Noname manuscript No.
(will be inserted by the editor)

Generating Data Series Query Workloads

Kostas Zoumpatianos · Yin Lou · Ioana Ileana · Themis Palpanas · Johannes Gehrke

Received: date / Accepted: date

Abstract Data series (including time series) 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. 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. 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 query workloads. 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. This is the first paper that introduces a method for quantifying hardness of data series queries, as well as the ability to generate queries of predefined hardness.

Keywords Time series · Data series · Similarity search · Indexing · Query workload generation

K. Zoumpatianos
Harvard University & LIPADE, Paris Descartes University
E-mail: kostas@seas.harvard.edu

Y. Lou
Airbnb Inc.
E-mail: yin.lou@airbnb.com

I. Ileana
LIPADE, Paris Descartes University
E-mail: ioana.ileana@parisdescartes.fr

T. Palpanas
LIPADE, Paris Descartes University
E-mail: themis@mi.parisdescartes.fr

J. Gehrke
Microsoft Inc.
E-mail: johannes@microsoft.com

1 Introduction

Data series (i.e., ordered sequences of values)[1] appear in many diverse domains ranging from audio signal [20] and image data processing [42], to financial [38] and scientific data [19] analysis, and have gathered the attention of the data management community for almost two decades [32,39,6,30,12,13,45].

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 [14,16,35,22,3]. However, such queries become challenging when performed on very large data series collections [7,34]. 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 [23,32,33,27,1,27,21,9,28] so that they can then be efficiently indexed [32,39,6,41,2,37].

Motivation. We note that despite the considerable amount of work on data series indexing [15,33,9,39, 21], no previous study paid particular attention to the

[1] Note that when these values are measured over time (usually at fixed time intervals), we call them *time series*. However, time series are just one special case of data series: a series can also be defined over other measures (e.g., mass in mass spectroscopy, position in genome sequences, angle in radial chemical profiles, etc.). For the rest of this paper, we will use the terms *sequence*, *data series*, and *time series* interchangeably.

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 thus these 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.

Query Workload Generation. 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, without assuming a single summarization and index or a specific test dataset.[2] 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 it is based on the properties of the lower bounding functions 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 first investigate the desirable size and structural properties of the area around the query's nearest neighbor, and then break down the problem of generating query workloads into three subproblems.

- Determine how to select candidate queries which can be independently controlled.
- Determine how many data series to add in that area.
- Determine how to add data series.

[2] Website: `http://www.mi.parisdescartes.fr/~themisp/bends/`

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.

Contributions. The contributions of this paper[3] can be summarized as follows.

- **Index-dependent query answering effort.** We identify the summarization- and query-specific factors that make query answering on data series indexes expensive. Such measures can capture the "effort" a *given index* needs to make in order to answer a *specific query*.
- **Intrinsic query hardness.** We define measures that can effectively capture the "effort" of various summarization/index combinations, while being both summarization- and index-independent. We call this intrinsic measure, the "hardness" of a query.
- **Queries with meaningful intrinsic query hardness.** We recommend a set of principles that should be followed when generating evaluation queries such that the intrinsic "hardness" measure can accurately capture various summarization/index-specific query "efforts."
- **Generating workloads.** 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.
- **Experimental evaluation.** We demonstrate how our workload generator can be used to produce query workloads, based on both real and synthetic datasets.

Paper outline. The rest of this paper is organized as follows. Section 2 presents the necessary background.

[3] This work (built on our preliminary version [46]) includes a more precise formal definition of the problem, a deeper analysis of previous workloads, a robust geometric solution for placing nearest neighbors at predefined distances from a query that removes earlier limitations, and an expanded experimental evaluation section.

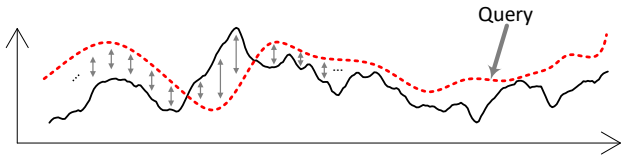


Fig. 1 An example data series and its distance to a query.

Symbol	Description
$\boldsymbol{x}$	A data series
$\boldsymbol{q}$	A query data series
$\mathcal{D}$	A set of N data series (dataset)
$\mathcal{Q}$	A set of M queries (query workload)
$DIST(\boldsymbol{x}, \boldsymbol{q})$	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.

We formally define the notion of hardness and structural requirements of data series workloads in Section 3. We then discuss common datasets widely used in the past in Section 4, and describe our approach for generating a query workload in Section 5. Finally, we present an experimental evaluation of our approach in Section 6, and conclude in Section 7.

2 Preliminaries

A data series $\boldsymbol{x} = [x_1, ..., x_d]$ is an ordered list of real values with length d. Since data series can be represented as points in a d-dimensional space, in this paper, we also call data series *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. An example data series and its distance to a query can be seen in Figure 1. 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 [6,39,41]. Table 1 summarizes the notation used in this paper.

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 [32,33,27,1], DHWT [27, 21,9], PCA and SVD [25,36], but also data series specific data summarization techniques such as SAX [28], PAA [23], APCA [8] and *i*SAX [39,6]. We briefly describe the most prominent ones below.

Piecewise Aggregate Approximation (PAA) [23] approximates a data series by splitting it into equal segments and calculating the average value for each segment.

Discrete Fourier Transform (DFT) [32,33,27,1] uses Fourier transforms to convert a data series to the frequency domain and represents it as a list of coefficients.[4]

Discrete Haar Wavelet Transform (DHWT) [27, 21,9] uses Haar wavelets in order to transform a data series into a list of coefficients.

Symbolic Aggregate approXimation (SAX) [28] is built above PAA with the addition that the value space is also discretized, leading to a symbolic representation with very small memory requirements.

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. To use such summarizations and make exact query answering feasible, indexes employ 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 L. Given two summarized data series, L returns a lower bound of their true distance in the original space.

2.2 Data Series Indexing

Data series 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. The goal is that each group contains series that are similar to each other, therefore, defining a partitioning of the data space. Moreover, the employed summarizations are carefully designed in order to support a distance measure (in the summarized space) that is always a lower bound of

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

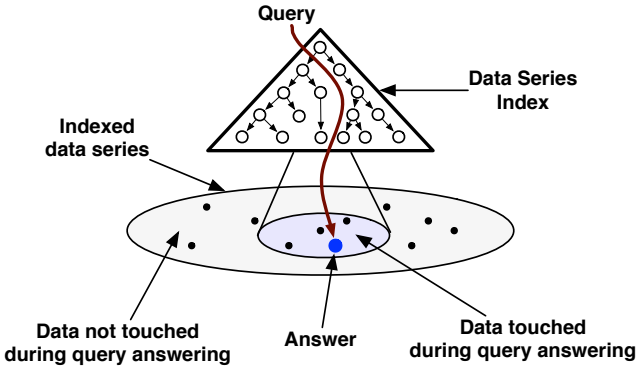


Fig. 2 An index structure built above a set of data series, pruning the search space for a query.

the corresponding distance in the original (raw data) space. This allows us to use the index for pruning the search space by removing true negatives, while at the same time guaranteeing no false negatives (i.e., no true positive is ever pruned).

The notion of an index is illustrated in Figure 2, where the multiple levels of aggregation are represented as nodes (empty circles in the figure) in a tree-like structure, and the raw data series (black points in the figure), lie below it.

In general, data series indexes that support exact nearest neighbor search can be divided into three broad categories as follows.

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 [1,15,32,33], DHWT [9] and Piecewise Linear Approximation (PLA) [10].

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 [41] (based on APCA), ADS [44] and *i*SAX index [39,6,7] (built on an indexable version of SAX), and SFA index [37] (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 [27,21].

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

In this section, we investigate the factors that affect the query answering performance of data series indexes and summarizations. We start the discussion with the requirements for meaningful and effective workloads (Section 3.1). We then give an introduction on lower bounding functions used by summarizations in Section 3.2, and study the *minimum effort* that an index can make in order to answer a given query. This allows us to define an index-dependent measure of query answering effort in Section 3.2.2. In order to do this, we connect the de facto measure for quantifying the quality of a summarization, called the tightness of the lower bound (TLB), to the percentage of data that an index will need to check, in the best case, in order to answer a query.

Since the amount of data an index has to check is tightly connected to a specific TLB, and is thus index-dependent, we continue our discussion with an index-independent, *intrinsic query hardness* definition in Section 3.3. This hardness notion will be based solely on the properties of the dataset and the query at hand. We will however show how, when certain conditions hold, this index-independent measure will be able to accurately capture the effort that various indexes and summarizations will need to do in order to answer this query. As we mentioned in Section 2.2, data series indexes can be considered as multi-level summarizations of data series (see also Figure 2). For this reason, in the rest of the section we reason using summarizations.

3.1 Requirements for Meaningful and Effective Workloads

According to the discussion above, a "good" workload should contain queries that effectively capture the quality of the summarizations. Moreover, the intrinsic hardness of each query should accurately capture the corresponding effort needed to answer each query. Accordingly, we need queries and a hardness definition with the following accuracy properties:

1. **Intra-index (inter-query) hardness accuracy.** Data series indexes are multi-level summarizations of data series. Therefore, at each level l_I of index I, this index summarizes data series with a specific summarization $Summarization(l_I)$. Assume that for *two queries* $\boldsymbol{q}_1, \boldsymbol{q}_2$ and *an index* I, the efforts of answering these queries using this index at some level $l_I \in \mathcal{S}$ are $\text{Effort}(l_I, \boldsymbol{q}_1)$, and $\text{Effort}(l_I, \boldsymbol{q}_2)$, respectively.[5] Using the set $\mathcal{S}$ we denote the set of all possible summarizations/index-levels. Then, an ideal intrinsic hardness definition would need to accurately capture how much bigger the effort for $\boldsymbol{q}_2$ is over the effort of $\boldsymbol{q}_1$ *for the given index level.* Moreover this ratio should be consistent across different summarizations/index levels. Formally, if the hardness values for $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ are $\text{Hardness}(\boldsymbol{q}_1)$ and $\text{Hardness}(\boldsymbol{q}_2)$, it should hold for every $l_I \in \mathcal{S}$ that:

$$\frac{\text{Effort}(l_I, \boldsymbol{q}_1)}{\text{Effort}(l_I, \boldsymbol{q}_2)} = \frac{\text{Hardness}(\boldsymbol{q}_1)}{\text{Hardness}(\boldsymbol{q}_2)} \; \forall \; l_I \in \mathcal{S}.$$

   Thus, given an effort definition for a given summarization, its ratio for any two queries should be:

   (R1.a) *stable accross different summarization efforts.*

   (R1.b) *equal to the ratio of their intrinsic query hardnesses.*

2. **Inter-index (intra-query) effort accuracy.** We now assume *a single query* $\boldsymbol{q}$ and *two indexes* I_1 and I_2. The summarization error is a commonly accepted measure of how good a summarization is: it is computed as the difference (e.g., when using Euclidean Distance) between the true values of the series, and the estimated, reconstructed values based on the summarization.

   At each level l_{I_1} of index I_1, the summarization error is denoted as $\text{SummError}(l_{I_1})$, and for index I_2 it is denoted as $\text{SummError}(l_{I_2})$. Given two summarization errors, for a specific level of each index (or two different levels of the same index), the corresponding efforts should capture how much bigger the error of one level is with respect to the other level. Thus, if the effort values for answering the query using each summary are $\text{Effort}(l_{I_1}, \boldsymbol{q})$, $\text{Effort}(l_{I_2}, \boldsymbol{q})$, then for every level of each index it should hold that:

   (R2) *The ratio of efforts of a single query accross two different summarizations is equal to the ratio of their summarization error.*

   Alternatively, this can be formally described as:

$$\frac{\text{Effort}(l_{I_1}, \boldsymbol{q})}{\text{Effort}(l_{I_2}, \boldsymbol{q})} = \frac{\text{SummError}(l_{I_1})}{\text{SummError}(l_{I_2})}.$$

   It is important to note that this is a property of the queries, and not a property of our hardness measure. Therefore, we should either generate, or select queries that respect this property.

We argue that a hardness definition that satisfies these criteria across different summarizations/indexes and different queries is the desired intrinsic hardness definition. Such a definition, will subsequently allow us to properly construct queries using a predefined hardness, effectively stressing indexes/summarizations to efforts that correspond to the error of their summarizations. We note that a coarse (bad) summarization (i.e., one that does not allow a lot of pruning to be performed), corresponds to a very rough representation of the indexed data, and is commonly found in the higher level nodes of an index, where it summarizes a large number of data series. On the contrary, a fine (good) summarization (i.e., one that corresponds to a very precise representation of the data series) allows more pruning to be performed, and is commonly found at the leaf level of an index, where it summarizes a small number of data series (usually the size of a disk-page). The smaller the total number of levels in an index is, and the better the pruning at each level is, the better the overall performance of the will be. Consequently, we study summarizations of varying precision as a proxy to quantifying global index effort and query hardness.

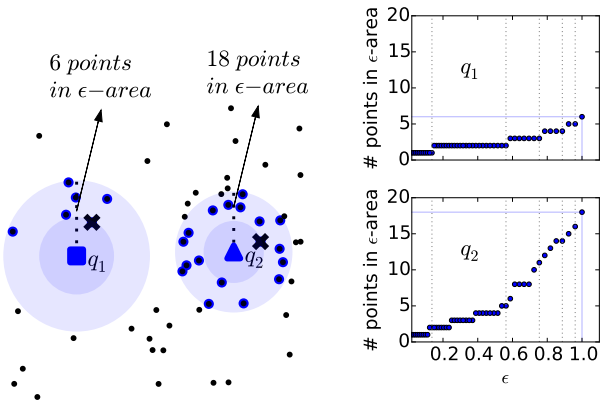
3.2 Index-Dependent Query Answering Effort

As we have seen earlier, certain properties should hold for generating "good" workloads. All of these properties are expressed as constraints on a quantified query answering effort. In this subsection we study summarizations and the tightness of their lower bounds in order to provide such a definition, which as we will see later on, we call *Minimum Effort* of an index to answer a given query.

3.2.1 Lower Bounds, ATLB and TLB

When navigating an index, we make use of the lower bounds of the true distances of data series in the original space (computed based on the summarizations of the series). This technique guarantees that there will be no false negatives in the candidate set. Nevertheless, it does not exclude the existence of false positives. The reason behind this is that during the query answering process, we can move to lower levels of the index that contain better (finer) summarizations, performing further lower bound computations, until we reach a leaf

[5] Informally, the *effort* is the amount of work that an index needs to perform. We formally define the notion of effort later in this section.



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

Fig. 3 Two random queries with nearest neighbors depicted with "×."

node. At that point, the summarization cannot be further improved. Therefore, we need to fetch the raw data series as well, and use those in order to make exact distance computations, and to filter out the false positives. This procedure then guarantees the correctness of the final answer (no false negatives and no false positives).

In order to capture this notion of how good (or useful) a summarization is, we can use the Tightness of Lower Bound (*TLB*) [40], which is measured as the average ratio of the lower bound distance 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 summarized data series pair.

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

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

Example 1 Figure 3(a) demonstrates the implications of *ATLB*. 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

$$L(\boldsymbol{x}, \boldsymbol{q}) \leq MINDIST(\boldsymbol{q}) \quad (2)$$

$$\Leftrightarrow DIST\boldsymbol{x}, \boldsymbol{q}) \leq \frac{MINDIST(\boldsymbol{q})}{ATLB(L, \boldsymbol{x}, \boldsymbol{q})}. \quad (3)$$

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 (3) is always no less than $MINDIST(\boldsymbol{q})$. These ranges are depicted as disks in Figure 3(a).

The *TLB* can then be defined as the aggregate *ATLB* over a set of queries and data series:

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

$$TLB(L) = \frac{1}{|\mathcal{Q}| \times |\mathcal{D}|} \sum_{\boldsymbol{q} \in \mathcal{Q}} \sum_{\boldsymbol{x} \in \mathcal{D}} ATLB(L, \boldsymbol{x}, \boldsymbol{q}). \quad (4)$$

Note that the *TLB* is small for an inaccurate summarization, i.e., such a 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. This is indeed an important criteria for measuring index performance. To formalize such difference in the effort of more or less effective summarization/indexing techniques, we hereafter introduce the notion of *Minimum Effort*. We will then examine in the following subsections how this notion can be linked to a meaningful concept of hardness of a query within a workload.

3.2.2 Index-dependent Measure of Minimum Effort

We define *ME* as the ratio of the number of series that an index has to fetch for answering some query, over the total number of series in the index, when the index uses the best (i.e., the most accurate) summarization it contains for these data series. The *ME* aims to describe the absolute minimum amount of raw data that an index will have to touch in order to answer a query.

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

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

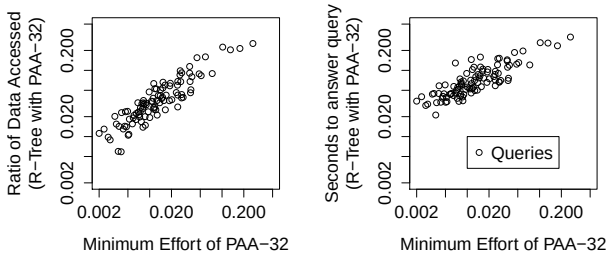


Fig. 4 100,000 random walk generated data series and 100 queries.

As we have seen in Example 1, given a fixed $ATLB$ between the query and the data series, the 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 it is not possible to prune these data series. 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; in most cases, the actual effort that the index will actually do will be larger. This is because the search for the solution hardly ever starts with the real answer as the best-so-far.

Nevertheless, in practice, the ME turns out to be a fairly good indicator of the actual query answering effort. To demonstrate this we have run a small experiment with 100 queries performed on 100,000 synthetic series, both generated with a simple randomwalk, a standard process that we describe in detail in Section 4.1. We converted all data series into PAA with 32 segments, and built an R-Tree index above them. We then fired the 100 queries to the index and measured the query answering time, as well as the ratio of the data accessed in order to answer each query. For every query we also measured its ME (Def. 3). We present the results in Figure 4. We can see that the ME is highly correlated to both the query answering time and the amount of data that an index checks. Additionally, we can see that the data organization of the index causes an additional overhead, since the ratio of data checked by the R-Tree is constantly higher than the minimum effort, and the time is higher than that required for simply accessing the data (due to the additional time cost of the index traversals).

3.3 Intrinsic Query Hardness

Recall that our goal is to investigate the general, summarization-independent, intrinsic hardness of queries in a workload. In this subsection, we discuss how one can define such a measure. We start by providing an index-dependent query hardness measure, and show under how and under which conditions this can be generalized to an index-independent measure.

3.3.1 ϵ-dependent Hardness

Since ME 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 ϵ-hardness of a query as follows.[6]

Definition 5 (ϵ-hardness) Given $\epsilon \geq 0$, the ϵ-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.

Example 2 Using the example in Figure 3(a), let us assume that the total number of points in the dataset is 100. ϵ-hardness computation for $\epsilon = 1.0$ accordingly yields $\alpha^{1.0}(\boldsymbol{q}_1) = 0.06$ and $\alpha^{1.0}(\boldsymbol{q}_2) = 0.18$.

In the following, we will discuss under which conditions the ϵ-hardness can be used as a meaningful intrinsic hardness measure for queries within a workload.

3.3.2 ϵ-independent Hardness

As mentioned in the introduction, we are interested in an intrinsic hardness measure for queries that would essentially enable a meaningful comparative analysis of the performances of various summarization/indexing techniques.

We introduced above the notion of ME, which provides a good indication of the efficiency of a summarization w.r.t. a query. We state the following two objectives:

[6] A similar definition has been proposed in the past [5].

1. Select, or generate queries (irrespective of their hardness values), which are able to reveal the differences between effective and non-effective summarizations. This is the inter-index (intra-query) effort accuracy property that we talked about in the beginning of Section 3.
2. Define an ϵ-independent intrinsic hardness measure for queries in a workload, whose values are *representative of the ME*, across *all summarizations*. This is the intra-index (inter-query) intrinsic hardness accuracy property.

We will now study the structural criteria that allow ϵ-hardness to be a robust hardness measure for the queries within a workload, according to the requirements stated above. In particular, we focus on two important aspects:

- What are the structural requirements the ϵ-area of a query should meet in order to reveal differences across summarizations.
- What is the ϵ value that can render an ϵ-bound hardness measure to be ϵ-independent.

In Section 4, we perform an evaluation of the aforementioned criteria in previous workloads and datasets, showing that they are inadequate for effective index comparison. Then, in Section 5, we present a method that allows users to generate workloads that fulfill the two objectives listed above.

The choice of an ϵ value. Let us go back to Examples 1 and 2, and assume a summarization with a (constant) *ATLB* of 0.5. As we have shown above, all points within a radius of $\frac{0.33}{0.5} = 0.66$ from q_1 participate in the *ME* of our considered summarization. Note that this radius corresponds to an ϵ-area with ϵ=1.

For this precise value of ϵ, we are sure that the respective ϵ-area covers exactly the points participating in the *ME*. It turns out that, in order to ensure that an ϵ-area around a query q in a workload covers all the points that participate in the *ME* of a specific summarization (or index) *with a constant ATLB*, the following must hold (direct derivation from Equation (3)):

$$\epsilon \geq \frac{1}{ATLB(L, \boldsymbol{x}, \boldsymbol{q})} - 1. \quad (5)$$

The equality in the above formula holds for an ϵ value, for which the corresponding area contains *all* the points involved in the *ME* of the specific summarization, and *exactly* those points. Accordingly, the ϵ-hardness is equal to the *ME*.

In order to demonstrate the relation between ϵ-hardness and minimum effort, we run an experiment on 100,000 randomwalk generated data of size 256, performing 1500 queries. For various ϵ values and summarizations we measured the correlation between ϵ-hardness and *ME*. As we can see in Figure 5, the correlation between the effort of each different summarization and ϵ-hardness peaks at a different ϵ value, and is thus tied to a different ϵ. Moreover, the highest the resolution of a summarization, the smaller this ϵ area to which *ME* is tied is.

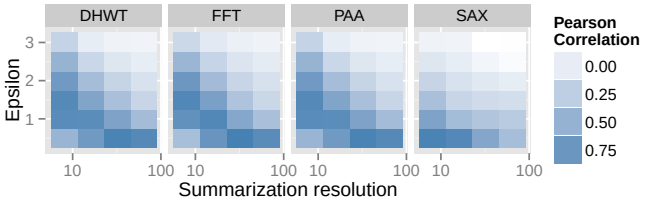


Fig. 5 Correlations for ϵ-hardnesses and Minimum Efforts in 100,000 data series and 1,500 queries.

The above observations have various implications. Let us assume now that for our example we had picked an $\epsilon = 0.1$ for the hardness computation. The computed hardness would then be *expectedly lower* that the *ME* of our considered summarization, since all *ME* points between $\epsilon = 0.01$ and $\epsilon = 1.0$ will be "unaccounted for." Assume further that in the interval between $\epsilon = 0.01$ and $\epsilon = 1.0$ we would have a very large number of dataset points. The estimated hardness of $\boldsymbol{q}_1$, computed as ϵ-hardness for $\epsilon = 0.01$, would then be *significantly lower* than the actual *ME*. The particular difference would be furthermore uncontrolled by the query hardness.

By the above, we argue that in order to be able to ensure that ϵ-hardness is a meaningful hardness measure within a workload, the value of ϵ to be chosen should be such that few/no points participating in the *ME* of the summarizations tested are located outside the ϵ-areas.

Considering again the case of constant *ATLB*s, it is obvious that if this "coverage" property holds for a summarization it will hold for "better" summarizations (i.e., with higher *ATLB*) as well. In fact, ensuring an equality in Equation 5 for the *worst* summarization tested guarantees that the *ME* points are covered by ϵ-areas for all summarizations.

Let us now go back to our original question: *which ϵ value should one choose to ensure a meaningful hardness measure?* The answer to this question could simply be *any value*. Nevertheless, the lower such a value is, the narrower the range of summarizations that can be meaningfully tested by employing the respective workload is. This is true for the following reason.

This ϵ value, should be chosen according to the worst summarization that we are interested in testing: all other summarizations equally bad, or worse than

this reference summarization, will be facing the same effort. As a result, we need an ϵ value large enough to cover all reasonably bad indexes. We call this measure: *Max-Hardness*. We will empirically show in Section 4 how to chose an ϵ, and which some possible values are for covering the most common summarizations.

Structure of the ϵ-area. The choice of a suitable ϵ value, ensuring that no *ME* points are "left aside" is a first necessary step towards turning ϵ-hardness into an intrinsic hardness measure. However, as we will demonstrate below, this first step is not sufficient.

The intra-index (inter-query) hardness accuracy property. As we mentioned in the first requirement of the beginning of Section 3, we need to ensure that the effort of answering queries using a single summarization is accurately captured by our hardness definition. That is, for any given summarization (or index), the ratio of the efforts of two different queries for this summarization should be equal to the ratio of the hardnesses of these two queries.

However, if we merely chose the largest ϵ that we can, as we can also see in Figure 5, very few *ME*s of different summarizations would be highly correlated to this ϵ-hardness (Max-Hardness). This is because the amount of points that exist in the larger area is not of fixed proportions to the amount of points that exist in every other smaller ϵ area. Thus, this query can be arbitrarily hard for arbitrary summarizations, based on the amount of data that exist in the ϵ-area tied to it.

In order to make this clear, we need to revisit Example 2. We now observe that queries $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ have hardnesses (computed as ϵ-hardnesses for $\epsilon = 1.0$) of 0.06 and 0.18, respectively. These values suggest that $\boldsymbol{q}_2$ is 3 times harder than $\boldsymbol{q}_1$. We now assume two hypothetical summarizations: S_1 with a (constant) $ATLB = 0.5$, and a "better" summarization, S_2, with a (constant) $ATLB = 0.83$. As we have seen earlier, for a constant ATLB, all points within a radius of 0.66 from q_1 participate in the *ME* of our considered summarization. this radius corresponds to an ϵ-area with ϵ=1.

Using the same reasoning, it is easy to show that S_2's *ME* on $\boldsymbol{q}_1$ is 0.02. However, S_2's *ME* on $\boldsymbol{q}_2$ is also 0.02! Therefore, for S_2, the two queries appear to be *equally hard*, making the relative performance of two queries behave differently, in relation to the summarization used.

Then, the following questions arise. How could we ensure that $\boldsymbol{q}_1$ is 3 times harder than $\boldsymbol{q}_2$ for S_2? How could we ensure this in general, for any other summarization within the limits of those addressed by the chosen ϵ? In other words, how can we make this property hold for all possible summarizations?

It turns out that, according to Equation 5, S_1's *ME* is best characterized by the ϵ-hardness computed for $\epsilon = 0.2$, which in fact yields $\alpha^{0.2}(\boldsymbol{q}_1) = 0.02$ and $\alpha^{0.2}(\boldsymbol{q}_2) = 0.02$. What we would want instead is for the 1:3 ratio to still hold at $\epsilon = 0.2$ and for any other ϵ. Consequently, we would like the following property to hold for all summarizations:

$$\frac{\alpha^{\epsilon}(\boldsymbol{q}_1)}{\alpha^{\epsilon}(\boldsymbol{q}_2)} = \frac{\alpha^{\hat{\epsilon}}(\boldsymbol{q}_1)}{\alpha^{\hat{\epsilon}}(\boldsymbol{q}_2)}, \forall \hat{\epsilon} \leq \epsilon. \tag{6}$$

Since each effort is tied to a given ϵ-hardness, (R1.a) would be satisfied, further on, since we can now use Max-Hardness (the hardness of the biggest ϵ) as our intrinsic hardness measure, (R1.b) would also be satisfied.

Note that in our example, this property does not hold for $\epsilon = 1.0$ for the query workload comprising $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$. Therefore, this workload is *unsuitable for comparing S_1 and S_2* (or for comparing any summarizations that include S_2 and others that are "better" than S_2).

We observe that the critical issue is related to the *distribution* of the ϵ-areas around $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$, i.e., how the points are distributed in these areas. To correct this issue, one needs to either *filter-out* non-suitable queries, or *alter* the distribution of points in their ϵ-areas. We will discuss the latter choice in Section 5.

The inter-index (intra-query) effort accuracy problem. Recall that we also need to ensure that the second requirement (R2) defined at the beginning of Section 3 holds. Therefore, we need to ensure the ability of each query to penalize bad summarizations. This means that, if for a given query the ϵ-hardness does not increase proportionally to the summarization errors and their corresponding ϵ values, this query will be *unsuitable for differentiating the quality of the tested summarizations*. Then, this query could appear equally hard for summarizations whose quality is objectively very different, or it could appear disproportionately hard for some summarizations and disproportionately easy for others. Intuitively, the difference in effort across different summarizations would not be representative of the actual quality of the summarization, but instead it would be artificially affected by a biased placement of points around the query. We will call this situation the *inter-index (intra-query) effort accuracy problem*.

As we mentioned earlier, in order to avoid this problem, it should hold that for a given query the effort for a specific summarization over the effort of any other one should be proportional to their relative summarization errors.

Each summarization has a specific summarization error, which can be meaningfully measured as, $1 - TLB$: the smaller(/larger) this number is, the smaller(/larger)

the error is, and equivalently, the better(/worse) the TLB is. Therefore the following condition is necessary in order to guarantee the effort accuracy, for any query q_1, and any two different summarizations S_1 and S_2:

$$\frac{\mu^{S_1}(\boldsymbol{q}_1)}{\mu^{S_2}(\boldsymbol{q}_1)} = \frac{1 - TLB(S_1)}{1 - TLB(S_2)}. \quad (7)$$

Note that the criteria derived above are once again a structural condition that the ϵ-areas in a workload need to respect. Thus, they technically form requirements that when satisfied, allow the ϵ-dependent hardness to generalize into an ϵ-independent hardness, assuming a large enough ϵ value is chosen.

In the following section, we evaluate workloads used in the past in terms of these structural criteria and demonstrate that these criteria are not met, since:

(a) Indeed the effort of each summarization is tied to a different ϵ-hardness.
(b) The ratio of efforts does not correspond to the quality of the summarizations but merely to the distribution of the data.

Moreover, we demonstrate that all these workloads are heavily skewed towards easy queries, thus not allowing for insightful comparisons accross different data structures.

4 Evaluation of Previous Work

In this section, we review the characteristics of synthetic and real datasets, as well as the corresponding query workloads, that have been commonly used in the literature.[7] For capturing trends and shapes, we z-normalize (mean=0, std=1) all data series following common practice. We initially describe each one of our datasets and all summarization methods used. We then analyze them both in terms of structural properties as well as in terms of their hardness.

Based on the results of our analysis, we will argue that these popular datasets and workloads are both structurally unfit for comparing index structures, and mostly composed by easy queries.

4.1 Datasets and Workloads

We use 3 datasets, both synthetically generated and real. RandWalk [1,15,32,9,39,2,6,21,26,7,37,44]: This is a dataset synthetically produced using a random walk data series generator, with a step size tat varies according to a Gaussian distribution. We start with a fixed random seed and produce 200,000 data series of length 64, 256, and 1024.

EEG [43,24,2,26,34]: We use the electroencephalograms dataset from the UCI repository [4], and uniformly sample 200,000 non-overlapping data series of length 64, 256, and 1024 from the dataset to be used as the dataset.

DNA [6,7,44]: We use the complete Human DNA (Homo Sapiens), obtained from the Ensembl project.[8] We converted the dataset into data series following the approach[9] described in [39]. We uniformly sample the dataset to create 200,000 non-overlapping data series of length 64, 256, and 1024.

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 [6,21,7,44].
2. The entire dataset is used for indexing. A subset of it is selected as a query set, and a small amount of random noise is added on top of it [1,27,26].

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. We note that in these experiments with 100,000 data series, our goal is to identify the factors that affect index performance and not to experimentally compare the index structures themselves. The choice of the relatively small number of series is dictated by the nature of our analysis, which is very time-consuming, requiring the computation of *all* pair-wise distances.

In all our experiments, we used a sample of 1500 queries to generate our plots. Moreover, we used 4 standard data series summarization techniques to compute efforts, namely, Discrete Haar Wavelet Transforms (DHWT), Fast Fourier Transforms (FFT), Piecewise Aggregate Approximation (PAA), and Symbolic Aggregate approXimation (SAX) in 4 distinct resolutions each, ranging from 8 to 64 bytes. Finally, following previous work [1], we repeated our experiments with queries from within the dataset, to which we added a small amount of gaussian noise (5% as in [1]).

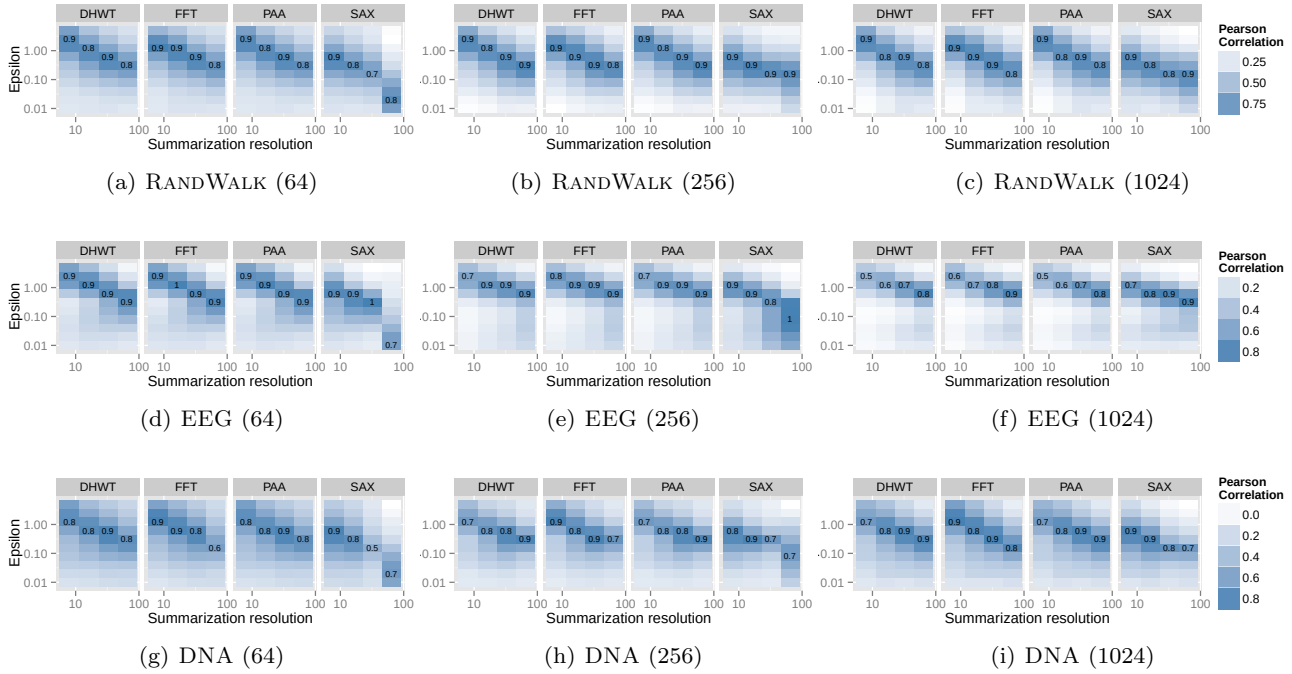
4.2 Structural Properties of Random Workloads

In the first part of our study, we analyze all workloads in terms of the structural requirements defined in the previous section.

[7] We also use the same datasets in our experimental section.

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

[9] This algorithm iterates over all symbols in the DNA sequence and constructs the series as a cumulative sum, which increases by 2 for every appearance of the base 'A'. By 1 for 'G'. And decreases by 1 and 2 for each appearance of 'C' and 'T', respectively.



(a) RandWalk (64) (b) RandWalk (256) (c) RandWalk (1024)

(d) EEG (64) (e) EEG (256) (f) EEG (1024)

(g) DNA (64) (h) DNA (256) (i) DNA (1024)

Fig. 6 Correlation between ϵ-hardness and Minimum Effort.

4.2.1 Effort and Intra-Index Hardness Accuracy

In our first experiment, we look into requirements (R1.a) and (R1.b). These two requirements allow us to guarantee that the ratio of efforts of two different queries remains stable across different summarizations, and that it is accurately captured by our intrinsic hardness definition. To achieve that we measured for every query the $\epsilon - hardness$ for various ϵ values, ranging from 0.01 to 2.5, and the *ME* for all resolutions of our 4 summarization techniques.

As with the experiment of Figure 5, we computed the Pearson correlation between each different $\epsilon - hardness$ and each summarization's *ME* across all queries. Figure 6 depicts the results. The x-axis corresponds to different resolutions of each summarization, while the y-axis represents different ϵ values that correspond to different $\epsilon - hardnesses$. The color represents the intensity of the correlation between each ϵ-hardness and *ME* (for each resolution). We also list the actual correlation values for the highest correlated pairs.

It is clear that each different summarization is tied to a specific ϵ value. This is the ϵ-hardness that best captures its *ME*. For every summarization, we call each best fitting ϵ-hardness the *Best-Hardness* for that summarization.

We additionally observe that the choice of a single ϵ value (possibly the largest one), is an inadequate measure for capturing the *ME* for all summarizations. This is because the number of points that lie at the largest possible ϵ area are not a fixed proportion of the number of points that lie in each smaller ϵ area. Thus, larger areas fail to consistently represent the amount of points existing in each different Best-Hardness ϵ area. As a result, choosing the largest ϵ value for random datasets violates our desired intra-index (inter-query) hardness accuracy property defined earlier (R1.a and R1.b).

The problems that arise from such violation are illustrated in Figure 7. In this plot, we used a query workload of 6 random walk data series and 4 distinct summarizations. For each query and summarization we computed the *ME* and the Best-Hardness. We compare this to the maximum ϵ hardness (Max-Hardness), which was fixed to 1.25 for all summarizations. (We chose this value, as this was the largest highly correlated ϵ found in our previous experiment.) All reported numbers are normalized in the range $[0, 1]$, where 1 corresponds to the largest observed value.

Looking at the *ME*s, it is obvious that for all summarizations $\boldsymbol{q}_2$ is the hardest query. However, for the first three summarizations, $\boldsymbol{q}_4$ was easier than $\boldsymbol{q}_3$, while for SAX32, $\boldsymbol{q}_4$ was harder than $\boldsymbol{q}_3$. Evidently, there is no ordering of the queries based on their *ME* for this workload that holds across all summarizations, since $\boldsymbol{q}_4$ is not consistently easier than $\boldsymbol{q}_3$. Consequently, the requirement (R1.a) does not hold.

In regards to requirement (R1.b), using a different *epsilon*-Hardness per summarization (Best-Hardness)

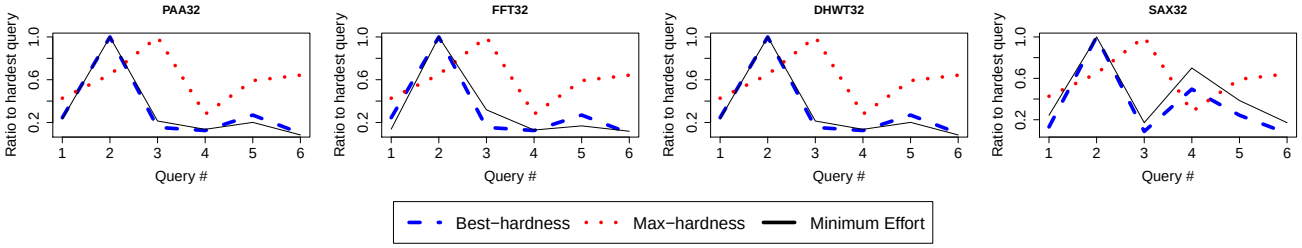
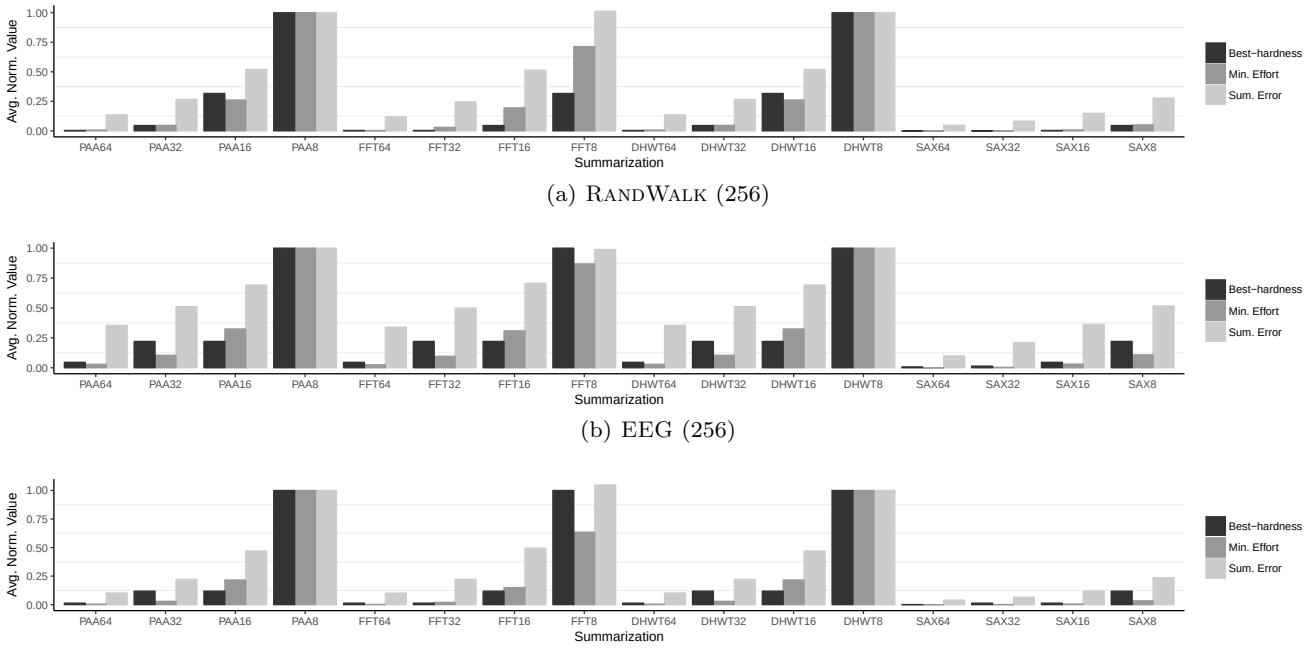


Fig. 7 A workload of 6 queries, their *ME*, Best-Hardness and Max-Hardness.



(a) RandWalk (256)

(b) EEG (256)

(c) DNA (256)

Fig. 8 Average normalized *ME*, Best-Hardness and Summarization Error for all datasets of length 256.

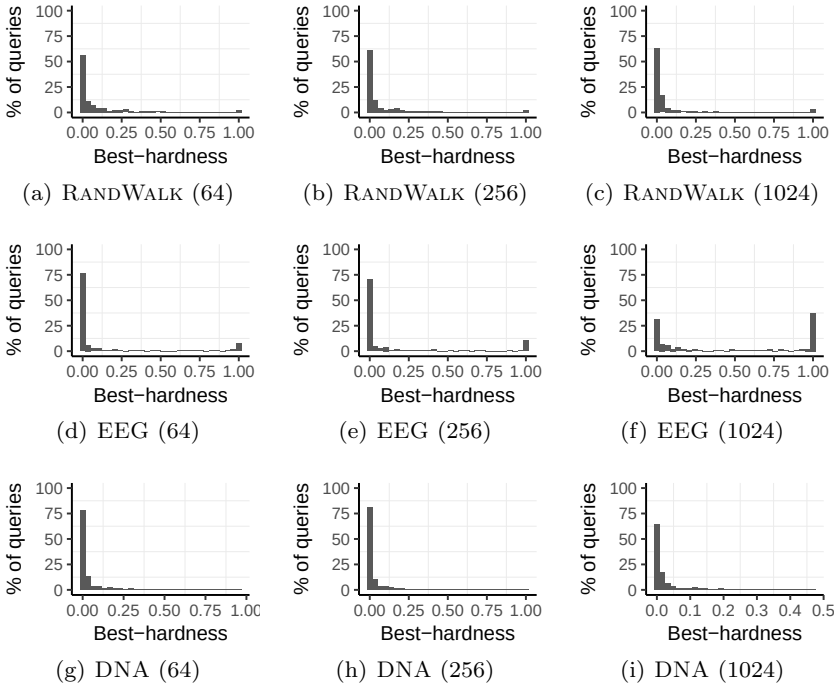
allows us to accurately capture all *ME*s. However, this is not a summarization-independent hardness definition. Looking at the summarization independent Max-Hardness in Figure 7, we can make two observations. First, we observe that it peaks at $\boldsymbol{q}_3$; however if we look at the *ME* lines across all summarizations, the hardest query is consistently $\boldsymbol{q}_2$. Therefore, Max-Hardness fails to identify the hardest query.

Second, it is also clear that the ranking of queries imposed by Max-Hardness completely fails to capture the *ME* ranking imposed by different summarizations. For example, as we can see in Figure 7, according to the *ME* of the first three summarizations (i.e., PAA32, FFT32, DHWT32), the queries are ordered from easiest to hardest as follows: $\boldsymbol{q}_6$, $\boldsymbol{q}_4$, $\boldsymbol{q}_5$, $\boldsymbol{q}_3$, $\boldsymbol{q}_1$, $\boldsymbol{q}_2$. This ordering is different for SAX32, though: $\boldsymbol{q}_3$, $\boldsymbol{q}_6$, $\boldsymbol{q}_1$, $\boldsymbol{q}_5$, $\boldsymbol{q}_4$, $\boldsymbol{q}_2$. Moreover, if we follow the Max-Hardness measure, then we get yet another ordering (common for all summarizations): $\boldsymbol{q}_4$, $\boldsymbol{q}_1$, $\boldsymbol{q}_5$, $\boldsymbol{q}_6$, $\boldsymbol{q}_2$, $\boldsymbol{q}_3$. As a result, apart from violating requirements (R1.a) and (R1.b) due to the inconsistencies across summarization *ME* rankings, requirement (R1.b) is additionally violated due to the *ME* ranking inconsistency to that of the one imposed by Max-Hardness.

4.2.2 Inter-Index Effort Accuracy

In our second experiment we turn our attention to requirement (R2). This is about the ratio of *ME*s of a single query across different summarizations, and how well it captures the relative summarization errors. If we observe queries $\boldsymbol{q}_3$ and $\boldsymbol{q}_4$ in Figure 7, we can see that query hardnesses can change for different summarizations: e.g., $\boldsymbol{q}_4$ becomes harder than $\boldsymbol{q}_3$ only for SAX32.

Then, a question that arises is the following: if the *ME* for a given query q varies across summarizations (i.e., it increases/decreases), how large should this variation be? This gets us back to requirement (R2). Our



(a) RandWalk (64) (b) RandWalk (256) (c) RandWalk (1024)

(d) EEG (64) (e) EEG (256) (f) EEG (1024)

(g) DNA (64) (h) DNA (256) (i) DNA (1024)

Fig. 9 Best-Hardness distributions (1,500 queries, 100,000 data series).

requirement for a meaningful workload was that the ratio of the *ME* of a single query between two different summarizations should increase in the same rate that the summarization error increases.

For the dataset used to create Figure 7, the *TLB* of the different summarizations were as follows. PAA32: 0.86, FFT32: 0.87, DHWT32: 0.86, SAX32: 0.96. Even though SAX32 has the lowest error (i.e., highest *TLB*), $\boldsymbol{q}_4$ is relatively harder for SAX32 than for the rest of the summarizations. We argue that this is merely an artifact of the distribution of the data, and does not capture the true pruning power of the summarization at hand.

To substantiate this claim, for each dataset and summarization we computed the theoretical Best-Hardness, *ME*, and the Summarization Error (defined as 1-*TLB*). We then normalized the value of every measurement for each query based on the corresponding value for PAA8, resulting in a scale of $[0-1]$ (PAA8 and DHWT8 are the worst-performing summarizations in our experiments). We plot the average ratios in Figure 8. While normalized Best-Effort is closer to the normalized *ME* for most of the cases, both these values are very different from the normalized Summarization Error. As a result, the relative hardnesses of queries across different summarizations are not consistent to their relative quality. This fact breaks also our last requirement (R2).

4.3 Hardness of Random Workloads

Remember that one of our goals is to generate queries of varying hardness. In order to characterize our workloads, we computed the Best-Hardness of 1,500 queries, across all our datasets of size 100,000 series each, and we report their distributions in Figure 9. We can see that all datasets are highly skewed towards easy queries. The only exception is EEG of length 1024 (Figure 9(f)), which includes a lot of hard queries. Even in this case though, the queries in the workload are either very easy, or very hard; there are no queries with hardness values in between.

We repeated the same experiments with the queries that were drawn from the indexed dataset with the addition of a small amount of gaussian noise. In this case, all the results show a negligible hardness (always very close to zero), and we omit the corresponding graphs for brevity. The reason for this behavior is that these queries have their nearest neighbor so close that almost no other points are close enough to be contained in their ϵ-areas.

In conclusion, the workloads commonly used in the literature not only violate our requirements for a fair comparison across methods, but are also heavily skewed towards easy queries.

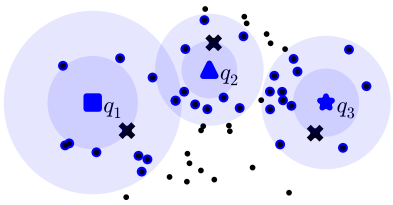


Fig. 10 Example of 3 queries, where the ϵ-area of $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ intersect. As a result we cannot control the hardness of these two queries independently, as densifying each one of the two zones might affect also the hardness of the other query.

5 Query workload generation

As we demonstrated in the previous section, all the widely-used (i.e., randomly generated) query workloads are both structurally unfit and biased towards easy queries. In this paper, apart from our structural conditions for a meaningful hardness definition, we also argue that an effective query workload should contain queries of varying hardness. Since most existing queries are easy, we start with these queries and make them harder by adding more points in their ϵ-areas, i.e., through the process of *densification*.

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 easy queries (produced through random generation).

We split our workload generation process in three stages:

- First, we select a subset $\mathcal{Q}'$ of $\mathcal{Q}$, comprising queries whose ϵ-areas do not intersect. This ensures that the densification process can be applied individually to each of the selected queries, without side-effects on the rest of the queries in $\mathcal{Q}'$. Observe that when the ϵ-areas of different queries intersect (for an example, refer to Figure 10 and the ϵ-areas of queries $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$), controlling the hardness of each query becomes difficult, since changes in the hardness of one query may also affect the intersecting queries.
- Second, we match (a subset of n chosen) queries in $\mathcal{Q}'$ with the provided hardness values, and identify the amount of points we need to add in each ϵ-area.
- Finally, we distribute these points in such a way that, as the *TLB* of the index gets worse, the minimum effort captured by the workload increases (following the requirements discussed in Section 3 and in Example 1).

The following subsections describe in detail each one of the three stages listed above.

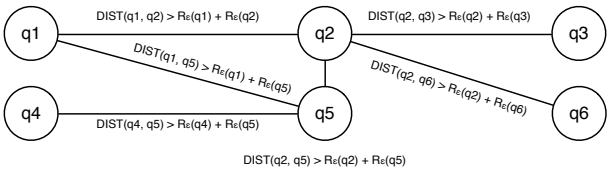


Fig. 11 Maximal clique formed by $\boldsymbol{q}_1$, $\boldsymbol{q}_2$, and $\boldsymbol{q}_5$.

5.1 Generating Non-intersecting Queries

The first stage in our query workload generation process is that of selecting a subset $\mathcal{Q}'$ of the initial set of queries $\mathcal{Q}$, comprising the queries whose pairwise ϵ-areas do not intersect. Moreover, we wish that $\mathcal{Q}'$'s cardinality is maximized, so as to accommodate a wide range of possible hardness values. In the following, we describe two approaches for achieving our goal.

5.1.1 An Initial Approach: Preserving Original Nearest Neighbors

In our first approach for query selection (also presented in our preliminary work [46]), we reason on ϵ-areas built around existing nearest neighbors. Therefore, query selection in this case does not imply any alteration of the original nearest neighbor.

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

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

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 $MINDIST(\boldsymbol{q})$.

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 11 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

Algorithm 1 FindNonIntersectingQueries

```
1: R_ε ← createRadius(Q, D, ε)
2: g ← createVerticesFromQueries(Q)
3: for (q_i, q_j) ∈ Q × Q do
4:     if DIST(q_i, q_j) > R_ε(q_i) + R_ε(q_j) then
5:         g.addEdge(q_i, q_j)
6: V ← g.getSortedVertices() {Sorted by ascending degree}
7: Q' ← ∅
8: for q ∈ V do
9:     if isCompatible(q, Q') then
10:        Q' ← Q' ∪ {q}
11: return Q'
```

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 starts by checking every pair of queries (Line 3) and compares the distance between them to the sum of the radiuses of their ϵ-areas. It then checks if this distance is bigger than the sum (Line 4), in which case it marks the two queries as compatible by connecting them with an edge (Line 5). The algorithm then sorts the vertices of the graph based on their degree (Line 6). 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 (Lines 8-10).

5.1.2 Improving Query Selection by Synthetic Nearest Neighbor Generation

Depending on the specific data and query set, as well as on the quantity of hardness values provided, the strategy presented in the previous subsection may produce an insufficient number of selected queries. Therefore, we now describe an approach, which consists in *synthetically generating nearest neighbors* for some (possibly all) the queries in the initial set $\mathcal{Q}$. The main advantage of this technique is that all provided queries can be part of the final workload (i.e., at the end of the process, we have $\mathcal{Q}' = \mathcal{Q}$).

We start by computing the minimum pairwise distances for all queries in $\mathcal{Q}$. For a given query $\boldsymbol{q}$, we denote this distance by $MINDISTQ(\boldsymbol{q})$. We then set:

$$R_\epsilon(\boldsymbol{q}) = \frac{MINDISTQ(\boldsymbol{q})}{2} - \omega,$$

where ω is a very small quantity necessary to avoid ϵ-areas tangency.

The above defines a valid radius for placing ϵ-spheres around each query in Q. Based on this computed radius, we define for each of the queries in Q a synthetic nearest-neighbor distance:

$$MINDIST_{syn}(\boldsymbol{q}) = \frac{R_\epsilon(\boldsymbol{q})}{1+\epsilon}.$$

Finally, for each query $\boldsymbol{q}$, we either keep its existing (i.e., in the original dataset) nearest neighbor, or generate a new, synthetic nearest neighbor as follows:

- if $MINDIST(\boldsymbol{q}) \leq MINDIST_{syn}(\boldsymbol{q})$ then the nearest neighbor is not changed;
- else, we pick a new point, p_q, uniformly at random among all points that satisfy the constraint $DIST(\boldsymbol{q}, p) = MINDIST_{syn}(\boldsymbol{q})$, and we let $\mathcal{D} = \mathcal{D} \cup p_q$.

5.2 Hardness Assignment and Number of Points to Add

Given the selected queries (as described above) and the desired hardness values $[\alpha_1^\epsilon, ..., \alpha_n^\epsilon]$, we proceed by (i) assigning queries to these hardness values, and (ii) determining the number of points to be added, so that the corresponding queries attain the desired hardness.

Hardness assignment is in part arbitrary. That is, any hardness value α_i^ϵ can be matched with any query $\boldsymbol{q}$, as long as the current ϵ-hardness value (for the given ϵ) of $\boldsymbol{q}$ is lower, or equal to α_i^ϵ. Note that it is possible to impose additional constraints on the queries (leading to a *ranking* function over the queries), in order to improve the overall workload quality (this is discussed in detail in Section 3).

Once each hardness value has been paired with a query, we need to identify the number of points to add 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,

$$\alpha_1^\epsilon = \frac{N_1 + x_1}{N + \sum_{i=1}^n x_i},\ ...,\ \alpha_n^\epsilon = \frac{N_n + x_n}{N + \sum_{i=1}^n x_i}. \tag{8}$$

Representing this linear system in matrix form, we have

$$(A - I)\boldsymbol{x} = \boldsymbol{b}, \tag{9}$$

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 indicate how many points to add in order to densify the ϵ-area for each selected query.

5.3 Densification Process

As we mentioned in Section 3, meaningful queries for effective index comparison should satisfy Equations (6) and (7). Given that for each summarization we can infer the ϵ areas that best characterize it using Equation (5), then we can assign an ϵ to each summarization.

Given *any two summarizations* S_1 and S_2, and a single query $\boldsymbol{q}_1$ it should hold that:

$$\frac{\mu^{S_1}(\boldsymbol{q}_1)}{\mu^{S_2}(\boldsymbol{q}_1)} = \frac{1 - TLB(S_1)}{1 - TLB(S_2)} = \frac{\alpha^{\epsilon_{S_1}}(\boldsymbol{q}_1)}{\alpha^{\epsilon_{S_2}}(\boldsymbol{q}_1)}. \quad (10)$$

Since this should hold for any summarization (and as a result, for every TLB and every ϵ), it automatically makes both Equations (6) and (7) hold. In this section, we describe how to densify the ϵ-areas for the selected queries in such a way that the above condition holds.

We call our solution *equi-densification*. The key idea is that if we add points to the right locations in the data space, then the desired properties will hold. Our method works for non z-normalized, as well as for z-normalized data. For z-normalized data, note that we cannot directly add random points in the desired ϵ-area, because after z-normalization these points could lie outside of the ϵ-area. Instead, when dealing with non z-normalized data, we avoid this problem. Given that most applications require the data to be z-normalized [17, 29], we present in the following our approach for the z-normalized case. The corresponding solution for the non z-normalized case is a simpler version of the approach described below.

To solve the densification problem we propose two different methods of EquiDensification.

- EquiDensification-LC: a baseline method that utilizes linear combinations of existing (z-normalized) points within the desired location, with points that lie outside the desired location (with the addition of noise).
- EquiDensification-GS: a method that uses the Gramm-Schmidt procedure for generating random, already normalized points at the desired locations.

Additionally, in order to demonstrate why Equations (6) and (7) should hold, we also consider two baseline candidate strategies for densification:

- BaselineRandom: randomly choosing points in $\mathcal{N}^{\epsilon}(\boldsymbol{q}(\alpha_i^{\epsilon}))$, and creating new points by adding noise to those;
- Baseline1NN: adding noise to the query's nearest neighbor (ignoring all other points in its ϵ-area).

In order to demonstrate the different densification strategies, we generate queries of hardness 0.2 ($\epsilon = 1.0$) for a dataset of 100,000 data series. According to Section 4, this ϵ allows us to test all representations in our set. In order to evaluate the effort for every query, we use four standard data series summarization techniques (namely, 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.

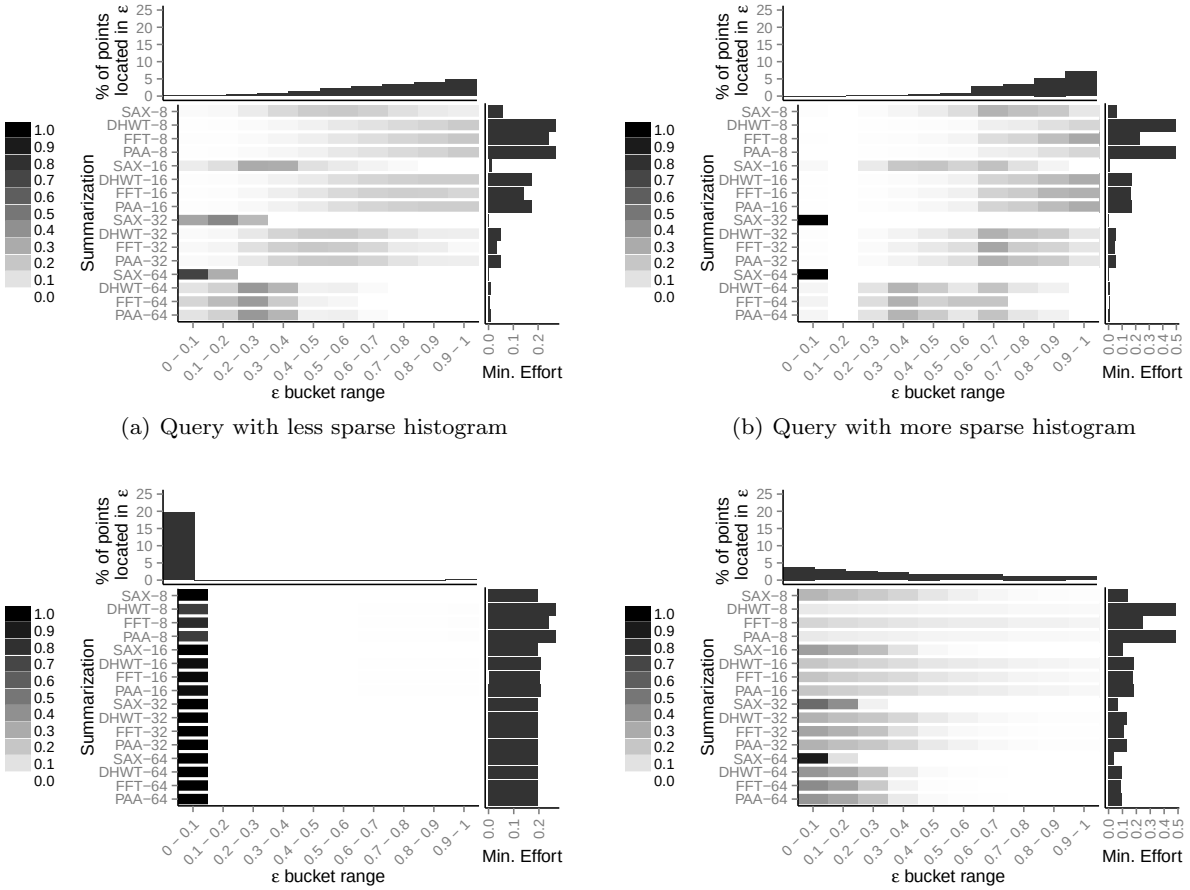
5.3.1 Baseline methods

We now briefly describe two baseline densification methods. As we will show later, these methods fall short of solving our problem, because they do not satisfy the criteria we have set (i.e., Equations (6) and (7)). Our solutions for equi-densification follow in the next subsection.

BaselineRandom. 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 12(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., $L(\boldsymbol{x}, \boldsymbol{q}) \leq MINDIST(\boldsymbol{q})$. 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 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



(a) Query with less sparse histogram

(b) Query with more sparse histogram

(c) 1NN densified query

(d) Equi-densified query

Fig. 12 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.

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 12(b), where the distribution of points in the ϵ-area is even more skewed.

Baseline1NN. 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 12(c).

5.4 EquiDensification

As discussed in Section 3.3.2, 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 the $ATLB$ values, and accordingly the ϵ values are bucketized (in a non-uniform way), as well. Note that all $ATLB$ buckets should have exactly the same number of points. We en-

force this by placing points at the desired locations using one of our proposed methods: EquiDensification-LC, or EquiDensification-GS. These algorithms are described in the next subsections.

A query produced with equi-densification is depicted in Figure 12(d). The histogram on the top shows that the first few buckets have many points, while the last few buckets have few. 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 lie within $[0.67, 1.0]$, and for *ATLB* values in $[0.6, 0.7]$ the corresponding ϵ values lie within $[0.43, 0.67]$ As we can see in the heat map, the effort points for this equi-densified query are more evenly distributed in the ϵ-areas than in the other cases (Figures 12(a)-(c)). It should be noted that the bucket limits in this plot are equi-width, while as we have seen, ϵ buckets during equi-densification are not of the same width. For this reason the bucket sizes appear to be decreasing for smaller ϵ values. 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.

5.4.1 EquiDensification-LC

In EquiDensification-LC, densification (i.e., the action of introducing points in the desired locations) is performed by creating linear combinations of points located within and outside of each *ATLB* 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.

We formally describe the process in Algorithm 2. The algorithm starts by randomly picking a data series that is within the ϵ-area of the query (Line 7) and one that is outside of it (Line 8). The loop in Lines 12-17 creates a linear combination between the inner point and the outer point (Line 14), increasing the ratio of the outer point (Line 15) until we reach the desirable distance (Line 17). If we do not succeed in placing the new point in the right location, we return an error (Line 19) and continue the search using another query.

Algorithm 2 EquiDensification-LC

```
1: Input: 𝒟: set of all data points
2: Input: Q: the query data point
3: Input: 𝒩: set of data points in ϵ-area of 𝒬.
4: Input: minB: The minimum acceptable distance
5: Input: maxB: The maximum acceptable distance
6: Input: noiseStep: The step we increase noise intensity

7: inPoint ← getRandomPoint(𝒩)
8: outPoint ← getRandomPoint(𝒟 − 𝒩)
9: newPoint ← inPoint
10: l ← noiseStep
11: found ← false
12: while found = false and l < 1 do
13:    noiseRatio ← noiseStep
14:    newPoint ← inPoint ∗ (1 − l) + outPoint ∗ l
15:    l ← l + noiseStep
16:    dist ← DIST(Z − Normalize(newPoint), Q)
17:    found ← (dist < maxB and dist > minB)
18: if found = false then
19:    return Not Found
20: else
21:    return newPoint
```

This algorithm produces a correct result, but has a high complexity since we need to exhaustively test various factors for the linear combination ranging from 0 to 1, in order to achieve the desired output.

5.4.2 EquiDensification-GS

Generating normalized points (optionally, with additional distance constraints) is an important primitive of our workload generation procedure, and its efficiency significantly affects the workload generation algorithm's complexity. We now describe the details of this method.

Generating unconstrained normalized points. A z-normalized N-dimensional point $\boldsymbol{x} = [x_1, \ldots, x_N]^T$ has mean 0 and standard deviation 1, that is, it respects the equations 11 and 12 below:

$$\sum_{i=1}^{N} x_i = 0, \tag{11}$$

$$\sum_{i=1}^{N} x_i^2 = N. \tag{12}$$

Note that these define the intersection of an N-hypersphere (of radius $\sqrt{N}$) with a hyperplane, thus we naturally expect these points to be found on an (N-1)-hypersphere. Our purpose, however, is an operational one, namely a procedure for synthetically generating such points. We start with an initial basis $\boldsymbol{V} = (\boldsymbol{V}_1, \ldots, \boldsymbol{V}_n)$, where:

$$\boldsymbol{V}_1 = [\frac{1}{\sqrt{N}}, \ldots, \frac{1}{\sqrt{N}}]^T$$

$$\boldsymbol{V}_2 \ = \ [0,1,\ldots,0]^T, \ldots, \boldsymbol{V}_N \ = \ [0,0,\ldots,1]^T$$

This basis of free vectors is specifically chosen to simplify the hyper-sphere expression process. We then proceed to transform this basis into an orthonormal one, required for generating synthetic points. To do this we use the Gram-Schmidt procedure, considering the orthonormal basis $\boldsymbol{U} = (\boldsymbol{U}_1, \ldots, \boldsymbol{U}_n)$ where:

$$\boldsymbol{U}_1 \ = \ \boldsymbol{V}_1 \ = \ [\frac{1}{\sqrt{N}}, \ldots, \frac{1}{\sqrt{N}}]^T, \tag{13}$$

$$U_i = \frac{W_i}{||W_i||}, W_i \ = \ V_i - \sum_{j=1}^{i-1} < V_i, U_j > U_j. \tag{14}$$

Let $b_1, \ldots, b_n$ be $\boldsymbol{x}$'s coordinates in basis $\boldsymbol{U}$. Because of equations (13) and (11) it holds that:

$$< \boldsymbol{x}, \boldsymbol{U}_1 > \overset{(13)}{=} \sum_{i=1}^{N} x_i \frac{1}{\sqrt{N}} \overset{(11)}{=} \frac{\sum_{i=1}^{N} x_i}{\sqrt{N}} = 0.$$

With $\boldsymbol{U}$ being an orthonormal basis, it also holds that $< \boldsymbol{x}, \boldsymbol{U}_1 >= b_1$. As a result, it follows that:

$$< \boldsymbol{x}, \boldsymbol{U}_1 >= b_1 = 0. \tag{15}$$

Moreover, the dot product of a vector with itself can be written as:

$$< \boldsymbol{x}, \boldsymbol{x} >= \sum_{i=1}^{N} x_i^2 \overset{(12)}{=} N.$$

Since $\boldsymbol{b}$ is just a transformation of $\boldsymbol{x}$ in basis $\boldsymbol{U}$, it follows that:

$$\sum_{i=1}^{N} b_i^2 = N. \tag{16}$$

Putting together equations (15) and (16) we end up with the (N-1) hypersphere equation expected, namely:

$$\sum_{i=2}^{N} b_i^2 = N. \tag{17}$$

Generating the required $\boldsymbol{x}$ is then a two-fold process:

– First, we aim at producing uniform random points on the (N-1) hypersphere, with radius $\sqrt{N}$ (from equation (17)). To achieve this, we use the standard procedure for sphere point picking that consists of generating (N-1) Gaussian random variables $b_2^G, \ldots, b_N^G$ and producing (partial) $\boldsymbol{b}$ vectors as:

$$[b_2, \ldots, b_N]^T = \frac{[b_2^G, \ldots, b_N^G]^T}{\sqrt{\sum_{i=2}^{N} (b_i^G)^2}} \sqrt{N}. \tag{18}$$

– Then, using the $\boldsymbol{U}$ basis coordinates of $\boldsymbol{x}$ obtained above and the $\boldsymbol{U}$ basis previously computed (equations (13) and (14)), we recover the canonical coordinates of $\boldsymbol{x}$ as:

$$x_i = \sum_{j=1}^{N} b_j * U_{ji}. \tag{19}$$

Based on the elements above, we thus obtain a procedure for generating N-dimensional z-normalized points. Indeed, to generate any number P of such points, we first construct the basis $\boldsymbol{U}$ as described above, then for each of the P points to be generated we produce random b vectors and plug these into equation (19).

While this procedure is reliable and efficient, it however only enforces z-normalization. Workload generation often requires an additional property, namely a "placement" constraint (for instance, a specific placement of a data point w.r.t. a query point).

Generating normalized points constrained by distance. We can further extend the above reasoning to obtain a refined primitive useful for workload generation, namely the ability of generating normalized points at a given distance D from an existing *normalized* point $p = [p_1, \ldots, p_N]^T$. Indeed, this implies the required points x further verifying the following equation:

$$\sum_{i=1}^{N} x_i^2 + \sum_{i=1}^{N} p_i^2 - 2 * \sum_{i=1}^{N} x_i * p_i = D^2, \tag{20}$$

where both x and p respect 11 and 12.

It follows that x must further respect the following:

$$\sum_{i=1}^{N} x_i * p_i = N - D^2/2. \tag{21}$$

Note that the above is another hyperplane equation. Coupled with the restrictions on x given by (11) and (12). It will thus lead us to an $(N-2)$-hypersphere definition for the required x points. Indeed, proceeding similarly as above, we will construct the orthonormal basis comprising:

$$\boldsymbol{U}_1 \ = \ [\frac{1}{\sqrt{N}}, \ldots, \frac{1}{\sqrt{N}}]^T, \tag{22}$$

and

$$\boldsymbol{U}_2 \ = \ [\frac{p_1}{\sqrt{N}}, \ldots, \frac{p_N}{\sqrt{N}}]^T. \tag{23}$$

Note that $< \boldsymbol{U}_1, \boldsymbol{U}_2 >= 0$ and that both U_1 and U_2 are unit vectors, because of p respecting 11 and 12. We generate the rest of the basis $\boldsymbol{U}$ as above, by employing the Gramm-Schmidt procedure. As above, let $b_1, \ldots, b_N$ be $\boldsymbol{x}$'s coordinates in the basis $\boldsymbol{U}$. We then have:

$$b_1 =< \boldsymbol{x}, \boldsymbol{U}_1 >= \sum_{i=1}^{N} x_i \frac{1}{\sqrt{N}} = \frac{\sum_{i=1}^{N} x_i}{\sqrt{N}} \overset{(11)}{=} 0 \tag{24}$$

$$b_2 =< \boldsymbol{x}, \boldsymbol{U}_2 >= \sum_{i=1}^{N} x_i \frac{p_i}{\sqrt{N}} = \frac{\sum_{i=1}^{N} x_i p_i}{\sqrt{N}} \tag{25}$$

$$\overset{(21)}{=} \frac{N - D^2/2}{\sqrt{N}}. \tag{26}$$

Using (12), i.e., $< \boldsymbol{x}, \boldsymbol{x} >= N$, we obtain the following:

$$\sum_{i=1}^{N} b_i^2 = N \tag{27}$$

or equivalently:

$$\sum_{i=3}^{N} b_i^2 = N - b_1^2 - b_2^2 \tag{28}$$

which, combined with equations (24) and (25), gives us the following equation for the remaining b coordinates:

$$\sum_{i=3}^{N} b_i^2 = D^2 * (1 - \frac{D^2}{4 * N}). \tag{29}$$

Note that this is indeed the equation of an $(N-2)$-hypersphere, as expected. To generate $\boldsymbol{x}$ points, we then sample $b_3^G, \ldots, b_N^G$ Gaussian random variables and produce (partial) $\boldsymbol{b}$ vectors as:

$$[b_3, \ldots, b_N]^T = \frac{[b_3^G, \ldots, b_N^G]^T}{\sqrt{\sum_{i=3}^{N} (b_i^G)^2}} \sqrt{\frac{N - D^2/2}{\sqrt{N}}}. \tag{30}$$

We can then recover $\boldsymbol{x}$'s canonical coordinates using the orthonormal basis $\boldsymbol{U}$ and equation (19), as previously.

We thus obtain an efficient and reliable procedure for generating N-dimensional z-normalized points placed at a given distance from a given z-normalized point (typically a query point). Indeed, to generate any number P of such points, we first construct the basis $\boldsymbol{U}$ as described above, then for each of the P points to be generated we produce random b vectors and plug these into equation (19).

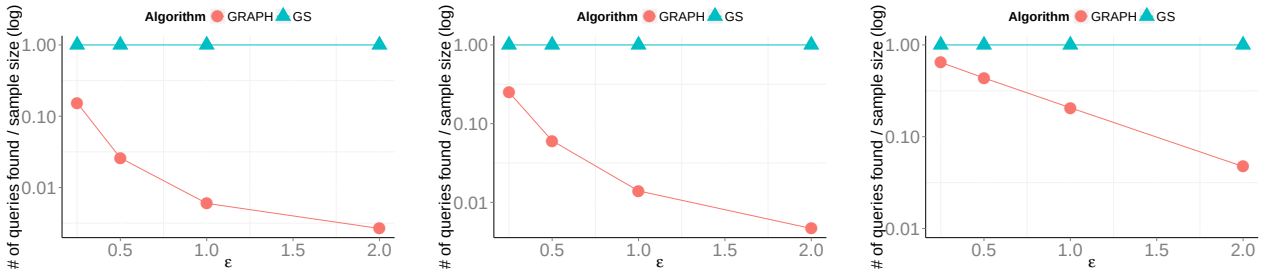
6 Experiments

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.

1. The first investigates the amount of non-interfering queries we can find for each dataset both with our baseline method, presented in our preliminary work [46], as well as with our new synthetic nearest neighbor placement method based on Gramm-Schmidt.
2. The second set of experiments is intended to compare EquiDensification to the two baseline densification methods with regards to the minimum effort of various common summarization techniques. We use PAA, FFT, DHWT and SAX. For each one of the summarizations, we used 8, 16, 32 and 64 bytes to represent each data series.
3. In the third set of experiments, we used two real world indexes, *i*SAX 2.0 [7] and the R-Tree [18] 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 Section 5.1, we presented two techniques for finding non-intersecting queries. Our graph based method maintains the original nearest neighbors, while our new method allows us to generate new nearest neighbors for each one of the queries in our sample size. These nearest neighbors are placed in the appropriate distance, such that they create non-intersecting areas. Moreover, they are generated in such a way that they are already normalized. In Figure 13, we can see that the ratio of non interfering queries found in a sample of 1500 queries and dataset sizes of 100,000 data series. We include the ratios for both the old graph based method (named as GRAPH in the plot), and the new Gram-Schmidt based method (named as GS in the plot). As we can see, using



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

Fig. 13 Number of non intersecting queries found using both methods as ϵ increases.

the old method, even in the case of very small ϵ (hardly useful in practice), we are able to use just above 10% of the queries from the sample for RandWalk and DNA. For the case of larger ϵ values, we can use less than 1% of the sample size for these two datasets. Using the EEG dataset, nevertheless, we can use a larger number of queries but only for a very small ϵ. As a result, for all the datasets, when an $\epsilon > 1.0$ is used, we can hardly ever use more than 10% of the queries. On the contrary, for the case of our Gram-Schmidt based method, we are able to select 100% of the sample size as queries, in all cases and all datasets.

6.1.1 Time Complexity

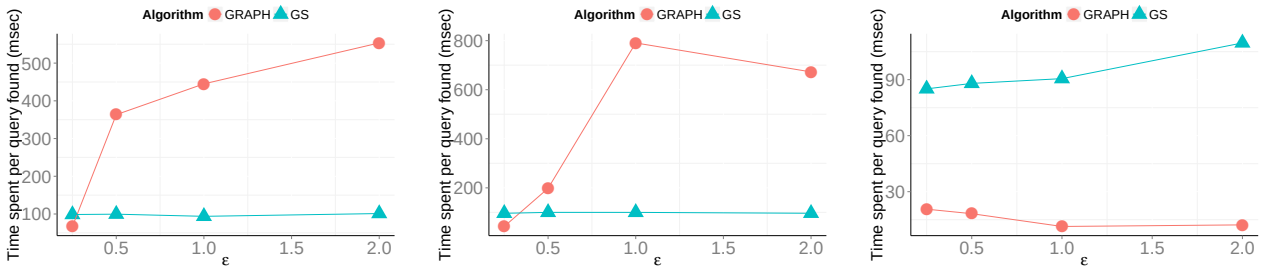
In regards to time complexity, the overall time spent in order to solve the Gram-Schmidt equations is much larger than the time spent for providing an approximate solution to the graph problem. On the contrary, the amount of queries found using Gram-Schmidt is much larger. If we normalize the time spent over the number of queries found, it becomes clear that the Gram-Schmidt based method is much more efficient. This means the time spent per query found is much less than what is required by the graph based approach. This can be seen in Figure 14, where we plot the time per query found for both methods and different ϵ values. It is clear that in most cases, Gram-Schmidt outperforms the graph based method. In the case of the EEG dataset, we are also able to generate many queries also using the graph based method, as a result the normalized time of the graph based method is less. However, both methods need less 100 milliseconds per query.

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 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 15 shows the average relative summarization errors for each dataset (averaged over the 100 different benchmarks generated). The results show that 1NN densification results in 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. As a result, our equi-densified workload satisfies requirement (R2.a).

In regards to (R1.a) and (R1.b), we repeated the experiment of Figure 5 in Section 3.3.2. This time with 10 equi-densified queries on an initial randomwalk dataset of 100,000 data series. We plot the Pearson correlations between efforts and all ϵ-hardnesses up to Max-Hardness in Figure 16 (our selected max ϵ was 1). It is clear that all ϵ-hardnesses for all methods are correlated to the effort and between-themselves for all summarizations. As a result, we satisfy both structural requirements for an ϵ-independent and accurate hardness definition.



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

Fig. 14 Time spent per query found by each method.

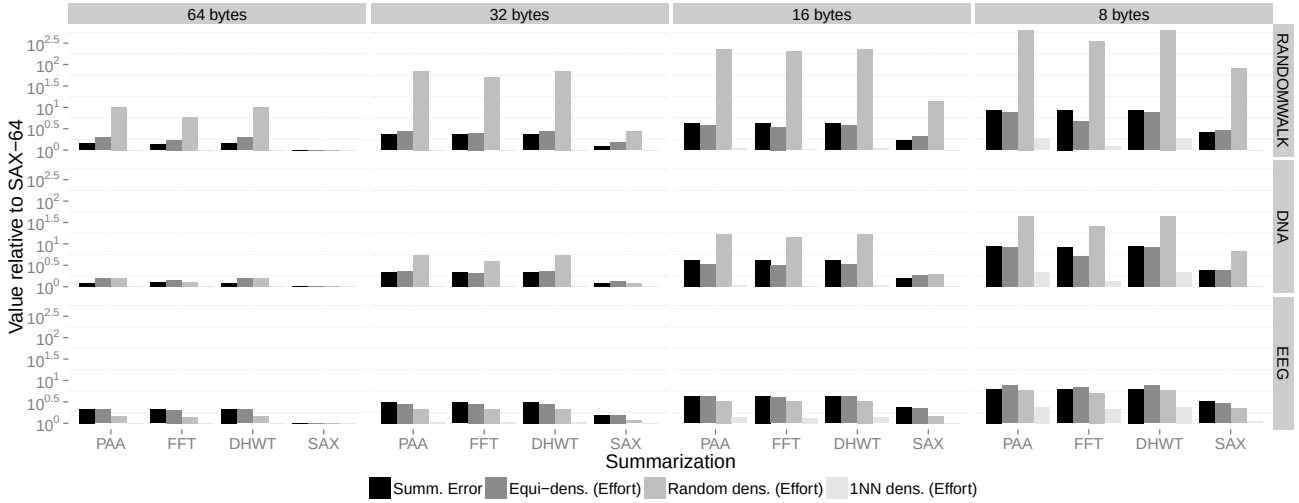
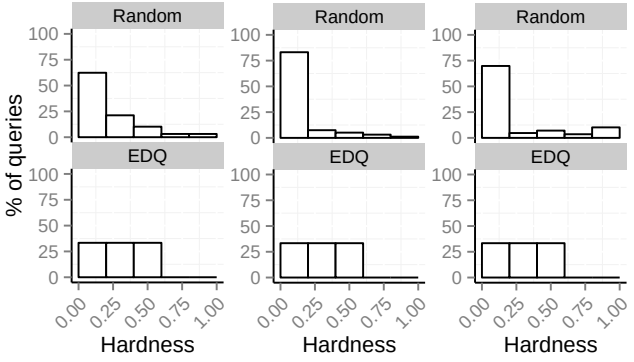


Fig. 15 Minimum efforts for different summarization techniques at different resolutions (8-64 bytes) representing 256 point data series, compared to the summarization error (1-TLB). All the values have been normalized against SAX-64 which was the overall tightest summarization method.

Fig. 16 Pearson correlation between effort and ϵ-hardness for an equi-densified workload.



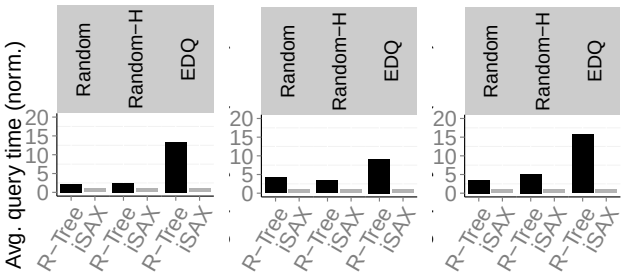
(a) RandWalk (b) DNA (c) EEG

Fig. 17 Histogram of hardnesses for $\epsilon = 1.0$.

On a side note, we observe that DHWT and PAA produce the same results in terms of performance (refer to Figures 15 and 16). The reason is that they both represent exactly the same information (they both partition the space in segments of equal size, and then DHWT represents them as a hierarchy of coefficients, while PAA represents them as a list of average values.)

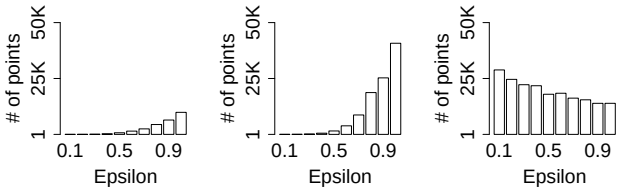
6.3 Case Study on Actual Indexes

Our last experiments goal is to demonstrate the qualitative difference of using our query workload versus



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

Fig. 18 Average query answering time comparison between *i*SAX (256 characters, 16 segments) and R-Tree (PAA with 8 segments) normalized over *i*SAX.



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

Fig. 19 Distribution of points in $\epsilon = 1.0$ area for 3 types of queries for RandWalk.

a workload of randomly generated queries. In order to have a sizable sample, we have aimed here at generating over 500 queries; more precisely, we have created 85 input datasets with 6 queries each, leading to a total of 510 queries. Among the 6 queries per dataset, 3 are equi-densified and referred hereafter as EDQ, whereas the remaining 3 are 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 ($\epsilon = 1.0$). Our goal for this study is to demonstrate the qualitative difference of using our query workload versus a workload of randomly generated queries.

Figure 17 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 queries with hardness larger than 0.5, and EDQ generated by our method. We indexed all three datasets with both *i*SAX [39] and R-Trees [18] with PAA [23], and measured the average query answering time per workload. Figure 18 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 19, 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 and Future Work

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.

Our long term goal is that of developing a generic framework for comparing and evaluating different data series index structures. A crucial part of this framework,

which we try to address in this paper, is the ability to generate query workloads. An important next step is to compare the various data series index structures using the workloads proposed in this work. Such an analysis will offer insights on the performance characteristics of the different approaches, and highlight the areas that merit future research efforts. In our future work, we also plan to extend our ideas to data series with multiple dimensions. Finally, the ability to scale our workload generator to raw datasets of multi-million data series is a very interesting engineering topic that would require the integration of indexes within the generator itself. Such indexes could either operate directly on the raw dataset, or draw inspiration from approaches that index queries [31, 11].

References

1. Agrawal, R., Faloutsos, C., Swami, A.: Efficient similarity search in sequence databases. In: FODO (1993)
2. Assent, I., Krieger, R., Afschari, F., Seidl, T.: The ts-tree: Efficient time series search and retrieval. In: EDBT (2008)
3. Bagnall, A., Lines, J., Bostrom, A., Large, J., Keogh, E.J.: The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Min. Knowl. Discov. **31**(3), 606–660 (2017). DOI 10.1007/s10618-016-0483-9. URL `https://doi.org/10.1007/s10618-016-0483-9`
4. Bay, S.D., Kibler, D., Pazzani, M.J., Smyth, P.: The uci kdd archive of large data sets for data mining research and experimentation. In: SIGKDD Explorations (2000)
5. Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is "nearest neighbor" meaningful? In: ICDT (1999)
6. Camerra, A., Palpanas, T., Shieh, J., Keogh, E.: iSAX 2.0: Indexing and mining one billion time series. In: ICDM (2010)
7. Camerra, A., Shieh, J., Palpanas, T., Rakthanmanon, T., Keogh, E.: Beyond one billion time series: indexing and mining very large time series collections with isax2+. KAIS (2013)
8. Chakrabarti, K., Keogh, E., Mehrotra, S., Pazzani, M.: Locally adaptive dimensionality reduction for indexing large time series databases. In: SIGMOD (2002)
9. Chan, K.P., Fu, A.C.: Efficient time series matching by wavelets. In: ICDE (1999)
10. Chen, Q., Chen, L., Lian, X., Liu, Y., Yu, J.X.: Indexable pla for efficient similarity search. In: VLDB (2007)
11. Chow, C., Mokbel, M.F., Bao, J., Liu, X.: Query-aware location anonymization for road networks. GeoInformatica **15**(3), 571–607 (2011). DOI 10.1007/s10707-010-0117-0. URL `https://doi.org/10.1007/s10707-010-0117-0`
12. Dallachiesa, M., Nushi, B., Mirylenka, K., Palpanas, T.: Uncertain time-series similarity: Return to the basics. In: VLDB (2012)
13. Dallachiesa, M., Palpanas, T., Ilyas, I.F.: Top-k nearest neighbor search in uncertain data series. In: VLDB (2015)
14. Das, G., Gunopulos, D., Mannila, H.: Finding similar time series. In: Principles of Data Mining and Knowledge Discovery, First European Symposium, PKDD '97, Trondheim, Norway, June 24-27, 1997, Proceedings, pp. 88–100 (1997). DOI 10.1007/3-540-63223-9_109. URL `https://doi.org/10.1007/3-540-63223-9_109`
15. Faloutsos, C., Ranganathan, M., Manolopoulos, Y.: Fast subsequence matching in time-series databases. In: SIGMOD (1994)
16. Fu, A.W., Leung, O.T., Keogh, E.J., Lin, J.: Finding time series discords based on haar transform. In: Advanced Data Mining and Applications, Second International Conference, ADMA 2006, Xi'an, China, August 14-16, 2006, Proceedings, pp. 31–41 (2006). DOI 10.1007/11811305_3. URL `https://doi.org/10.1007/11811305_3`
17. Goldin, D.Q., Kanellakis, P.C.: On similarity queries for time-series data: Constraint specification and implementation. In: Principles and Practice of Constraint Programming (1995)
18. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: SIGMOD (1984)
19. Huijse, P., Estévez, P.A., Protopapas, P., Principe, J.C., Zegers, P.: Computational intelligence challenges and applications on large-scale astronomical time series databases. IEEE Comp. Int. Mag. **9**(3) (2014)
20. Kashino, K., Smith, G., Murase, H.: Time-series active search for quick retrieval of audio and video. In: ICASSP (1999)
21. Kashyap, S., Karras, P.: Scalable knn search on vertically stored time series. In: KDD (2011)
22. Keogh, E.: Machine learning in time series databases (and everything is a time series!). In: Tutorial at the AAAI Int. Conf. on Artificial Intelligence, vol. 2 (2011)
23. Keogh, E., Chakrabarti, K., Pazzani, M., Mehrotra, S.: Dimensionality reduction for fast similarity search in large time series databases. KAIS **3** (2000)
24. Keogh, E., Pazzani, M.: Scaling up dynamic time warping to massive datasets. In: PKDD (1999)
25. Korn, F., Jagadish, H.V., Faloutsos, C.: Efficiently supporting ad hoc queries in large datasets of time sequences. In: SIGMOD (1997)
26. Kremer, H., Günnemann, S., Ivanescu, A.M., Assent, I., Seidl, T.: Efficient processing of multiple dtw queries in time series databases. In: SSDBM (2011)
27. Li, C.S., Yu, P., Castelli, V.: Hierarchyscan: a hierarchical similarity search algorithm for databases of long sequences. In: ICDE (1996)
28. Lin, J., Keogh, E., Lonardi, S., Chiu, B.: A symbolic representation of time series, with implications for streaming algorithms. In: DMKD (2003)
29. Lin, J., Keogh, E.J., Wei, L., Lonardi, S.: Experiencing SAX: a novel symbolic representation of time series. Data Min. Knowl. Discov. **15**(2), 107–144 (2007)
30. Lin, J., Khade, R., Li, Y.: Rotation-invariant similarity in time series using bag-of-patterns representation. J. Intell. Inf. Syst. **39**(2) (2012)
31. Prabhakar, S., Xia, Y., Kalashnikov, D.V., Aref, W.G., Hambrusch, S.E.: Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. IEEE Trans. Computers **51**(10), 1124–1140 (2002). DOI 10.1109/TC.2002.1039840. URL `https://doi.org/10.1109/TC.2002.1039840`
32. Rafiei, D., Mendelzon, A.: Similarity-based queries for time series data. In: SIGMOD (1997)
33. Rafiei, D., Mendelzon, A.: Efficient retrieval of similar time sequences using dft. In: ICDE (1998)

34. Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., Keogh, E.: Searching and mining trillions of time series subsequences under dynamic time warping. In: KDD (2012)
35. Ratanamahatana, C.A., Lin, J., Gunopulos, D., Keogh, E.J., Vlachos, M., Das, G.: Mining time series data. In: Data Mining and Knowledge Discovery Handbook, 2nd ed., pp. 1049–1077 (2010). DOI 10.1007/978-0-387-09823-4_56. URL https://doi.org/10.1007/978-0-387-09823-4_56
36. Ravi Kanth, K.V., Agrawal, D., Singh, A.: Dimensionality reduction for similarity searching in dynamic databases. In: SIGMOD (1998)
37. Schäfer, P., Högqvist, M.: Sfa: A symbolic fourier approximation and index for similarity search in high dimensional datasets. In: EDBT (2012)
38. Shasha, D.: Tuning time series queries in finance: Case studies and recommendations. IEEE Data Eng. Bull. **22**(2) (1999)
39. Shieh, J., Keogh, E.: isax: Indexing and mining terabyte sized time series. In: KDD (2008)
40. Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., Keogh, E.: Experimental comparison of representation methods and distance measures for time series data. DMKD **26**(2) (2013)
41. Wang, Y., Wang, P., Pei, J., Wang, W., Huang, S.: A data-adaptive and dynamic segmentation index for whole matching on time series. In: VLDB (2013)
42. Ye, L., Keogh, E.J.: Time series shapelets: a new primitive for data mining. In: KDD (2009)
43. Yi, B.K., Jagadish, H., Faloutsos, C.: Efficient retrieval of similar time sequences under time warping. In: ICDE (1998)
44. Zoumpatianos, K., Idreos, S., Palpanas, T.: Indexing for interactive exploration of big data series. In: SIGMOD (2014)
45. Zoumpatianos, K., Idreos, S., Palpanas, T.: Rinse: Interactive data series exploration. In: VLDB (2015)
46. Zoumpatianos, K., Lou, Y., Palpanas, T., Gehrke, J.: Query workloads for data series indexes. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015, pp. 1603–1612 (2015)