

Snake Table: A Dynamic Pivot Table for Streams of k -NN Searches

Juan Manuel Barrios¹, Benjamin Bustos¹, and Tomáš Skopal^{2,*}

¹ KDW+PRISMA, Department of Computer Science, University of Chile
`{jbarrios, bebustos}@dcc.uchile.cl`

² SIRET Research Group, Faculty of Mathematics and Physics,
Charles University in Prague
`skopal@ksi.mff.cuni.cz`

Abstract. We present the Snake Table, 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, when the set of query objects are consecutive frames from a query video. 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.

Keywords: Similarity Search, Metric Indexing, Multimedia Information Retrieval, Content-Based Video Retrieval.

1 Introduction

In this paper we present the Snake Table, which is an indexing structure designed for supporting streams of k -NN searches. The index is intended to be used when the stream of query objects fits a “snake distribution”, which we define formally in this work. This kind of distribution is usually present in content-based video retrieval systems, when the set of query objects corresponds to consecutive frames from a query video. In particular, we evaluate the Snake Table on a Content-based Video Copy Detection (CBVCD) system where the query objects present a snake distribution. Unlike most of the index structures, the Snake Table is a session-oriented and short-lived index.

An existing indexing structure with similar objectives and properties is called the D-cache [9]. In this work, we show that the D-cache suffers from high internal realtime complexity making it unviable to use in a CBVCD system or other systems with computationally inexpensive (i.e., fast) distance functions (like

* This research has been supported in part by Czech Science Foundation project GA CR 202/11/0968.

Manhattan distance or Euclidean distance). Also, we compare the Snake Table with D-cache and LAESA index, and we show that Snake Table achieves the best performance.

The structure of the paper is as follows. Section 2 gives a background of metric spaces and efficiency issues. Section 3 reviews the related work. Section 4 gives the definition of the Snake Table and snake distribution. Section 5 presents the experimental results. Finally, Section 6 concludes the paper and outlines some future work.

2 Background

Let $\mathcal{M} = (\mathcal{D}, d)$ be a metric space [11]. 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) [3] 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 [11] guarantees that:

$$\forall a, b, p \in \mathcal{D}, |d(a, p) - d(p, b)| \leq d(a, b) \leq d(a, p) + d(p, b) \quad (1)$$

One index structure that uses pivots for indexing is the Approximating and Eliminating Search Algorithm (AESA) [10]. It first computes a matrix of distances between every pair of objects $x, y \in \mathcal{R}$. The structure is simply an $|\mathcal{R}| \times |\mathcal{R}|$ matrix holding the distances between every pair (due to the symmetry property of d only a half of the matrix needs to be stored). The main drawback of the AESA approach is the quadratic space of the matrix. Linear AESA (LAESA) [8] 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 [2].

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 $\text{LB}_{\mathcal{P}}$:

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

Note that $\text{LB}_{\mathcal{P}}$ 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 $\text{LB}_{\mathcal{P}}(q, r) > \epsilon$ then r can be safely discarded because r cannot be part of the search result. In the case of k -NN searches, if $\text{LB}_{\mathcal{P}}(q, r) \geq d(q, o^k)$ then r can be safely discarded, where o^k is the current k^{th} nearest neighbor candidate to q . If r could not be discarded, then 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 $\text{LB}_{\mathcal{P}}(q, r)$ for each object r in \mathcal{R} , thus it increases linearly with $|\mathcal{P}|$. The amount of distances discarded by LAESA depends on the size and quality of \mathcal{P} and on the metric space itself.

In order to analyze the efficiency that any MAM can achieve in a collection $\mathcal{R} \subseteq \mathcal{D}$ for some metric space $\mathcal{M} = (\mathcal{R}, d)$, Chávez et al. [3] 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 $\rho(\mathcal{M}) = \frac{\mu^2}{2\sigma^2}$, 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 probably become useless 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, like the M-Tree, are dynamic. 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 can manage the insertion or deletion of objects and pivots [7] by adding or removing rows or columns from the pivot table. However, depending on its actual implementation, LAESA is usually a static index, mainly because the pivot table might not support to dynamically modify its structure. In that case, a new table is required, copying values from the old table to the new one, evaluating the distances for new objects or pivots, and discarding the old table. Also, after many modifications in \mathcal{R} the set of pivots can begin to perform poorly and a new set of pivots should be selected.

Most of the MAMs are designed to be created during the offline phase, that is, a time-expensive process creates the index structure prior to resolve any search.

It is expected that the MAM will resolve many similarity searches, amortizing its creation time, but no information is a priori known about the query objects that will be resolved afterwards. In the online phase, the MAM efficiently receives and resolves any similarity search that may proceed from different sources and users. All the searches share the same MAM, and the MAM should achieve good performance for any search.

However, depending on the domain, the query objects may have some special properties that can be exploited to improve the performance of the MAM. In particular, frame-based CBVCD systems usually divide a query video into shots and many keyframes are extracted. A similarity search is then performed for consecutive keyframes, thus it can be expected that two consecutive keyframes will frequently be similar. In the case of interactive Content-based Multimedia Information Retrieval (CBMIR) systems, a user starts a search with some example or some 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.

3.1 Related Work – D-Cache and D-File

To take advantage of the online indexing process and a stream of correlated queries, there is a recently proposed structure called D-file [9]. The D-file is the database file itself accompanied by a main-memory structure, called the D-cache. The D-cache stores the evaluated distances $d(q_i, o_j)$ while processing queries in the stream. When the n^{th} query in the stream is processed, the D-cache calculates a lower-bound distance for $d(q_n, o_j)$ evaluating the distance from q_n to previous q_i and treating the previous queries as pivots. Hence, if the calculated 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-cache 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 is implemented with: 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.

In CBVCD systems, a similarity search is performed for consecutive keyframes, thus it can be expected that the D-cache will achieve high performance. However, as we show in the experimental section, the D-cache suffers from high internal realtime complexity rendering it unviable to use in a CBVCD system. 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. 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.

4 Snake Table

In this work we propose a new dynamic indexing structure, called a Snake Table, which is designed to: 1) improve the search time for streams of queries where consecutive query objects are similar; and 2) keep its internal complexity low to be applied in systems that use fast distance functions, like CBVCD systems and interactive CBMIR that use global descriptors and Minkowski distances.

The life cycle of the Snake Table is as follows: First, when a new session is created, 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. Then, when a new query object q_i is received, a k -NN is performed using the previous query objects q_1, \dots, q_{i-1} as pivots to accelerate the search. Finally, when the session ends, the Snake Table is discarded. Therefore, like D-cache and unlike most of MAMs, the Snake Table is a session-oriented and short-lived MAM.

The Snake Table is implemented with a fixed-size $|\mathcal{R}| \times p$ matrix used as a dynamic pivot table. As in LAESA, the j^{th} 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. However, each cell in the j^{th} 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 $\text{LB}_{\mathcal{P}}(q_i, o_j)$ for the distance $d(q_i, o_j)$ is calculated (see Eq. 2), with \mathcal{P} dynamically determined by the query objects and distances in the j^{th} row. The object o_j is discarded when $\text{LB}_{\mathcal{P}}(q_i, o_j)$ is greater than the distance between q_i and its current k^{th} 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 evaluated, added to some cell in the j^{th} row, and the NN candidates are updated when o_j is closer than the current k^{th} NN to q_i .

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

1. Each query q_i picks a column in round-robin mode, i.e., the distance $d(q_i, o_j)$ is stored in the $(i \bmod p)$ column of j^{th} row, eventually replacing the stored distance $d(q_{i-p}, o_j)$. If the distance was not evaluated because it was discarded by $\text{LB}_{\mathcal{P}}(q_i, o_j)$ there are two options: 1) its corresponding cell is

- either updated with an ∞ distance; 2) the cell is left unmodified but before any read the query stored in the cell is matched with the last query for that column (the experimental section uses the latter). With this strategy, each row is sparse, containing at most p distances between $d(q_{i-p}, o_j)$ and $d(q_i, o_j)$.
2. The distance $d(q_i, o_j)$ is compared to every distance in j^{th} row and the highest distance in the row is replaced. With this strategy, each row stores p unsorted distances between $d(q_1, o_j)$ and $d(q_i, o_j)$.
 3. Each distance $d(q_i, o_j)$ is stored in a cell chosen in an independent round-robin for every row. With this strategy, every row compactly stores the last p evaluated distances for o_j replacing the old ones. $\text{LB}_{\mathcal{P}}$ starts its evaluation from the last stored distance and goes backwards, therefore favoring the most recent stored distances.

D-cache uses a combination of strategies 1 and 2. It always replaces an old distance (older than q_{i-p}), but if there is not an old distance in the collision interval, then it replaces the worst distance, defined as the distance closer to the median (or to some predefined percentile of distances). Note that a very high distance can achieve better discarding performance than a medium distance (as used in D-cache strategy), however they are unlikely to appear in a snake distribution. In order to reduce the internal complexity of strategy 2 this case is not considered.

For strategy 1, distances $d(q_i, q_j)$ with $j \in \{i-p, \dots, i-1\}$ are calculated and stored in memory at the beginning of every search. For strategies 2 and 3, distances $d(q_i, q_j)$ with $j \in \{1, \dots, i-1\}$ are calculated on-demand by $\text{LB}_{\mathcal{P}}$. Note that the internal complexity of strategy 3 is slightly higher than strategy 1, because it needs to manage an independent index for each row to mark the position of the last stored distance.

The performance achieved by these three replacement strategies are 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.

4.1 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 when the distance between two consecutive objects in the stream is small compared to the average distance between any two objects (see Fig. 1). 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:

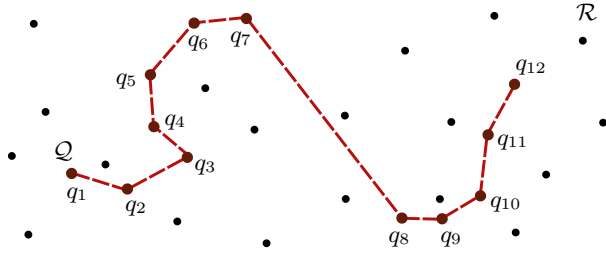


Fig. 1. Stream of queries $\mathcal{Q}=\{q_1, \dots, q_{12}\}$ with a snake distribution: most of distances $d(q_i, q_{i+1})$ are smaller than $d(x, y)$ for randomly selected pairs x, y in \mathcal{R}

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. $\Delta(F_1, F_2)$ is greater than zero when the distances in F_1 are smaller than the distances in 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 $\mathcal{Q} = \{q_1, \dots, q_m\}$. 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 $d(q_i, q_{i-p}) \forall i \in \{p+1, \dots, m\}$. \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 i^{th} 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 Experimental Evaluation

In this section we evaluate the performance of the Snake Table with the three presented strategies, and we compare them with the performance achieved by

D-cache and LAESA. The comparison is performed under six different metric spaces, with a stream of queries with a snake distribution.

5.1 Preliminaries

Dataset. We tested the Snake Table on our frame-based CBVCD system [1] using different configurations over the MUSCLE-VCD-2007 dataset [5]. MUSCLE-VCD-2007 is a publicly available and widely-used video copy database, and it was the corpus used at the CIVR 2007 video copy detection evaluation. The reference collection is composed of 101 videos, 59 hours total length, and the query videos are divided into collections called ST1 and ST2. ST2 has three videos with a total length of 45 minutes. The ST2 collection contains 21 video excerpts copied from a video in the reference collection. Each copied excerpt may have some transformations like blur, flip, subtitles, zoom, insertion of logo, noise, etc. Every video in the dataset has 25 fps.

For the present evaluation, each reference video and each video in ST2 is partitioned into short fixed-length segments of 1 second. For each segment, four global descriptors are calculated: the Edge Histogram (EH), captures the spatial distribution of edges in a frame [6]. 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 [4]. We used 9×9 blocks, producing a vector of 81 bytes. The Color Histogram (CH), divides a frame into 4 horizontal slices. Each slice calculates a histogram of 16 bins for R, G, and B channels, and each bin with 8-bits linear quantization, producing a vector of 192 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.

\mathcal{R} is the set of reference segments ($|\mathcal{R}|=211,479$ segments), and \mathcal{Q} is the set of query segments ($|\mathcal{Q}|=2,692$ segments). The correct answer for a segment $q \in \mathcal{Q}$ is the reference segment $r_q \in \mathcal{R}$ for which q is a copy. Because we stated that there is only one correct answer, the mean average precision (MAP) corresponds to the average of the inverse of the ranks of r_q , for all copied segment in \mathcal{Q} .

Configurations. We test six 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 $L_1(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n |x_i - y_i|$ for n -dimensional vectors \mathbf{x} and \mathbf{y} :

1. **OM**: compares OM descriptors $d(r, s)=L_1(\text{OM}(r), \text{OM}(s))$.
2. **KF**: compares KF descriptors $d(r, s)=L_1(\text{KF}(r), \text{KF}(s))$.
3. **EH**: compares EH descriptors $d(r, s)=L_1(\text{EH}(r), \text{EH}(s))$.
4. **CH**: compares CH descriptors $d(r, s)=L_1(\text{CH}(r), \text{CH}(s))$.
5. **ECK**: weighted combination of EH, CH, and KF descriptors:

$$\begin{aligned}
d(r, s) &= 0.6 \times L_1(\text{EH}(r), \text{EH}(s)) \times \frac{1}{7996} \\
&+ 0.2 \times L_1(\text{CH}(r), \text{CH}(s)) \times \frac{1}{6219} \\
&+ 0.2 \times L_1(\text{KF}(r), \text{KF}(s)) \times \frac{1}{24721}
\end{aligned}$$

Each left factor (0.6, 0.2, 0.2) is the weight in the combination, and each normalization factor (7996, 6219, 24721) is the maximum distance value for the respective distance.

6. **EK3**: temporal combination of EH and KF descriptors:

$$\begin{aligned}
g(r, s) &= 0.5 \times L_1(\text{EH}(r), \text{EH}(s)) \times \frac{1}{7996} \\
&+ 0.5 \times L_1(\text{KF}(r), \text{KF}(s)) \times \frac{1}{24721} \\
d(r_i, s_j) &= \frac{1}{3} [g(r_{i-1}, s_{j-1}) + g(r_i, s_j) + g(r_{i+1}, s_{j+1})]
\end{aligned}$$

Where r_{i-1} and r_{i+1} are the previous and the next segments of r_i in a video.







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

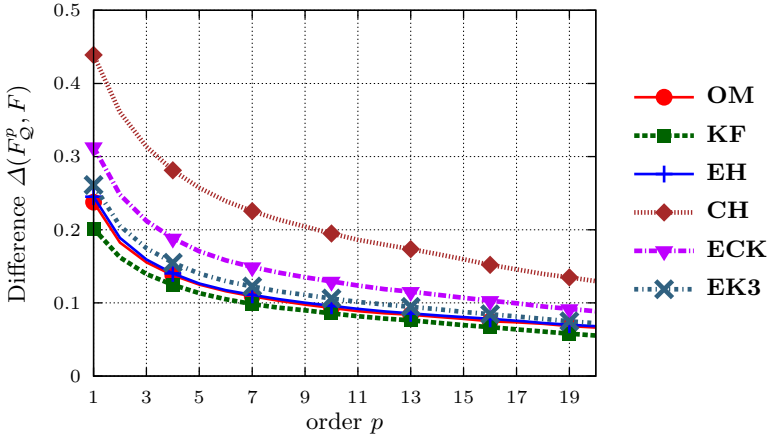
1. **D-cache**: It uses a hash table with fixed size $|\mathcal{R}|*p$, the collision interval to the minimum (1), and the hash function is $h(q_i, o_j) = (rnd_i * rnd_j) \bmod (|\mathcal{R}|*p)$, where rnd_i and rnd_j are unique random IDs assigned to each object. We checked that the hash function generates a uniform distribution through the whole table, producing almost no collisions.
2. **LAESA**: Following its definition, LAESA does not require any information of the query objects, but for a fair comparison, we allow LAESA to use \mathcal{Q} in the selection process. **LaesaR** chooses p static pivots from \mathcal{R} , and **LaesaQ** chooses p static pivots from \mathcal{Q} . Both selections are performed using the SSS algorithm [2]. Four different sets are selected and the average value of LB_p is calculated for each one by sampling pairs from $\mathcal{Q} \times \mathcal{R}$. The set of pivots with higher average LB_p is finally selected while the other sets are discarded.
3. **Snake Table**: We test the three strategies depicted in Section 4. **SnakeV1** uses a sparse row with the last p queries, **SnakeV2** uses an unsorted row discarding the highest distance, and **SnakeV3** uses a compact row with the last p evaluated distances.

5.2 Experiments

Table 1 shows for the different configurations the total time spent by a linear scan (in seconds), the achieved MAP, and some indicators for the metric space. The

Table 1. Effectiveness and efficiency for the base configurations

	Time	MAP	max	μ	σ	ρ	H_d
Group 1							
OM	282 s.	0.125	3285	1489	416	6.4	
KF	304 s.	0.509	24721	7264	2636	3.8	
Group 2							
EH	541 s.	0.639	7996	3198	751	9.1	
CH	501 s.	0.482	6219	3661	970	7.1	
Group 3							
ECK	1258 s.	0.646	0.888	0.416	0.09	11.4	
EK3	2214 s.	0.732	0.870	0.347	0.08	10.2	

**Fig. 2.** Snake distribution of order $p \in \{1, \dots, 20\}$ for the six configurations

histogram of distances was created by evaluating $d(x, y)$ with pairs x, y sampled from $Q \cup \mathcal{R}$. The configurations are split into three groups. Group 1 (**OM** and **KF**) contains the configurations where the linear scan takes less amount of time. Group 2 (**EH** and **CH**) contains the configurations where linear scan takes about twice as much time as Group 1. Group 3 (**ECK** and **EK3**) contains the configurations in which the linear scan is slower by one order of magnitude. In the following experiments, the performance of each index is presented as a ratio with the performance of the linear scan for that configuration. The MAP achieved by each configuration is also shown in the table. The configuration with best detection effectiveness is **EK3** followed by **ECK**. Also, **KF** shows a promising tradeoff between effectiveness and efficiency.

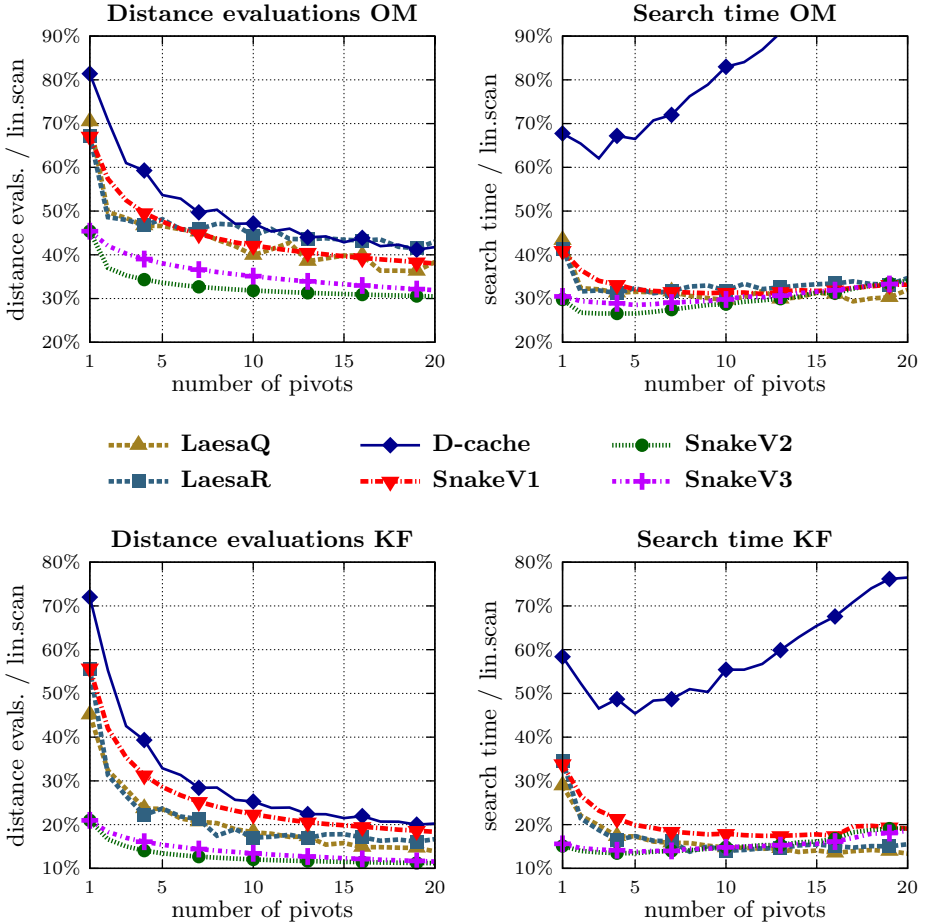


Fig. 3. Search time and distance evaluations for **OM** and **KF** (Group 1)

Snake Distribution. Figure 2 depicts the snake distribution of order p for the six configurations. The value of difference $\Delta(F_Q^p, F)$ for $p \in \{1, \dots, 20\}$ is shown. The six 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 good a fit for the six configurations, but as p increases, the snake distributions tend to disappear. As shown in the following experiments, the different configurations present satisfactory results for roughly between 1 and 5 pivots.

Group 1. Figure 3 shows the efficiency achieved by the six indexes for the **OM** and **KF** configurations varying the number of pivots from 1 to 20. It shows the amount of distances evaluations as a proportion of the evaluations required by the linear scan (i.e., a fraction of $|Q|*|\mathcal{R}|$). This value includes the distances between query and pivots but does not include the distance required for pivot

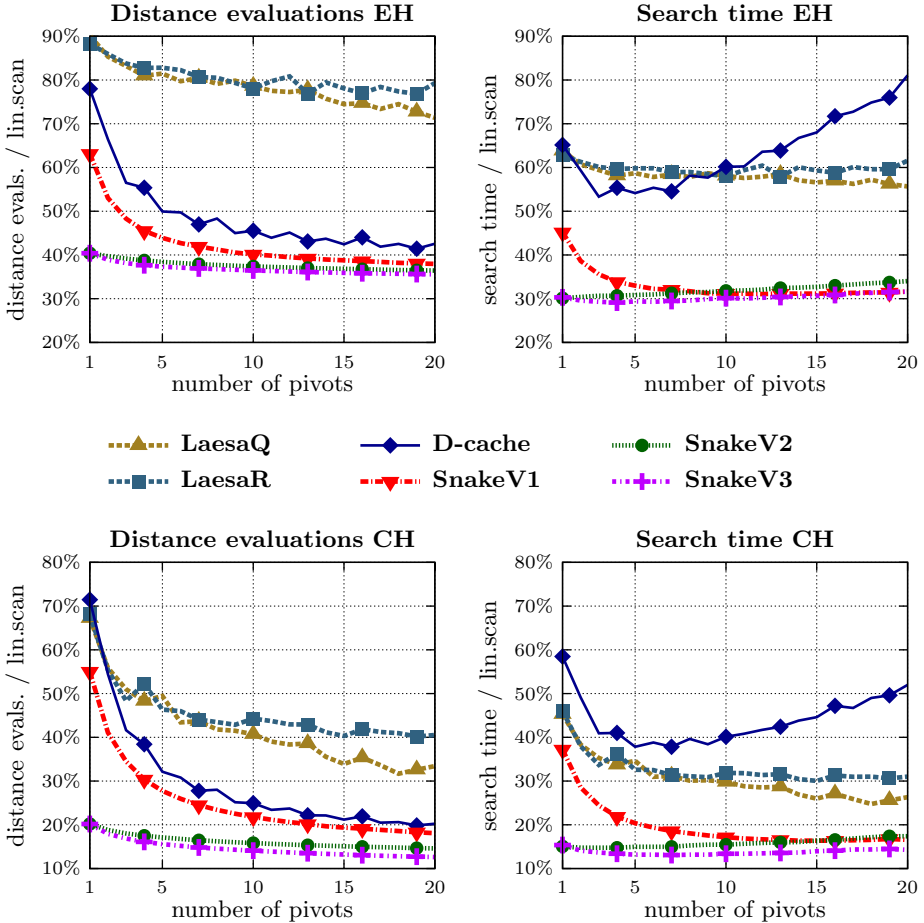


Fig. 4. Search time and distance evaluations for **EH** and **CH** (Group 2)

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. To reduce this issue we calculate three different sets of pivots with SSS and the average value for evaluated distances and search time is presented.

The disparity between saved distances and saved time reveals that **D-cache** suffers from high internal complexity at these two configurations: while most of the distance computations are discarded, the search time increases even beyond the time required by a linear scan in **OM**. Both **LaesaR** and **LaesaQ** (i.e., static pivots) perform better than **D-cache**. On average, **LaesaQ** performs slightly better than **LaesaR** in both experiments. The Snake Table achieves the best performance by its combination of good pivot selection (due to the snake distribution) and its low internal complexity. Between the different strategies, the **SnakeV2** (i.e., replacement of the highest distance) achieves the best performance.

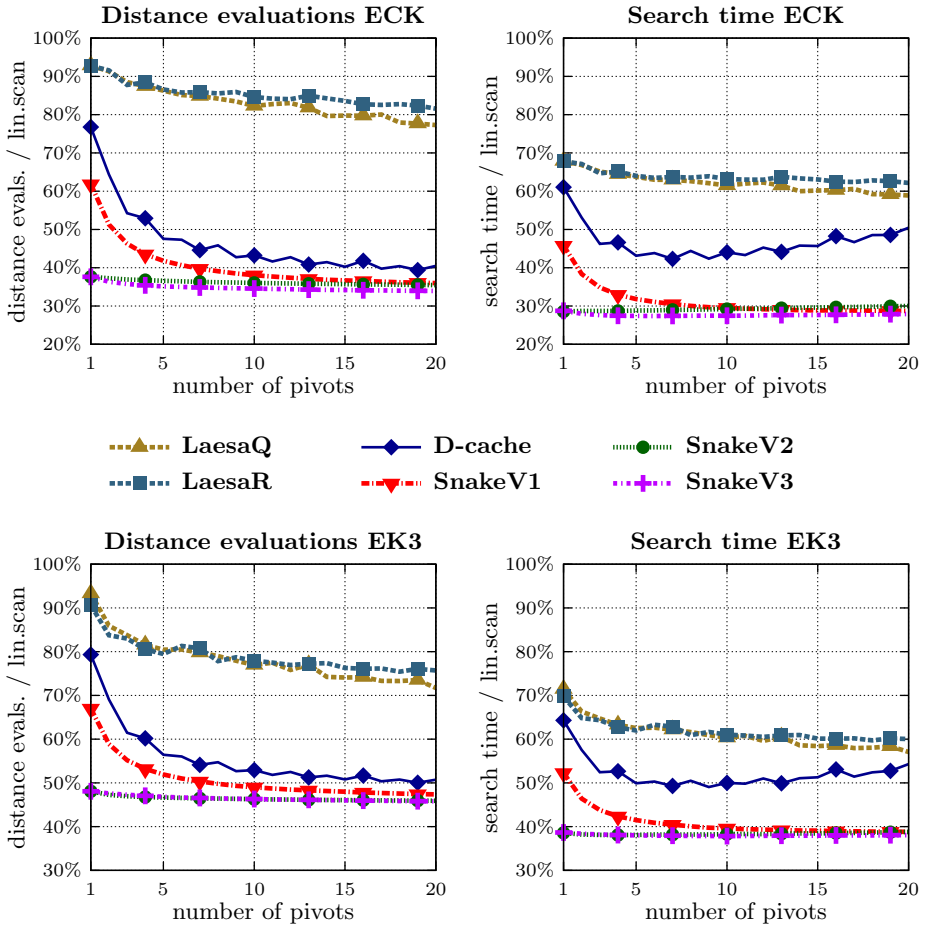


Fig. 5. Search time and distance evaluations for **ECK** and **EK3** (Group 3)

Group 2. Figure 4 shows the efficiency achieved by the six indexes for the **EH** and **CH** configuration varying the number of pivots. **D-cache** again suffers from high internal complexity, reducing the linear scan time by 10% for 20 pivots at **EH**, even though it can save more than 50% of distance evaluations. Both **LaesaR** and **LaesaQ** starts slightly better than linear scan, but then their internal complexity dominates the search-time. This behavior might be due to the high intrinsic dimensionality that **EH** presents implying that any static selection of pivots will achieve bad performance. However, because **EH** fits a snake distribution, with just a few dynamic pivots **D-cache** and the Snake Table can discard more than 50% of distance evaluations. For a few pivots, **SnakeV3** achieves the best performance, however as the number pivots increases, the **SnakeV1** improves, mainly due to its lower internal complexity. As a summary for this experiment, the Snake Table achieves the best performance, enabling the indexing of metric spaces with high intrinsic dimensionality if they present a snake distribution.

Group 3. Figure 5 shows the efficiency achieved by the six indexes for the **ECK** and **EK3** configurations varying the number of pivots. In these configurations, **D-cache** starts to show better results, outperforming **LAESA**, because the saved distance computations pay for the internal complexity. However, with more than 10 pivots, **D-cache** begins to increase its search times. In this case, the exploitation of snake distribution becomes a remarkable approach for improving efficiency. While **LaesaQ** and **LaesaR** can discard 20% distances, the Snake Table can discard more than 50% of distances. In particular, **SnakeV3** (i.e., to store the last p pivots by object) shows the fastest search times.

6 Conclusions and Future Work

In this work we presented the Snake Table, which achieves high performance for 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 than D-cache in the tested scenarios.

The Snake Table presents an approach to index spaces when consecutive queries are similar among them. This behavior usually appears in content-based video retrieval (when the queries are consecutive keyframes), and it also may appear in interactive multimedia retrieval systems (when the user selects a new query object from the answers of a previous query). 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 connects to the database, an empty Snake Table may be associated with the session. As the user performs queries with snake distribution, the Snake Table improves its performance because it will contain pivots close to the next queries. However, the Snake table is not memory efficient, because 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 the 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 other 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. One approach to overcome this issue is to combine dynamic pivots with static pivots while resolving the stream. As it is shown in the experimental section, static pivots in the queries (**LaesaQ**) perform almost identically (sometimes even better) than static pivots in the

reference objects (**LaesaR**). An improvement may be a combination between the SSS algorithm and the Snake Table. The Snake Table may choose to fix one of the dynamic pivots (i.e., to not remove it from the table) when it is far away from all the previous pivots, thus when the sliding window moves away, the fixed pivots will start to behave as static pivots complementary to the dynamic ones. Finally, LAESA can benefit of multi-core architectures by sharing the pivot table and resolving each query in different threads. However, it is not evident how to efficiently resolve parallel queries in the Snake Table due to the dynamic nature of its structure. Every thread should lock the pivot table to add the new distances, but this will interfere with other threads reading the table. A possible solution for this issue is to partition the queries into independent subsets, and each subset is resolved in an independent thread using its own Snake Table. We plan to address these issues in the future.

References

1. Barrios, J.M., Bustos, B.: Competitive content-based video copy detection using global descriptors. *Multimedia Tools and Applications*, 1–36 (2011)
2. Bustos, B., Pedreira, O., Brisaboa, N.: A dynamic pivot selection technique for similarity search. In: *Proc. of the Int. Workshop on Similarity Search and Applications (SISAP 2008)*, pp. 105–112. IEEE (2008)
3. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. *ACM Computing Surveys* 33(3), 273–321 (2001)
4. Kim, C., Vasudev, B.: Spatiotemporal sequence matching for efficient video copy detection. *IEEE Transactions on Circuits and Systems for Video Technology* 15(1), 127–132 (2005)
5. Law-To, J., Joly, A., Boujemaa, N.: MUSCLE-VCD-2007: a live benchmark for video copy detection (2007), <http://www-rocq.inria.fr/imedia/civr-bench/>
6. Manjunath, B.S., Ohm, J.-R., Vasudevan, V.V., Yamada, A.: Color and texture descriptors. *IEEE Transactions on Circuits and Systems for Video Technology* 11(6), 703–715 (2001)
7. Micó, L., Oncina, J.: A constant average time algorithm to allow insertions in the LAESA fast nearest neighbour search index. In: *Proc. of the Int. Conf. on Pattern Recognition (ICPR 2010)*, pp. 3911–3914. IEEE (2010)
8. Micó, M.L., Oncina, J., Vidal, E.: A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements. *Pattern Recognition Letters* 15(1), 9–17 (1994)
9. Skopal, T., Lokoč, J., Bustos, B.: D-cache: Universal distance cache for metric access methods. *IEEE Transactions on Knowledge and Data Engineering* 24(5), 868–881 (2012)
10. Vidal, E.: New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (AESA). *Pattern Recognition Letters* 15(1), 1–7 (1994)
11. Zezula, P., Amato, G., Dohnal, V., Batko, M.: *Similarity Search: The Metric Space Approach (Advances in Database Systems)*. Springer (2005)