarded, then q_i will achieve poor efficiency. In order to diminish this "min-max effect" a larger p or a different strategy should be chosen.
2. Highest/Compact: The distance $d(q_i, o_j)$ is compared to every distance in the jth row and the lowest distance is replaced, independently of its position in the row. With this strategy, every row stores the highest p distances between $d(q_1, o_j)$ and $d(q_i, o_j)$ that have not been discarded.
3. FIFO/Compact: The distance $d(q_i, o_j)$ is stored in a cell chosen in an independent round-robin for every row. With this strategy, the jth row stores the last p computed distances for o_j, discarding the old ones. $\mathrm{LB}_{\mathcal{P}}$ starts its evaluation from the last stored distance and goes backwards, therefore favoring the most recent stored distances.

For strategy FIFO/Sparse, distances $d(q_i, q_j)$ with $j \in \{i-p, \ldots, i-1\}$ are calculated and stored in memory at the beginning of every search. For strategies Highest/Compact and FIFO/Compact, distances $d(q_i, q_j)$ with $j \in \{1, \ldots, i-1\}$ are calculated on-demand by $\mathrm{LB}_{\mathcal{P}}$. Note that the internal complexity of strategy FIFO/Compact is slightly higher than strategy FIFO/Sparse because it needs to manage an independent index for each row to mark the position of the last stored distance.

D-file uses a combination of FIFO/Sparse and Highest/Compact. It always replaces an old distance (older than q_{i-p}), but if there is no old distance in the collision interval, it replaces the worst distance (defined as the distance closer to the median or some predefined percentile).

The performance achieved by these three replacement strategies is compared in the experimental section. However, despite the replacement strategy used by the Snake Table, the overall performance of the Snake Table mainly depends on the distribution of the query objects.

5.1. Snake distribution

The Snake Table is intended to be used when the query objects in a stream fit a "snake distribution". Intuitively, we define that a set of objects fits a snake distribution

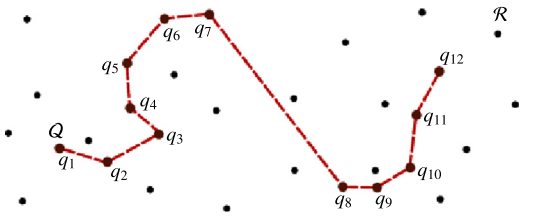

Fig. 2. Stream of queries $\mathcal{Q}=\{q_1,\ldots,q_{12}\}$ with a snake distribution: most of distances $d(q_i,q_{i+1})$ are smaller than $d(x,y)$ for randomly selected pairs x, y in $\mathcal{R}$.

when the distance between two consecutive objects in $\mathcal{Q}$ is small compared to the median distance between any two objects (see Fig. 2). To measure and compare this fit, we define an indicator using the histogram of distances of d for $\mathcal{Q}$ and $\mathcal{R}$.

Because the area of the histogram of distances is normalized to 1, the histogram can be seen as a probability distribution of the distances calculated by d. Then, we define the cumulative distribution in a similar way as in probabilities:

Definition 1 (*Cumulative distribution*). Let H be a normalized histogram of distances, the Cumulative Distribution of Distances $F:\mathbb{R}^+\rightarrow[0,1]$ is defined as

$$F(x)=\int_0^x H(t)\,dt$$

For comparing the distribution of distances of two sets of objects, we compare their cumulative distributions:

Definition 2 (*Difference* Δ). Let F_1 and F_2 be two cumulative distributions, the difference Δ between F_1 and F_2 is defined as

$$\Delta(F_1,F_2)=\int_0^\infty F_1(t)-F_2(t)\,dt$$

The difference Δ is meaningful only when both F_1 and F_2 originate from the same metric space. Note that $\Delta(F_1,F_2)$ is greater than zero when the distances accumulated by F_1 are smaller than distances accumulated by F_2.

Definition 3 (*Snake distribution*). Let $\mathcal{M}=(\mathcal{D},d)$ be a metric space, let $\mathcal{R}\subset\mathcal{D}$ be the collection of objects, and let $\mathcal{Q}\subset\mathcal{D}$ be a set of m query objects $\mathcal{Q}=\{q_1,\ldots,q_m\}$. Let F be the cumulative distribution of $d(x,y)$ with random pairs $x,y\in\mathcal{Q}\cup\mathcal{R}$, p be a number between 1 and $m-1$, and $F_{\mathcal{Q}}^p$ be the cumulative distribution of $d(q_i,q_{i-p})\ \forall\ i\in\{p+1,\ldots,m\}$. $\mathcal{Q}$ fits a snake distribution of order p if $\Delta(F_{\mathcal{Q}}^p,F)>s$, for some threshold value $s\in\mathbb{R}^+$.

Note that when both $\mathcal{Q}$ and $\mathcal{R}$ are random samples of $\mathcal{D}$ without any special ordering (i.e., the ith sample does not depend on previous samples), then $\Delta(F_{\mathcal{Q}}^p,F)\approx 0$. When a distribution fits a Snake Distribution of order 1 to p then a Snake Table can be created with a sliding window containing up to p query objects.

5.2. Re-ordering the query set

In cases when a set of query objects is a priori known or is discovered in groups, and the snake distribution is not strong enough (i.e., a query set Full/Random or Groups/Random), one technique usually worth trying is to apply some reordering to the query set in order to produce or enhance the similarity between consecutive queries.

Following this idea, we propose a simple approach which consists in reordering $\mathcal{Q}$ by consecutively selecting the first nearest neighbor, as depicted in Algorithm 1. We should note that the reordering approach is affordable only when $|\mathcal{Q}|\ll|\mathcal{R}|$, otherwise the reordering step may turn out to be more expensive than directly resolving $\mathcal{Q}$.

Algorithm 1. Reordering algorithm based on consecutive Nearest Neighbors.

Input: $\mathcal{Q}=\{q_1,\ldots,q_m\}$ an unordered set of query objects.
Output: $\mathcal{Q}'=(q'_1,\ldots,q_m)$ an ordered list of query objects.
$\mathcal{Q}'\leftarrow$ empty list;
object$\leftarrow q_1$; // `seed object`
add object to $\mathcal{Q}'$;
$\mathcal{Q}\leftarrow\mathcal{Q}-\{\text{object}\}$;
while $\mathcal{Q}\neq\emptyset$ **do**
  $q_j\leftarrow NN(\text{object},\mathcal{Q})$; // object's `NN in` $\mathcal{Q}$
  add q_j at the end of $\mathcal{Q}'$;
  $\mathcal{Q}\leftarrow\mathcal{Q}-\{q_j\}$;
  object$\leftarrow q_j$;
return $\mathcal{Q}'$;

6. Experimental evaluation

In this section we compare the performance achieved by the Snake Table using the three replacement strategies, with the performance achieved by D-file and LAESA under different scenarios.

6.1. Video copy detection scenario

We tested the Snake Table on our frame-based CBVCD system [1]. The MUSCLE-VCD-2007 [21] is a publicly available and widely-used video copy dataset. The reference collection contains 101 videos with 59 h total length, and the query collection (ST2) contains three videos with 45 min total length.

In this evaluation, each reference and query video is partitioned into short fixed-length segments of 1 s. For each segment, four global descriptors are calculated: the Edge Histogram (EH), captures the spatial distribution of edges in a frame [22]. We used 10 orientations and 8-bits linear quantization, producing a vector of 160 bytes. The Ordinal Measurement (OM), captures the spatial distribution of intensities in a frame [11]. We used 9×9 blocks, producing a vector of 81 bytes. The Keyframe (KF), reduces the frame to 11×9 pixels and uses the value for each pixel, producing a vector of 99 bytes. These descriptors are calculated for all the frames in a segment and then averaged. Finally, the set of query segments $\mathcal{Q}$ contains 2692 objects, and the set of reference segments $\mathcal{R}$ contains 211,479 objects.

6.1.1. Configurations

We present three configurations, each one defining a distance function $d(r,s)$ between the video segments r and s. The distances are based on linear combinations of L_1 (Manhattan) distance between descriptors, where $L_1(\vec{x},\vec{y}) = \sum_{i=1}^{n} |x_i - y_i|$ for n-dimensional vectors $\vec{x}$ and $\vec{y}$:

1. **OM**: compares OM descriptors $d(r,s) = L_1(\mathrm{OM}(r), \mathrm{OM}(s))$.
2. **EH**: compares EH descriptors $d(r,s) = L_1(\mathrm{EH}(r), \mathrm{EH}(s))$.
3. **EK3**: temporal combination of EH and KF descriptors:

$$g(r,s) = 0.5 \cdot L_1(\mathrm{EH}(r), \mathrm{EH}(s)) \cdot \frac{1}{7996} + 0.5 \cdot L_1(\mathrm{KF}(r), \mathrm{KF}(s)) \cdot \frac{1}{24,721}$$

$$d(r_i, s_j) = \frac{1}{3}[g(r_{i-1}, s_{j-1}) + g(r_i, s_j) + g(r_{i+1}, s_{j+1})]$$

where constants 7996 and 24,721 are the maximum distances for EH and KF, respectively, and r_{i-1} and r_{i+1} are the previous and the next segments of r_i in a video.

We compare the efficiency of six indexes with p pivots (static pivots for LAESA and dynamic pivots for D-file and the Snake Table), where p varies between 1 and 20:

1. **D-file**: It uses a D-cache with a fixed size hash table of $|\mathcal{R}|*p$ cells, collision interval 1, and hash function $h(q_i, o_j) = (rnd_i * rnd_j) \bmod (|\mathcal{R}|*p)$, where rnd_i and rnd_j are unique random IDs assigned to each object.
2. LAESA: The selection of static pivots follows the SSS algorithm [8] and assumes the query set is Full/*. Four different sets are selected and the average value of $LB_{\mathcal{P}}$ is calculated for each one by sampling pairs from $\mathcal{Q} \times \mathcal{R}$. The set of pivots with higher average $LB_{\mathcal{P}}$ is selected while the other sets are discarded. **LaesaQ** chooses p static pivots from $\mathcal{Q}$. **LaesaR** chooses p static pivots from $\mathcal{R}$.
3. Snake Table: We test the three strategies depicted in Section 5. **SnakeFS** uses FIFO/Sparse strategy (sparse row with the last p queries), **SnakeHC** uses Highest/Compact strategy (compact row containing p prior query in any order), and **SnakeFC** uses FIFO/Compact strategy (compact row with the last p evaluated distances for each object).

Table 2 shows some indicators for the metric space defined by each configuration, including the total time spent by a linear scan (in seconds). The histogram of distances was created by evaluating $d(x,y)$ with pairs x, y sampled from $\mathcal{Q} \cup \mathcal{R}$. If we compare the time spent by a linear scan, **OM** takes less amount of time, **EH** takes about twice as much time as **OM**, and **EK3** is slower by one order of magnitude than **OM**. In the following experiments, the performance of each index is presented as a ratio with the performance of the linear scan for that configuration.

Table 2
Metric spaces for the video copy detection scenario.

Configuration	**Time** (s)	μ	σ	ρ	H_d
OM	282	1489	416	6.4	
EH	541	3198	751	9.1	
EK3	2214	0.347	0.08	10.2	

6.1.2. Snake distribution

Fig. 3 analyzes the snake distribution for the three configurations by showing the difference $\Delta(F^p_{\mathcal{Q}}, F)$ for $p \in \{1, \ldots, 20\}$. The three configurations present a difference Δ higher than zero, hence the streams of queries have a snake distribution (distances between q_i and q_{i-p} are smaller than distances between random sampled pairs). The first orders show a good fit for the three configurations, but as p increases, the snake distribution tends to disappear. As shown in the following experiments, the different configurations present satisfactory results for roughly between 1 and 5 pivots.

6.1.3. Performance comparison

Fig. 4 shows the efficiency achieved by the six indexes for **OM**, **EH**, and **EK3** configurations varying the number of pivots from 1 to 20. It shows the amount of distance evaluations as a proportion of the evaluations required by the linear scan (i.e., a fraction of $|\mathcal{Q}|*|\mathcal{R}|$). This value includes the distances between query and pivots, but does not include the distance required for pivot selection in LAESA. It also shows the search time as a proportion of the time spent by a linear scan. The instability in LAESA indexes for consecutive p is due to the random-base pivot selection. In order to reduce this effect we calculated three different sets of pivots with SSS and the average values for evaluated distances and search time are presented.

In the case of static pivots, **LaesaQ** shows an almost identical performance as **LaesaR** for the three configurations, hence knowing a priori the set of queries does not produce a noticeable improvement in the quality of the index. This behavior shows that a static index cannot profit from snake distributions. LAESA achieves good performance for **OM**, which is a configuration with low intrinsic dimensionality, but the performance decreases for **EH** and **EK3**. These two configurations have higher intrinsic

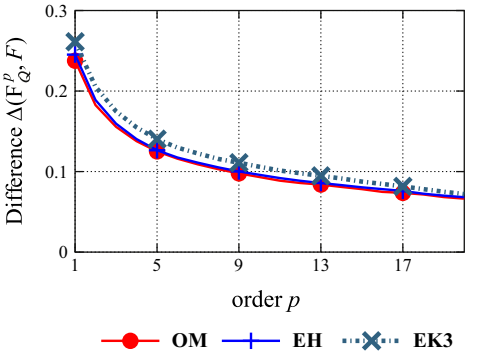

Fig. 3. Snake distribution of order $p \in \{1, \ldots, 20\}$ for the three configurations.

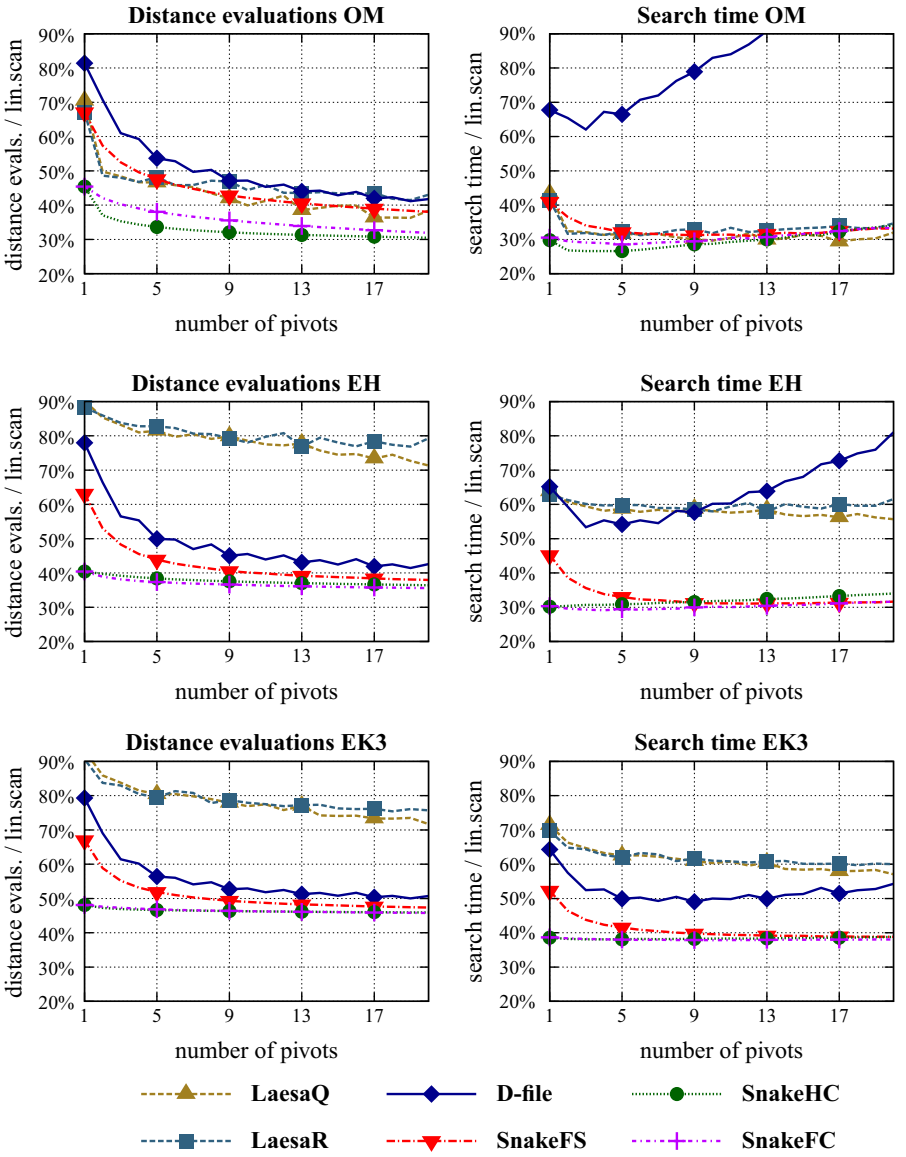

Fig. 4. Comparison of indexes performance (distance evaluations and search times) for three configurations (**OM**, **EH**, and **EK3**). This experiment uses the MUSCLE-VCD-2007 dataset.

dimensionality than **OM**, implying that any static selection of pivots will achieve worse performance. In fact, increasing the number of pivots from 5 to 20 pivots produces almost no improvement on the performance even though the required memory space increases four times.

In the case of **D-file**, the disparity between saved distances and saved time reveals that it suffers from high internal complexity. In **OM** and **EH**, it can discard most of the distance computations, but the search time increases even beyond the time required by a linear scan. In **EK3**, due to the more expensive distance, the saved computations pay for the internal complexity and **D-file** decreases its search times compared with linear scan, even outperforming LAESA.

In the case of the Snake Table, it achieves the best performance due to its good pivot selection and low internal complexity. There is not a clear difference in performance between **SnakeHC** and **SnakeFC**. However, the "min-max" effect described in Section 5 for FIFO/Sparse strategy becomes apparent: **SnakeFS** with one and two pivots cannot achieve high performance because every discarded distance implies an empty cell in the pivot table, which affects the performance for subsequent pivots. This undesired effect decreases as the number of pivots increases. On the other hand, Highest/Compact and FIFO/Compact strategies do not suffer the "min-max" effect because a compact pivot table prevents harming the performance with empty cells.

In summary, this experiment shows that the exploitation of snake distributions is a remarkable approach for improving the efficiency: while LAESA struggles with high intrinsic dimensionality, discarding about 25% of distance

computations in **EH** and **EK3**, the Snake Table and D-file discard more than 50% of distance computations with just a few pivots. However, D-file fails at transferring those savings into faster searches due to its high internal complexity. The graphs also show that the Snake Table achieves better performance with just a few pivots. In fact, a large window size p mostly adds redundant pivots, hence discarding almost no new objects and worsening the search times.

6.2. Image similarity using local descriptors

In this experiment, we test the Snake Table in the scenario of similarity search based on local descriptors. Given a collection of images, the local descriptors are extracted from every image, and the whole set corresponds to $\mathcal{R}$. Given a query image, its local descriptors $\{l_1, \ldots, l_n\}$ are calculated, and for every l_i a similarity search is performed in $\mathcal{R}$ for $i \in \{1, \ldots, n\}$. This is a straightforward approach used by some image classification systems [15] and object recognition systems [23].

The Pisa Dataset [24] is a publicly available image classification dataset. It comprises 1127 images of 12 constructions located in Pisa, Italy. The images were crawled from Flickr and have a maximum resolution of 500×500 pixels. Using the extraction software in [25] we extracted for each image many 192-d color SIFT descriptors using the Hessian-Laplace detector. The whole dataset produced near 470,000 vectors with an average of 418 vectors per image. This experiment works as follows: first, it chooses one image I at random and defines $\mathcal{Q}$ as the local descriptors of I (removing them from $\mathcal{R}$); then, it resolves the $|\mathcal{Q}|$ similarity searches using LAESA and the Snake Table; and finally, it compares the number of computed distances and search time against the linear scan. This process is repeated several times for different query images and the results are averaged.

The result for this experiment is summarized in Fig. 5. Seven indexes were used to resolve the searches: **LaesaR** chooses p static pivots from $\mathcal{R}$ using the SSS algorithm. **SnakeFS-RND**, **SnakeHC-RND**, and **SnakeFC-RND** correspond to the Snake Table using the three strategies depicted in Section 5 and processing $\mathcal{Q}$ in random order; and **SnakeFS-NN**, **SnakeHC-NN**, and **SnakeFC-NN** correspond to the Snake Table using the three replacement strategies and processing reordering $\mathcal{Q}$ according to the algorithm depicted in Section 5.2.

The graphs show that static pivots in $\mathcal{R}$ can be outperformed by dynamic pivots. The fact that **Snake*XX*-RND** indexes outperform **LaesaR** proves that query sets have some degree of internal similarity, i.e., the local descriptors from the same image are more similar than local descriptors from random images. Moreover, **Snake*XX*−NN** indexes outperform **Snake*XX*−RND** because the reordering applied to query objects enhances the snake distribution in the query stream. In particular, the fastest search times are achieved by **SnakeHC-NN** and **SnakeFC-NN** using a single dynamic pivot, both reducing search times by more than 50%. In order to achieve this improvement it is necessary to consider the cost of reordering query sets: a reordering implemented with linear scans requires $|\mathcal{Q}| \cdot (|\mathcal{Q}|-1)/2$ distance computations which is usually negligible compared to the saving in the search (some fraction of $|\mathcal{Q}| \cdot |\mathcal{R}|$). For instance, in this experiment the reordering needed 87,153 distance evaluations, while discarding 10% of the distances of the linear scan implies 20 million evaluations less.

6.3. Image similarity using global descriptors

The following experiment tests the existence of snake distributions on near-random data. The MIRFLICKR-1M [26] is a widely known image dataset consisting of one million images downloaded from Flickr website. In this experiment we extracted a global visual descriptor from the content of each image. The extracted global descriptor is a color histogram by zones: it divides the image into 3×3 zones, and for each zone it calculates an RGB histogram of $3 \times 3 \times 3$ bins, i.e., the RGB color space is divided into 27 regular regions, and the pixels for each

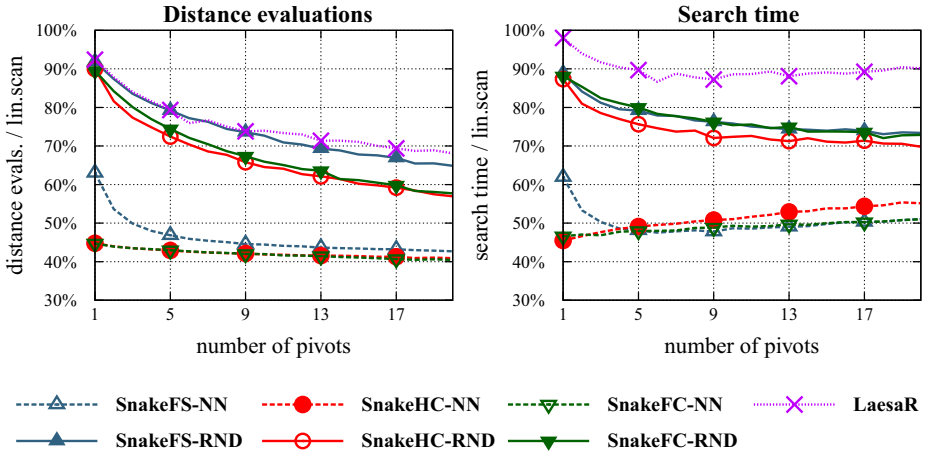

Fig. 5. Distance evaluations and search times for local descriptors, comparing the effect of reordering query sets. This experiment uses Pisa Dataset.

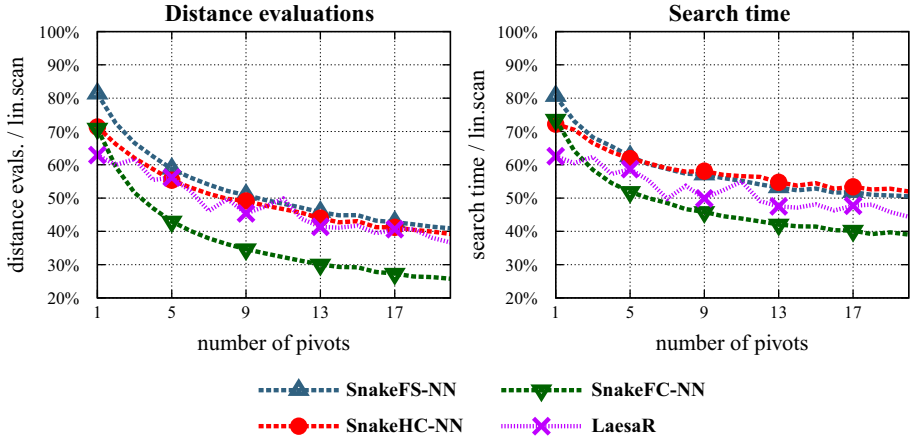

Fig. 6. Distance evaluations and search times for reordering query sets for a random sample of global descriptors. This experiment uses MIRFLICKR-1M dataset.

color region and zone are counted. The final descriptor is a 243-d vector which is compared with L_1 distance.

The experiment extracts a random sample of images and produces the query dataset $\mathcal{Q}$ and the rest of the images define $\mathcal{R}$. Then, a similarity search is performed for each query object in order to retrieve its nearest neighbor from $\mathcal{R}$. The $\mathcal{Q}$ similarity searches are resolved using four different index structures: **LaesaR**, **SnakeFS-NN**, **SnakeHC-NN**, and **SnakeFC-NN**, i.e., the Snake Table with the three replacement strategies using the reordering of the query set. As in the previous experiments, the number of distance evaluations and the search time is presented as a fraction of the linear scan. This process is repeated several times with random samples of $|\mathcal{Q}| = 1000$ images.

Fig. 6 summarizes the results. In general, the graphs show that Snake Table with reordering of queries behaves relatively similar to LAESA. This is an expected result because $\mathcal{Q}$ is just a random sample of $\mathcal{R}$, thus the query set does not show a strong snake distribution to profit from. However, an interesting behavior occurs with the redundancy of pivots: in the previous experiments, the Snake Table with $p > 10$ showed just increases in search times due to little savings in distance computations. On the other hand, in this experiment, even considering $p = 20$, the Snake Table does not show that redundancy behavior, hence the reordering of random objects produces a weak snake distribution. Despite this, **SnakeFC-NN** is able to outperform **LaesaR** by about 10% in distance computations and about 5% in search time.

This result shows the potentialities of dynamically indexing the query set: even in random samples of query objects, a reordering of the query set produces a weak snake distribution from which the Snake Table may profit from, achieving a performance that can outperform static pivots.

7. Conclusions and future work

In this work we have studied the problem of indexing the query set instead of the reference set. Following this approach, we presented the Snake Table, which is a dynamic pivot table for query objects that achieves high performance at processing streams of queries with snake distribution. This satisfactory performance is due to its properties of dynamic selection of good pivots and low internal complexity. The Snake Table is able to reduce the search time for both fast and time-expensive distances, even in spaces with high intrinsic dimensionality. In particular, the Snake Table is a better alternative to LAESA and D-file in the tested scenarios.

The Snake Table presents an approach to index spaces when consecutive queries are similar to each other. This behavior usually arises in content-based video retrieval (when the queries are consecutive keyframes), interactive multimedia retrieval systems (when the user selects a new query object from the answers of a previous query), and similarity searches using local descriptors. In a more general domain, given an unsorted set of queries, the test of snake distribution presented in this work may be useful to determine an optimal ordering of queries which will achieve a high performance in the Snake Table.

One usage of the Snake Table is to create an index for each stream of queries. When a user starts a session, an empty Snake Table is associated with it. As the user performs queries with snake distribution, the index improves its performance because it will select pivots close to following queries. However, the Snake table is not memory efficient as it requires space proportional to the size of the dataset and to the number of sessions connected. This approach is more suitable for medium-sized databases with long k-NN streams. Moreover, because it does not need to use a central shared index structure, it is also suitable for highly dynamic datasets.

On one hand, pivots in a sliding window with snake distribution satisfy one desirable property: they should be close to either the query or the collection objects. On the other hand, those pivots do not satisfy another desirable property: they should be far away from each other. Hence, using a Snake Table with many pivots will only increase the internal complexity without increasing the efficiency because pivots will be mostly redundant. In order to overcome this issue, the Snake Table and LAESA can

seamlessly be combined by with a unique pivot table containing both static and dynamic pivots. Moreover, the SSS algorithm can also be combined with the Snake Table by fixing one pivot when it is far away from all the previous ones. This combined approach enables to profit from both dynamic pivots close to queries and non-redundant static pivots.

LAESA can benefit from multi-core architectures by sharing the pivot table and resolving each query in different threads. In the case of Snake Table, in order to efficiently resolve parallel queries we recommend partitioning the queries into independent subsets, and resolving each subset by a Snake Table in an independent thread.

References

[1] J.M. Barrios, B. Bustos, Competitive content-based video copy detection using global descriptors, Multimedia Tools and Applications 62 (1) (2013) 75–110.

[2] E.J. Keogh, L. Wei, X. Xi, S.-H. Lee, M. Vlachos, Lb_keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures, in: VLDB, 2006, pp. 882–893.

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

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

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

[6] E. Vidal, New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (AESA), Pattern Recognition Letters 15 (1) (1994) 1–7.

[7] M. Micó, J. Oncina, E. Vidal, A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements, Pattern Recognition Letters 15 (1) (1994) 9–17.

[8] B. Bustos, O. Pedreira, N. Brisaboa, A dynamic pivot selection technique for similarity search, in: Proceedings of the International Workshop on Similarity Search and Applications (SISAP'08), IEEE, 2008, pp. 105–112.

[9] P. Ciaccia, M. Patella, P. Zezula, M-tree: an efficient access method for similarity search in metric spaces, in: Proceedings of the International Conference on Very Large Databases (VLDB'97), Morgan Kauffman, 1997, pp. 426–435.

[10] L. Micó, J. Oncina, A constant average time algorithm to allow insertions in the LAESA fast nearest neighbour search index, in: Proceedings of the International Conference on Pattern Recognition (ICPR'10), IEEE, 2010, pp. 3911–3914.

[11] C. Kim, B. Vasudev, Spatiotemporal sequence matching for efficient video copy detection, IEEE Transactions on Circuits and Systems for Video Technology 15 (1) (2005) 127–132.

[12] J.M. Barrios, B. Bustos, Text-based and content-based image retrieval on Flickr: demo, in: Proceedings of the International Workshop on Similarity Search and Applications (SISAP'09), IEEE, 2009, pp. 156–157.

[13] M. Batko, V. Dohnal, D. Novak, J. Sedmidubsky, Mufin: a multi-feature indexing network, in: Proceedings of the International Workshop on Similarity Search and Applications (SISAP'09), IEEE, 2009, pp. 158–159.

[14] M. Batko, F. Falchi, C. Lucchese, D. Novak, R. Perego, F. Rabitti, J. Sedmidubsky, P. Zezula, Building a web-scale image similarity search system, Multimedia Tools and Applications 47 (2010) 599–629.

[15] G. Amato, F. Falchi, C. Gennaro, Geometric consistency checks for KNN based image classification relying on local features, in: Proceedings of the International Workshop on Similarity Search and Applications (SISAP), ACM, 2011, pp. 81–88.

[16] G. Muhammad, M. Hussain, G. Bebis, Passive copy move image forgery detection using undecimated dyadic wavelet transform, Digital Investigation 9 (1) (2012) 49–57.

[17] B. Braunmüller, M. Ester, H.-P. Kriegel, J. Sander, Multiple similarity queries: a basic DBMS operation for mining in metric databases, IEEE Transactions on Knowledge and Data Engineering 13 (1) (2001) 79–95.

[18] R. Paredes, E. Chávez, K. Figueroa, G. Navarro, Practical construction of k-nearest neighbor graphs in metric spaces, in: Proceedings of the 5th International Workshop on Experimental Algorithms (WEA'06), Lecture Notes in Computer Science, vol. 4007, Springer, 2006, pp. 85–97.

[19] F. Falchi, C. Lucchese, S. Orlando, R. Perego, F. Rabitti, A metric cache for similarity search, in: LSDS-IR '08: Proceedings of the 2008 ACM Workshop on Large-Scale Distributed Systems for Information Retrieval, ACM, 2008, pp. 43–50.

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

[21] J. Law-To, A. Joly, N. Boujemaa, MUSCLE-VCD-2007: a live benchmark for video copy detection, ⟨http://www-rocq.inria.fr/imedia/civr-bench/⟩, 2007.

[22] B.S. Manjunath, J.-R. Ohm, V.V. Vasudevan, A. Yamada, Color and texture descriptors, IEEE Transactions on Circuits and Systems for Video Technology 11 (6) (2001) 703–715.

[23] J.M. Barrios, B. Bustos, Prisma-orand team: instance search based on parallel approximate searches, in: TRECVID, NIST, 2012.

[24] Pisa landmarks dataset, ⟨http://www.fabriziofalchi.it/pisaDataset/⟩, 2011.

[25] Feature detection code, ⟨http://www.featurespace.org/⟩, 2010.

[26] Mirflickr, ⟨http://press.liacs.nl/mirflickr/⟩, 2008.