DIDS: Double Indices and Double Summarizations for Fast Similarity Search

Han Hu
Harbin Institute of Technology
huhan@stu.hit.edu.cn

Jiye Qiu
Harbin Institute of Technology
qiujiye@stu.hit.edu.cn

Hongzhi Wang*
Harbin Institute of Technology
wangzh@hit.edu.cn

Bin Liang
Harbin Institute of Technology
liangbin@stu.hit.edu.cn

Songling Zou
Harbin Institute of Technology
zousongling@stu.hit.edu.cn

ABSTRACT

Data series has been one of the significant data forms in various applications. It becomes imperative to devise a data series index that supports both approximate and exact similarity searches for large data series collections in high-dimensional metric spaces. The state-of-the-art works employ summarizations and indices to reduce the accesses to the data series. However, we discover two significant flaws that severely limit performance enhancement. Firstly, the state-of-the-art works often employ segment-based summarizations, whose lower bound distances decrease significantly when representing a data series collection, resulting in numerous invalid accesses. Secondly, the disk-based indices for the exact search mainly rely on tree-based indices, which results in low-quality approximate answers, consequently impacting the exact search.

To address these problems, we propose a novel solution, Double Indices and Double Summarizations (DIDS). Besides segment-based summarizations, DIDS introduces reference-point-based summarizations to improve the pruning rate by the sorted-based representation strategy. Moreover, DIDS employs reference points and a cost model to cluster similar data series, and uses a graph-based approach to interconnect various regions, enhancing approximate search capabilities. We conduct experiments on extensive datasets, validating the superior search performance of DIDS.

PVLDB Reference Format:
Han Hu, Jiye Qiu, Hongzhi Wang, Bin Liang, and Songling Zou. DIDS: Double Indices and Double Summarizations for Fast Similarity Search. PVLDB, 17(9): 2198 - 2211, 2024.
doi:10.14778/3665844.3665851

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/imarcher/DIDS.

1 INTRODUCTION

Modern applications generate an abundance of data series. Investigating the efficient management and application of data series has been one of the foremost challenges [8, 28, 65, 81, 92]. Similarity search, as a pivotal operation in data analysis and mining, holds significance across various domains such as finance [76], engineering [71, 73], telecommunication [67], astronomy [38], machine learning [78] and so on [43, 62].

Firstly, we briefly review existing similarity search techniques. The studies can primarily be classified into four categories [57]: hash-based [32, 37, 50, 56, 80], tree-based [3, 17, 59, 84, 89], graph-based [21, 30, 31, 40, 60], and quantization-based approaches [22, 33, 41]. Graph-based approaches achieve the highest approximate search performance and precision with high memory cost [4].

While there are many studies on the Approximate Nearest Neighbor (ANN), the number of data series indices supporting both approximate and exact searches for large data series collections remains relatively scarce [24, 25]. In this domain, the EAPCA and iSAX families [23, 66] are the state-of-the-art works [89]. Due to their high pruning rates and training-free feature, they can significantly reduce the search time. Their approaches can be summarized as follows: constructing data series indices based on summarizations with lower bound distances, utilizing the indices to obtain approximate answers, with the answers, using the summarizations that represent a data series collection for initial pruning, employing the summarizations corresponding to one data series for further pruning, and ultimately accessing the remaining data series.

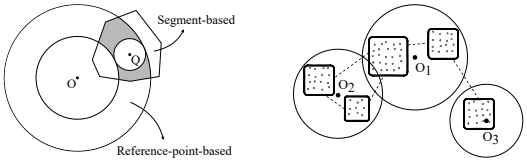
From previous works, we find two paramount flaws that severely limit the search capabilities of data series indices.

Firstly, for initial pruning, the EAPCA and iSAX families mainly employ the segment-based summarizations, where data series are segmented and each segment undergoes dimensionality reduction, resulting in low initial pruning rate. The segment-based summarizations like EAPCA [23], employ a boundary-based strategy to represent a collection of data series, by considering the common boundary of the collection for each dimension. The more data series they represent, the smaller the lower bound distances. Consequently, the initial pruning rate is insufficient.

Secondly, the quality of the approximate answers obtained from these tree-based disk indices is low. The tree-based indices partition data based on rules, and each partition step can separate numerous similar data series alongside the partition boundary, resulting in reduced recall rates. Additionally, the approximate answers often serve as inputs to the exact search. The low-quality approximate answers may result in a low pruning rate. Previous works [13, 89] use some strategies with random access to enhance the pruning rate, leading to performance degradation.

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 17, No. 9 ISSN 2150-8097.
doi:10.14778/3665844.3665851



(a) The pruning region **(b) The graph of DIDS**

Figure 1: (a) shows the pruning regions for different types of summarizations. DIDS further partitions the data for each reference point and constructs the graph at this level, as shown in (b).

In conclusion, previous works suffer from low pruning rates, resulting in excessive invalid accesses. To address these issues, we introduce a novel approach, DIDS. DIDS employs the reference-point-based summarizations for initial pruning. Their lower bound distances do not decrease when representing a data series collection, by a sorted-based representation strategy rather than a boundary-based one. Moreover, DIDS combines reference points with graphs, providing a data-based rather than rule-based approximate search algorithm, enhancing the quality of approximate answers.

For the first constraint, we use reference-point-based summarizations perform a swift initial pruning before employing segment-based summarizations. DIDS selects many reference points rather than few points in previous works, providing a higher pruning rate. However, for large datasets, an excess of points yield unacceptable construction and search cost. For construction, we devise a graph-based reference-point search algorithm, swiftly assigning a reference point to each data series. Then, we construct a B^+-tree for each point to cluster data series. For search, we sequentially access the B^+-trees in disk order, and rely on approximate answers (from the method to be discussed later) to locate the boundaries of the data series which cannot be pruned. Finally, we employ the costlier but higher pruning-rate SAX to process the rest data series. We use the SAX as in many previous works [23, 89], even though SAX can be replaced here by any other more updated summarizations [75, 87] with lower bound distances.

We improve the initial pruning rate to reduce the accesses to the summarizations, lowering the pruning cost. Moreover, we enhance the overall pruning rate. In Figure 1a, the circle centered at query Q represents the true answers. The region that cannot be pruned by the reference-point-based summarizations forms a ring centered around the reference point O. For segment-based summarizations, the region expands outward from Q. The overlap of these two summarizations may be a smaller area, improving the pruning rate.

For the second constraint, we introduce a novel approximate search algorithm based on reference points and graph. In Figure 1b, we cluster data series by reference points and employ binary trees to partition them within a cluster into leaf nodes based on a cost model. We represent leaf nodes with their centroids. Then, we insert all leaf nodes into a similarity search graph (we select HNSW [60]). Clearly, our memory is consistently maintained within a cluster level. As a result, DIDS employs the graph to locate the nodes nearest to the query, obtaining high-quality approximate answers.

Contributions. Our primary contributions are as follows:

- We propose an approach that combines reference-point-based summarizations and segment-based summarizations to reduce the pruning cost and enhance the pruning rate. We combine the approximate answers, B^+-trees, and SAX summarizations to provide an exceptionally efficient sequential pruning method.
- We integrate the reference points and the binary trees base on a cost model to partition the data series into numerous small regions. Then, we leverage these regions and the graph to obtain high-quality approximate answers rapidly.
- We conduct experiments with various datasets, offering insights for the future research. In comparison to the state-of-the-art works, DIDS boasts a around 60% higher recall rate in the approximate search under low execution times, and achieves a around 90% improvement in exact search speed on average.

2 PRELIMINARIES AND RELATED WORK

In this section, we introduce the preliminaries, discuss some related works and summarize their shortcomings. In § 2.1, we present several related definitions. In § 2.2, we show commonly used summarizations for similarity search. In § 2.3, we discuss reference-point-based methods. Then we introduce data series indices that support both approximate and exact similarity searches in § 2.4. Lastly, we provide a brief overview of graph-based methods in § 2.5.

2.1 Definitions

2.1.1 Data series. A data series $S = \{s_1, ..., s_d\}$, is an ordered point sequence that can be viewed as a d-dimensional vector.

2.1.2 Similarity search query. A query $Q = \{q_1, ..., q_d\}$, also is a d-dimensional data series, employed to identify a subset from a collection of data series.

2.1.3 Similarity search problem. The definition of the similarity search (k-NN) in metric spaces is as follows: given a data series collection $Arr_s = \{S_1, ..., S_n\}$, a query Q and a metric distance calculation formula $D(\cdot, \cdot)$, the expected answers are $Arr_a = \{S_{a_1}, ..., S_{a_k}\}$, where $|Arr_a| = k$, $Arr_a \subseteq Arr_s$ and $\forall S_{a_i} \in Arr_a, \forall S_j \in Arr_s - Arr_a, D(Q, S_{a_i}) \leq D(Q, S_j)$. Search requiring a recall rate of 1 is referred to as exact search, otherwise approximate search [24, 25].

2.2 Summarization techniques

We often employ summarizations to represent high-dimensional data series due to the high cost of directly manipulating them. The distance between two summarizations approximates or is lower than the distance between the data series they represent.

The Product Quantization (PQ) [33, 41] utilizes centroids obtained through K-means to represent data series. The Discrete Fourier Transform (DFT) [2, 46, 63] converts the time domain into the frequency domain. The Singular Value Decomposition (SVD) [6, 35, 49] decomposes a matrix to extract the essence of the matrix. The Discrete Haar Wavelet Transform (DHWT) [44] transforms data series into a multi-level structure. The Piecewise Aggregate Approximation (PAA) [94] and its more updated variants [29, 34, 45, 87, 88] segment data series and compress them by the the average or variance of each segment.

In data series indices for large datasets, Symbolic Aggregate Approximation (SAX) [51] is a popular summarization. SAX possesses

lower bound distance, compact storage, high pruning rate, and is training-free. To obtain SAX, the data series is divided into several segments, and the average value for each segment is computed to obtain PAA. Lastly, a Gaussian mapping function with equiprobable divisions is employed to discretize the values of PAA. SAX has numerous variants that can enhance pruning rates [55, 75, 79, 95], with the cost of increased computation and storage requirements.

2.3 Works on reference points

Reference-point-based summarizations involve selecting reference points and precomputing the distances from data series to one or more reference points. During searching, they utilize the triangle inequality for pruning. More specifically, for a query Q, a reference point O, a queried data series S and the distance of the best so far (BSF) answers D_b, if $D(S,O) \leq D(Q,O) - D_b$ or $D(S,O) \geq D(Q,O) + D_b$, we have $D(Q,S) \geq D_b$, then we can prune the S.

Some approaches [39, 61] cluster the data series and utilize cluster centroids or boundaries as reference points. Others [9–11, 19, 83] primarily use outliers or maximize the distances between reference points. There are also methods [15, 36, 85, 86] that determine reference points by cost models or queries. iDistance [39] constructs a B$^+$-tree by K-means [58] and conducts the search outward from the middle of the B$^+$-tree. [15] selects a few reference points, reduces dimensionality using a Hilbert curve [74], and then employs B$^+$-trees for indexing. HD-Index [3] segments the data series, uses a Hilbert curve and a B$^+$-tree for each segment. The ML-Index [1] is a learning index based on distances to reference points.

In conclusion, previous works often select few reference points. On the contrary, we select more reference points to achieve a higher initial pruning rate and employ approximate answers and segment-based summarizations to perform a sequential search algorithm.

2.4 Data series indices

For large high-dimensional datasets, methods that support the exact similarity search are not as abundant as ANN methods [24, 25]. Some sequential scanning methods [27, 72, 90] support similarity matching of short query series against long data series. The reference-point-based methods [16, 39, 64] can support this but may not achieve high pruning rate. The EAPCA and iSAX families are currently the state-of-the-art data series indices that support both the approximate and exact similarity searches [24, 25, 89]. We introduce their development in the following.

Many methods are built upon the binary tree and SAX. iSAX [12, 13, 77] is the first to propose using a binary tree to index SAX and to use iSAX to represent a SAX collection. It uses iSAX for initial pruning, SAX for further pruning. ADS [97, 98] introduces an adaptive index construction method, shifting construction time to the search phase. ParIS [69] employs multithreading. ULISSE [54] supports variable-length data series. MESSI [68], SING [70], DPiSAX [93] and Odyssey [14] utilize memory, GPU, and distributed system.

Some methods employ different index structures. Tardis [96] is a multi-way tree index, but it suffers from serious space wastage as it splits each segment of SAX for each partition. Coconut [47, 48] utilizes the Z-order curve [74] to sort SAX and employs a B$^+$-tree for searching. Dumpy [89] strikes a balance between Tardis and iSAX. It selects a portion of SAX segments to split by cost models. When the splitting is complete, it merges the smaller nodes.

Within the EAPCA family, DS-Tree [88] employs adaptive segmentation, and uses EAPCA to represent and prune a data series collection. Hercules [23], after utilizing DS-Tree to partition data series, further prunes them using SAX.

In summary, previous works utilize iSAX or EAPCA to represent a collection, reducing the accesses to SAX. However, iSAX and EAPCA exhibit very small lower bound distances and nearly lost their effectiveness as examples in § 7.3. Furthermore, they are all tree-based indices with limited abilities to obtain approximate answers as examples in § 7.4.

Table 1: Notations for complexity analysis.

Notation	Explanation
z	the summarization for initial pruning
x	the summarization for further pruning
p_z	the pruning rate of z
p_x	the pruning rate of x
n	the number of data series in the dataset
d	the data series dimension
w	the number of segments in summarizations
th	the leaf size of the binary tree
th_b	the leaf size of the B$^+$-tree
n_r	the number of reference points
n_g	the number of graph nodes for approximate search
e	the iterations of K-means
v	the sampling rate of K-means

2.5 Graph-based methods

Here is a brief overview of the state-of-the-art graph-based works. Hierarchical Navigating Small World (HNSW) [60] is based on the navigating small-world graph and employs a multi-level structure. Navigating Spreading-Out (NSG) [31] reduces the average node outdegree, thereby shortening search paths. HVS [57] utilizes the Hierarchical Voronoi structure in place of HNSW's upper-level structure. ELPIS [4] utilizes DS-Tree [88] to partition the data and constructs a graph for each leaf node. There are many other works in this field, which can be found in two comprehensive reviews on graph methods [52, 53]. We adopt a graph-based approach since it can yield the approximate answers with high recall rates.

3 THE STRUCTURE OF DIDS

We first discuss the design ideas of DIDS in § 3.1. Then, we propose the specific structure of DIDS based on these ideas in § 3.2.

3.1 Motivations

We begin our analysis by the time complexity of the exact search, elucidating how we design appropriate strategies to address the two issues mentioned in § 1. The notations used for analyzing algorithm complexity are presented in Table 1.

Previous works often employ two types of summarizations, z and x, to conduct initial pruning and further pruning respectively, reducing the accesses to x and data series. Hence, the cost of accessing x is $O((1-p_z)nw)$ and that of accessing data series is $O((1-p_z-p_x)nd)$.

We also assume the total cost of accessing z is c_z, the cost of the approximate search is c_a. Then, the time complexity for the exact search is given by $O(c_a + c_z + (1 - p_z)nw + (1 - p_z - p_x)nd)$.

Typically, z represents over thousands of data series, making c_z relatively small, and the c_a is also low which allows us to disregard them. Then, the complexity can be simplified to $O((1-p_z)nw+(1-p_z-p_x)nd)$. For the comparison with previous works, we utilize the SAX summarization as x, so nw and nd are both constants. Therefore, the two parameters that we can optimize are p_z and p_x.

3.1.1 Improving p_z. Existing works employ segment-based summarizations like iSAX or EAPCA as z [23, 89]. They achieve high pruning rate when representing an individual data series. However, when representing a collection, iSAX and EAPCA compute the minimum distance of the collection for each dimension, resulting in a decrease in lower bound distances. Consequently, for challenging real workloads, the p_z tends to be quite low (around 10% in § 7.3).

The reference-point-based summarizations offer a completely different sorted-based approach to represent a collection. Their have lower pruning rates compared to segment-based summarizations for an individual data series. However, their lower bound distances don't decrease when representing a collection. The reason is as follows. Given a reference point O, a query Q and a top distance D_b of the BSF answers, we can deduce that the collection Arr_s which can be pruned satisfies that $\forall S \in Arr_s, D(S,O) \leq D(Q,O) - D_b$ or $D(S,O) \geq D(Q,O) + D_b$. In other words, we only need to sort their distances to quickly prune a collection. Their lower bound distances remain unaffected for more data series. Clearly, reference-point-based summarizations are a superior choice for z.

3.1.2 Enhancing p_x. When x is fixed, improving p_x can only be achieved by enhancing the quality of the BSF answers. Previous works [23, 89] often calculate the distances between queries and summarizations of leaf nodes. They traverse the leaf nodes in ascending order of the distances and prioritize visiting nodes that might contain answers, to continuously update the BSF answers. However, it leads to random accesses. While we can enhance sequential access by increasing the size of leaf nodes like Hercules [23], it results in lower distances for iSAX or EAPCA, rendering them more inaccurate. Therefore, for previous work, the sequential access and pruning rate are inversely proportional.

To address this issue, obtaining high-quality approximate answers is essential. Because with high-quality approximate answers, there is no need to prioritize visiting specific nodes. Instead, we can sequentially traverse the entire dataset. Therefore, we consider graph-based methods, which are widely recognized for their capability to yield high-quality answers [4]. However, graphs incur substantial memory overhead and may not be well-suited for large data series collections. Consequently, we devise an approach similar to quantization: further partitioning each reference point and employing a regional centroid to represent the data series within that region. This strategy serves to reduce memory consumption.

3.2 The structure

From the analysis above, for DIDS, it is necessary to sort reference-point-based summarizations for each reference point for initial pruning. Additionally, a small-memory graph is needed to provide approximate answers.

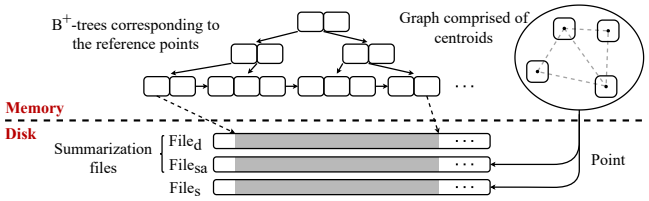


Figure 2: The structure of DIDS in memory and disk.

Algorithm 1 ClusterDataSeries

Input: Dataset Arr_s; Reference points size n_r; Read buffer size n_{rb}; Write buffer size n_{rw};
Output: The temporary files $File_{gd}[n_r], File_{gs}[n_r]$;
1: $Arr_r \leftarrow$ Select n_r reference points from Arr_s;
2: $Graph_r \leftarrow$ Construct the HNSW graph by Arr_r;
3: $Buffer_{wd}[n_r], Buffer_{ws}[n_r] \leftarrow$ Initialize each write buffer with a size of n_{rw};
4: $File_{gd}[n_r], File_{gs}[n_r] \leftarrow$ Create two files for each reference point;
5: **for** $i \leftarrow 1$ **to** $Arr_s.Size/n_r$ **do**
6: $Buffer_r \leftarrow$ load n_{rb} data series from Arr_s;
7: **for all** S **in** $Buffer_r$ **do**
8: $Id_r, Dis \leftarrow$ Obtain 1 reference point and distance from $Graph_r$ by S;
9: Push Dis, S into $Buffer_{wd}[Id_r]$ and $Buffer_{ws}[Id_r]$;
10: **if** $Buffer_{wd}[Id_r]$ is full **then**
11: Write two buffers into files and clear;
12: **return** $File_{gd}[n_r], File_{gs}[n_r]$;

The structure of DIDS is depicted in Figure 2. In memory, DIDS consists of the internal nodes of some B$^+$-trees and a graph. On disk, there are three files: $File_d$, which corresponds to the leaf nodes of the B$^+$-trees (also the reference-point-based summarizations), $File_{sa}$ for SAX summarizations, and $File_s$ for data series. The data in these three files are one-to-one corresponding.

The graph is utilized for quickly obtaining the approximate answers firstly. Each node in the graph points to a continuous space within the files, and uses the centroid of a subset of data series in that space as the representative. Subsequently, the B$^+$-trees and summarization files are used for pruning. DIDS selects a reference point for each data series and constructs a B$^+$-tree for each reference point. This aids in rapidly locating the positions of data series that cannot be pruned by reference-point-based summarizations. Then, the SAX summarizations are loaded for further pruning.

4 THE CONSTRUCTION OF DIDS

The construction can be divided into three phases: clustering data series, preparing summarizations and constructing the graph, as shown in Figure 3. In the following three subsections, we introduce them respectively.

4.1 Data series clustering

We begin by clustering data series using reference points. We employ a reference point to represent a collection of data series near it, allowing DIDS to perform the approximate search on a cluster-level graph in limited memory. Moreover, we compute the distances between the data series and the reference points, for the initial pruning in the exact search. Clustering the data series first also allows

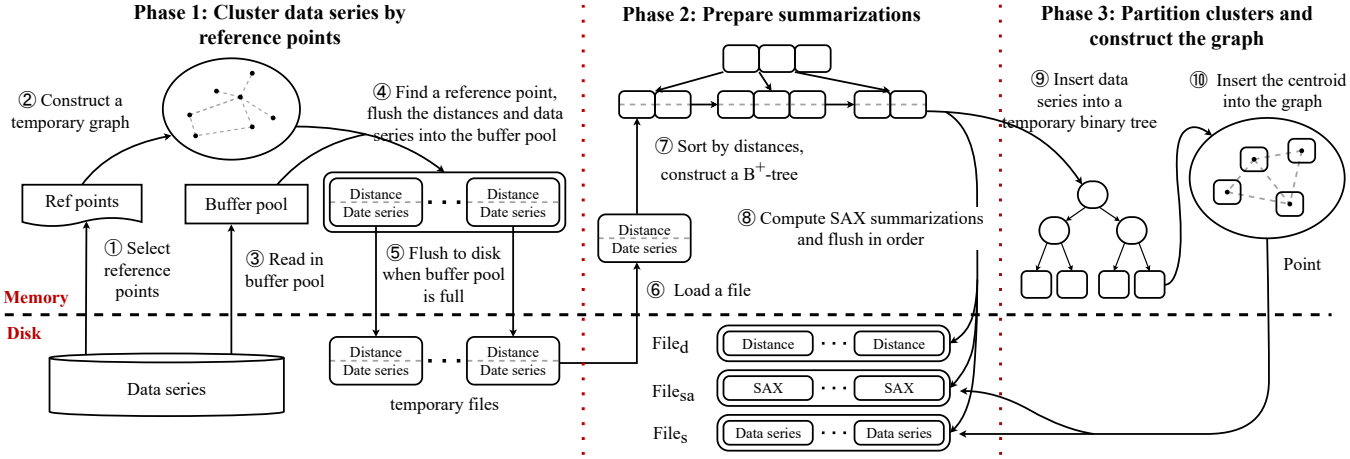


Figure 3: The overall process of DIDS construction.

Algorithm 2 PrepareSummarizations

Input: The temporary files $File_{gd}[n_r], File_{gs}[n_r]$; The distance file $File_d$; The SAX file $File_{sa}$; The data series file $File_s$; The global B$^+$-tree array Arr_b;

1: **for** $i \leftarrow 1$ **to** n_r **do**
2: $Arr_{gd}, Arr_{gs} \leftarrow$ Load from $File_{gd}[i], File_{gs}[i]$;
3: Sort Arr_{gd}, Arr_{gs} by Arr_{gd};
4: $Tree_b \leftarrow$ Construct a B$^+$-tree by Arr_{gd};
5: Push $Tree_b$ into Arr_b;
6: $Arr_{sa} \leftarrow$ Compute SAX summarizations for Arr_{gs};
7: Write $Arr_{gd}, Arr_{sa}, Arr_{gs}$ into $File_d, File_{sa}, File_s$;

Algorithm 3 ConstructTheGraph

Input: Data series array Arr_{gs}; The start position of Arr_{gs} in $File_s$ Pos_s; The global HNSW graph $Graph_l$;

1: $Tree_{bi} \leftarrow$ Initialize a binary tree;
2: **for** $i \leftarrow 1$ **to** $Arr_{gs}.Size$ **do**
3: Insert i into $Tree_{bi}$ with Arr_{gs};
4: $Arr_l \leftarrow$ Obtain all leaf nodes from $Tree_{bi}$;
5: **for all** $Node$ **in** Arr_l **do**
6: $Centroid \leftarrow$ Compute the centroid of the data series within the $Node$;
7: $Pos_{min}, Pos_{max} \leftarrow$ Calculate the minimum and maximum positions within the $Node.Arr_p$;
8: $Pos_{min}, Pos_{max} \leftarrow Pos_{min} + Pos_s, Pos_{max} + Pos_s$
9: Insert $Centroid, Pos_{min}, Pos_{max}$ into $Graph_l$;

subsequent phases to be performed at the cluster level, reducing memory consumption.

The pseudo code is shown in Algorithm 1. Firstly, we consider the reference point selection. Since our objective is to cluster data series, which can be used for the approximate search, K-means is indeed a good choice. However, the complexity of K-means can be too high for large data series collections, so we have resorted to sampling to select n_r reference points from the dataset (Line 1).

For each data series, we select the nearest reference point. However, given many reference points and data series, computing the distances for all them is highly time-consuming. Therefore, we insert all reference points into the HNSW graph (Line 2) and employ this graph to identify the nearest reference point (Line 8).

We establish a read buffer for batch reading (Line 6). Additionally, for each reference point, we create two temporary files along with corresponding write buffers (Lines 3, 4). One is for distances to the reference points, while the other is for data series. This separation of storage is intended to expedite subsequent sorting. We determine a reference point for a data series and push both the data series and its distance to the buffers (Line 9). When the buffers reach the capacity, we flush them into the corresponding files (Lines 10, 11).

4.2 Summarization preparation

In this section, we prepare two summarizations for the initial pruning and further pruning in the exact search. For initial pruning, based on § 3.1, we need to sort reference-point-based summarizations for each reference point, to rapidly locate the boundaries of the data that cannot be pruned. Thus, the B$^+$-tree is a good choice for ordered data. For further pruning, we use SAX as in the previous works [23, 89], with a higher cost but a superior pruning rate.

In Algorithm 2, we first read in the two temporary files of a reference point saved in the previous phase (Line 2). We then sort distances and data series based on distances (Line 3). Next, we build the B$^+$-tree with the distance to the reference point as the key, from the bottom to up using Arr_{gd} and record the B$^+$-tree in the global array Arr_b (Lines 4, 5). For each data series, we compute the corresponding SAX (Line 6). Finally, we write the distances, SAX, and data series to their respective files (Line 7). All reference points share the same three files, for sequential access to all B$^+$-trees during searching which is good for disk-based structures.

4.3 Graph construction

Since the cluster level remains excessively large and there are variations in the sizes of clusters, in pursuit of a more uniform partitioning and more precise approximate answers, we employ a cost model to further partition each cluster into smaller regions. Then, we construct a graph by the centroids of small regions, to conduct high-precision approximate search under low memory.

In Algorithm 3, we first insert the data series represented by array subscript i into a temporary binary tree to partition this cluster (Lines 1-3). We use an example to illustrate this binary tree. In Figure 4, we have a binary tree populated with 4 data series

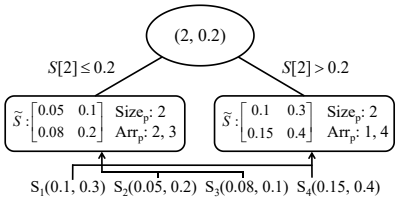


Figure 4: A binary tree construction example.

with d=2. The internal node stores routing information (2, 0.2), where 2 indicates which segment to observe, and 0.2 corresponds to the value for partitioning. The second segments of S_2, S_3 in the left node are ≤ 0.2, and those of S_1, S_4 in the right node are > 0.2. The leaf nodes have three attributes: (i) $\widetilde{S}$: for a data series collection $Arr_s = \{S_1, ..., S_n\}$ where $S_i = (s_{i_1}, ..., s_{i_d})$, the summarization $\widetilde{S} = \left[\begin{pmatrix} s_{min_1} \\ s_{max_1} \end{pmatrix}, ..., \begin{pmatrix} s_{min_d} \\ s_{max_d} \end{pmatrix}\right]$, where $s_{min_j} = \min_{1 \leq i \leq n} \{s_{i_j}\}$, $s_{max_j} = \max_{1 \leq i \leq n} \{s_{i_j}\}$. (ii) $Size_p$: the number of data series within the leaf node. (iii) Arr_p: the positions of the data series in the Arr_{gs}.

These attributes are updated when a data series is inserted. We partition a leaf node when it attains a threshold. Difference [88] and variance [89] are popular reference metrics for selecting a partitioning dimension. Here, we select difference as the reference, sacrificing precision compared to variance, but gaining a faster construction speed. Assuming $Size_p = n_p$, we define a cost model to assist in node partition: $c_l = n_p \sum_{j=1}^{d} (s_{max_j} - s_{min_j})$. This model represents the boundary differences of a leaf node, ensuring that the boundaries of the resulting leaf nodes have smaller differences, since we always choose the dimension with the largest difference for partitioning. The theoretical reasons are as follows. If we uniformly partition a node by a value *mid* and the i-th segment, and only focus on the cost of the i-th segment as the others remain unchanged, the cost of the left node is $(mid - s_{min_i})n_p/2$, and the cost of the right node is $(s_{max_i} - mid)n_p/2$. The total cost is reduced by $(s_{max_i} - s_{min_i})n_p/2$. Thus, to maximize the reduction, we create an internal node (i, mid) for partitioning, where $i = \arg\max_{1 \leq j \leq d}(s_{max_j} - s_{min_j})$ which is also the dimension with the largest difference, and *mid* is the median of the i-th segment to make partitioning more uniform. Finally, the leaf node is partitioned by the (i, mid).

We compute the centroid $Centroid$ of the data series within each leaf node and record the minimum and maximum positions Pos_{min}, Pos_{max} within the Arr_p of each leaf node (Lines 4-7). Subsequently, we add the starting position Pos_s of this cluster in $File_s$, to Pos_{min} and Pos_{max} (Line 8). Thus, Pos_{min} and Pos_{max} can point to a continuous space within $File_s$. Finally, we utilize $\{Centroid, Pos_{min}, Pos_{max}\}$ to represent a leaf node, and employ the distances of centroids to represent the distances between leaf nodes, inserting all leaf nodes into the global HNSW graph $Graph_l$ (Line 9).

5 THE SEARCH IN DIDS

From § 3, DIDS employs reference-point-based summarizations graph to enhance the pruning rate, reducing the time complexity of the exact search. Furthermore, for disk-based indices, the sequential access also holds substantial significance. DIDS constructs B$^+$-trees to cluster data that cannot be pruned into a contiguous space on disk. And with high-quality approximate answers, DIDS sequentially accesses all B$^+$-trees to reduce the cost of disk reads. Figure 5 describes the search in DIDS. We discuss the details of the approximate search in § 5.1 and the exact search in § 5.2.

Algorithm 4 ApproximateSearch

Input: Query Q; Size of graph nodes to be searched n_g; The global HNSW graph $Graph_l$; The two files $File_{sa}$, $File_s$;
Output: The approximate answers; The positions that have already been searched;
1: $Ans_a \leftarrow$ Initialize a priority queue;
2: $Arr_{sg} \leftarrow$ Select n_g nodes from $Graph_l$ by Q;
3: Sort Arr_{sg} by the Pos_{min} of each node;
4: Merge the nodes in Arr_{sg} that have overlapping regions for Pos_{min} and Pos_{max};
5: $PAA_q \leftarrow$ compute PAA summarization by Q;
6: **for all** $Node$ **in** Arr_{sg} **do**
7: $Ans_a \leftarrow$ PruneWithSAX(Q, PAA_q, $Node.Pos_{min}$, $Node.Pos_{max}$, Ans_a, $File_{sa}$, $File_s$);
8: **return** Ans_a, Arr_{sg};

Algorithm 5 PruneWithSAX

Input: Query Q; PAA PAA_q; The begin and end positions Pos_b, Pos_e; The answers Ans; The two files $File_{sa}$, $File_s$;
Output: updated answers;
1: $Arr_{sa} \leftarrow$ Load SAX summarizations from $File_{sa}$ between Pos_b and Pos_e;
2: **for** $i \leftarrow Pos_b$ **to** Pos_e **do**
3: **if** $D(PAA_q, Arr_{sa}[i - Pos_b + 1]) < Ans.TopDist$ **then**
4: $S \leftarrow$ Load i-th data series from $File_s$;
5: **if** $D(Q, S) < Ans.TopDist$ **then**
6: Update Ans by S;
7: **return** Ans;

5.1 Approximate search

DIDS employs a centroid to represent a data series collection. By a graph, DIDS can obtain the high-quality approximate answers in the limited memory. The high-quality approximate answers provide enough pruning opportunities for subsequent steps.

The algorithm for the approximate search is shown in Algorithm 4. We first initialize a priority queue for approximate answers (Line 1). Then, we search n_g nodes near to the Q from the $Graph_l$ (Line 2). We sort the nodes in ascending order of the Pos_{min}, and merge nodes with intersecting position ranges (Lines 3, 4). Merging nodes enables sequential access, thereby expediting the approximate search. Next, We compute the PAA of the query to prepare for subsequent pruning (Line 5). We iterate through each node, and process the data series between $Node.Pos_{min}$ and $Node.Pos_{max}$ using SAX summarizations sequentially (Lines 6, 7).

The algorithm for using the SAX to process the data series is shown in Algorithm 5. We read the corresponding SAX from the disk (Line 1). Next, we access each SAX, compute the lower bound distance between the PAA of the query and each SAX with the formula from [77] (Lines 2, 3). If the distance is lower than the top distance of the BSF answers, we load the data series from $File_s$, compute the actual distance, and update the answers (Lines 4-6).

5.2 Exact search

With high-quality approximate answers and a well-structured DIDS index, the exact search as depicted in Figure 5, just sequentially accesses continuous spaces and achieves the high pruning rate.

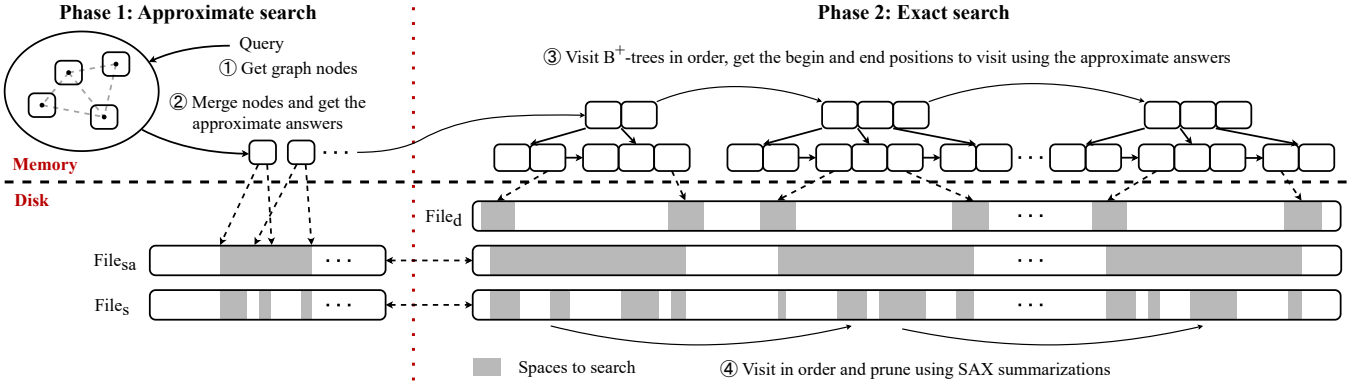


Figure 5: The overall process of the exact search in DIDS.

Algorithm 6 ExactSearch

Input: Query Q; Size of graph nodes to be searched n_g; The global B$^+$-tree array Arr_b; The global HNSW graph $Graph_l$; The three files $File_d$, $File_{sa}$, $File_s$;
Output: Exact answers;
1: $Ans, Arr_{sg} \leftarrow$ ApproximateSearch(Q, n_g, Arr_b, $Graph_l$, $File_{sa}$, $File_s$);
2: **for all** $Tree_b$ **in** Arr_b **do**
3: $Ans \leftarrow$ SearchTheB$^+$-tree(Q, $Tree_b$, $Tree_b.RefPoint$, Arr_{sg}, Ans, $File_d$, $File_{sa}$, $File_s$);
4: **return** Ans;

Algorithm 7 SearchTheB$^+$-tree

Input: Query Q; The B$^+$-tree $Tree_b$; The reference point O; The positions that have already been searched Arr_{sg}; The BSF answers Ans; Three files $File_d, File_{sa}, File_s$;
Output: The updated answers;
1: $Pos_b, Pos_e \leftarrow$ Search the positions of $D(Q,O) - Ans.TopDist$ and $D(Q,O) + Ans.TopDist$ in $Tree_b$ by $File_d$;
2: $Arr_{range} \leftarrow$ Utilize Arr_{sg} to eliminate the positions between Pos_b and Pos_e that have already been searched in the approximate search.
3: $PAA_q \leftarrow$ compute PAA summarization by Q;
4: **for all** Pos_{b_i}, Pos_{e_i} **in** Arr_{range} **do**
5: $Ans \leftarrow$ PruneWithSAX(Q, PAA_q, Pos_{b_i}, Pos_{e_i}, Ans, $File_{sa}$, $File_s$);
6: **return** Ans;

The algorithm is shown in Algorithm 6. We first obtain the approximate answers and the positions that have been searched by the approximate search (Line 1). Then, we search each B$^+$-tree in disk order (Lines 2, 3). The algorithm for accessing the B$^+$-tree is shwon in Algorithm 7. From § 3.1, data series with distances from $D(Q,O) - Ans.TopDist$ to $D(Q,O) + Ans.TopDist$ cannot be pruned using reference-point-based summarizations. Thus, we use B$^+$-tree to locate the positions of these two distances (exclusive), and only process data in between (Line 1). Then, we remove the positions that have been searched during the approximate search, obtaining several smaller ranges (Line 2). Finally, we compute the PAA of the query and sequentially employ SAX to prune data series within each of the small ranges (Lines 3-5).

Note that the algorithms are used for SSD, which allows for simultaneous access to all three files. However, for HDD, DIDS should access a whole file to obtain all results before next file.

6 COMPLEXITY ANALYSIS

The notations are shown in Table 1 and we have discussed the exact search complexity in § 3.1, here we discuss the construction in § 6.1 and the approximate search in § 6.2. From [60], for the HNSW graph with m nodes, the search complexity is around $O(d\log m)$, the construction is $O(md\log m)$, and the space is around $O(md)$.

6.1 Construction complexity

6.1.1 Time complexity. The original complexity of the K-means is $O(evn_rnd)$. We can construct a graph with means for search purposes during each iteration like § 4.1, and degrade it to $O(evnd\log n_r)$. The total complexity for selecting reference points for n data series is $O(nd\log n_r)$. In the latter two phases of construction, we construct a B$^+$-tree and a binary tree for each cluster ($O(n/n_r)$ data series), with the cost of around $O(nd/n_r \log n/n_r)$ for one cluster. Computing training-free summarizations takes approximately $O(nd)$. With $O(n/th)$ centroids, the complexity of constructing a graph is $O(nd/th\log n/th)$. In summary, the total time complexity is $O(nd((ev+1)\log n_r+\log n/n_r+1+1/th\log n/th))$. We often set $ev < 1$, and $th > 1000$ which makes $1/th\log n/th < 1$. As a result, the time complexity can be simplified to the $O(nd(\log n_r+\log n/n_r))$.

Subsequently, we consider disk writes in DIDS. DIDS writes temporary files firstly, then constructs three final files, with a volume of writes slightly exceeding $O(2nd)$. Note that DIDS performs a sequential appending for each file. The disk reads are similar.

Evidently, the complexity of DIDS is lower than that of graph-based methods, which is $O(nd\log n)$. Given that the search in graph entails higher constants, DIDS often exhibits a longer construction time compared to tree-based indices.

6.1.2 Space complexity. In memory, we have two structures, the internal nodes of the B$^+$-trees, which occupy $O(n/th_b)$ space, and $Graph_l$, which occupies around $O(nd/th)$ space. Furthermore, we store reference points which occupy $O(n_rd)$ space. Since th_b and th are often similar, n/th_b is much smaller than nd/th. And we often set n_r smaller than n/th. Therefore, the total memory consumption is $O(nd/th)$. We often set $th > 1000$, making the memory around 0.1% of the dataset size. On disk, we have three files: $File_d$ with a size of $O(n)$, $File_{sa}$ with a size of $O(nw)$, and $File_s$ with a size of $O(nd)$. In total, the disk space required is around $O(n(w+d))$.

Table 2: Data statistics.

Dataset	n	d	Storage	Type	n_r	th	n_g
DEEP500M-1B	500M-1B	96	192-384GB	Image	20000	8000	200
SIFT	100M	128	51.2GB	Image	10000	2000	200
SALD	100M	128	51.2GB	Neuro			
SPACEV	100M	100	40GB	Text			
DEEP	100M	96	38.4GB	Image			
TINY	79M	150	47.5GB	Image			
DEEP10M-50M	10M-50M	96	3.8-19.2GB	Image	5000	2000	50
SYNTHETIC10-1024	10M	10-1024	0.4-38GB	Synthetic			
IMAGENET	2.3M	150	1.3GB	Image	1000	800	10
MSONG	1M	420	1.5GB	Audio			

6.2 Approximate search complexity

The majority of the approximate search time is spent on processing the data series from the graph nodes. When we search a leaf node, we search all data series that fall between the minimum and maximum positions of that node. Therefore, the size of a node is no more than a cluster $O(n/n_r)$. So, the upper bound of accessed data series is $O(n_g n/n_r)$. However, due to the overlapping of many nodes, the actual complexity is much smaller. The complexity is similar to tree-based indices, involving the accesses to data within several leaf nodes. However, DIDS offers higher recall rates.

7 EXPERIMENTS

7.1 Experimental setup

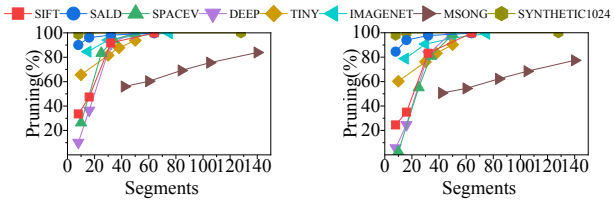
7.1.1 Environment. We conduct the experiments on a server with an Intel Core i9-13900, 128GB of RAM, a 2T NVMe SSD with 3GB/sec sequential access throughput and 1GB/sec random access throughput. We implement all algorithms in C++, and use the GCC 9.4.0 with the O3 optimization flag on Ubuntu Linux 20.04.

7.1.2 Datasets. As shown in Table 2, we leverage 8 datasets. These datasets have been extensively employed in prior experiments: (i) SIFT [42] comprises a collection of SIFT vectors for image representation. (ii) SALD [91] contains a collection of neuroscience MRI samples. (iii) SPACEV [18] encompasses natural language encodings from the commercial search engine. (iv) DEEP [5] contains deep learning network vectors associated with images. (v) TINY [82] comprises images for the detection of small targets. (vi) IMAGENET [20] is a popular image dataset. (vii) MSONG [7] contains the feature analysis for one million songs. (viii) SYNTHETIC [26] is a random dataset to model the financial data. For each new number, we add a new number from a Gaussian distribution to the last number.

All datasets are z-normalized, a prerequisite for SAX.

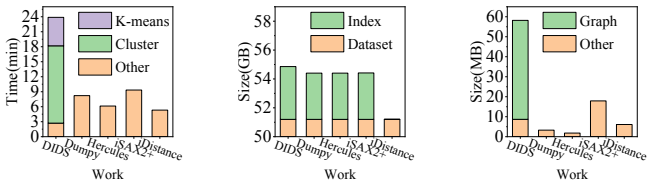
Queries consist of data series with dimensions identical to the datasets but are not present in the datasets (except for SPACEV and TINY). We extract 100 queries from the original query sets of datasets. Unless otherwise specified, we set k to 10 and set memory to 25% of the size of datasets, ensuring that all methods are disk-based. Experimental results represent the averages across these 100 queries.

7.1.3 Competitors and parameters. In § 2, we introduce previous works. For each category of works, we select a representative. All parameters used are adopted from the respective papers. We use



(a) k=10 (b) k=100

Figure 6: The theoretical maximum pruning rates for different SAX segments when k=10 and k=100. We use the exact answers for pruning.



(a) Time (b) Disk (c) Memory

Figure 7: The construction time and the index size in disk and memory for SIFT.

euclidean distance and disable multithreading and SIMD since they introduce inaccuracies in our evaluations.

- Dumpy [89]: the latest work that introduces a novel multi-way tree index structure based on iSAX. The leaf size is set to 10,000.
- Hercules [23]. It claims the fastest data series index for exact search, and is the latest EAPCA work. The leaf size is set to 100,000, and *SAX_TH* is set to 0.5. For *EAPCA_TH*, while the original setting was 0.25, in our more challenging workload, the pruning rate of EAPCA always falls below 0.25, leading to full sequential scan of data series. Consequently, we set it to 0.
- iSAX2+ [13]: the most classic work with a binary tree index on SAX. The leaf size is set to 2,000. We delete the first level of iSAX2+ due to our longer SAX segment configuration, since the first level results in fewer than 5 data series per node.
- iDistance [39]: a classic reference-point-based exact similarity search index. The number of reference points is set to 64, and the initial radius r is set to 0.01, with a Δr of 0.01.

The SAX cardinality is set to 256. Although prior works set SAX segments to 16, the choice of SAX segments heavily depends on the dataset and cannot be uniformly applied. In Figure 6, we vary the number of SAX segments and compute the exact answers for queries when k=10 and k=100. Then, we use the exact answers and SAX to perform pruning. The resulting pruning rates represent the theoretical maximum pruning rates achievable with SAX. These pruning rates must be sufficiently high. Otherwise, SAX would become ineffective. The final segments selected for each dataset are as follows: SIFT, DEEP: 32; SALD, SYNTHETIC: 16 (5 for SYNTHETIC10); SPACEV 34; TINY, IMAGENET: 50; MSONG: 140.

As shown in Table 2, we select the DIDS parameters mainly based on the size (n) of the dataset. We sample 1% for each dataset

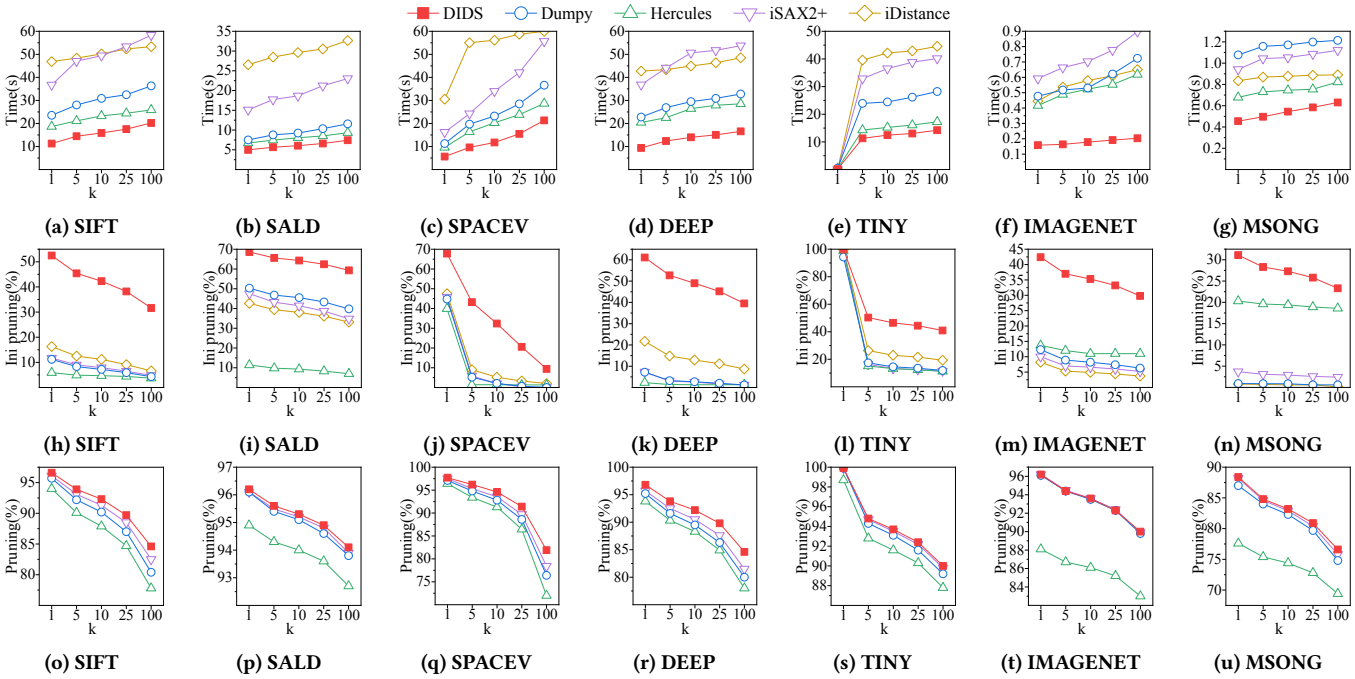


Figure 8: The exact similarity search performance of various works under different values of k and across different datasets. (a)-(g) represent the execution time. (h)-(n) denote the initial pruning rate. (o)-(u) show the overall pruning rate.

to execute the K-means algorithm with at most 30 iterations. We discuss how to select parameters for DIDS in § 7.5.

7.2 Index construction

We conduct experiments for index construction using SIFT, which is independent of the dataset. In Figure 7a, we record the construction times. The construction time for DIDS is 2-3 times longer than that of other works due to its more complex structures, with the K-means and cluster phases consuming 88%. Since DIDS targets scenarios involving offline construction for multiple queries, the construction time is entirely acceptable. And these two principal time-consuming phases can be easily parallelized in an offline environment.

Regarding the index size on disk in Figure 7b, there is not a significant difference between DIDS and other works, since the most of files primarily consist of SAX summarizations. For the size of the index structures in memory in Figure 7c, DIDS is larger than other works, with the graph structure accounting for 84%. However, the size of DIDS is around 0.1% of the overall dataset size, which is entirely acceptable for the size of memory, often measured in GB.

7.3 Exact search performance

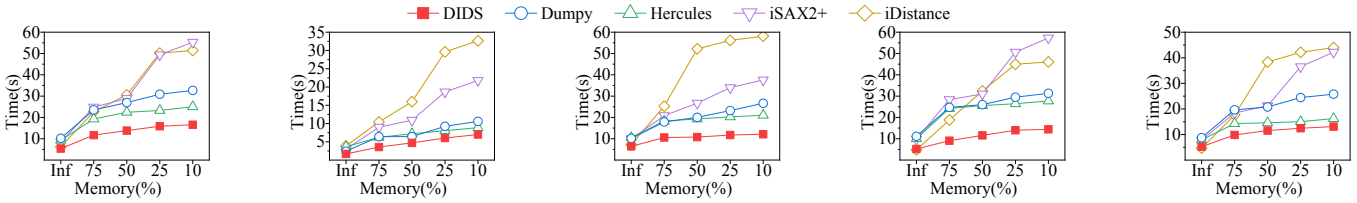
We evaluate the exact search performance across various workloads, which is the foremost functionality of these works.

7.3.1 The k of queries. In Figure 8, we vary the value of k in the queries which is also equivalent to altering the query difficulty across 7 distinct datasets for the exact similarity search. We record three metrics: (i) execution time; (ii) initial pruning rate; (iii) overall pruning rate = initial pruning rate + further pruning rate. The overall pruning rate is usually above 80%, so the percentage of the initial pruning rate to the overall pruning rate is approximately similar to the initial pruning rate itself. Subsequently, we proceed to analyze each of these three metrics.

Figure 8a-Figure 8g illustrate execution time, a paramount metric for gauging the efficacy of a work. DIDS significantly outperforms the previous works. On average, DIDS achieves search speeds, which are 2.16 times that of Dumpy, 1.7 times that of Hercules, 2.98 times that of iSAX2+, and 3.37 times that of iDistance.

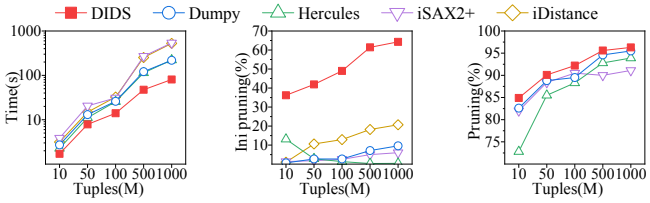
Figure 8h-Figure 8n illustrate the initial pruning rate. For initial pruning, DIDS and iDistance employ reference-point-based summarizations, Dumpy and iSAX2+ utilize iSAX, and Hercules employs EAPCA. Evidently, a higher initial pruning rate translates to fewer SAX visits and, consequently, faster search speed. DIDS significantly outperforms its competitors in initial pruning rate. On average, DIDS boasts an initial pruning rate that is 12 times over Dumpy, 9.36 times over Hercules, 10.35 times over iSAX2+, and 11.26 times over iDistance. The initial pruning rates of Dumpy, Hercules and iSAX2+ are often less than 10%. This observation underscores that while segment-based summarizations have the tight lower bound distance for individual data series, they prove ineffective in representing a collection. In contrast, reference-point-based summarizations exhibit higher pruning rate for initial pruning.

Figure 8o-Figure 8u represent the overall pruning rate. iDistance's overall pruning rate is identical to its initial pruning rate, and significantly differs from the other works. Hence, we don't include iDistance in these graphs. The overall pruning rates of the remaining works are close, because they all rely on SAX for pruning. DIDS exhibits a slight advantage over the others due to its



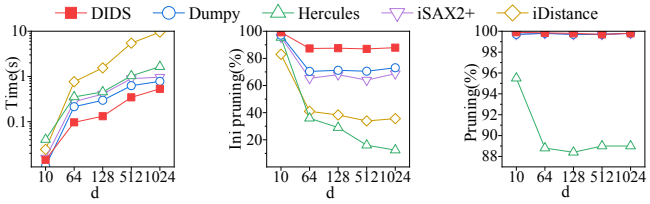
(a) SIFT (b) SALD (c) SPACEV (d) DEEP (e) TINY

Figure 9: The exact search performance of different works under different memory sizes.



(a) Time (b) Initial pruning (c) Overall pruning

Figure 10: The exact search performance of works on datasets with various sizes.



(a) Time (b) Initial pruning (c) Overall pruning

Figure 11: The exact search performance of works on datasets with various dimensions.

reference-point-based summarizations and highest-quality approximate answers. Hercules exhibits notably lower pruning rate, but achieves faster search speed. This is attributed to Hercules having larger leaf size, resulting in lower pruning rate but more sequential accesses. In summary, both the pruning rate and sequential access are crucial factors. However, in previous works, they tend to be inversely proportional. DIDS has successfully optimized both aspects, thereby attaining the fastest search speed.

7.3.2 The runtime memory. In Figure 9, we vary the memory size as a certain proportion of the datasets and record the times. On average, the search speed for DIDS is 1.9 times that of Dumpy, 1.68 times that of Hercules, 2.54 times that of iSAX2+, and 2.9 times that of iDistance. While the memory of DIDS index is larger than that of other works, it remains just under 100MB. In the context of modern memory, typically measured in GB, the memory of DIDS is less than 1%, so the impact of runtime memory is independent of the index size. For DIDS, the impact of memory size is not significant, and it does not degrade as severely as it does for iDistance and iSAX2+. iDistance exhibits similar performance to DIDS when running in memory, indicating that SAX is less suitable in memory scenarios.

7.3.3 The size of dataset. In Figure 10, we vary the size of the DEEP dataset to assess the performance of various works. The larger the size, the faster the performance of DIDS compared to other works, ranging from 1.7x to 4.6x on average. We can find the reason from Figure 10b, where the initial pruning rates of DIDS exhibit greater superiority at larger sizes. This is reasonable, as larger datasets typically make the reference points of DIDS more effective.

7.3.4 The dimension of data series. In Figure 11, we alter the dimensions of the SYNTHETIC dataset to examine the scalability. We find when d=10 and d=1024, the performance of DIDS exhibits no discernible difference compared to Dumpy and iSAX2+. We explicate the reason from the perspective of time complexity. The complexity of the exact search is $O((1-p_z)nw + (1-p_z-p_x)nd)$. DIDS reduces complexity by improving p_z and p_x. For d=10, the low dimension obscures the degradation of the lower bound distances in segment-based summarizations that consider each dimensional boundary, thereby improving the p_z of other works in Figure 11b. For d=1024, we always set w=16, leading to a high d/w=64. Then, $(1-p_z)nw$ is significantly smaller than $(1-p_z-p_x)nd$, thereby weakening the impact of p_z. In summary, a small d or a large d/w reduces the gap in complexity between DIDS and other works.

7.4 Approximate search performance

In this subsection, we examine the approximate search capabilities of various works. Although the approximate search is not a forte of disk-based data series indices compared to memory-based graph methods, the approximate answers serve as the inputs to the exact search, influencing the search strategies and pruning rate significantly. In Figure 12, when k=10 and k=100, we vary certain parameters (n_g of DIDS, the number of iterations of iDistance, and the number of leaf nodes to be searched in other works) to control the time of the search. We record the corresponding recall rates, a vital metric for evaluating the quality of approximate answers.

As demonstrated in Figure 12, on recall rates, DIDS significantly outperforms other works. Under the same execution time (we consider the median time for each graph), on average, DIDS exhibits a recall rate 67% higher than Dumpy, 59% higher than Hercules, 69% higher than iSAX2+, and 73% higher than iDistance. DIDS stands as the sole work within exact data series indices that can achieve a high recall rate, with the relatively short execution time. Relying on the high-quality approximate answers, DIDS manages to maintain the highest overall pruning rate in Figure 8, while simultaneously enabling efficient sequential access to all B$^+$-trees.

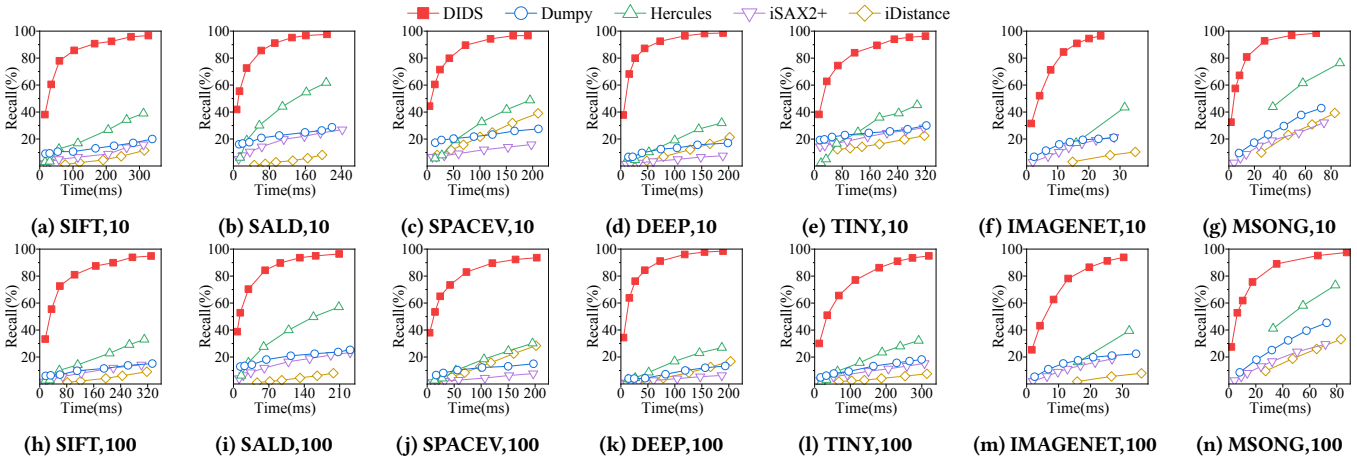


Figure 12: The recall rates of various works for the approximate search at different execution times. (a)-(g) present the search results for k=10, while (h)-(n) illustrate the search results for k=100.

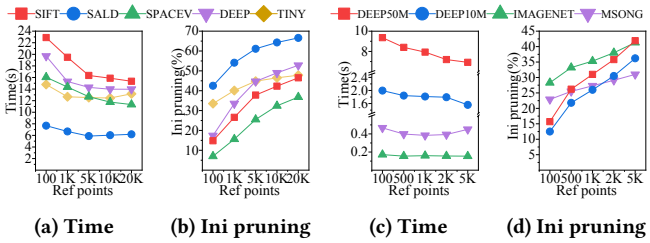


Figure 13: The exact search time and initial pruning rate for DIDS at different numbers of reference points. Large datasets are in (a)-(b) and small datasets are in (c)-(d).

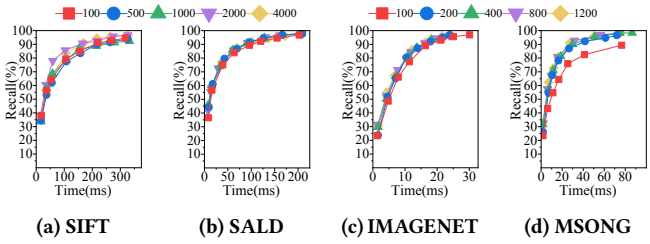


Figure 14: The recall rates of DIDS with different leaf size of the binary tree on four datasets.

7.5 The impact of parameters

In this subsection, we investigate the influence of various parameters on the performance of DIDS and discuss how to choose them.

7.5.1 The number of reference points n_r. In Figure 13, an increased n_r yields faster exact search speed, since it improves the initial pruning rate in Figure 13b and Figure 13d. However, excessively large n_r results in a decrease in the average cluster size n/n_r, leading to a decline in sequential access. We hope that both n_r and n/n_r are substantial, concurrently enhancing the initial pruning rate and sequential access. Hence, from both theoretical and experimental perspectives, we set n_r to around $\sqrt{n}$, ensuring that n_r and n/n_r increase or decrease simultaneously with variations in dataset size. The leaf size in previous works is like n/n_r, and improving them makes a lower pruning rate and an increase in sequential access.

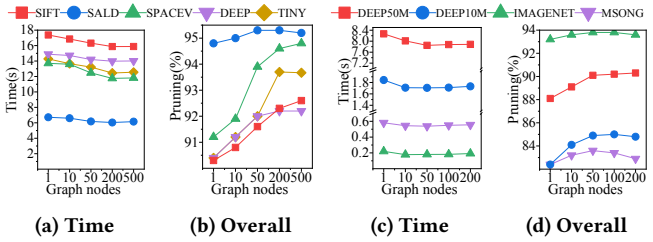
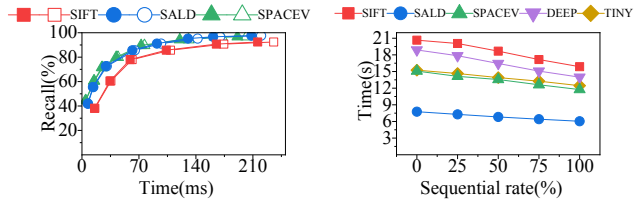


Figure 15: The exact search performance of DIDS under varying numbers of graph nodes for the approximate search. Large datasets are in (a)-(b) and small datasets are in (c)-(d).

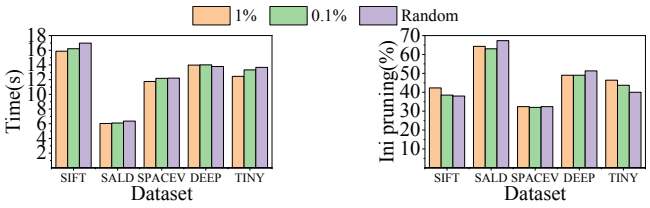
7.5.2 The leaf size of the binary tree th. In Figure 14, we find that the approximate search ability of DIDS exhibits a relatively insensitivity to th, with the disparity in recall rates within 10%. th is used to achieve a more equitable partitioning of clusters, so it is less than the average cluster size n/n_r. An excessively small th results in an increase in the number of graph nodes, improving the cost of search on the graph. Thus, we set th slightly less than n/n_r, as DIDS often demonstrates commendable performance with it in Figure 14.

7.5.3 The number of graph nodes n_g. In Figure 15, more graph nodes slightly enhance the exact search speed because they yield higher-quality approximate answers and a greater overall pruning rate in Figure 15b and Figure 15d. A graph node accesses th data series on average. We hope the accessed data series $n_g th$ is much less than the dataset size n like 1%, preventing a performance decline due to the inability to apply the exact search pruning on them. Considering the experiments, we set n_g slightly less than $(n/th)\%$.



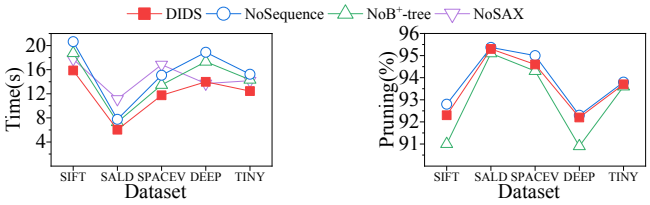
(a) Approximate search (b) Exact search

Figure 16: The search performance of DIDS under the various degrees of sequential access. The hollow symbols in (a) signifies the omission of sequential access.



(a) Time (b) Initial pruning rate

Figure 17: The exact search performance of DIDS under K-means and random reference point selection strategies.



(a) Time (b) Overall pruning rate

Figure 18: An ablation experiment about the exact search for DIDS at different datasets.

7.5.4 The sequential access. In Figure 16, we assess the impact of sequential access on the approximate and exact performance. In Figure 16a, we evaluate two approximate search methods: sequential access nodes in § 5.1 and access based on the distances between graph nodes and query. As the recall rate increases, sequential access outperforms the other one 5%-10% in performance. In Figure 16b, we conduct sequential access on partial B^+-trees, while accessing the rest based on the distances between reference points and query, to explore the impact on exact search. Evidently, as the degree of sequential access increases, the performance improves.

7.5.5 The reference point selection algorithm. In Figure 17, we conduct data sampling at rates of 1% and 0.1% to employ K-means and compare it to random selection, recording the exact search times in Figure 17a and initial pruning rates in Figure 17b. We find that, as the sampling rate increases, both the exact search speed and the initial pruning rate, overall improves. Furthermore, it is evident that K-means outperforms random selection.

7.5.6 An ablation experiment. *NoSequence* signifies accessing B^+-trees in ascending order of the distances between query and reference points. *NoB^+-tree* entails the omission of B^+-tree pruning. *NoSAX* indicates the omission of SAX pruning. In Figure 18a, the incomplete DIDS exhibits varying degrees of performance degradation. Apart from DEEP, since it achieves a high initial pruning rate, further utilization of SAX summarizations may not provide significant benefits. In Figure 18b (*NoSAX* is not included here due to its significantly lower pruning rate), we observe that when DIDS employs sequential scanning directly, its pruning rates are comparable to that of *NoSequence* with optimization approach. This demonstrates the high quality of DIDS's approximate answers. *NoB^+-tree* exhibits a lower pruning rate, indicating that using both summarizations concurrently can enhance the pruning rate.

7.6 Experimental summary

Our experiments provide insights into how to design an effective data series index that supports both approximate and exact searches.

Firstly, the reference-point-based summarizations are more suitable for initial pruning. Segment-based summarizations, like iSAX or EAPCA, provide very small lower bound distances, resulting in limited initial pruning rates in § 7.3. The lower bound distances of reference-point-based summarizations don't decrease when representing a collection, so their initial pruning rates are much higher.

Secondly, the high-quality approximate answers are important. In previous works, the pruning rate and sequential access are inversely proportional, since they cannot obtain the high-quality approximate answers in a short time, requiring optimizing answers by various strategies during the exact search, leading to random accesses. Thus, we introduce graph-based methods to obtain high-quality approximate answers as a solution to this issue.

8 CONCLUSION

We initially identify two flaws in existing data series indices supporting both approximate and exact similarity searches: one being the use of segment-based summarizations leading to lower initial pruning rates, and the other being the reliance on tree-based indices resulting in inferior quality of the approximate answers.

We introduce a novel approach, DIDS, to address these flaws. DIDS employs reference-point-based summarizations, whose lower bound distances remains unaffected for initial pruning, reducing the pruning cost. Moreover, DIDS combines reference points and a graph to obtain the high-quality approximate answers in limited memory. The high-quality approximate answers contribute to DIDS higher pruning rate and more sequential accesses. Based on extensive experiments, DIDS outperforms previous works in terms of the pruning cost, pruning rate, and sequential access. Consequently, it exhibits superior search performance.

In the future, we will parallelize DIDS and explore the possibility of adopting more advanced segment-based summarizations.

ACKNOWLEDGMENTS

This paper was supported by NSFC grant (62232005, 62202126) and The National Key Research and Development Program of China (2020YFB1006104).

REFERENCE

[1] E. Milchevski A. Davitkova and S. Michel. 2020. The ML-Index: A multidimensional, learned index for point, range, and nearest-neighbor queries. *EDBT* (2020), 407–410.

[2] Rakesh Agrawal, Christos Faloutsos, and Arun Swami. 1993. Efficient similarity search in sequence databases. In *Foundations of Data Organization and Algorithms: 4th International Conference, FODO'93 Chicago, Illinois, USA, October 13–15, 1993 Proceedings 4*. Springer, 69–84.

[3] Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. 2018. HD-Index: Pushing the Scalability-Accuracy Boundary for Approximate kNN Search in High-Dimensional Spaces. *Proceedings of the VLDB Endowment* 11, 8 (2018).

[4] Ilias Azizi, Karima Echihabi, and Themis Palpanas. 2023. ELPIS: Graph-Based Similarity Search for Scalable Data Science. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1548–1559.

[5] Artem Babenko and Victor Lempitsky. 2016. Efficient indexing of billion-scale datasets of deep descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2055–2063.

[6] Štěpán Beneš and Jaroslav Kruis. 2018. Singular value decomposition used for compression of results from the finite element method. *Advances in Engineering Software* 117 (2018), 8–17.

[7] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. 2011. The Million Song Dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*.

[8] Paul Boniol, Michele Linardi, Federico Roncallo, and Themis Palpanas. 2020. Automated anomaly detection in large sequences. In *2020 IEEE 36th international conference on data engineering (ICDE)*. IEEE, 1834–1837.

[9] Nieves R Brisaboa, Antonio Farina, Oscar Pedreira, and Nora Reyes. 2006. Similarity search using sparse pivots for efficient multimedia information retrieval. In *Eighth IEEE International Symposium on Multimedia (ISM'06)*. IEEE, 881–888.

[10] Benjamin Bustos, Gonzalo Navarro, and Edgar Chávez. 2003. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters* 24, 14 (2003), 2357–2366.

[11] Benjamin Bustos, Oscar Pedreira, and Nieves Brisaboa. 2008. A dynamic pivot selection technique for similarity search. In *2008 IEEE 24th International Conference on Data Engineering Workshop*. IEEE, 394–401.

[12] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn Keogh. 2010. isax 2.0: Indexing and mining one billion time series. In *2010 IEEE International Conference on Data Mining*. IEEE, 58–67.

[13] Alessandro Camerra, Jin Shieh, Themis Palpanas, Thanawin Rakthanmanon, and Eamonn Keogh. 2014. Beyond one billion time series: indexing and mining very large time series collections with SAX2+. *Knowledge and information systems* 39, 1 (2014), 123–151.

[14] Manos Chatzakis, Panagiota Fatourou, Eleftherios Kosmas, Themis Palpanas, and Botao Peng. 2023. Odyssey: A Journey in the Land of Distributed Data Series Similarity Search. *Proceedings of the VLDB Endowment* 16, 5 (2023), 1140–1153.

[15] Lu Chen, Yunjun Gao, Xinhan Li, Christian S Jensen, and Gang Chen. 2015. Efficient metric indexing for similarity search. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 591–602.

[16] Lu Chen, Yunjun Gao, Xuan Song, Zheng Li, Yifan Zhu, Xiaoye Miao, and Christian S Jensen. 2022. Indexing metric spaces for exact similarity search. *Comput. Surveys* 55, 6 (2022), 1–39.

[17] Qi Chen, Haidong Wang, Mingqin Li, Gang Ren, Scarlett Li, Jeffery Zhu, Jason Li, Chuanjie Liu, Lintao Zhang, and Jingdong Wang. 2018. SPTAG: A library for fast approximate nearest neighbor search.

[18] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. Spann: Highly-efficient billion-scale approximate nearest neighborhood search. *Advances in Neural Information Processing Systems* 34 (2021), 5199–5212.

[19] Sanjoy Dasgupta and Philip M Long. 2005. Performance guarantees for hierarchical clustering. *J. Comput. System Sci.* 70, 4 (2005), 555–569.

[20] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[21] Wei Dong. 2014. Kgraph, an open source library for k-nn graph construction and nearest neighbor search.

[22] Matthys Douze, Alexandre Sablayrolles, and Hervé Jegou. 2021. Fast indexing with graphs and compact regression codes on online social networks. US Patent 11,093,561.

[23] Karima Echihabi, Panagiota Fatourou, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2022. Hercules against data series similarity search. *Proceedings of the VLDB Endowment* 15, 10 (2022), 2005–2018.

[24] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2020. The lernaean hydra of data series similarity search: An experimental evaluation of the state of the art. *arXiv preprint arXiv:2006.11454* (2020).

[25] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2020. Return of the lernaean hydra: Experimental evaluation of data series approximate similarity search. *arXiv preprint arXiv:2006.11459* (2020).

[26] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. 1994. Fast subsequence matching in time-series databases. *ACM Sigmod Record* 23, 2 (1994), 419–429.

[27] Hakan Ferhatosmanoglu, Ertem Tuncel, Divyakant Agrawal, and Amr El Abbadi. 2000. Vector approximation based indexing for non-uniform high dimensional data sets. In *Proceedings of the ninth international conference on Information and knowledge management*. 202–209.

[28] Luciano Floridi and Massimo Chiriatti. 2020. GPT-3: Its nature, scope, limits, and consequences. *Minds and Machines* 30 (2020), 681–694.

[29] Vanel Steve Siyou Fotso, Engelbert Mephu Nguifo, and Philippe Vaslin. 2019. Grasp heuristic for time series compression with piecewise aggregate approximation. *RAIRO-Operations Research* 53, 1 (2019), 243–259.

[30] Cong Fu, Changxu Wang, and Deng Cai. 2021. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 8 (2021), 4139–4150.

[31] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proceedings of the VLDB Endowment* 12, 5 (2019), 461–474.

[32] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. 2012. Locality-sensitive hashing scheme based on dynamic collision counting. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data*. 541–552.

[33] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization. *IEEE transactions on pattern analysis and machine intelligence* 36, 4 (2013), 744–755.

[34] Chonghui Guo, Hailin Li, and Donghua Pan. 2010. An improved piecewise aggregate approximation based on statistical features for time series mining. In *Knowledge Science, Engineering and Management: 4th International Conference, KSEM 2010, Belfast, Northern Ireland, UK, September 1-3, 2010. Proceedings 4*. Springer, 234–244.

[35] Jiangfeng Guo, Ranhong Xie, and Guowen Jin. 2018. An efficient method for NMR data compression based on fast singular value decomposition. *IEEE Geoscience and Remote Sensing Letters* 16, 2 (2018), 301–305.

[36] Christian Hennig and Longin Jan Latecki. 2003. The choice of vantage objects for image retrieval. *Pattern Recognition* 36, 9 (2003), 2187–2196.

[37] Qiang Huang, Jianlin Feng, Qiong Fang, Wilfred Ng, and Wei Wang. 2017. Query-aware locality-sensitive hashing scheme for lp norm. *The VLDB Journal* 26, 5 (2017), 683–708.

[38] Pablo Huijse, Pablo A Estevez, Pavlos Protopapas, Jose C Principe, and Pablo Zegers. 2014. Computational intelligence challenges and applications on large-scale astronomical time series databases. *IEEE Computational Intelligence Magazine* 9, 3 (2014), 27–39.

[39] Hosagrahar V Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. 2005. iDistance: An adaptive B+-tree based indexing method for nearest neighbor search. *ACM Transactions on Database Systems (TODS)* 30, 2 (2005), 364–397.

[40] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in Neural Information Processing Systems* 32 (2019).

[41] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

[42] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. 2011. Searching in one billion vectors: re-rank with source coding. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 861–864.

[43] K. Kashino, G. Smith, and H. Murase. 1999. Time-series active search for quick retrieval of audio and video. In *Acoustics, Speech, and Signal Processing, 1999. Proceedings., 1999 IEEE International Conference on*.

[44] Shrikant Kashyap and Panagiotis Karras. 2011. Scalable knn search on vertically stored time series. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1334–1342.

[45] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. 2001. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*. 151–162.

[46] Chamari I Kithulgoda, Russel Pears, and M Asif Naeem. 2018. The incremental Fourier classifier: Leveraging the discrete Fourier transform for classifying high speed data streams. *Expert Systems with Applications* 97 (2018), 1–17.

[47] Haridimos Kondylakis, Niv Dayan, Kostas Zoumpatianos, and Themis Palpanas. 2018. Coconut: A Scalable Bottom-Up Approach for Building Data Series Indexes. PVLDB 11, 6 (2018), 677–690.

[48] Haridimos Kondylakis, Niv Dayan, Kostas Zoumpatianos, and Themis Palpanas. 2019. Coconut: sortable summarizations for scalable indexes over static and streaming data series. *The VLDB Journal* 28 (2019), 847–869.

[49] Flip Korn, Hosagrahar V Jagadish, and Christos Faloutsos. 1997. Efficiently supporting ad hoc queries in large datasets of time sequences. *Acm Sigmod Record* 26, 2 (1997), 289–300.

[50] Yifan Lei, Qiang Huang, Mohan Kankanhalli, and Anthony KH Tung. 2020. Locality-sensitive hashing scheme based on longest circular co-substring. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2589–2599.

[51] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. 2003. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*. 2–11.

[52] Peng-Cheng Lin and Wan-Lei Zhao. 2019. A comparative study on hierarchical navigable small world graphs. *Computing Research Repository (CoRR) abs/1904.02077* (2019).

[53] Peng-Cheng Lin and Wan-Lei Zhao. 2019. Graph based nearest neighbor search: Promises and failures. *arXiv preprint arXiv:1904.02077* (2019).

[54] Michele Linardi and Themis Palpanas. 2020. Scalable data series subsequence matching with ULISSE. *The VLDB Journal* 29, 6 (2020), 1449–1474.

[55] Battuguldur Lkhagva, Yu Suzuki, and Kyoji Kawagoe. 2006. New time series data representation ESAX for financial applications. In *22nd International Conference on Data Engineering Workshops (ICDEW'06)*. IEEE, x115–x115.

[56] Kejing Lu and Mineichi Kudo. 2020. R2LSH: A nearest neighbor search scheme based on two-dimensional projected spaces. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1045–1056.

[57] Kejing Lu, Mineichi Kudo, Chuan Xiao, and Yoshiharu Ishikawa. 2021. HVS: hierarchical graph structure based on voronoi diagrams for solving approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 15, 2 (2021), 246–258.

[58] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.

[59] Gloria Mainar-Ruiz and J Perez-Cortes. 2006. Approximate nearest neighbor search using a single space-filling curve and multiple representations of the data points. In *18th International Conference on Pattern Recognition (ICPR'06)*, Vol. 2. IEEE, 502–505.

[60] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[61] Mauricio Marin, Veronica Gil-Costa, and Roberto Uribe. 2008. Hybrid index for metric space databases. In *Computational Science–ICCS 2008: 8th International Conference, Kraków, Poland, June 23-25, 2008, Proceedings, Part I 8*. Springer, 327–336.

[62] Katsiaryna Mirylenka, Vassilis Christophides, Themis Palpanas, Ioannis Pefkianakis, and Martin May. 2016. Characterizing home device usage from wireless traffic time series. In *19th International Conference on Extending Database Technology (EDBT)*.

[63] Pravin Nair, Anmol Popli, and Kunal N Chaudhury. 2017. A fast approximation of the bilateral filter using the discrete Fourier transform. *Image Processing On Line* 7 (2017), 115–130.

[64] David Novak, Michal Batko, and Pavel Zezula. 2011. Metric index: An efficient and scalable solution for precise and approximate similarity search. *Information Systems* 36, 4 (2011), 721–733.

[65] Themis Palpanas. 2015. Data series management: The road to big sequence analytics. *ACM SIGMOD Record* 44, 2 (2015), 47–52.

[66] Themis Palpanas. 2020. Evolution of a Data Series Index: The iSAX Family of Data Series Indexes: iSAX, iSAX2. 0, iSAX2+, ADS, ADS+, ADS-Full, ParIS, ParIS+, MESSI, DPiSAX, ULISSE, Coconut-Trie/Tree, Coconut-LSM. In *Information Search, Integration, and Personalization: 13th International Workshop, ISIP 2019, Heraklion, Greece, May 9–10, 2019, Revised Selected Papers 13*. Springer, 68–83.

[67] Pavlos Paraskevopoulos, Thanh-Cong Dinh, Zolzaya Dashdorj, Themis Palpanas, Luciano Serafini, et al. 2013. Identification and characterization of human behavior patterns from mobile phone data. *D4D Challenge session, NetMob* (2013).

[68] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2020. Messi: In-memory data series indexing. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 337–348.

[69] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2020. Paris+: Data series indexing on multi-core architectures. *IEEE Transactions on Knowledge and Data Engineering* 33, 5 (2020), 2151–2164.

[70] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2021. SING: Sequence Indexing Using GPUs. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 1883–1888.

[71] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. 2012. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 262–270.

[72] Thanawin Rakthanmanon and Eamonn J Keogh. 2013. Data Mining a Trillion Time Series Subsequences Under Dynamic Time Warping.. In *IJCAI*. 3047–3051.

[73] Usman Raza, Alessandro Camerra, Amy L Murphy, Themis Palpanas, and Gian Pietro Picco. 2015. Practical data prediction for real-world wireless sensor networks. *IEEE Transactions on Knowledge and Data Engineering* 27, 8 (2015), 2231–2244.

[74] Hans Sagan. 1994. Space-Filling Curves. *Universitext* (1994).

[75] Patrick Schäfer and Mikael Högqvist. 2012. SFA: a symbolic fourier approximation and index for similarity search in high dimensional datasets. In *Proceedings of the 15th international conference on extending database technology*. 516–527.

[76] Dennis Shasha. 1999. Tuning time series queries in finance: Case studies and recommendations. *IEEE Data Eng. Bull.* 22, 2 (1999), 40–46.

[77] Jin Shieh and Eamonn Keogh. 2008. i SAX: indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 623–631.

[78] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. 2022. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990* (2022).

[79] Youqiang Sun, Jiuyong Li, Jixue Liu, Bingyu Sun, and Christopher Chow. 2014. An improvement of symbolic aggregate approximation distance measure for time series. *Neurocomputing* 138 (2014), 189–198.

[80] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin. 2014. SRS: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. *Proceedings of the VLDB Endowment* (2014).

[81] Chang Wei Tan, Geoffrey I Webb, and François Petitjean. 2017. Indexing and classifying gigabytes of time series under time warping. In *Proceedings of the 2017 SIAM international conference on data mining*. SIAM, 282–290.

[82] Antonio Torralba, Rob Fergus, and William T Freeman. 2008. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence* 30, 11 (2008), 1958–1970.

[83] Caetano Traina, Roberto F Santos Filho, Agma JM Traina, Marcos R Vieira, and Christos Faloutsos. 2007. The omni-family of all-purpose access methods: a simple and effective way to make similarity search more efficient. *The VLDB Journal* 16 (2007), 483–505.

[84] Eduardo Valle, Matthieu Cord, and Sylvie Philipp-Foliguet. 2008. High-dimensional descriptor indexing for large multimedia databases. In *Proceedings of the 17th ACM conference on Information and knowledge management*. 739–748.

[85] Reinier H Van Leuken and Remco C Veltkamp. 2011. Selecting vantage objects for similarity indexing. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 7, 3 (2011), 1–18.

[86] Jayendra Venkateswaran, Tamer Kahveci, Christopher Jermaine, and Deepak Lachwani. 2008. Reference-based indexing for metric spaces with costly distance measures. *The VLDB Journal* 17, 5 (2008), 1231–1251.

[87] Haiquan Wang. 2017. An APCA-enhanced compression method on large-scale time-series data. In *Proceedings of the ACM Turing 50th Celebration Conference-China*. 1–6.

[88] Yang Wang, Peng Wang, Jian Pei, Wei Wang, and Sheng Huang. 2013. A data-adaptive and dynamic segmentation index for whole matching on time series. *Proceedings of the VLDB Endowment* 6, 10 (2013), 793–804.

[89] Zeyu Wang, Qitong Wang, Peng Wang, Themis Palpanas, and Wei Wang. 2023. Dumpy: A compact and adaptive index for large data series collections. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–27.

[90] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, Vol. 98. 194–205.

[91] Dongtao Wei, Kaixiang Zhuang, Lei Ai, Qunlin Chen, Wenjing Yang, Wei Liu, Kangcheng Wang, Jiangzhou Sun, and Jiang Qiu. 2018. Structural and functional brain scans from the cross-sectional Southwest University adult lifespan dataset. *Scientific data* 5, 1 (2018), 1–10.

[92] Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. 2020. mT5: A massively multilingual pre-trained text-to-text transformer. *arXiv preprint arXiv:2010.11934* (2020).

[93] Djamel Edine Yagoubi, Reza Akbarinia, Florent Masseglia, and Themis Palpanas. 2017. Dpisax: Massively distributed partitioned isax. In *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1135–1140.

[94] Byoung-Kee Yi and Christos Faloutsos. 2000. Fast time sequence indexing for arbitrary Lp norms. (2000).

[95] Chaw Thet Zan and Hayato Yamana. 2016. An improved symbolic aggregate approximation distance measure based on its statistical features. In *Proceedings of the 18th international conference on information integration and web-based applications and services*. 72–80.

[96] Liang Zhang, Noura Alghamdi, Mohamed Y Eltabakh, and Elke A Rundensteiner. 2019. TARDIS: Distributed indexing framework for big time series data. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1202–1213.

[97] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. 2014. Indexing for interactive exploration of big data series. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1555–1566.

[98] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. 2016. ADS: the adaptive data series index. *The VLDB Journal* 25 (2016), 843–866.