Query Workloads for Data Series Indexes

Kostas Zoumpatianos
University of Trento
zoumpatianos@disi.unitn.it

Yin Lou*
LinkedIn Corporation
ylou@linkedin.com

Themis Palpanas
Paris Descartes University
themis@mi.parisdescartes.fr

Johannes Gehrke*
Microsoft Corporation
johannes@microsoft.com

ABSTRACT

Data series are a prevalent data type that has attracted lots of interest in recent years. Most of the research has focused on how to efficiently support similarity or nearest neighbor queries over large data series collections (an important data mining task), and several data series summarization and indexing methods have been proposed in order to solve this problem. Nevertheless, up to this point very little attention has been paid to properly evaluating such index structures, with most previous work relying solely on randomly selected data series to use as queries (with/without adding noise). In this work, we show that random workloads are inherently not suitable for the task at hand and we argue that there is a need for carefully generating a query workload. We define measures that capture the characteristics of queries, and we propose a method for generating workloads with the desired properties, that is, effectively evaluating and comparing data series summarizations and indexes. In our experimental evaluation, with carefully controlled query workloads, we shed light on key factors affecting the performance of nearest neighbor search in large data series collections.

Categories and Subject Descriptors

H.2 [**Database Management**]; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing

Keywords

data series; indexing; similarity search; workloads

1. INTRODUCTION

Data series (ordered sequences of values) appear in many diverse domains ranging from audio signal [15] and image data processing [33], to financial [29] and scientific data [14] analysis, and have gathered the attention of the data management community for almost two decades [24, 30, 5, 23, 10, 11, 36]. Note that *time series* are a special case of data series, where the values are measured over time, but a series can also be defined over other measures (e.g., mass in Mass Spectroscopy).

Nearest neighbor queries are of paramount importance in this context, since they form the basis of virtually every data mining and analysis task involving data series. However, such queries become challenging when performed on very large data series collections [6, 26]. The state-of-the-art methods for answering nearest neighbor queries mainly rely on two techniques: *data summarization* and *indexing*. Data series summarization is used to reduce the dimensionality of the data [17, 24, 25, 21, 1, 21, 16, 8, 22] so that they can then be efficiently indexed [24, 30, 5, 32, 2, 28].

We note that despite the considerable amount of work on data series indexing [12, 25, 8, 30, 16], no previous study paid particular attention to the query workloads used for the evaluation of these indexes. Furthermore, since there exist no real data series query workloads, all previous work has used random query workloads (following the same data distribution as the data series collection). In this case though, the experimental evaluation does not take into account the hardness of the queries issued.

Indeed, our experiments demonstrate that in the query workloads used in the past, the vast majority of the queries are easy. Therefore, they lead to results that only reveal the characteristics of the indexes' performance under examination for a rather restricted part of the available spectrum of choices. The intuition is that easy queries are easy for all indexes, and therefore queries cannot capture well the differences among various summarization and indexing methods (the same also holds for extremely hard queries as well). In order to understand how indexes perform for the entire range of possible queries, we need ways to *measure* and *control* the hardness of the queries in a workload. Being able to generate large amounts of queries of predefined hardness will allow us to stress-test the indexes and measure their relative performance under different conditions.

In this work, we focus on the study of this problem and we propose the first principled method for generating query workloads with controlled characteristics under any situation, *irrespective* of the data series summarization used by

*Work done while at Cornell University. This work was funded by NSF Grants IIS-0911036 and IIS-1012593, and by the iAd Project from the National Research Council of Norway. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

KDD'15, August 10-13, 2015, Sydney, NSW, Australia.
© 2015 ACM. ISBN 978-1-4503-3664-2/15/08 ...$15.00.
DOI: http://dx.doi.org/10.1145/2578726.2578744.

the index or the available test dataset.[1] To this end, we investigate and formalize the notion of *hardness* for a data series query. This notion captures the amount of effort that an index would have to undertake in order to answer a given query, and is based on the properties of the lower bounding function employed by all data series indexes. Moreover, we describe a method for generating queries of controlled hardness, by increasing the density of the data around the query's true answer in a systematic way.

Intuitively adding more data series around a query's nearest neighbor forces an index to fetch more raw data in that area for calculating the actual distances, which makes a query "harder". In this paper, we break down this problem into three subproblems.

- Determine how large the area should be around the query's nearest neighbor (Section 3).
- Determine how many data series to add in that area (Section 5.2).
- Determine how to add data series (Section 5.3).

The proposed method leads to data series query workloads that effectively and correctly capture the differences among various summarization methods and index structures. In addition, these workloads enable us to study the performance of various indexes as a function of the amount of data that have to be touched. Our study shows that queries of increased hardness (when compared to those contained in the random workloads used in past studies) are better suited for the task of index performance evaluation.

Evidently, a deep understanding of the behavior of data series indexes will enable us to further push the boundaries in this area of research, developing increasingly efficient and effective solutions. We argue that this will only become possible if we can study the performance characteristics of indexes under varying conditions, and especially under those conditions that push the indexes to their limits.

Our contributions can be summarized as follows.

- We propose a methodology for characterizing a nearest neighbor query workload for data series collections, as well as the set of recommended principles that should be followed when generating evaluation queries.
- We describe the first nearest neighbor query workload generator for data series indexes, which is designed to stress-test the indexes at varying levels of difficulty. Its effectiveness is independent of the inner-workings of each index and the characteristics of the test dataset.
- We demonstrate how our workload generator can be used to produce query workloads, based on both real and synthetic datasets.

2. PRELIMINARIES

A data series $\boldsymbol{x} = [x_1, ..., x_d]$ is an ordered list of real values with length d. In this paper, we also call data series as *points*. Given a dataset $\mathcal{D} = \{\boldsymbol{x}_i\}_1^N$ of N points and a query set $\mathcal{Q} = \{\boldsymbol{q}_i\}_1^M$ of M data series, a query workload W is defined as a tuple $(\mathcal{D}, \mathcal{Q}, k, DIST)$, where each query point $\boldsymbol{q}_i \in \mathcal{Q}$ is a k nearest neighbors (k-NN) query and $DIST(\cdot, \cdot)$ is a distance function. When the context is clear, we use $\boldsymbol{x}^{(k)}$ to denote k-th nearest neighbor of a query $\boldsymbol{q}$.

In this work, we focus on the nearest neighbor query, i.e., $k = 1$, and define $MINDIST(\boldsymbol{q})$ as $DIST(\boldsymbol{x}^{(1)}, \boldsymbol{q})$. However, our methods can be naturally extended to higher values of k by employing the distance to the k-th nearest neighbor. For the rest of this study, we consider the Euclidean distance $DIST(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_2$, due to its wide application in the data series domain [5, 30, 32]. Table 1 summarizes the notations in this paper.

Symbol	**Description**
$\boldsymbol{x}$	A data series
$\boldsymbol{q}$	A query data series
$\mathcal{D}$	A set of N data series
$\mathcal{Q}$	A set of M queries
$DIST(\boldsymbol{x}, \boldsymbol{q})$	Euclidean distance between $\boldsymbol{x}$ and $\boldsymbol{q}$
$MINDIST(\boldsymbol{q})$	Distance between $\boldsymbol{q}$ and its nearest neighbor
$L(\boldsymbol{x}, \boldsymbol{q})$	Lower bound of $DIST(\boldsymbol{x}, \boldsymbol{q})$
$ATLB(L, \boldsymbol{x}, \boldsymbol{q})$	Atomic tightness of lower bound of L for $\boldsymbol{x}, \boldsymbol{q}$
$TLB(L)$	Tightness of lower bound of L
$\mu^L(\boldsymbol{q})$	Minimum effort to answer $\boldsymbol{q}$ using L
$\mathcal{N}^\epsilon(\boldsymbol{q})$	ϵ-Near Neighbors of $\boldsymbol{q}$
$\alpha^\epsilon(\boldsymbol{q})$	Hardness of $\boldsymbol{q}$ for a given ϵ

Table 1: Table of symbols.

2.1 Data Series Summarization

Since data series are inherently high-dimensional, different summarization techniques are used in order to reduce the total number of dimensions. Popular techniques not only include well known transforms and decompositions such as DFT [24, 25, 21, 1], DHWT [21, 16, 8], PCA and SVD [19, 27], but also data series specific data summarization techniques such as SAX [22], PAA [17], APCA [7] and iSAX [30, 5]. We briefly describe the most prominent ones below.

Piecewise Aggregate Approximation (PAA) [17] approximates a data series by splitting it into equal segments and calculating the average value for each segment.

Discrete Fourier Transform (DFT) [24, 25, 21, 1] uses Fourier transforms to convert a data series to the frequency domain and represents it as a list of coefficients.[2]

Discrete Haar Wavelet Transform (DHWT) [21, 16, 8] uses Haar wavelets in order to transform a data series into a list of coefficients.

Symbolic Aggregate approXimation (SAX) [22] is built above PAA with the addition that the value space is also discretized, leading to a symbolic representation with very small memory requirements.

To use such summarizations and make exact query answering feasible, indexes use lower and upper bounds of the distances between two data series in the original data space. These bounds are computed based on the summarizations of the data series. Throughout our study, we refer to the lower bounding function of a given summary as function L; given two summarized data series, L returns a lower bound of their true distance in the original space.

2.2 Data Series Indexing

Nearest neighbor search can be an intensive task; the naïve approach requires a full scan of the dataset. Fortunately, lower bounding functions on summarizations along with indexing make it possible to significantly prune the search space.

Indexes are built by hierarchically organizing data series in one or many levels of aggregation. At each level multiple groups of data series are summarized under a common representation. This is illustrated in Figure 1. Data series indexes that support exact nearest neighbor search can be divided into three broad categories as follows.

[1] Website: `http://disi.unitn.it/~zoumpatianos/edq`

[2] In this work, we use the well known FFT algorithm.

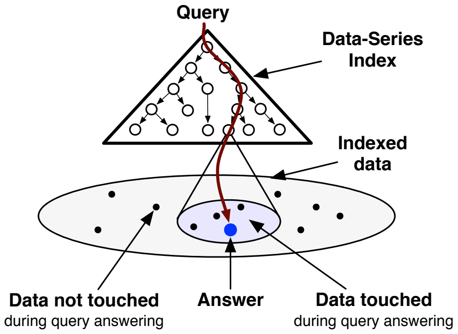

Query
Data-Series Index
Indexed data
Data not touched during query answering
Answer
Data touched during query answering

Figure 1: An index structure built above a set of data series, pruning the search space for a query.

Summarization & spatial access method. The first category involves the use of a summarization technique and a (general) spatial access method. Previous work has proposed the use of R-Trees with summarizations like DFT [1, 12, 24, 25], DHWT [8] and Piecewise Linear Approximation (PLA) [9].

Data series specific summarization & index. The second category involves the use of a summarization method specific to data series, and a specialized index that is built on top of it. Such indexes include TS-Tree [2] (based on a symbolic summarization), DS-Tree [32] (based on APCA), ADS [35] and *i*SAX index [30, 5, 6] (built on an indexable version of SAX), and SFA index [28] (it uses a symbolic summarization of data series in the frequency domain based on DFT).

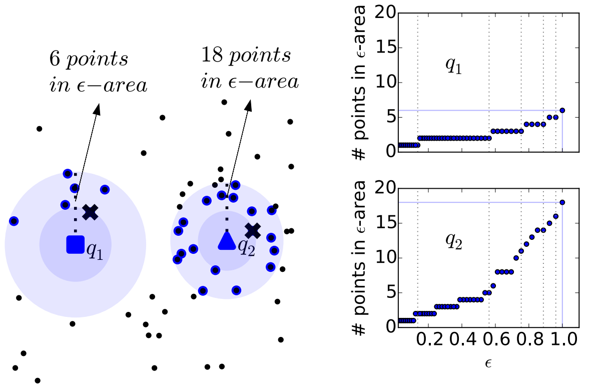
Summary reorganization & multi-level scan. This last category skips the step of building an index structure; it rather relies on carefully organizing and storing the data series representations on disk. Using this approach, data can be read in a step-wise function, where distance estimations for all data series are gradually refined as we read the summarizations in increasing detail. Both the DFT and DHWT summarizations have been studied in this context [21, 16].

Although the problem of data series indexing has attracted considerable attention, there is little research so far in properly evaluating those indexes. In this work, we focus on studying the properties of data series query workloads. The aim is to better understand the characteristics of different queries, and how these can be used to effectively test a data series index under different, but controllable conditions.

3. CHARACTERIZING WORKLOADS

When navigating an index, we make use of the lower bounds (computed based on the summarizations) of the true distances of data series in the original space. This technique guarantees that there will be no false negatives in the candidate set, but it does not exclude the false positives. Therefore, the indexes need to fetch the raw data series as well, and check them before returning the answer in order to filter out the false positives, and thus guarantee the correctness of the final answer.

The use of lower bounds can be conceptually thought of as the cut-off point in the distance between two summarized data series. Below this point, the corresponding raw data have to be checked. To capture this notion, we can use the Tightness of Lower Bound (TLB) [31], which is measured as the average ratio of the lower bound over the true distance. We formalize this notion by first introducing here the Atomic Tightness of Lower Bound ($ATLB$), which is the same ratio, but defined for a specific query and data series pair.



(a) Points in ϵ-area ($\epsilon = 1.0$) (b) # of points in ϵ-area

Figure 2: Two random queries with nearest neighbors depicted with "×".

Definition 1 (Atomic Tightness of Lower Bound). *Given a lower bounding function L, the atomic tightness of lower bound (ATLB) between two data series $\boldsymbol{q}$ and $\boldsymbol{x}$ is defined as*

$$ATLB(L, \boldsymbol{x}, \boldsymbol{q}) = L(\boldsymbol{x}, \boldsymbol{q})/DIST(\boldsymbol{x}, \boldsymbol{q}) \qquad (1)$$

Definition 2 (Tightness of Lower Bound). *Given a summarization with lower bounding function L, a set of queries $\mathcal{Q}$ and a set of data series $\mathcal{D}$, the tightness of lower bound (TLB) for this summarization is defined as*

$$TLB(L) = \frac{1}{N \times M} \sum_{\boldsymbol{q} \in \mathcal{Q}} \sum_{\boldsymbol{x} \in \mathcal{D}} ATLB(L, \boldsymbol{x}, \boldsymbol{q}) \qquad (2)$$

Note that the TLB is small for an inaccurate summarization, i.e., the summarization tends to significantly underestimate the distance. As a result, data series under this summarization will look much closer than they actually are. Consequently, the index will have to check more raw data, leading to a longer execution time.

Example 1. *Figure 2(a) demonstrates the implications of the TLB . For simplicity, we represent each data series as a point in a two-dimensional space, i.e., $d = 2$. In this example, we plot two queries $\boldsymbol{q}_1, \boldsymbol{q}_2$ and mark their nearest neighbors with a bold "×". Assume $MINDIST(\boldsymbol{q}_1) = 0.33$, $MINDIST(\boldsymbol{q}_2) = 0.26$, and all data series are summarized using the same summarization method. Let the $ATLB$ between the queries and any data point be 0.5, i.e., the lower bound of the distances between $\boldsymbol{q}_1$ or $\boldsymbol{q}_2$ and all other points is 0.5 times their actual distance. According to the definition of $ATLB$, a point $\boldsymbol{x}$ cannot be pruned if*

$$L(\boldsymbol{x}, \boldsymbol{q}) \leq MINDIST(\boldsymbol{q}) \qquad (3)$$

$$\Leftrightarrow DIST(\boldsymbol{x}, \boldsymbol{q}) \leq \frac{MINDIST(\boldsymbol{q})}{ATLB(L, \boldsymbol{x}, \boldsymbol{q})}. \qquad (4)$$

This means that for $\boldsymbol{q}_1$, all points whose actual distance is within a radius $\rho = \frac{0.33}{0.5}$ from $\boldsymbol{q}_1$'s nearest neighbor can not be pruned, because their lower bound distances are less than the distance to the answer. Since $ATLB(L, \boldsymbol{x}, \boldsymbol{q}) \in (0, 1]$, the right hand side of Inequality (4) *is always no less than $MINDIST(\boldsymbol{q})$. These ranges are depicted as disks in Figure 2(a).*

Since different summarizations (and hence different TLBs) can be employed, we need a method to generate query workloads that are summarization-independent so that it does not accidentally favor one summarization over another. To achieve that, we start by formalizing Example 1 using the notion of minimum effort that an index has to undertake for a specific query. We then generalize this concept to a measure that is summarization/index-independent, called *hardness*, for a given query and a given set of data series, and describe the connection between hardness and $ATLB/TLB$.

3.1 Minimum Effort

We define Minimum Effort (ME) as the ratio of points over the total number of series that an index has to fetch to answer a query.

Definition 3 (Minimum Effort). *Given a query $\boldsymbol{q}$, its $MINDIST(\boldsymbol{q})$ and a lower bounding function L, the minimum effort that an index using this lower bounding function has to do in order to answer the query is defined as*

$$\mu^L(\boldsymbol{q}) = |\{\boldsymbol{x} \in \mathcal{D} | L(\boldsymbol{x}, \boldsymbol{q}) \leq MINDIST(\boldsymbol{q})\}| / |\mathcal{D}|$$

As we have seen in Example 1, given a fixed $ATLB$ between the query and data series, data series that contribute to ME are within a radius $\rho = \frac{MINDIST(\boldsymbol{q})}{ATLB(L,\boldsymbol{x},\boldsymbol{q})}$ from the query's nearest neighbor and these data series cannot be pruned. The size of this radius is inversely proportional to $ATLB$ and proportional to $MINDIST(\boldsymbol{q})$.

It is important to clarify that this is the *minimum* possible effort that an index will have to undertake, and in most cases it will be smaller than the actual effort that the index will actually do. This is because the search for the solution hardly ever starts with the real answer as a best-so-far.

3.2 Hardness

Recall that our goal is to generate query workloads that are summarization-independent. Since minimum effort is tied to a specific summarization, we need a more general notion to capture how hard a query is. Intuitively, the hardness of a query is related to the number of points around its nearest neighbor (true answer). Given this intuition, we define the ϵ-Near Neighbors (ϵ-NN) of a query $\boldsymbol{q}$ as follows.

Definition 4 (ϵ-Near Neighbors). *Given $\epsilon \geq 0$, the ϵ-near neighbors of a query $\boldsymbol{q}$ is $\mathcal{N}^\epsilon(\boldsymbol{q}) = \{\boldsymbol{x} \in \mathcal{D} | DIST(\boldsymbol{x}, \boldsymbol{q}) \leq (1+\epsilon)MINDIST(\boldsymbol{q})\}$, i.e., all the points in $\mathcal{D}$ that are within $(1+\epsilon)MINDIST(\boldsymbol{q})$ of the query's nearest neighbor.*

The ϵ-NN naturally defines a hypersphere around the nearest neighbor of the query. In the rest of this paper, we will refer to this hypersphere as the **ϵ-area**. Now we define the hardness of a query as follows.[3]

Definition 5 (Hardness). *Given $\epsilon \geq 0$, the hardness of a query $\boldsymbol{q}$ is $\alpha^\epsilon(\boldsymbol{q}) = \frac{|\mathcal{N}^\epsilon(\boldsymbol{q})|}{|\mathcal{D}|}$, i.e., the fraction of $\mathcal{D}$ that is within $(1+\epsilon)MINDIST(\boldsymbol{q})$ of the query.*

Note that for an ϵ-area to cover all the points that cannot be pruned, ϵ needs be no less than $\frac{1}{ATLB(L,\boldsymbol{x},\boldsymbol{q})} - 1$ according to Inequality (4). Using the example in Figure 2(a) and assuming the total number of points in the dataset is 100, since $ATLB$ is 0.5, $\epsilon = 1.0$ defines the area where no points can be pruned. In this case, $\alpha^\epsilon(\boldsymbol{q}_1) = 0.06$ and $\alpha^\epsilon(\boldsymbol{q}_2) = 0.18$, i.e., $\boldsymbol{q}_1$ is a much easier query than $\boldsymbol{q}_2$.

Even though we have drawn a connection between $ATLB$ and hardness, it is worth noting that hardness and $ATLB$ (and hence TLB) are two intrinsically different notions. While we can use the $ATLB$ as a way to calibrate our ϵ values, TLB is tied to a specific index structure and summarization technique combination. Hardness on the other hand is a property of a dataset; when the hardness of a query is high, there are more points distributed at a close distance to the query's nearest neighbor, and as a result if the TLB is low, the index will have to do more effort, since it will have to check a larger amount of data.

3.3 Discussion

Figure 2(b) shows the number of points in the ϵ-area when ϵ increases for both $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$. These two queries show very different behavior; the hardness of $\boldsymbol{q}_2$ increases significantly as ϵ increases while the hardness of $\boldsymbol{q}_1$ stays relatively still for large ϵ. As a result, "easy" queries such as $\boldsymbol{q}_1$ will be easy for a reasonably "smart" summarization and index, and therefore lead to similar performance. On the other hand, queries such as $\boldsymbol{q}_2$ are good candidates to test various summarizations under examination because 1) it is reasonably "hard" so that any stress test could be conducted and 2) the number of points around the true answer increases almost linearly as ϵ increases, which makes it easier to observe the subtle differences for various summarizations. Recall that we could think of the lower bounding function of a summarization as a cut-off point, below which one cannot prune any points. When points are roughly uniformly distributed in the ϵ-area, such "cut-off" point can be better captured.

4. EVALUATION OF PREVIOUS WORK

4.1 Datasets and Workloads

In this section, we review some common datasets and their corresponding workloads that have been used in the literature. We use the same datasets in our experimental section as well. For capturing trends and shapes, we z-normalize (mean=0, std=1) all data series following common practice.

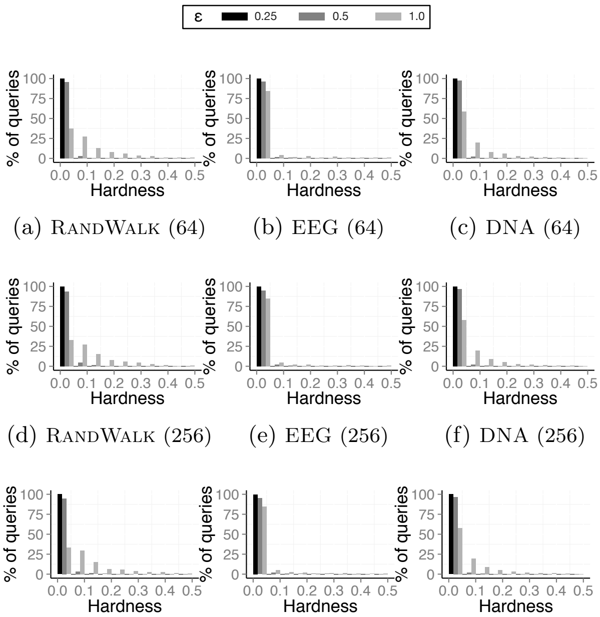
RandWalk [1, 12, 24, 8, 30, 2, 5, 16, 20, 6, 28, 35]. This dataset is synthetically produced using a random walk data series generator. The step size in each data series varies according to a Gaussian distribution. We start with a fixed random seed and produce 200,000 data series of length 1024, 256 and 64.

EEG [34, 18, 2, 20, 26]. We use the electroencephalograms dataset from the UCI repository [3], and sample 200,000 data series of length 1024, 256 and 64 from the dataset to be used as the dataset.

DNA [5, 6, 35]. We use the complete Human DNA (Homo Sapiens), obtained from the Ensembl project.[4] We sample the dataset to create 200,000 data series of length 1024, 256 and 64.

[3] A similar definition can also be found in [4].

[4] ftp://ftp.ensembl.org/pub/release-42/



(a) RandWalk (64) (b) EEG (64) (c) DNA (64)

(d) RandWalk (256) (e) EEG (256) (f) DNA (256)

(g) RandWalk (1024) (h) EEG (1024) (i) DNA (1024)

Figure 3: Histograms of query hardnesses.

The query workloads that have been used in all past studies are generated in one of the following two ways.

1. A subset of the dataset is used for indexing and a disjoint subset is used as queries [5, 16, 6, 35].
2. The entire dataset is used for indexing. A subset of it is selected and a small amount of random noise is added on top. This is used as the query set [1, 21, 20].

In our study, we shuffle the datasets, and use half of each dataset (100,000 data series) as queries, and the other half (100,000 data series) as the indexed dataset.

4.2 Hardness Evaluation

One of our key requirements is the ability to test how indexes scale as they need to check an increasing amount of data. This is the case with hard queries, for which indexes are not able to easily identify the true nearest neighbor. In this subsection, we choose 1,000 random queries from our initial query set and evaluate the hardness of each one of them for $\epsilon \in \{0.25, 0.5, 1\}$.

The results are depicted in Figure 3. As the histograms show, for all data series (length 64, 256, 1024) the query workloads are mainly concentrated towards easy queries. For $\epsilon = 0.5$, the average hardness is less than 0.1, while for $\epsilon = 1.0$, the average of hardness is less than 0.25. Additionally, as the ϵ decreases to 0, the hardness of the queries drops very rapidly for both the RandWalk and the DNA datasets. These low hardness values further motivate us for the need of a controlled way to generate workloads with queries of varying hardness.

5. QUERY WORKLOAD GENERATION

As we demonstrated in the previous section, all the widely-used (i.e., randomly generated) query workloads are biased towards easy queries. In this paper, we argue that an effective query workload should contain queries of varying hardness. Since most existing queries are easy, we start with those easy queries and make them harder by adding more points in their ϵ-areas.

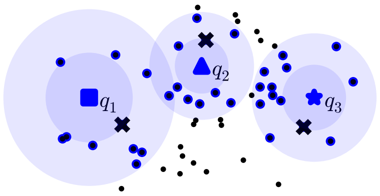


Figure 4: Example of 3 queries, where the ϵ-area of $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ intersect. As a result we cannot control the hardness of these two queries independently, as densifying each one of the two zones might affect also the hardness of the other query.

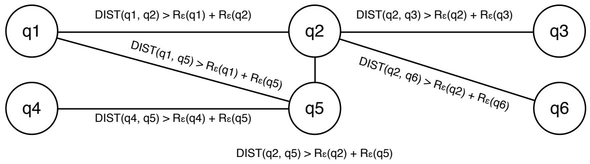

q1
q2
q3
q4
q5
q6
DIST(q1, q2) > Rε(q1) + Rε(q2)
DIST(q2, q3) > Rε(q2) + Rε(q3)
DIST(q1, q5) > Rε(q1) + Rε(q5)
DIST(q2, q6) > Rε(q2) + Rε(q6)
DIST(q4, q5) > Rε(q4) + Rε(q5)
DIST(q2, q5) > Rε(q2) + Rε(q5)

Figure 5: Maximal clique formed by $\boldsymbol{q}_1$, $\boldsymbol{q}_2$, and $\boldsymbol{q}_5$.

We start with a list of different hardness values in non-decreasing order $[\alpha_1^\epsilon, ..., \alpha_n^\epsilon]$ with respect to some ϵ that is provided by the user ($\sum_{i=1}^{n} \alpha_i^\epsilon \leq 1$, $\alpha_i^\epsilon \leq \alpha_j^\epsilon$ for $i < j$), and an input sample query set $\mathcal{Q}$ that contains many easy queries (produced through random generation). Therefore, for each hardness value α_i^ϵ, we need to *densify* (i.e, adding more points) a certain region in $\mathcal{D}$ so that we could select a subset of queries of size n from $\mathcal{Q}$ such that their hardness values match the predefined values α_i^ϵ for $i \in \{1, ..., n\}$.

The key problem here is to ensure the ϵ-areas of selected queries do not intersect with each other so that when we densify the ϵ-areas the hardness value will stay exactly as required. Figure 4 shows an example that $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ intersect. In this case, we can either choose $\boldsymbol{q}_1$ and $\boldsymbol{q}_3$, or $\boldsymbol{q}_2$ and $\boldsymbol{q}_3$. The next step is to identify the amount of points we need to add in each ϵ-area. Finally, we spread these points in such a way that as the TLB of the index gets worse, the minimum effort captured by the workload increases, following our intuitions described in Section 3 and Example 1.

5.1 Finding Non-intersecting Queries

Given an initial sample of queries $\mathcal{Q}$, we want to find the largest subset of $\mathcal{Q}$, where it holds that the ϵ-areas around each query do not intersect.

Our first step is to calculate the radius of each ϵ-area. In order to do this we need to find the distance to the nearest neighbor and multiply it by $(1 + \epsilon)$. Since we are using Euclidean distance (a metric), we can use the triangle inequality in order to find non-intersecting queries.

Given a distance function $DIST$ in metric space, we set $R_\epsilon(\boldsymbol{q}) = (1 + \epsilon)MINDIST(\boldsymbol{q})$ as the radius of the ϵ-area. Two queries $\boldsymbol{q_i}, \boldsymbol{q_j} \in \mathcal{Q}, \boldsymbol{q_i} \neq \boldsymbol{q_j}$ are non-intersecting if and

Algorithm 1 FindNonIntersectingQueries

```
$R_\epsilon \leftarrow \mathit{createRadius}(\mathcal{Q}, \mathcal{D}, \epsilon)$
$g \leftarrow \mathit{createVerticesFromQueries}(\mathcal{Q})$
for $(\boldsymbol{q}_i, \boldsymbol{q}_j) \in \mathcal{Q} \times \mathcal{Q}$ do
    if $\mathit{DIST}(\boldsymbol{q}_i, \boldsymbol{q}_j) > R_\epsilon(\boldsymbol{q}_i) + R_\epsilon(\boldsymbol{q}_j)$ then
        $g.\mathit{addEdge}(\boldsymbol{q}_i, \boldsymbol{q}_j)$
$\mathcal{V} \leftarrow g.\mathit{getSortedVertices}()$ {Sorted by ascending degree}
$\mathcal{Q}' \leftarrow \emptyset$
for $\boldsymbol{q} \in \mathcal{V}$ do
    if $\mathit{isCompatible}(\boldsymbol{q}, \mathcal{Q}')$ then
        $\mathcal{Q}' \leftarrow \mathcal{Q}' \cup \{\boldsymbol{q}\}$
return $\mathcal{Q}'$
```

only if the following holds (by the triangle inequality):

$$\mathit{DIST}(\boldsymbol{q}_i, \boldsymbol{q}_j) > R_\epsilon(\boldsymbol{q}_i) + R_\epsilon(\boldsymbol{q}_j)$$

In order to validate this constraint, we first need to calculate all the pairwise distances for all the queries in $\mathcal{Q}$, and for each query $\boldsymbol{q}$, the distance to its nearest neighbor $\mathit{MINDIST}(\boldsymbol{q})$.

Given the set of queries and their pairwise distances, we can create a graph $\mathcal{G}$, where each vertex represents a query, and an edge exists if two queries $\boldsymbol{q}_i$ and $\boldsymbol{q}_j$ do not interfere with each other. Now it is clear that our problem is closely related to the maximum clique problem. Figure 5 illustrates an example graph with 6 queries, where queries $\boldsymbol{q}_1$, $\boldsymbol{q}_2$, $\boldsymbol{q}_5$ form the maximum clique, being mutually non-intersecting.

Note that finding the maximum clique in graph $\mathcal{G}$ is NP-complete, we therefore employ a greedy approach to select queries by assigning a query $\boldsymbol{q}$ to some α_i^ϵ (denoted as $\boldsymbol{q}(\alpha_i^\epsilon)$) if its current hardness is smaller than α_i^ϵ and if its ϵ-area does not intersect with the ϵ-areas of all previously assigned queries. This ensures that when densifying the ϵ-area for $\boldsymbol{q}(\alpha_i^\epsilon)$, the hardness of other selected queries $\boldsymbol{q}(\alpha_j^\epsilon)$ $(j \neq i)$ will remain unaffected.

Algorithm 1 describes how to find non-intersecting queries. The algorithm sorts the vertices of the graph based on their degree. The intuition is that high-degree vertices have more compatible vertices. We then keep reading vertices in that order, adding compatible ones to a list while skipping incompatible ones.

5.2 Deciding the Number of Points to Densify

As we discussed, it is important to generate a benchmark that has queries of varying hardness. In the earlier section, we have established the notion of hardness and have discussed how to select a set of candidate queries given a query set. In this section, we will describe how to determine the number of points to densify.

Given the list of n input hardness values $[\alpha_1^\epsilon, ..., \alpha_n^\epsilon]$ with respect to some ϵ. Each hardness value now has a corresponding query, and we need to identify the number of points to densify to the ϵ-area of each query in order to achieve the target hardness. Let x_i be the number of points to add for $\mathcal{N}^\epsilon(\boldsymbol{q}(\alpha_i^\epsilon))$ and $N_i = |\mathcal{N}^\epsilon(\boldsymbol{q}(\alpha_i^\epsilon))|$ is the current number of points in $q(\alpha_i^\epsilon)$'s ϵ-area, we have the following linear system.

$$\alpha_1^\epsilon = \frac{N_1 + x_1}{N + \sum_{i=1}^n x_i},\ ...,\ \alpha_n^\epsilon = \frac{N_n + x_n}{N + \sum_{i=1}^n x_i} \qquad (5)$$

Representing this linear system in matrix form, we have

$$(A - I)\boldsymbol{x} = \boldsymbol{b}, \qquad (6)$$

where

$$A = \begin{pmatrix} \alpha_1^\epsilon & \alpha_1^\epsilon & ... & \alpha_1^\epsilon \\ \alpha_2^\epsilon & \alpha_2^\epsilon & ... & \alpha_2^\epsilon \\ ... & ... & ... & ... \\ \alpha_n^\epsilon & \alpha_n^\epsilon & ... & \alpha_n^\epsilon \end{pmatrix}$$

and

$$\boldsymbol{b} = [N_1 - \alpha_1^\epsilon N, ..., N_n - \alpha_n^\epsilon N]^T.$$

This linear system can be easily solved and it will tell us how many points to densify in the ϵ-area for each selected query.

5.3 Densification Process

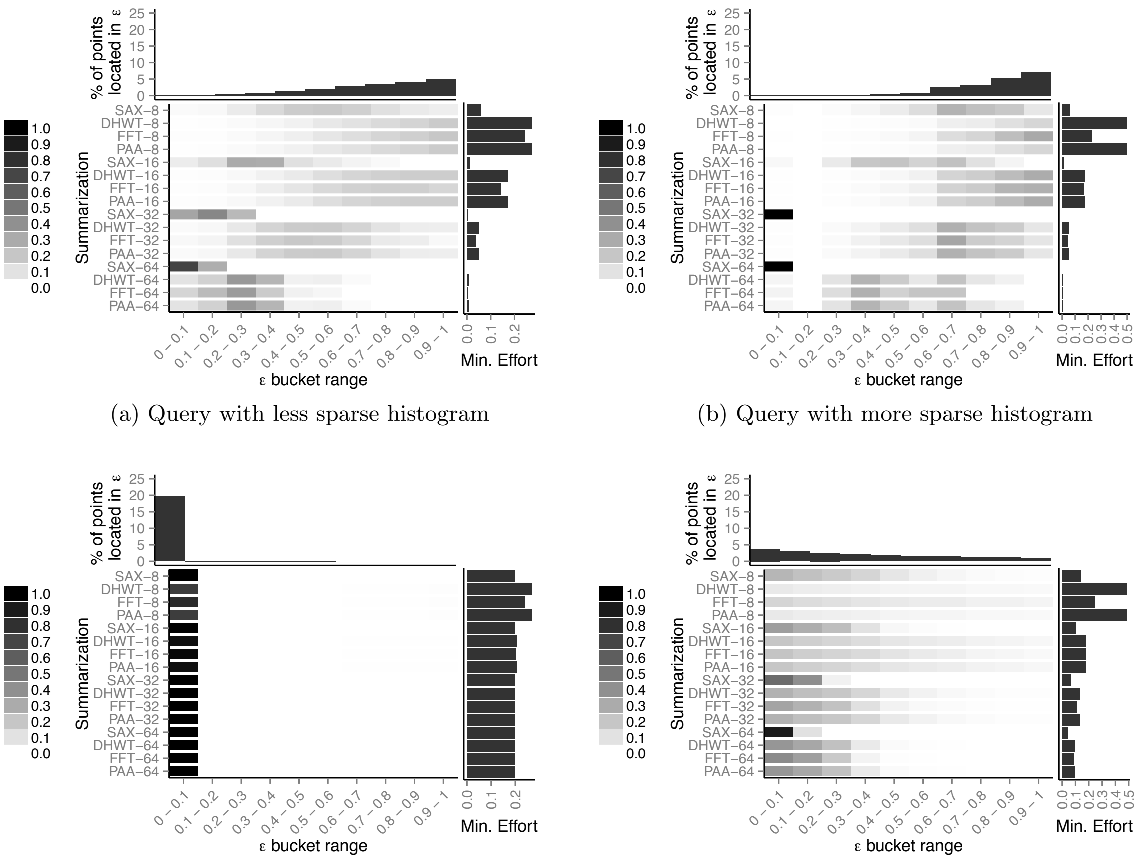
In this section we describe how to densify the ϵ-areas for the selected queries. Given that most summarization and index methods require the data to be z-normalized, we could not directly add random points in the ϵ-area since after z-normalization those points could be outside of the ϵ-area. Instead, we utilize existing points in the ϵ-area. Therefore, the candidate strategies for densification are the following: (a) randomly choosing a point in $\mathcal{N}^\epsilon(\boldsymbol{q}(\alpha_i^\epsilon))$ and adding noise to create a new point; (b) adding noise to the query's nearest neighbor (ignoring all other points in its ϵ-area); and (c) adding points as uniformly as possible in the ϵ-area.

To demonstrate different densification strategies, we generate queries of hardness 0.2 ($\epsilon = 1.0$) for a dataset of 100,000 data series. According to Section 3.2, this ϵ allows us to test less tight representations with *TLB*s as low as 0.5. To evaluate the effort for every query, we use four standard data series summarization techniques (SAX, FFT, DHWT, PAA) at various resolutions, ranging from 8 to 64 bytes per data series. The data series are of length 256, and for each summarization we measure the minimum effort required.

Random densification. A naïve method to increase the hardness in an ϵ-area is to choose random points from this area and add noise to them, thus producing the desired amount of extra points. A property of this method is that the original distribution of the points will not change significantly. The problem with this method, however, is that for very good summarization methods (large *TLB* values), as we increase the number of points in the ϵ-area, the minimum effort will not necessarily increase relatively to it. As a result, indexes with different *TLB* values might have the same effort to answer a query.

The result of a query generated with this method can be seen in Figure 6(a). The histogram at the top of the figure displays the distribution of the points in the densified ϵ-area. As we can see the further away we get from the nearest neighbor, the more points we find at each area. The heat map in the center represents the locations of the points that contribute to the minimum effort, i.e., $L(\boldsymbol{x}, \boldsymbol{q}) \leq \mathit{MINDIST}(\boldsymbol{q})$. The color represents the portion of these points in the corresponding bucket of ϵ. Finally, the vertical graph on the right side represents the minimum efforts of the different summarization methods.

As expected, the results show that crude summarizations (SAX-8, DHWT-8, FFT-8, PAA-8) that use less bytes for representing the data series have much larger minimum efforts. From the plot we could infer that a significant portion



(a) Query with less sparse histogram

(b) Query with more sparse histogram

(c) 1NN densified query

(d) Equi-densified query

Figure 6: Two randomly densified, one 1NN densified and one equi-densified queries on a 100,000 data series randomwalk dataset. Distribution of distances of all data series in the dataset on top, minimum effort for each summarization technique on the right. Heat maps represent the amount of points that are part of the effort located at the corresponding bucket of ϵ.

of points that contribute to minimum effort may not be included by this ϵ-area. On the other hand, fine summarizations (SAX-64, DHWT-64, FFT-64, PAA-64) are well captured by this ϵ. Actually, we only need $\epsilon = 0.6$ to capture all points contributing to the minimum efforts. With the histogram on the top, it is easy to see that the minimum effort is related to the distribution of points in the original space. For example, while the heat map for FFT-64, DHWT-64 and PAA-64 spans a larger range of ϵ values, their minimum effort is not much greater than that of SAX-64, which spans a much smaller range. This is because, as we can see in the histogram at the top, there is a very small amount of data within $\epsilon = 0.5$ and it does not increase too much as ϵ increases. This situation is more pronounced with another query example shown in Figure 6(b), where the distribution of points in the ϵ-area is even more skewed.

1NN densification. Another naïve method for increasing hardness in the ϵ-area is by just adding noise to the query's nearest neighbor itself. This will force all summarizations to make (almost) the same effort, as the area very close to the nearest neighbor is now very dense and all the rest of the ϵ-area is very sparse. In this case, all efforts for all summarizations are almost identical. An example 1NN densified query is shown in Figure 6(c).

Equi-densification. As discussed in Section 3.3, we want to ensure that the hardness points are distributed as uniformly as possible within the ϵ-area corresponding to each possible $ATLB$ value. This ensures that we capture the subtle differences for various summarizations. To this end, we propose equi-densification that aims to distribute the extra points we need to add in such a way that buckets that are originally almost empty get a large number of points, and buckets that are almost full get a small number of points.

In order to achieve this, we bucketize $ATLB$ values, and accordingly the ϵ values are bucketized (in a non-uniform way). For each $ATLB$ bucket we want to make sure there is an equal amount of points. Densification is done by creating linear combinations of points located within and outside of each bucket. This ensures the diversity of the generated data series, allowing us to control the location of the data points in the ϵ-area, and also ensures that the resulting data series after z-normalization will fall in the desired location with high probability.

A query produced with equi-densification is depicted in Figure 6(d). The histogram on the top shows that the first few buckets have more points, while the last few buckets have less. This happens because ϵ is inversely proportional to $ATLB$, and as a result ϵ bucket ranges are small for large $ATLB$ values and large for small $ATLB$ values. For example for $ATLB$ values in $[0.5, 0.6]$ the corresponding ϵ values are in $[0.67, 1.0]$ and for $ATLB$ values in $[0.6, 0.7]$ the corresponding ϵ values are in $[0.43, 0.67]$ As we can see in the heat map, the effort points are now evenly distributed in the ϵ-areas. Note also that as the bounds of a summarization get worse, we need to increase the ϵ to include all points that contribute to the minimum effort.

Therefore, equi-densification achieves the desired result, accurately capturing the relative differences among different summarizations, and consequently leading to correct performance comparisons of indexes based on their TLB. We further validate this claim in the experimental evaluation.

6. EXPERIMENTS

In this section, we provide an experimental evaluation of the proposed method. We generate query workloads on the three datasets in Section 4 using our method described in the previous section. All our datasets contain 100,000 data series with length 256. Given a set of desired hardness values, ϵ, and the densification mode, our method produces a new dataset that is the original dataset with extra points, and a set of queries that forms the workload that matches the desired hardness values.

We performed three sets of experiments. The first investigates the amount of non-interfering queries we can find for each dataset. The second is intended to compare the three different densification methods with regards to the minimum effort of various common summarization techniques, i.e., PAA, FFT, DHWT and SAX. For each one of the summarizations, we used 8, 16, 32 and 64 bytes to represent each data series. In the third set of experiments, we used two real world indexes, iSAX 2.0 [6] and the R-Tree [13] using PAA as a summarization method. This last experiment aims to show the impact of our benchmark on these indexes compared to choosing random points from the dataset (queries are left outside of the indexed data). A comprehensive experimental comparison of various data series indexes is out of the scope of this study and is part of our future work.

6.1 Non-interfering Queries

In this experiment, we used 100,000 data series from each dataset as the indexed data, and 100,000 data series as sample queries. We generated sets of 1,000 (100 sets), 2,000 (50 sets) and 4,000 (25 sets) queries, and run our non-interfering queries discovery algorithm on each one of them for $\epsilon \in \{0.25, 0.5, 1.0\}$. Our algorithm evaluates each set of queries against the corresponding dataset, and we report the average number of queries found per dataset, query set size and ϵ, as well as the corresponding error bars.

The results are depicted in Figure 7 (error bars are very small). We observe that for RandWalk and DNA, using a large ϵ only allows us to find an average of 7-10 non-interfering queries, and as ϵ decreases we can find up to 300-600 queries. For the EEG dataset, the data distribution allows us to find a much higher number of non-interfering queries, which is in the order of thousands. Note that since we are mainly interested in generating queries with high hardness values, we do not need too many queries. Furthermore, the constraint $\sum_{i=1}^{n} \alpha_i^{\epsilon} \leq 1$ restricts the number of hard queries that we could produce for one dataset. For example, we can generate 5 queries of hardness 0.2, but only 2 queries of hardness 0.5 and 0.5, respectively. Since this number of queries is small for a comprehensive benchmark, the solution is to use multiple datasets with corresponding query workloads; even in the case of $\epsilon = 1.0$, we can run our algorithm 100 times to get 100 different output datasets with at least 3 different queries each, for a total of 300 queries.

6.2 Densification Mode

In this experiment, we generated 3 different queries with a hardness of 0.2 for each one of them ($\epsilon = 1.0$). For each query we used a different densification method. Our goal is to measure how well the different densification methods capture the relative summarization errors of different summarization techniques. We use $1 - TLB$ as the summarization error for each technique. This number intuitively captures how far the lower bound of a summarization is from the true distance. We report the relative summarization errors in the results (normalized by the smallest summarization error). In our experiments, the summarization with the smallest error was SAX (64 bytes). The TLBs for each summarization were computed by comparing the distances to the lower bounds for 100 random queries against all the other points of the dataset, for each of the datasets we generated.

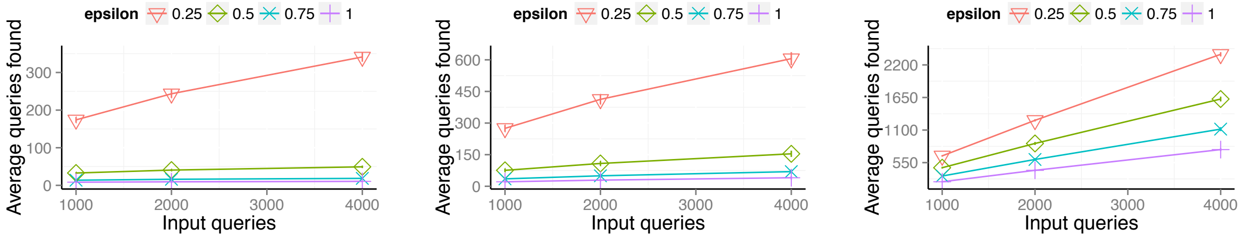
Figure 8 shows the average relative summarization errors for each dataset (averaged over the 100 different benchmarks generated). The results show that 1NN densification results to almost equal effort for all summarizations, while random densification tends to over-penalize bad summarizations and favor good ones. Both situations are not desirable and cannot be useful. In contrast, equi-densification has an effort much more closely related to the summarization error across all datasets. As a result, equi-densification well captures the actual pruning power of each summarization and does not over-penalize or under-penalize any of the summarizations.

6.3 Case Study on Actual Indexes

In our last experiment, we generated 85 datasets with 3 equi-densified queries corresponding to each one of them, which we will refer to as EDQ, and 3 additional queries (per dataset) that were randomly selected from the input queries sample without any densification. The 3 EDQ queries have hardness values of 0.1, 0.3 and 0.5 ($\epsilon = 1.0$). We generated 510 queries in total, half of which were random and half equi-densified. Our goal for this study is to demonstrate the qualitative difference of using our query workload versus a workload of randomly generated queries.

Figure 9 shows the histograms of the distribution of the hardnesses for the queries on each workload for every dataset for $\epsilon = 1.0$. Again, the random workloads are concentrated on easy queries with only a very small number of hard queries. On the contrary, the EDQ workload has been designed to produce queries of varying hardness values, and as a result their histograms contain equal number of queries in the 0.1, 0.3 and 0.5 bucket. This confirms that our method produces queries with desired properties.

In order to specifically evaluate the effect of hard queries, we further split the random workload into two sets, resulting in 3 different workloads: Random, where we use all the randomly selected queries, Random-H, where we only use



(a) RandWalk dataset (b) DNA dataset (c) EEG dataset

Figure 7: Number of independent queries found for each dataset for various input sample query sizes.

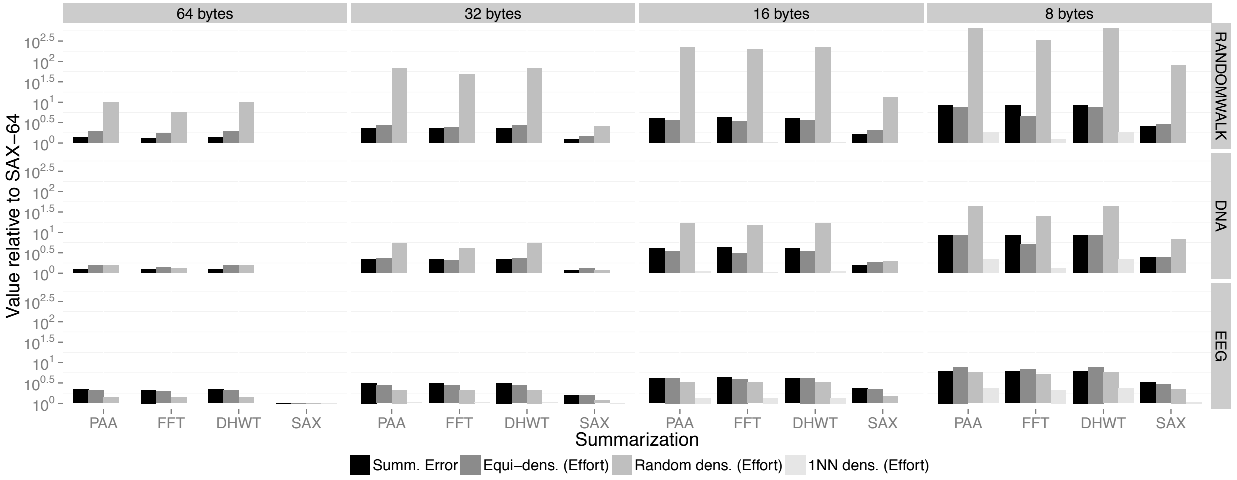


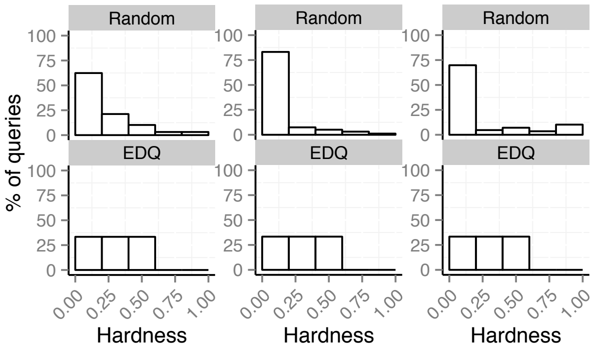
Figure 8: Minimum efforts for different summarization techniques at different resolutions (8-64 bytes) representing 256 point data series, compared to the summarization error (1-*TLB*). All the values have been normalized against SAX-64 which was the overall tightest summarization method.

queries with hardness larger than 0.5, and EDQ generated by our method. We indexed all three datasets with both *i*SAX [30] and R-Trees [13] with PAA [17], and measured the average query answering time per workload. Figure 10 illustrates the normalized query answering time. The results show that when the Random workload is used, queries are on average easy, and consequently, the two indexes seem to have similar performance. The same observation also holds when only the hard queries are selected using Random-H, indicating that simply selecting the hard queries of a randomly generated query workload cannot lead to a good query workload.

The real difference comes when the workload becomes harder using the EDQ workload. In this case, the differences between the indexes become more prominent. The reason behind this can be intuitively seen in Figure 11, where we plot the distribution of the distances to the query's nearest neighbor in the ϵ-area for the three different workloads. We can see that with random queries, (Random and Random-H), the vast majority of the points are located towards the large ϵ values. The difference between Random and Random-H is just on the number of points in each bucket. As we discussed earlier, such a distribution of points cannot capture the relative TLB of different indexes, as there are fewer points in small range to the true answer and many more points in larger range. On the other hand, the distribution of EDQ is very different from the others, which ensures there are roughly equal number of points for the corresponding $ATLB$ bucket.

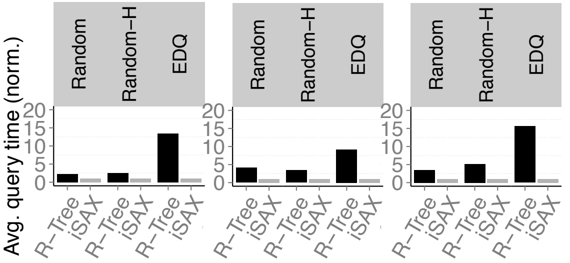
7. CONCLUSIONS

In this work, we focus on the problem of how to systematically characterize a data series query workload, and subsequently, how to generate queries with desired properties, which is a necessary step for studying the behavior and performance of data series indexes under different conditions. We demonstrate that previous approaches are not viable solutions as they are biased toward easy queries. We formally define the key concept of query hardness and conduct an extensive study on hardness of a data series query. Finally, we describe a method for generating data series query workloads, which can be used for the evaluation of data series summarizations and indexes. Our experimental evaluation demonstrates the soundness and effectiveness of the proposed method.



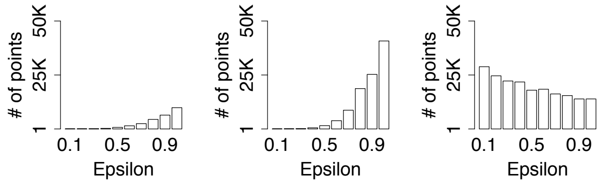
(a) RandWalk (b) DNA (c) EEG

Figure 9: Histogram of hardnesses for $\epsilon = 1.0$.



(a) RandWalk (b) DNA (c) EEG

Figure 10: Average query answering time comparison between iSAX (256 characters, 16 segments) and R-Tree (PAA with 8 segments) normalized over iSAX.



(a) Random (b) Random-H (c) EDQ

Figure 11: Distribution of points in $\epsilon = 1.0$ area for 3 types of queries for RandWalk.

References

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *FODO*, 1993.

[2] I. Assent, R. Krieger, F. Afschari, and T. Seidl. The ts-tree: Efficient time series search and retrieval. In *EDBT*, 2008.

[3] S. D. Bay, D. Kibler, M. J. Pazzani, and P. Smyth. The uci kdd archive of large data sets for data mining research and experimentation. In *SIGKDD Explorations*, 2000.

[4] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *ICDT*, 1999.

[5] A. Camerra, T. Palpanas, J. Shieh, and E. Keogh. iSAX 2.0: Indexing and mining one billion time series. In *ICDM*, 2010.

[6] A. Camerra, J. Shieh, T. Palpanas, T. Rakthanmanon, and E. Keogh. Beyond one billion time series: indexing and mining very large time series collections with isax2+. *KAIS*, 2013.

[7] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD*, 2002.

[8] K.-P. Chan and A.-C. Fu. Efficient time series matching by wavelets. In *ICDE*, 1999.

[9] Q. Chen, L. Chen, X. Lian, Y. Liu, and J. X. Yu. Indexable pla for efficient similarity search. In *VLDB*, 2007.

[10] M. Dallachiesa, B. Nushi, K. Mirylenka, and T. Palpanas. Uncertain time-series similarity: Return to the basics. In *VLDB*, 2012.

[11] M. Dallachiesa, T. Palpanas, and I. F. Ilyas. Top-k nearest neighbor search in uncertain data series. In *VLDB*, 2015.

[12] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, 1994.

[13] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, 1984.

[14] P. Huijse, P. A. Estévez, P. Protopapas, J. C. Principe, and P. Zegers. Computational intelligence challenges and applications on large-scale astronomical time series databases. *IEEE Comp. Int. Mag.*, 9(3), 2014.

[15] K. Kashino, G. Smith, and H. Murase. Time-series active search for quick retrieval of audio and video. In *ICASSP*, 1999.

[16] S. Kashyap and P. Karras. Scalable knn search on vertically stored time series. In *KDD*, 2011.

[17] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *KAIS*, 3, 2000.

[18] E. Keogh and M. Pazzani. Scaling up dynamic time warping to massive datasets. In *PKDD*, 1999.

[19] F. Korn, H. V. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *SIGMOD*, 1997.

[20] H. Kremer, S. Günnemann, A.-M. Ivanescu, I. Assent, and T. Seidl. Efficient processing of multiple dtw queries in time series databases. In *SSDBM*, 2011.

[21] C.-S. Li, P. Yu, and V. Castelli. Hierarchyscan: a hierarchical similarity search algorithm for databases of long sequences. In *ICDE*, 1996.

[22] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *DMKD*, 2003.

[23] J. Lin, R. Khade, and Y. Li. Rotation-invariant similarity in time series using bag-of-patterns representation. *J. Intell. Inf. Syst.*, 39(2), 2012.

[24] D. Rafiei and A. Mendelzon. Similarity-based queries for time series data. In *SIGMOD*, 1997.

[25] D. Rafiei and A. Mendelzon. Efficient retrieval of similar time sequences using dft. In *ICDE*, 1998.

[26] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *KDD*, 2012.

[27] K. V. Ravi Kanth, D. Agrawal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *SIGMOD*, 1998.

[28] P. Schäfer and M. Högqvist. Sfa: A symbolic fourier approximation and index for similarity search in high dimensional datasets. In *EDBT*, 2012.

[29] D. Shasha. Tuning time series queries in finance: Case studies and recommendations. *IEEE Data Eng. Bull.*, 22(2), 1999.

[30] J. Shieh and E. Keogh. isax: Indexing and mining terabyte sized time series. In *KDD*, 2008.

[31] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh. Experimental comparison of representation methods and distance measures for time series data. *DMKD*, 26(2), 2013.

[32] Y. Wang, P. Wang, J. Pei, W. Wang, and S. Huang. A data-adaptive and dynamic segmentation index for whole matching on time series. In *VLDB*, 2013.

[33] L. Ye and E. J. Keogh. Time series shapelets: a new primitive for data mining. In *KDD*, 2009.

[34] B.-K. Yi, H. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *ICDE*, 1998.

[35] K. Zoumpatianos, S. Idreos, and T. Palpanas. Indexing for interactive exploration of big data series. In *SIGMOD*, 2014.

[36] K. Zoumpatianos, S. Idreos, and T. Palpanas. Rinse: Interactive data series exploration. In *VLDB*, 2015.