MS-Index: Fast Top-k Subsequence Search for Multivariate Time Series under Euclidean Distance

Jens E. d'Hondt
j.e.d.hondt@tue.nl
Eindhoven University of Technology
Eindhoven, the Netherlands

Teun H. Kortekaas
teun@jelter.net
Eindhoven University of Technology
Eindhoven, the Netherlands

Odysseas Papapetrou
o.papapetrou@tue.nl
Eindhoven University of Technology
Eindhoven, the Netherlands

Themis Palpanas
themis@mi.parisdescartes.fr
Université Paris Cité & IUF
Paris, France

Abstract

Modern applications frequently collect and analyze temporal data in the form of multivariate time series (MTS) – time series that contain multiple channels. A common task in this context is subsequence search, which involves identifying all MTS that contain subsequences highly similar to a query time series. In practical scenarios, not all channels of an MTS are relevant to every query. For instance, airplane sensors may gather data on a plethora of components and subsystems, but only a few of these are relevant to a specific query, such as identifying the cause of a malfunctioning landing gear, or a specific flight maneuver. Consequently, the relevant query channels are often specified at query time. In this work, we introduce the *Multivariate Subsequence Index* (MS-Index), a novel algorithm for nearest neighbor MTS subsequence search under Euclidean distance that supports ad-hoc selection of query channels. The algorithm is *exact* and demonstrates query performance that scales sublinearly to the number of query channels. We examine the properties of MS-Index with a thorough experimental evaluation over 34 datasets, and show that it outperforms the state-of-the-art one to two orders of magnitude for both raw and normalized subsequences.

PVLDB Reference Format:
Jens E. d'Hondt, Teun H. Kortekaas, Odysseas Papapetrou, and Themis Palpanas. MS-Index: Fast Top-k Subsequence Search for Multivariate Time Series under Euclidean Distance. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/JdHondt/MS-Index.

1 Introduction

Time series are ubiquitous in diverse domains, such as astrophysics, seismology, meteorology, health care, finance, video and audio recordings [5, 37, 39, 42, 58]. Due to advances in sensor technology, the amount of time series being collected has increased dramatically over the last years, particularly in the form of multivariate time series (MTS) [63]. Each MTS is a collection of time series, often sourced from different sensors that measure different aspects of the same phenomenon or object. These different time series are referred to as the *channels* of the MTS. Examples of MTS include health monitoring data of patients in a hospital (e.g., heart rate, blood pressure, and body temperature), climate sensor arrays (e.g., an array measuring the temperature, humidity, and air pressure, at a certain location) [45], or motion capture data (e.g., the position and acceleration of different body parts).

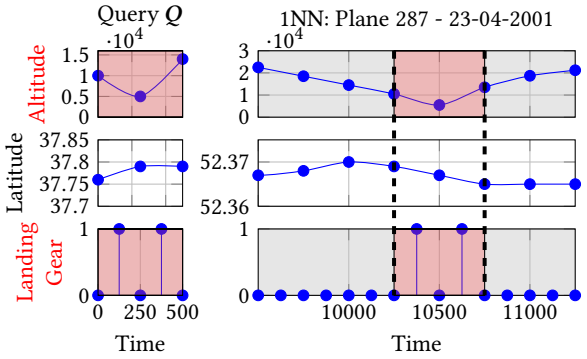

Figure 1: Example query and 1NN for MTS of synthetic airplane data. Altitude and landing gear are the query channels. The highlighted boxes (red) are the considered subsequences.

A key operation on time series is similarity search, which involves finding the most similar time series to a query time series (also known as its *Nearest Neighbors*) [26, 27]. Similarity search can be performed as a standalone task [16, 22, 29, 41, 43, 46, 60, 61, 67], but is also frequently used as a subroutine in tasks such as outlier detection [14, 22, 71], classification [8, 35, 64, 68, 69, 77], and clustering [9, 13, 24, 38, 65].

To get the most out of the ever-growing datasets, modern-day similarity search algorithms should support queries with high flexibility. For example, when analyzing patient data, a doctor might need to find similar occurrences of a short-term pattern in a patient's vital signs, to understand the root cause of their condition. Such a query would require the algorithm to support (a) MTS (multiple sensors), (b) comparison of subsequences rather than whole time series, and (c) deciding the query channels at query time (only

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

the sensors relating to the patient's relevant vital signs). Another example requiring such queries is the analysis of sensor data from airplanes [12], where the user is analyzing the failed landing of an airplane due to a malfunctioning landing gear, and wants to find similar occurrences in historical data to understand the root cause. In this case, the user might select the period of time of the failed landing (i.e., the left time series in Figure 1) as well as the relevant channels (highlighted red in the figure) to find similar occurrences in historical data.

While much work has been done to address the challenges that come with similarity search on large datasets of univariate time series (UTS), the work on MTS search is still rudimentary. Current solutions only support searching for whole time series (whole-matching) instead of subsequences (subsequence search), and only on a fixed, pre-determined set of channels. In this work, we show that the performance of whole-matching state-of-the-art solutions suffers when these are extended for subsequences. A further extension of these solutions to handle multivariate data only exacerbates the problem, due to the additional challenges that come with the number of channels. In summary, existing algorithms fall short in at least one of the following ways: (a) they natively support only UTS, and their extension to MTS is non-trivial, (b) they are focused on whole matching, and their performance becomes unacceptable when inserting subsequences, or, (c) they rely on severely restrictive assumptions of the user query, such as user-defined thresholds on each channel.

In this work, we propose *MS-Index*, a novel algorithm for k-nearest neighbor (k-NN) subsequence search on MTS under Euclidean distance. MS-Index allows for selection of the query channels at query time, and works for both normalized and un-normalized subsequences.[1] MS-Index supports *fixed-length queries*, i.e., it requires the length of the query to be known at index construction. The algorithm is *exact*: it always returns the correct and complete result.

MS-Index differs from existing methods in three ways. First, it is designed specifically for MTS, leveraging the additional pruning potential that comes with multiple channels. Second, it combines index-powered pruning of the search space with efficient distance computation on the remaining candidates through the convolution theorem (MASS) [2, 51]. This is done with the help of an R-tree that indexes Discrete Fourier Transform (DFT) approximated subsequences *on all channels*, and a two-pass search algorithm that prunes candidates based on the lower-bound distance to the query. To the best of our knowledge, this is the first time an index has been combined with MASS for subsequence search; previous works have only used each of these techniques in isolation [29, 51], arguing for either an index-based or a sequential-scan approach. Third, it utilizes two novel methods to tighten the lower bound distance to the query, further improving the pruning potential of the search. These methods are *generic*, meaning that they can be applied to any search solution that uses R-trees and/or DFT approximations. Our key contributions are as follows:

- We introduce MS-Index, a novel algorithm for k-NN subsequence search on MTS under Euclidean distance (Section 3).
- We propose a general set of optimizations to improve any search solution using R-trees and/or DFT approximations (Section 3.4). These are shown to improve the performance of MS-Index by a factor of 4, when combined.
- We provide a general approach to extending current solutions for UTS to the multivariate case, which provide baselines for our evaluation (Section 4).
- We conduct a thorough evaluation of MS-Index across 34 datasets, comparing it to a range of baselines, and showing that MS-Index outperforms the state-of-the-art by two orders of magnitude for both raw and normalized subsequences (Section 5).

2 Preliminaries

2.1 Definitions and Notation

Time series. A UTS $\boldsymbol{t}$ of length m is denoted as $\boldsymbol{t} = [\boldsymbol{t}_1, \ldots, \boldsymbol{t}_m]$, where $\boldsymbol{t}_i$ is the data point at time i. An MTS $\boldsymbol{T}$ with c channels is denoted as a matrix $\boldsymbol{T} = [\boldsymbol{t}_1, \ldots, \boldsymbol{t}_c]^T$, where $\boldsymbol{t}_i$ is the UTS of channel i. A subsequence of length s starting at timepoint i is denoted as $\boldsymbol{T}_i^s = \boldsymbol{T}_{*,i:i+s-1}$. The number of available channels in the dataset is denoted as c, and the set of indices to the query channels is denoted as $c_{\boldsymbol{Q}}$, with $c_{\boldsymbol{Q}} \subseteq \{1, \ldots, c\}$.

Similarity Search Queries. There are two forms of similarity search queries; a k-Nearest-Neighbor (k-NN) query and an r-range query (also called threshold query). A k-NN query is defined as finding the k time series in a dataset $\mathcal{D}$ with the smallest distances to a query time series $\boldsymbol{Q}$, under a given distance measure d [26]. In the case of MTS, all time series in $\mathcal{D}$ contain the same c channels. However, the query time series $\boldsymbol{Q}$ may contain only a subset of these channels $c_{\boldsymbol{Q}}$. Conversely, an r-range query is defined as finding all time series in $\mathcal{D}$ with a distance to a query time series $\boldsymbol{Q}$ smaller than a given threshold r [26]. These query types can be further categorized into whole-matching and subsequence search queries. In *whole-matching*, we consider the distance between an entire query time series and an entire candidate series [26]. All the series involved in the search need to be of the same length. In *subsequence search*, we consider the distance between an entire query series and all subsequences of a candidate series with the same length as the query [26]. In this case, the candidate series from which the subsequences are extracted need not have the same length. There exist two variants of subsequence search; (a) *fixed-length* search, where the query length $|\boldsymbol{Q}|$ is predetermined and fixed across queries, and (b) *variable-length* search, where the query length is not fixed and can vary between queries [26]. In this work, we focus on the problem of fixed-length subsequence search, which is a common assumption in the literature [12, 17, 29, 72, 73]. Variable-length search is left for future work.

Distance. In line with previous studies on MTS similarity search [8, 12, 72], we use Euclidean distance (ED) to measure the distance between time series, which is defined over MTS as:

$$d(\boldsymbol{X}, \boldsymbol{Y}) = ||\boldsymbol{X} - \boldsymbol{Y}||_2 = \sqrt{\sum_{i \in c_{XY}} \sum_{j=1}^{m} (\boldsymbol{X}_{i,j} - \boldsymbol{Y}_{i,j})^2} \qquad (1)$$

[1] Supporting normalized subsequences is more challenging than normalized time series, as it cannot be done through preprocessing the data. Furthermore, while most applications require normalized subsequences for shape matching, some applications require querying for raw subsequences to preserve scale differences [7, 54].

Table 1: Nomenclature

$\boldsymbol{t}$	Univariate time series
$\boldsymbol{T}$	Multivariate time series
$\lvert\boldsymbol{t}\rvert$	Length of time series $\boldsymbol{T}$
$\boldsymbol{T}_i$	Channel i of $\boldsymbol{T}$
$\boldsymbol{T}_{i,j}$	Data point j of channel i of $\boldsymbol{T}$
$\boldsymbol{T}_i^s$	Subsequence of $\boldsymbol{T}$ of length s starting at index i
$\tilde{\boldsymbol{T}}$	DFT approximation of $\boldsymbol{T}$
$\tilde{\boldsymbol{T}}'$	Flattened DFT approximation of $\boldsymbol{T}$ (i.e. feature vector)
$\boldsymbol{Q}$	Query time series
c	Total number of channels in the dataset
c_Q	The set of channel ids of $\boldsymbol{Q}$
n	Number of time series in the dataset $\mathcal{D}$
m	Length of the time series in $\mathcal{D}$

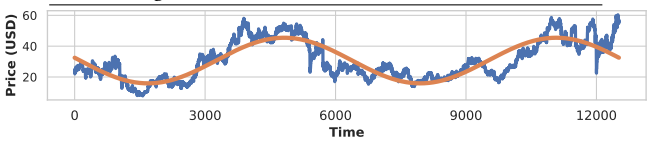

Figure 2: The price of a stock over time (blue), and its reconstruction through its first three DFT coefficients (orange).

with $c_{XY} = c_X \cap c_Y$ the common channels of $\boldsymbol{X}$ and $\boldsymbol{Y}$, and $m = \min(m_X, m_Y)$ the length of the shortest time series. ED was chosen due to its simplicity, efficiency, well-understood properties for univariate data, its natural extension to the multivariate case, and its popularity in the literature [8, 12, 23, 72]. In fact, while temporal alignment through more complex measures like Dynamic Time Warping (DTW) and Shape-Based Distance (SBD) has proven effective for whole-matching in both univariate and multivariate contexts [23, 25, 56, 57, 65, 68], the marginal gain of using these measures over ED for subsequence search is negligible [23, 62, 67, 75]. This is because the consideration of all subsequences in time series is effectively a form of temporal alignment, comparing the query with candidate time series under different shifts, naturally correcting for temporal distortions. Table 1 summarizes the notations used throughout the paper.

2.2 DFT Approximation

A key challenge in similarity search is the high dimensionality of the data, sourcing from the length of the time series. A common approach to address this challenge is the use of dimensionality reduction, or "summarization", techniques that transform the data into a lower-dimensional space where distances can be approximated at a low cost. A popular summarization technique for Euclidean distance (L_2) is the DFT approximation, which involves performing a DFT on the time series and keeping only a small fraction of the resulting vector [29, 50, 60, 70]. The DFT decomposes a time series into a sum of sinusoids of different frequencies (also called the *coefficients*), where the amplitudes of the sinusoids represent the importance of the corresponding frequency in the time series (also called the *energy* of the coefficient). For most real-world time series, the amplitudes of the high-frequency sinusoids are small, which means that these sinusoids can be discarded without losing much information. This means that we can accurately approximate a time series using only the first f values of the DFT, resulting in a vector of size f instead of the original size m [29]. The accuracy of such an approximation is demonstrated with an example from the *Stocks* dataset (cf. Section 5) in Figure 2. DFT approximation is popular for estimating the Euclidean distance, as the distance between two DFT-approximated time series is a lower bound on their true Euclidean distance [50]. Namely, the Euclidean distance between two time series $\boldsymbol{t}, \boldsymbol{Q} \in \mathbb{R}^m$ is bounded by their DFT approximations $\tilde{\boldsymbol{t}}, \tilde{\boldsymbol{Q}} \in \mathbb{C}^f$ as [50]:

$$d(\boldsymbol{t}, \boldsymbol{Q}) \geq \sqrt{\frac{\sum_{i=1}^{f} |\tilde{\boldsymbol{t}}_i - \tilde{\boldsymbol{Q}}_i|^2}{m}} = \frac{d(\tilde{\boldsymbol{t}}, \tilde{\boldsymbol{Q}})}{\sqrt{m}} \tag{2}$$

As the value of f increases (i.e., we account for more coefficients), the DFT bound converges to the exact distance [50]. This way, DFT approximation allows for a trade-off between the accuracy of the distance estimation and the dimensionality of the data, which is particularly useful for high-dimensional data such as time series. In the case of MTS, the same concept can be used by summarizing each individual channel, independently.

Table 2: Overview of related work

		Name	Ref	Type	Notes
UTS	Whole	VA+ file	[31]	Index	
		ISAX2+	[17]	Index	
		DSTree	[73]	Index	
	Sub.	ST-index	[29]	Index	
		KV-match	[74]	Sequential	Custom definition of normalized subsequences
		MASS	[51]	Sequential	
MTS	Whole	Vlachos	[72]	Index	Does not support normalized subsequences
	Sub.	Bhaduri	[12]	Index	Only supports r-range queries
		MS-Index (ours)		Index	

2.3 R-tree Construction Techniques

An R-tree is a tree-based index for spatial data that groups nearby objects into Minimum Bounding Rectangles (MBRs), which are recursively grouped into larger MBRs [10, 34]. R-trees can be constructed top-down by splitting a single MBR containing all data points into smaller MBRs using strategies like Quadratic Split [34], Linear Split [34], or R*-tree topological split [10]. Alternatively, bottom-up construction (or *bulk loading*) starts with individual data points as MBRs and merges them into larger MBRs based on a *leaf size* L and partitioning strategy, generally resulting in less overlap but requiring all data to be known in advance. A popular bulk loading algorithm is the *Sort-Tile-Recursive* (STR) algorithm [44]. STR first sorts the children based on their coordinates in each dimension (using the middle in case the children are rectangles instead of points). Then, it recursively partitions the entries into groups of size $\lceil \frac{N}{L} \rceil^{1/d}$, where N is the number of entries to index, L is the desired leaf size, and d is the dimensionality of the tree. For example, given a 1D space with entries $[1, 2, \ldots, 10]$ and $L = 2$, the algorithm partitions the entries into $[[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]]$, to subsequently create the nodes $[[1, \ldots, 4], [5, \ldots 8], [9, 10]]$, etc.

2.4 Related Work

We first discuss related work on UTS, followed by recent work on MTS. An overview of all related work is given in Table 2.

Univariate Time Series (UTS) whole-matching. In a large-scale comparison of algorithms for UTS whole-matching by Echihabi,

et al. [26], index-based algorithms generally showed to outperform their sequential scan counterparts. Furthermore, for small in-memory datasets, VA+ file [31] and iSAX2+ [17] excel, while DSTree [73] dominates for larger datasets. DSTree uses Extended Adaptive Piecewise Constant Approximation (EAPCA) to estimate distances between query and groups of time series, differentiating them by mean and variance at increasing resolution. Whole-matching indices could be extended to support subsequence search by indexing each subsequence individually. However, this approach would overwhelm the indices when $|\boldsymbol{Q}| \ll |\boldsymbol{T}|$, degrading performance as verified in Section 5.

UTS subsequence search indices. Multiple indices have been proposed for UTS subsequence search. The ST-index [29] for r-range queries first extracts a DFT approximation of each subsequence of length $|\boldsymbol{Q}|$ in a time series. These f-dimensional vectors then form a trail in the f-dimensional space, which is segmented into sub-trails using MBRs. The MBRs are then indexed in an R*-tree [10], to enable efficient search. More recents improvements through Dual Match and General Match [32, 48] used different window types to reduce index size. However, these extensions inherently restrict the indices to raw subsequences, as supporting normalized subsequences would require a complete redesign. Our work adopts the idea of indexing DFT approximations in an R-tree but (a) focuses on MTS, (b) removes explicit time series segmentation, (c) leverages convolution theorem for faster distance computation, and (d) employs a different search algorithm to support k-NN queries with *ad-hoc* selection of query channels. Another notable index is KV-match [74], which indexes subsequences within a single time series using the means and variances of disjoint windows over the time series. The index supports both normalized and raw subsequences, though for normalized subsequences it restricts the search space to subsequences with similar means and variances to the query *before normalization*. KV-Match can be adapted to our problem setting by (a) building one KV-match per time series and iteratively querying each, and (b) dropping the filter on means and variances before querying normalized subsequences.

Other UTS subsequence indices include L-match [30] and TS-index [19], which we do not consider further because: L-match improves KV-match's indexing time but has higher query time; and TS-index uses Chebyshev distance rather than our Euclidean distance. As we show in Section 5, our work outperforms KV-match, which (by transivity) also suggests how our work is expected to relate to L-match.

UTS subsequence search sequential scans. Mueen's Algorithm for Similarity Search (MASS) [51] is an exact subsequence search algorithm that computes the distance between a query $\boldsymbol{Q}$ and all subsequences of a time series $\boldsymbol{t}$ in time $O(|\boldsymbol{t}| \log |\boldsymbol{t}|)$ rather than the exhaustive $O(|\boldsymbol{t}||\boldsymbol{Q}|)$ time. It does so through the convolution theorem, which states that the cross-correlation (i.e., sliding dot product) of two time series is equivalent to the point-wise multiplication of their Fourier transforms [2]. Namely, the convolution theorem states that the dot-products $\langle \cdot, \cdot \rangle$ between $\boldsymbol{Q}$ and all subsequences of $\boldsymbol{t}$ of length $|\boldsymbol{Q}|$ can be computed through:

$$[\langle \boldsymbol{Q}, \boldsymbol{t}_1^{|\boldsymbol{Q}|} \rangle, \ldots, \langle \boldsymbol{Q}, \boldsymbol{t}_{|\boldsymbol{t}|-|\boldsymbol{Q}|+1}^{|\boldsymbol{Q}|} \rangle] = \mathcal{F}^{-1}(\mathcal{F}(\boldsymbol{Q}) \otimes \mathcal{F}(\boldsymbol{t})) \quad (3)$$

where $\mathcal{F}$, $\mathcal{F}^{-1}$, and $\otimes$ are the Fourier transform, inverse Fourier transform, and point-wise multiplication, respectively. Then, as the Euclidean distance between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined as $d(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{||\boldsymbol{x}||^2 + ||\boldsymbol{y}||^2 - 2\langle \boldsymbol{x}, \boldsymbol{y} \rangle}$, MASS computes the distance between $\boldsymbol{Q}$ and all subsequences of $\boldsymbol{t}$ in time $O(|\boldsymbol{t}| \log |\boldsymbol{t}|)$ using Equation 3 and the sliding squared sums of $\boldsymbol{Q}$ and $\boldsymbol{t}$. MASS can be used as a subsequence search algorithm for MTS, by repeating the distance computation for each channel separately and summing the results to obtain the multivariate Euclidean distance. We will be using MASS in this work, both as a baseline and as a component of our method.

MTS whole-matching. The first index for MTS whole-matching was proposed by Vlachos et al. [72], focusing on r-range queries under DTW and Longest Common Subsequence (LCSS) distance, besides Euclidean distance. It works by splitting the MTS into MBRs that span across the time, channel, and value axes, and storing those in an R*-tree [10]. Then, at query time, a *Minimum Bounding Envelope* (MBE) is constructed for the query, which covers all the possible matching areas of the query under warping conditions. This MBE is further decomposed into MBRs and probed in the R*-tree to efficiently find the candidates. However, using the method for subsequence search under Euclidean distance essentially reduces to a simplified variant of Dual Match [32], which – as discussed earlier – prevents the index from supporting normalized subsequences, making it unsuitable for our problem setting.

MTS subsequence search. Recently, a novel algorithm for *variable-length* MTS subsequence search was proposed named MULISSE [59], as a multivariate extension of the state-of-the-art UTS subsequence search algorithm ULISSE [46]. The method extends iSAX2+ [17] representations to multiple channels and introduces channel-aware node splitting for improved pruning. To accommodate for the vast search space of variable-length subsequence search, the ULISSE index (and thus MULISSE) uses a lightweight design that sacrifices pruning power for a reduced index size. MULISSE can be applied to fixed-length search by restricting the query length range to a single value, as we show in Section 5.

To the best of our knowledge, the only solution for *fixed-length* MTS subsequence search is by Bhaduri, et al [12]. The method answers *r-range* queries under Euclidean distance by indexing subsequences per channel based on distances to univariate reference points. However, its reliance on channel-level thresholds prevents support for k-NN queries, which require simultaneous querying of all channels. Adapting the method to our problem setting would require non-trivial modifications likely to degrade its performance, so we do not consider it further in our work.

Summary. Concluding, literature includes a multitude of similarity search solutions for UTS and MTS data under different query types and solution designs. However, only few of the existing solutions can be directly applied/extended to our problem setting. Namely, all MTS solutions either (a) do not support k-NN queries [12] or querying on normalized subsequences [72], or (b) optimize for index size rather than query performance (MULISSE [59]). There are UTS solutions that could be extended to MTS, but those extensions are not expected to scale well. This is either because (a) they are not build for subsequence search (*DSTree*), (b) they are

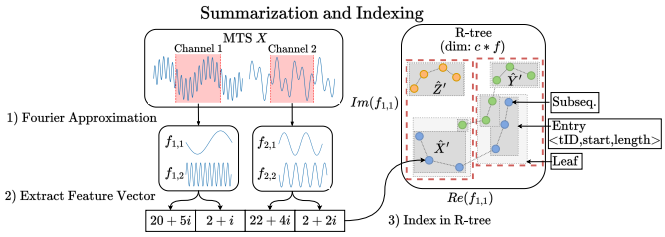

Figure 3: Summarization and indexing of MTS subsequences. We represent the indexed subsequences of three MTS in the feature space with blue, green, and orange nodes. Time-neighbouring subsequences are connected with lines.

sequential scan algorithms (*MASS*), or (c) they are simply designed for UTS (*ST-index*).

3 MS-Index for Subsequence Search on MTS

MTS subsequence search involves two key challenges. First, the number of subsequences in the dataset grows rapidly, even for fairly small datasets. For a dataset with n time series, each of length m, an exhaustive search would require computing the Euclidean distance between all $n * (m - |Q| + 1)$ subsequences and the query Q, with each computation costing $|c_Q| * |Q|$ time. Second, the requirement to support ad-hoc selection of query channels precludes the use of summarizations that involve merging channels, such as PCA [77].

Our solution, named MS-Index, addresses these challenges by combining the pruning potential of an index with the efficiency of the MASS algorithm for exact distance computation (cf. Section 2.4). The solution involves three key steps (cf. Figure 3): (a) summarizing the MTS subsequences through DFT approximations, (b) building an R-tree index on these approximations, [2] and (c) at query time, using the index to heavily prune the candidate subsequences, followed by an exact distance computation on the remaining candidates using MASS. Through the index, MS-Index is able to prune over 99% of the candidate subsequences for a variety of datasets, at a very low cost, thereby achieving a speedup up to 100x compared to the state-of-the-art. We also propose a number of optimizations to further improve the efficiency of the search process (Section 3.4). These novel optimizations are interesting in their own right, as they can be applied to any search algorithm that uses DFT approximations and/or spatial indices, such as [21, 29, 31, 50, 60, 72].

We now present the key components of MS-Index, starting with time series summarization (Section 3.1) and indexing (Section 3.2), followed by the query execution algorithm (Section 3.3), and the optimizations (Section 3.4).

3.1 Summarizing All Subsequences with DFTs

As discussed in Section 2.2, a big challenge in working with time series is their length. Using a subset of DFT coefficients to approximate time series has been extensively used in the literature [29, 31, 50]. This approach is effective because it provides a compact representation that enables efficient lower-bounding of distances between time series. However, our analysis of real-world time series revealed two important observations that can further improve the accuracy of such approximations;

Observation 1: The *first-f* coefficients are not always the *top-f* coefficients in terms of their contribution to the total energy of the time series; there exist domains where certain high-frequency contributions are also substantial. This is illustrated in the two plots in Figure 4a, which show the absolute and cumulative contribution to the total energy and distance across DFT coefficients, extracted from 100 time series with real-world temperature readings at different locations (see Section 5 for details on the dataset). Both lines show a 90% confidence interval over a wide range of measurements (i.e., all 100 time series to form the line for energy, and all 100*99/2 pairwise distances to form the line for total distance). The figure shows a sudden jump in the energy contribution at the 62nd coefficient, which indicates that this coefficient has a substantial contribution to the shape of the time series and should thus be included in the approximation. The top-f most significant coefficients can be derived similarly to the configuration process in other adaptive summarization techniques [33, 55, 76]; by computing statistics on a sample of the dataset pre-index construction.

Observation 2: The contribution of DFT coefficients to the total *distance* between time series is even more significant than their contribution to energy. Intuitively, this means that the top coefficients are not only larger in scale, but also vary much more between time series. This evident from Figure 4a, which shows that the cumulative contribution to distance increases much faster than the contribution to energy, implying that one can already cover ~90% of the total distance with the top 5 coefficients, even though these only cover roughly ~60% of the total energy.

MS-Index exploits these observations to create accurate approximations of time series subsequences, adapted to the dataset at hand. In particular, we summarize the MTS in the dataset as follows. First, before index construction, we extract a uniformly random sample S of subsequences of length $|Q|$ from different time series in the dataset[3], and derive the *average relative distance contribution* (ARDC) of each DFT coefficient and each channel to the Euclidean distance between these subsequences (i.e., the process that generated the plots in Figure 4a). Then, for each channel in the dataset, the top-f coefficients with the largest ARDC are derived, with f chosen such that the total ARDC of the top-f coefficients is above a certain threshold d_{target} (e.g., 90%). Parameterizing the algorithm on d_{target} rather than f allows the algorithm to adapt the size of the summarization to the dataset at hand, ensuring a predetermined level of accuracy of the approximations. Those top-f coefficients are then used to compute the DFT approximations of *all* $|Q|$-length subsequences in the dataset. Notice that the summation in Equation 2 can be over any permutation or subset of coefficients; as long as the same coefficients are used for both subsequences the bound still holds.

In the following, with $\tilde{T}$ we will denote the matrix of size $c \times f$ that stores the DFT approximations for subsequence T. Also, $\tilde{T}'$

[2] We would like to stress that, despite the use of DFTs, which is an approximation technique, MS-Index remains an *exact* algorithm. Approximation only influences the bounding efficiency and pruning power (i.e., the speed) of MS-Index, and not the accuracy or completeness of its results. Specifically, after applying dimensionality reduction, the distances between objects in the R-tree are lower-bounds on the actual distances. This potentially leads to false positives, but never false negatives as we query the R-tree with a threshold that is an upper bound on the *actual* k-NN distance (i.e., without dimensionality reduction). We formally prove this in Lemma 3.1.

[3] The sample size is a parameter of the algorithm, and is typically set to 100 subsequences. This value was chosen after empirical testing on different datasets, which showed that larger sample sizes did not lead to a noticeable difference in performance.

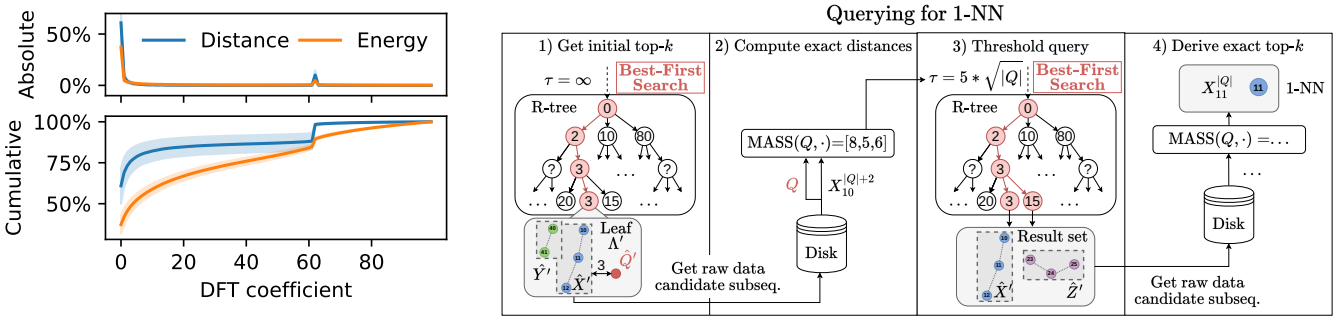

Figure 4: (a) Cumulative and absolute % of total distance and energy across DFT coefficients on the temperature channel of a weather dataset; (b) Query execution in MS-Index. Numbers in tree nodes indicate the lower bound distance of the respective MBR to the query.

will be used to denote the *feature vector* of subsequence $\boldsymbol{T}$, which is of length $c * f$ and is constructed by flattening the matrix $\tilde{\boldsymbol{T}}$ (step 2 in Figure 3). Summarizing a dataset of n time series costs $O(n * m * |\boldsymbol{Q}| * f)$, and leads to a total of $O(n * m)$ feature vectors – one per subsequence.

3.2 Indexing of the MTS

The next step involves indexing the generated feature vectors in an R-tree (step 3 in Figure 3).[4] Notice, however, that the total number of feature vectors is $O(n * m)$, which can lead to a large and inefficient R-tree when the number of time series n is large, negatively impacting the query performance.

We mitigate this problem as follows. First, we build the R-tree in a bottom-up fashion with leaves containing more than one entry (i.e., leaf size $L > 1$), to reduce the number of nodes in the tree. Second, once the leaves are constructed, we revisit them and modify their internal representation, to allow pruning of multiple subsequences (i.e., entries) with a single distance computation. Particularly, for each leaf that contains two or more entries, we group together all entries that originate from *time-neighbouring subsequences* of the same MTS, i.e., feature vectors corresponding to a series of subsequences that are a single time shift apart (e.g., $\boldsymbol{T}_i^{|\boldsymbol{Q}|}$, $\boldsymbol{T}_{i+1}^{|\boldsymbol{Q}|}$, $\boldsymbol{T}_{i+2}^{|\boldsymbol{Q}|}$, …). These groups correspond to a continuous subsequence of the original MTS. For example, assume a leaf Λ that includes the following subsequences: $\Lambda = \{\boldsymbol{X}_{10}^{|\boldsymbol{Q}|}, \boldsymbol{X}_{11}^{|\boldsymbol{Q}|}, \boldsymbol{X}_{12}^{|\boldsymbol{Q}|}, \boldsymbol{Y}_{40}^{|\boldsymbol{Q}|}, \boldsymbol{Y}_{41}^{|\boldsymbol{Q}|}\}$, with X and Y being two different MTS. We compress these five entries into two, representing the relatively larger subsequences $\Lambda' = \{\boldsymbol{X}_{10}^{|\boldsymbol{Q}|+2}, \boldsymbol{Y}_{3}^{|\boldsymbol{Q}|+1}\}$. These entries are now represented with Minimal Bounding Rectangles (MBR) rather than with single points in the feature space, with the start and end positions of the continuous subsequence stored in the entry along with the MBR.

The intuition behind this grouping is that time-neighboring subsequences are typically very similar due to their large overlap [29]. This means that their feature vectors are also similar, often ending up in the same R-tree leaf. This allows many subsequences to be represented by compact entries with tight MBRs, which both reduces the number of entries in the R-tree and enables efficient distance computation with MASS. In our experiments, this compression typically merges 8-50 subsequences into a single entry, depending on the dataset, with a leaf size optimized for query performance.

Consequently, the indexing process of MS-Index is as follows. (a) We build an R-tree bottom-up on the feature vectors of all subsequences in the dataset. In our running example with X and Y, this step involves creating the leaf node Λ along with all other leafs that contain the other subsequences in the dataset. In the recursion of the R-tree, we then create the parent nodes of Λ, which contain the MBRs of the leafs (and are MBRs themselves), and so on until we reach the root node. (b) We then revisit the leaf nodes and group together all time-neighbouring entries (i.e., creating Λ') to reduce the index size. After these steps, we end up with an R-tree with each entry storing (a) an MBR covering the indexed feature vectors, (b) the start and end positions of the subsequences in the original MTS, and (c) the MTS identifier. This index is visualized on the right of Figure 3, where the nodes represent the indexed subsequences of different MTS (differentiated by color), and the edges connect time-neighbouring subsequences of the same MTS.

3.3 Query Execution

Our algorithm probes the index twice. The first probe is for finding a rough estimate (an upper bound) of the distance to the k-th nearest neighbor, whereas the second probe retrieves all entries within this distance, and computes the exact result. The second probe is necessary to guarantee correctness of the answer.

Queries are executed as follows. First, we extract a feature vector $\tilde{\boldsymbol{Q}}'$ of the query $\boldsymbol{Q}$. Then, starting with a distance threshold $\tau_k = \infty$, we perform a best-first search on the R-tree, to find the k entries (i.e., groups of $|\boldsymbol{Q}|$-length subsequences) with the smallest distance to $\tilde{\boldsymbol{Q}}'$. Since the R-tree relies on the feature vectors to compute all distances, the computed distances are lower bounds (LBs) on the true distances to $\boldsymbol{Q}$. Then, for each retrieved entry, we compute the exact distances between $\boldsymbol{Q}$ and the $|\boldsymbol{Q}|$-length subsequences in the entry using MASS, and set the distance threshold τ_k to the k-th smallest distance found in this step. Finally, we perform a threshold query on the index, using $\tau_k * \sqrt{|\boldsymbol{Q}|}$ as the distance threshold, and compute the exact distances for these entries with MASS to obtain

[4] We also experimented with other spatial indexes like KD-trees [11], but found that the recall stage of our algorithm was the most time-consuming part of our solution, rather than the probing of the index. We thus opted for using an R-tree to enable a more clear comparison with ST-index [29].

the final k-NN.[5] These results are then returned to the user along with the corresponding timestamps of the subsequences, which are derived from the offset and the timestamp of the MTS. Since the query Q does not necessarily contain all channels (i.e., $c_Q \subseteq \{1, \ldots, c\}$), the R-tree is only queried on the dimensions of the feature space that correspond to the channels in Q. This is natively handled by the R-tree algorithm [66].

To illustrate this process, consider a query Q for which the 1-NN is the subsequence $X_{11}^{|Q|}$ in leaf Λ' from the previous section. The algorithm will return the entry $X_{10}^{|Q|+2}$ in the first probe, as it has the smallest lower bounding distance to Q (Step 1 in Figure 4b). It then computes MASS$(Q, X_{10}^{|Q|+2})$ on the original subsequences, which outputs the distances $[8, 5, 6]$, and adds $X_{11}^{|Q|}$ to the running 1-NN set as it corresponds to the smallest distance of 5 (Step 2). Accordingly, the threshold is set to $5 * \sqrt{|Q|}$ and the algorithm performs the second probe, retrieving the entries $Z_{23}^{|Q|+2}$ and $X_{10}^{|Q|+2}$ (Step 3). After computing the exact distances for these entries with MASS, the algorithm returns $X_{11}^{|Q|}$ as the 1-NN to Q (Step 4).

We now prove that the query process described above is guaranteed to return the complete and correct k-NN to the query Q. Specifically, we aim to prove the following lemma:

Lemma 3.1. *Any indexed subsequence T of length $|Q|$ that is part of the k-NN of Q is guaranteed to be in the set of subsequences returned by MS-Index.*

Proof. The first probe of the index returns k subsequences of length $s \geq |Q|$, for which the exact distances with the query are computed. As these k subsequences collectively contain at least k subsequences of length $|Q|$, by setting τ_k as the k'th smallest of these distances we get an upper bound on the distance of the true k-th nearest neighbor. In the second probe, we perform a range query with threshold $\tau_k * \sqrt{|Q|}$ on the R-tree index, which contains the feature vectors. Based on Equation 2, the distance between any two points in the feature space is a lower bound on the true distance of the subsequences they represent. Specifically, the distance between a $|Q|$-length subsequence T and the query Q is lower-bounded by the distance between their feature vectors $\tilde{T}'$ and $\tilde{Q}'$ as:

$$d(T, Q) \geq \frac{1}{\sqrt{|Q|}} \sqrt{\sum_{i \in c_Q} d(\tilde{T}_i, \tilde{Q}_i)^2} = \frac{d(\tilde{T}'_{c_Q}, \tilde{Q}'_{c_Q})}{\sqrt{|Q|}} \tag{4}$$

where $\tilde{T}'_{c_Q}$ and $\tilde{Q}'_{c_Q}$ correspond to the feature vectors of T and Q limited to the channels of Q. Now, assume that an indexed subsequence T has distance with Q less than τ_k. Then, by Equation 4 we know that $\frac{d(\tilde{T}_{c_Q}, \tilde{Q}'_{c_Q})}{\sqrt{|Q|}} \leq d(T, Q) \leq \tau_k$. Therefore, by querying an R-tree a threshold $\tau_k * \sqrt{|Q|}$, T is guaranteed to be included in the returned set.[6] □

[5]Note that MASS is not executed on the feature vectors, but rather on the original subsequences, to ensure that the distances are exact. This data is retrieved by chasing the pointers stored in the R-tree entries to the original MTS, residing on disk.

[6]Note that the $\sqrt{|Q|}$ is simply a scaling factor that needs to be accounted when transforming the distances from the feature space to the time domain. It does not weaken the bounds.

The lemma proves that MS-Index is complete by guaranteeing that all subsequences in the k-NN are found. Since exact distances are computed for all candidates using MASS (which is exact), and the k-NN is derived from these distances, the algorithm is also correct, never returning false positives.

3.4 Optimizations

We now present three optimizations of MS-Index aimed at improving the pruning power of the index and the efficiency of the search process. These optimizations do not affect the correctness of the algorithm or the final result – *the algorithm still remains exact.*

Tightening the DFT bounds. As discussed in Section 3.1, in most real-world datasets, around 60%-80% of the distance between time series is covered by the top 2-5 DFT coefficients. Formally, this concept is expressed as:

$$d^2(T, Q) = (\underbrace{d^2(\tilde{T}_f, \tilde{Q}_f)}_{\sim 80\%} + \underbrace{d^2(\tilde{T}_{f+}, \tilde{Q}_{f+})}_{\sim 20\%})/|Q| \tag{5}$$

where $\tilde{T}_f$ denotes the DFT approximation of an MTS T using the top-f coefficients, and $\tilde{T}_{f+}$ denotes the set of c vectors composed of the remaining coefficients. In the DFT bound of Equation 4, we effectively throw away the second term, which leads to a lower bound of the true distance. This bound can be computed very efficiently (in $O(f)$), but it may also lead to a weaker pruning in the R-tree, and to unnecessary distance computations for subsequences that are outside the final k-NN.

We improve this lower bound such that it approaches the true distance more closely, by introducing a correction term at the distance calculation that approximates (again by lower-bounding) the distance over the remaining $(|Q| - f)$ coefficients, *without actually having to compute these coefficients.* Computing this correction term costs $O(1)$ at query time for each probed R-tree node, and improves the pruning power of the index.

Namely, during index construction, after computing the DFT approximation for each subsequence T, we also derive the part of T that is not covered by the top-f coefficients, called its *remainder* $T_{f+} = [T_1 - \text{IDFT}(\tilde{T}_{f,1}), \ldots, T_c - \text{IDFT}(\tilde{T}_{f,c})]$. This remainder can be computed solely based on the top-f coefficients, and does not require computing all other $(|Q| - f)$ coefficients. The key observation here is that the distance between the remainders of two time series captures the remaining ~20% of their distance as shown in Equation 5. Then, Equation 5 can be rewritten to utilize the remainders:

$$d^2(T, Q) = \frac{d^2(\tilde{T}_f, \tilde{Q}_f)}{|Q|} + d^2(T_{f+}, Q_{f+}) \tag{6}$$

The remainders are of length $|Q|$, meaning that the cost is as much as computing the full distance on the original data (i.e., $O(|c_Q| * |Q|)$). To avoid this cost at query time, we precompute the distances of the remainders for each subsequence to a small fixed set of *pivot points* at index time, and store these distances in a map. Then, during query execution, we do the same for Q_{f+}, and use the reverse triangle inequality to lower-bound the distance between Q_{f+} and the remainders of the indexed subsequences through their distances to pivots. We refer to this latter bound as the *correction term.* We

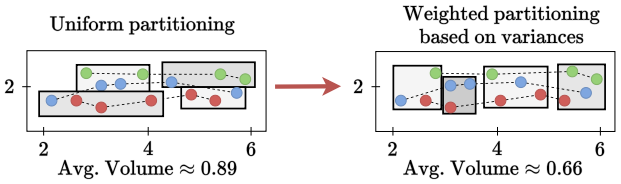

Figure 5: Different partitioning strategies for a 2-dimensional feature space; the STR algorithm (left) and the proposed weighted partitioning (right), that leads to smaller MBRs and to tighter bounds.

apply the correction term to Equation 6 as follows:

$$\begin{aligned} 0 \quad &\leq (|d(\boldsymbol{T}_{f+}, \boldsymbol{P}) - d(\boldsymbol{Q}_{f+}, \boldsymbol{P})|)^2 \leq d^2(\boldsymbol{T}_{f+}, \boldsymbol{Q}_{f+}) \Rightarrow \\ d^2(\boldsymbol{T}, \boldsymbol{Q}) \quad &\geq \underbrace{d^2(\tilde{\boldsymbol{T}}_f, \tilde{\boldsymbol{Q}}_f)/|\boldsymbol{Q}|}_{\text{DFT distance}} + \underbrace{(|d(\boldsymbol{T}_{f+}, \boldsymbol{P}) - d(\boldsymbol{Q}_{f+}, \boldsymbol{P})|)^2}_{\text{Correction term}} \end{aligned} \tag{7}$$

where $\boldsymbol{P}$ is a pivot. At index construction time, we derive k pivots by running k-means clustering on a sample of the remainders of subsequences in the dataset. Then, during query execution, the algorithm detects the closest pivot to the query's remainder, and uses this pivot $\boldsymbol{P}$ to compute the bound, according to Equation 7. In Section 5 we show that this optimization leads to a 2x speedup in query execution.

Tightening the MBRs. Pruning efficiency of the R-tree can be further enhanced by reducing the volume of the MBRs in the R-tree nodes. The R-tree is constructed in a bottom-up fashion using the STR algorithm [44], briefly presented in Section 2.3. STR's approach of splitting the entries into an equal number of partitions per dimension works well when the data is uniformly distributed at all dimensions, but it can lead to sub-optimal partitioning when the data is skewed or concentrated in certain areas of the space. To illustrate the reason, consider a simple example where the R-tree is used to index a two-dimensional space, with the values in the first dimension ranging from 2 to 6, and in the second from 1.5 to 2.5 (see Figure 5). The first dimension will likely be the main contributor of the Euclidean distances between different entries. Therefore, if we devote more splits/partitions on the first dimension while constructing the R-tree, the resulting MBRs will be tighter in this dimension, leading to tighter lower distance bounds and to a more aggressive pruning. Figure 5 illustrates this idea.

DFT approximations, in particular, are prone to have such a heavy concentration of variance in the first few dimensions, as discussed in Section 3.1. We leverage this observation by weighing the dimensions of the R-tree (i.e., the DFT coefficients across channels) based on the variance of their values, which is a good proxy for the contribution of the dimension to the distance between points. Namely, before constructing the R-tree, we use the sample S of subsequences extracted during the summarization step (Section 3.1) to estimate the variance $\hat{\sigma}_i^2$ of the distribution of feature vectors across each dimension i of the feature space. Next, we compute ω_i (the weight of dimension i) through softmax normalization of the variances [15]. Then, given a desired leaf size L, the number of splits p_i for a dimension i is determined by $p_i = \lceil (N/L)^{\omega_i} \rceil$, with N being the total number of entries to index in the whole dataset. This definition of p_i ensures that the desired leaf size L is reached as $\prod_{i=1}^{c*f*2} \lceil (N/L)^{\hat{\omega}_i} \rceil \approx \frac{N}{L}$. This way, the algorithm effectively redistributes the number of partitions across the dimensions, such that it is proportional to the contribution of each dimension to the distance. In Section 5 we will show that this optimization leads to a 2-4x speedup in query execution, due to the more aggressive pruning of the index.

Algorithm 1: UTSBaseline($\mathcal{I}, \boldsymbol{Q}, k$)

Input : A set of channel-level indices $\mathcal{I}$, a query MTS $\boldsymbol{Q}$, a result set size k.

Output : The top-k subsequences in $\mathcal{T}$ with the lowest distance to $\boldsymbol{Q}$.

```
1  R̂ ← {}
2  for i ∈ c_Q do                                  // Iterate over indices
3  |  R̂ ← R̂ ∪ Query(I_i, Q_i, k)                   // Query index
4  R̂ ← ExhaustiveTopK(R̂, Q)                       // Compute est. top-k
5  for i ∈ c_Q do                                  // Set thresholds
6  |  τ_i ← max_{S∈R̂} d²(S_i, Q_i);
7  R ← {};
8  for i ∈ c_Q do                                  // Re-query
9  |  R_i ← Query(I_i, Q, τ_i);
10 |  R ← R ∪ {T | T ∈ R_i ∧ d²(Q_i, T_i) ≤ τ_i}
11 return ExhTopK(R, Q);
```

Distance browsing. Notice that, as the threshold τ_k used in the second probe is set to the k-th smallest *exact* distance of the subsequences returned by the first probe, the second probe is guaranteed to return a superset of the entries returned by the first probe. To avoid redundant lower bound distance computations, we preserve the priority queue of entries used in the best-first searches between the two probes, so that the second probe continues from where the first one left off. This optimization is commonly used for querying spatial indices, and is typically referred to as *distance browsing* [36].

4 Extending Existing Algorithms to the Multivariate Case

As mentioned in Section 2.4, there exist several efficient algorithms that address similarity search for UTS. Therefore, a natural question is whether these algorithms can be extended to query MTS, and how they would perform in such a setting. We will now present a unified extension – a wrapper algorithm – that can be used to extend any UTS search algorithm to work with MTS. We also note that this alternative design approach comes with several deficiencies compared to MS-Index, such as low pruning power when the channels are uncorrelated, as shown in Section 5. Still, it is a useful approach to enable comparison of MS-Index with out-of-the-box extensions of UTS algorithms such as DSTree [73], ST-index [29], and KV-match [74] on MTS data.

The general approach is based on the well-known Threshold algorithm [28], which can be used to derive a global top-k from multiple sorted lists with different attribute values of the same objects, given a monotonic aggregation function to compute the target value upon which the top-k is based. The approach works as follows (cf., Alg. 1): (a) We initialize one index (e.g., one DSTree or one ST-index) per channel. (b) At query time, we first obtain an

initial top-k estimate $\hat{R}_i$ for each channel $i \in c_Q$ by querying the corresponding index (lines 2-3). (c) For each subsequence in the union of the local estimates, we compute the full distance to the query (i.e., using all query channels), constructing an intermediate global top-k $\hat{R}$ (line 4). (d) We set a distance threshold τ_i for each channel $i \in c_Q$ to the largest *univariate* Euclidean distance in $\hat{R}$ on that channel (lines 5-6). (e) Finally, we re-query the channel-level indices with their respective thresholds τ_i. We compute the full distances to the query for the results, update the global top-k accordingly, and return it as a final result (lines 7-11). Since any subsequence that belongs to the global top-k must have a distance at most τ_i on at least one channel $i \in c_Q$, the result of this algorithm is guaranteed to be correct. Lastly, MASS is used to speed up the exact distance computations (lines 4 & 10).

5 Evaluation

The purpose of our experiments was threefold: (a) to assess the scalability and efficiency of MS-Index under various query configurations, (b) to compare MS-Index to other methods, and, (c) to test the effectiveness of the optimizations proposed in Section 3.4. Since our method and all baselines are exact, our evaluation only focuses on efficiency; experiments on accuracy would always yield 100%.

Compared methods. First of all, we compare MS-Index to **MULISSE** [59], applying it to fixed-length search by restricting the supported range of query lengths to $|Q|$. This way, MULISSE also exploits the knowledge of a preknown query length, which slightly improved its performance without affecting the quality of the results. As no other method natively supports k-NN MTS subsequence search, our other baselines comprise of SOTA methods for UTS search extended through Algorithm 1.[7] (a) **ST-Index*** [29], a well-known index for subsequence search [49, 78], (b) **KV-Match*** [74], a SOTA index for subsequence search on single long series, extended with per-MTS indices, and (c) **DSTree*** [73], an index for whole-matching [26], adapted by indexing all subsequences. We also include two sequential scan baselines: (a) **MASS** [51] applied to all MTS, and (b) **Brute-force** exhaustive comparison. Detailed descriptions of the compared methods can be found in Section 2.4.

Hardware and implementations. All experiments were executed on a server equipped with a 64-core 2.4 GHz AMD Genoa 9654 processor and 128 GB of RAM. The code for MULISSE (C++) was provided by the authors of [59], and was executed directly with the original parameters as described in the paper. The code for DSTree* (Java-11) was provided by the authors of [73], and was invoked as a subroutine in Alg. 1. Due to the lack of publicly available code, all other algorithms were implemented from scratch in Java-11.[8] For a fair comparison, all methods were implemented to operate fully in *main memory* and run in *single-threaded*.

Datasets. We used 32 real-world publicly available datasets from different domains, and a set of synthetic datasets: (a) **Stocks.** Daily volumes, opening, closing, high, and low-prices of 28678 stocks over the period Jan. 2, 1987 to Feb. 26, 2021. (b) **Weather.** Segment of the ISD weather dataset [53] containing hourly readings of wind speed, sea level pressure, atmospheric temperature, dew point temperatureof 13545 sensors taken between Jan. 1, 2020 and Dec. 31, 2021. (c) **Wind.** Sensor output of an active wind turbine sampled every 2 seconds for 10 days [52], resulting in a single MTS with 432,000 observations and 10 channels covering power output, rotor speed, wind speed, and other wind-related variables. (d) **Synthetic.** Random walk datasets with 1600 MTS, of 64 channels and 4096 observations each. Following [26, 27], these datasets were generated by drawing random numbers from a normal distribution with a mean of 0 and a standard deviation sampled uniformly at random from the interval $[0, \ldots, 10]$, with starting points sampled uniformly at random from the interval $[0, \ldots, 100]$. (e) **UEA archive.** Popular benchmark archive for MTS classification consisting of 30 labeled real-world datasets from different domains [8], with varying numbers of MTS, channels, and observations. Train and test splits for each dataset were merged, to form a single dataset.

Table 3: Default query configurations.

	Stocks	Weather	Synthetic	Wind	UEA (30x)
n	2000	1300	1600	1	all, see [8]
Avg. m	5590	8692	4096	432,000	see [8]
$\|Q\|$	730 (2 y.)	1488 (2 mth.)	1024	1800 (1 h.)	20%
#Channels	5	4	64	10	see [8]

We evaluated both raw and normalized subsequences, focusing on raw results and highlighting normalized results only when they provide additional insights. The Stocks dataset serves as our primary benchmark due to its size, with other datasets discussed in designated experiments or when they reveal notable patterns. Table 3 summarizes the default query confirms for each dataset. We include more details about the datasets in our code repository.[8]

Queries and evaluation metrics. As per standard practice [26], we generated the query workloads for each dataset by randomly selecting $|Q|$-length subsequences from the dataset and adding Gaussian noise with standard deviation $0.1 * \sigma_{Q_i}$ on all channels. As part of sensitivity analysis experiments, we also investigate varying noise levels and query generation from out-of-dataset subsequences in Section 5.2.6. Unless otherwise stated, we query on all channels of the dataset.When querying on a subset of channels (as in Section 5.2.7), the query channels are selected uniformly at random from all available channels. For each algorithm, we measured: (a) initialization time, e.g., index construction index, (b) query execution time, and, (c) size of the index and/or auxiliary data. We set a timeout of 12 hours for total execution time, and report median results over 10 runs with 100 queries each.

5.1 Tuning the Algorithms

*5.1.1 Number of DFT coefficients for MS-Index and ST-Index** Recall from Section 3.1 that the number of DFT coefficients f is derived based on the distance d_{target} covered by the top-f coefficients, rather than parameterizing the number of coefficients directly. For both algorithms, d_{target} was tuned through a grid search over the values $[20\%, 40\%, \ldots 100\%]$, across all datasets and with queries on both raw and normalized subsequences. The results showed that a coverage of 60% was the most robust choice for both algorithms across the datasets, resulting in a query time at most 20% larger than the optimal choice for each dataset. Therefore, d_{target} was set to 60% for the following experiments.

5.1.2 Leaf size DSTree*, ST-Index*, and MS-Index support tuning of the leaf size L, i.e., the *maximum* number of entries per leaf.

[7] We add an asterisk to their names to differentiate them from their original versions.
[8] Our code is available at https://github.com/JdHondt/MS-Index

Table 4: Average query time (ms) of MS-Index for different leaf size values. Leaf size is expressed as a percentage of the total number of subsequences in each dataset (raw datasets).

Leaf size	Insectwingbeat	Stocks	Synthetic	Weather
0.0001 %	**8.61**	8.50	2.98	4.67
0.01 %	**8.61**	8.37	**2.97**	4.64
0.05* %	9.01	**8.16**	**2.98**	**4.61**
0.1 %	10.79	8.36	3.01	4.69
1 %	22.70	11.83	3.62	5.06
100 %	38.12	158.10	4.59	100.87

Table 5: Index size (MB) and the corresponding percentage of the dataset size (MB) of different algorithms across different Stocks dataset sizes.

	Raw				Normalized			
n	**1000**	**2000**	**3000**	**%**	**1000**	**2000**	**3000**	**%**
Dataset size (MB)	210	430	641	100%	210	430	641	100%
MULISSE	14	15	16	5%	16	18	20	5%
KV-Match*	259	536	793	124%	10	20	30	5%
MASS	315	646	964	150%	315	646	964	150%
MS-Index	968	1923	2861	452%	1094	2248	3348	522%
DSTree*	1491	3230	4773	731%	620	1276	1900	296%
ST-Index*	2904	5771	8584	1355%	3282	6746	10045	1567%

Similar to the number of DFT coefficients, we tuned the leaf size for query time through a grid search over values ranging from 0.0001% to 100% (i.e., no constraints on the leaf size) of the total number of entries to index.[9] Looking at the results in Table 4, we see that a leaf size of 0.05% provides consistently good performance across all datasets for MS-Index. Therefore, this leaf size was chosen for all experiments. For ST-Index*, the optimal leaf size was also 0.05%. For DSTree*, the optimal leaf size was found to be 10%. This contradicts the results of Echihabi et al. [26], who showed that the optimal leaf size for query time was 0.9% for time series of size 256. Both 0.9% and 10% had a similar query time in our experiments, but the 10% choice had a substantially lower initialization cost.

5.1.3 KV-Match segment size* The segment size parameter in KV-Match* controls the number of piecewise means and variances used to index each subsequence. We found the optimal segment size to be 1 for all datasets. For raw subsequences, this reduces the index to a single lookup table per channel and time series. For normalized subsequences, all entries end up in the same bucket due to zero mean and unit variance, effectively reducing KV-Match* to MASS with overhead.

5.2 Evaluation Results

5.2.1 Initialization time Figures 6a-b show the initialization time of all index-based algorithms on the Stocks dataset as the number of MTS (n) increases. All methods scale linearly with n, which is expected since initialization of these methods requires iterating over all time series subsequences. MS-Index and ST-Index* have comparable initialization times, both primarily spent on computing DFT approximations. The remainder of the initialization cost relates to index construction which for ST-Index* involves building channel-level trees (scaling linearly with the number of channels). In contrast, MS-Index builds a single tree for all channels, making it 2-3 times faster for this phase. DSTree* shows comparable initialization time to other methods for normalized subsequences, but is two orders of magnitude slower for raw subsequences. This difference occurs because raw subsequences with varying scales and means cause DSTree* to create deep, unbalanced trees with costly node splitting operations, while normalized subsequences lead to more balanced trees based on pattern differences. KV-Match* and MULISSE have the lowest initialization time. This is because KV-Match* only requires a small lookup table for each time series, and MULISSE is originally build and parameterized for variable-length subsequence search, which adds an additional degree of freedom to the problem and makes indexing performance a key priority.

5.2.2 Size of the data structures Table 5 presents the memory requirements of each algorithm for the Stocks dataset both in MB and as a percentage of the dataset size. All methods scale linearly with n, with MULISSE being the most space-efficient (storing only n envelopes in a shallow iSAX tree), followed by KV-Match*, MASS, MS-Index, DSTree*, and ST-Index*. MS-Index and ST-Index* have a worst-case space complexity of $O(n * (m_{\max} - |\boldsymbol{Q}|) * f * c)$, with ST-Index* requiring approximately three times more space than MS-Index due to its channel-level indices. Reflecting on the relative memory requirements, we see that many methods require more than 100% of the dataset size to store their indices. While this may sound counterintuitive, recall that these structures index the space of *subsequences*, which is significantly larger than the dataset, given that subsequences overlap. Indicatively, while a $n = 1000$ subset of Stocks is 210 MB large, storing all subsequences of length $|\boldsymbol{Q}| = 730$ would require 133 GB of memory, implying that MS-Index already compresses this space by a factor of 138. Still, the memory footprint of subsequence indices may be a concern for some applications, and is a commonly acknowledged limitation of these methods [46, 61].

5.2.3 Query time Figures 6c-d present the query execution time for all methods, when executing queries on subsets of the time series in the Stocks dataset. We see that MS-Index significantly outperforms all other methods. In fact, it outperforms its closest competitor, MASS, by over two orders of magnitude, and the other methods by over three orders of magnitude. These results can be explained by the fact that MS-Index adds pruning power to the MASS algorithm, which already offers a big performance boost over the Brute-force algorithm. A deeper investigation revealed that MS-Index reaches a median pruning effectiveness of 99% across all datasets, i.e., 99% of the subsequences are already pruned from the index and do not need to be compared with MASS. While ST-Index*, KV-Match*, and DSTree* also act as a pre-filter on top of MASS, their pruning power is significantly lower compared to MS-Index, with pruning effectiveness of 52%, 65%, and 46% respectively on average on the Stocks dataset with raw subsequences. This is because the pruning power of these methods depends on the relative ordering of the subsequences to the partial (per-channel) results. When this relative ordering differed significantly over the different query channels, it resulted in relatively high thresholds for the per-channel distances (set in Lines 5-7 of Alg. 1), and therefore to a high number of MASS-based comparisons. MS-Index avoids this issue by querying all query channels simultaneously. Lastly, MULISSE performs second-slowest (after Brute-force) with only 9% pruning rate. This shows that while its extremely memory-efficient

[9] ST-Index*'s leaf size was tuned after tuning the number of DFT coefficients, using a leaf size of 0.1% in the first step.

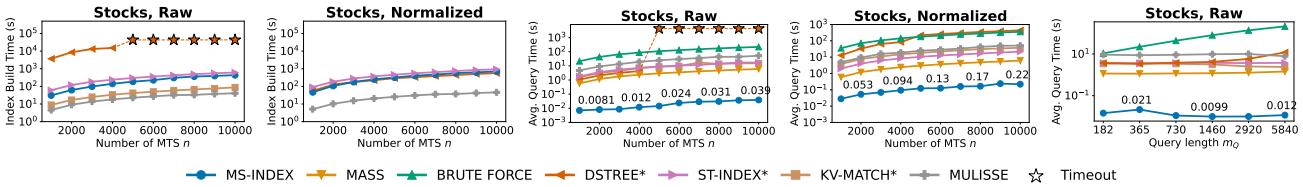

Figure 6: Scalability over number of MTS n on *Stocks*. (a) Initialization time for raw subsequences, and (b) normalized subsequences; (c) Query time for raw subsequences, and (d) normalized subsequences. (e) Query time for raw subsequences with varying query length. All y axes are presented in log scale.

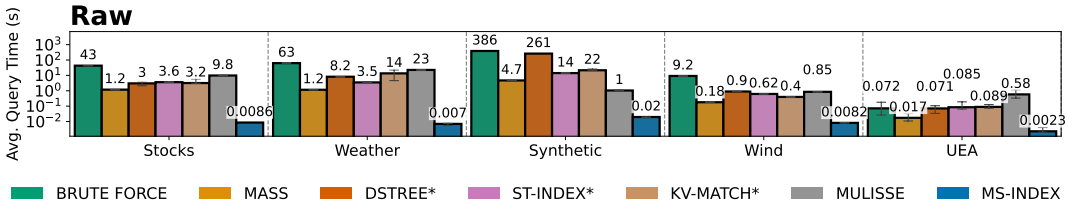

Figure 7: Query time of algorithms on different datasets. All y axes are presented in log scale.

design achieves the smallest index size among all methods, this comes at the cost of loose distance bounds that result in many exhaustive comparisons.

5.2.4 Query length Since query length must be specified before index construction, we evaluate its impact on performance. Figure 6e shows that all algorithms except Brute-force are invariant to query length, as they use MASS which scales with subsequence length ($O(m \log m)$ with $m \geq |Q|$) rather than query length. This suggests MS-Index's performance is independent of $|Q|$, allowing users to choose lengths that best fit their use case.

5.2.5 Different datasets Figure 7 presents the query execution performance across all datasets using default query parameters from Table 3.[10] For the 30 UEA datasets, we present average results as individual dataset results showed similar patterns. MS-Index consistently outperforms competitors by 2-3 orders of magnitude on all datasets, with the exception of the UEA datasets where MS-Index's advantage is reduced to 5-7 times compared to MASS. This is because UEA datasets capture short, single events (e.g., a person lifting their arm, or a duck producing a sound) rather than continuous long series with multiple events. Subsequences in such datasets are typically more similar to each other than those in the Stocks or Weather datasets. This creates more challenging queries for index-based methods, further discussed in Section 5.2.6. Another notable observation is that MS-Index's advantage over competitors remains substantial also on extremely long time series, with a 22x speedup over MASS and a ~100x speedup over other methods on the Wind dataset. This is relevant, as such datasets are becoming increasingly common in domains such as IoT and sensor data [52]. All in all, we conclude that MS-Index is significantly faster than all competitors across various datasets with different characteristics and different domains, confirming its robustness and general applicability.

5.2.6 Effect of query difficulty Pruning-based algorithms rely on the "left-tail" assumption [3]: only a small fraction of the dataset is similar to the query. Their performance depends on the *relative contrast* between the query and indexed data – the ratio between distances to the closest and farthest indexed entities [1, 18]. Following [4, 79], we test algorithm sensitivity by varying noise levels and using out-of-distribution (OOD) queries from held-out data. Figure 8a shows results for Stocks and Weather, revealing that query times are inversely proportional to relative contrast (grey bars on secondary y-axis), consistent with previous work [4, 18, 79]. Normalized subsequence queries have lower contrast than raw ones, making them more challenging. For high-noise or OOD queries, MS-Index's pruning power weakens to match MASS (plus tree traversal overhead), suggesting sequential scans are more appropriate as they don't rely on pruning. This motivates future work on a hybrid approach that selects between MS-Index and MASS based on estimated query difficulty.

Table 6: Relative contrast of queries and percentage of nodes pruned by MS-Index, over different numbers of query channels on the DuckDuckGeese dataset (UEA).

Query channels $\lvert c_Q \rvert$		**16**	**64**	**256**	**1024**
Rel. contrast	Raw	62	107	110	120
	Normalized	14.5	14.4	13.9	13.9
Perc. pruned	Raw	70.54%	79.36%	92.20%	97.94%
	Normalized	89.58%	84.66%	83.68%	83.61%

5.2.7 Number of query channels We investigate algorithm sensitivity to the number of query channels using the *DuckDuckGeese* dataset from UEA, containing 1024 channels. Figure 8b shows MS-Index's query time scales sublinearly with the number of channels, while other methods scale linearly. This improvement comes from increased pruning power outweighing the cost of additional distance computations. For instance, query time decreases from 8.4 to 4.4 ms between 8-16 channels, then grows slowly as marginal selectivity diminishes. Table 6 confirms that MS-Index prunes 70.5-97.9% of R-tree nodes as channels increase from 16 to 1024 when querying for raw subsequences, demonstrating effective use of multivariate

[10] The results on normalized subsequences are omitted as they show similar patterns.

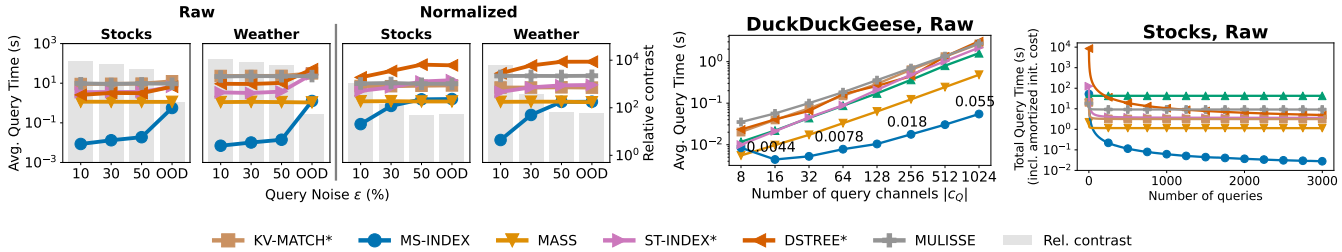

Figure 8: Impact of query workload on algorithms. (a) Query time of algorithms with varying levels of query noise; (b) Query time over queried channels; (c) Amortized initialization time over queries. All y axes are presented in log scale.

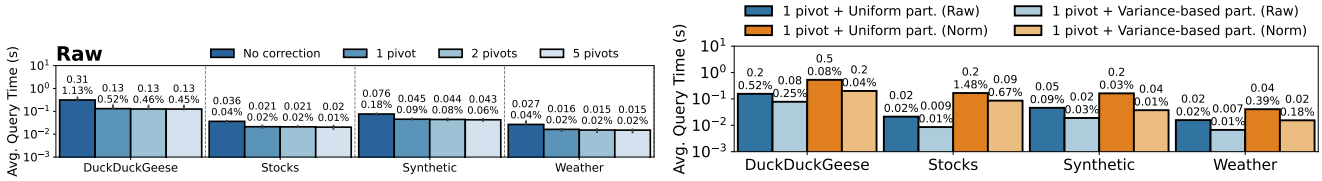

Figure 9: Impact analysis of the optimizations on MS-Index: (a) Query time over the number of pivots in DFT-bound correction; (b) Query time of weighted partitioning vs. uniform partitioning. Annotations indicate the average query time and the percentage of subsequences exhaustively considered (the rest is pruned).

information.[11] In contrast, DSTree*, ST-Index*, and KV-Match* suffer with more query channels as their union of per-channel results grows almost linearly. For normalized subsequences, MS-Index maintains superior performance but experiences a decreasing pruning power due to the curse of dimensionality – as the number of channels increases, distances between points become more uniform, reducing discriminative power [10, 47]. This effect is less pronounced for raw subsequences, where the scale of values in a channel can still provide additional discriminative information that can help in pruning. The relative contrast values in Table 6 confirm this: increasing with $|c_Q|$ for raw subsequences while slightly decreasing for normalized ones.

5.2.8 Amortizing the initialization cost Since algorithms have different initialization costs, their relative efficiency depends on the number of queries executed. Figure 8c shows the amortized total cost (initialization + query time) up to 3000 queries. While MASS is fastest for few queries due to its low initialization cost, MS-Index becomes more efficient after just 45 queries, with its advantage growing with more queries due to its superior query performance.

5.2.9 Effect of DFT bound correction optimization Recall from Section 3.4 that we tighten the distance bound using a triangle-inequality-based correction term, enabling more pruning at the cost of added initialization overhead. We evaluate this correction's impact on both initialization and query times with varying number of pivots. Figure 9a shows that adding just a single pivot improves query performance by a factor of 2, while additional pivots yield only marginal improvements. This is known as the *coverage saturation effect*, where the effectiveness of pivot-based bounding stabilizes as the space becomes increasingly well-covered with more pivots [6, 20]. In our case, diminishing returns after the first pivot indicate that the space of remainders (i.e., the high-frequency DFT coefficients) has low complexity: the data is concentrated around a single point. This observation is not surprising, as – by definition – these remainders comprise lower-energy frequencies that mostly represent noise following a normal distribution [40]. Nevertheless, the improvement over not correcting shows that the remainders still contain valuable information. Our analysis shows pivot comparisons increase initialization time by 15%, 30%, and 75% for 1, 2, and 5 pivots respectively. Given the query time improvements, a single pivot provides the best cost-benefit tradeoff.

5.2.10 Effect of the optimization for tightening the MBRs We evaluate the optimization for tightening MBRs via weighted R-tree partitioning (Section 3.4) against uniform partitioning. The results in Figure 9b show that weighted partitioning improves query time by 1.5-3x by allowing the R-tree to better adapt to dataset characteristics, with larger gains on high-dimensional datasets like DuckDuckGeese and Synthetic. From these results we can conclude that the weighted partitioning strategy is beneficial.

6 Conclusions

We considered the problem of fixed-length subsequence search on MTS, and proposed MS-Index, an exact algorithm that adaptively partitions and indexes a DFT-based feature space, allowing for a cheap pruning of over 99% of the subsequences. Our evaluation with 34 datasets demonstrated that MS-Index outperforms the state-of-the-art by two orders of magnitude for both raw and normalized subsequences, and that it scales sublinearly with the amount of query channels.

Acknowledgments

This work has received funding from the Horizon Europe research and innovation programmes STELAR (101070122), AI4Europe (101070000), TwinODIS (101160009), ARMADA (101168951), DataGEMS (101188416), and RECITALS (101168490).

[11] The pruning percentage of nodes is not the same as the pruning power (the ratio between number of considered subsequences over all subsequences, around 99.9% for MS-Index), since many nodes in the tree contain groups of subsequences.

References

[1] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. 2001. On the Surprising Behavior of Distance Metrics in High Dimensional Space. In *Proc. ICDT'01.*

[2] George Arfken. 1985. *Mathematical Methods for Physicists* (third ed.). Academic Press, Inc., San Diego.

[3] M. Aumüller and M. Ceccarello. 2021. The role of local dimensionality measures in benchmarking nearest neighbor search. *Information Systems* 101 (2021).

[4] Ilias Azizi, Karima Echihabi, and Themis Palpanas. [n.d.]. ELPIS: Graph-Based Similarity Search for Scalable Data Science. In *Proc. VLDB'23.* 1548–1559.

[5] Martin Bach-Andersen, Bo Rømer-Odgaard, and Ole Winther. 2017. Flexible non-linear predictive models for large-scale wind turbine diagnostics. *Wind Energy* 20 (2017), 753–764.

[6] Ricardo Baeza-Yates, Walter Cunto, Udi Manber, and Sun Wu. 1994. Proximity matching using fixed-queries trees. In *Annual Symposium on Combinatorial Pattern Matching*. Springer, 198–212.

[7] Anthony J. Bagnall, Richard L. Cole, Themis Palpanas, and Konstantinos Zoumpatianos. 2019. Data Series Management (Dagstuhl Seminar 19282). *Dagstuhl Reports* 9, 7 (2019), 24–39.

[8] Anthony J. Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn J. Keogh. 2018. The UEA multivariate time series classification archive. (2018). http://arxiv.org/abs/1811.00075

[9] A. J. Bagnall and G. J. Janacek. 2004. Clustering time series from ARMA models with clipped data. In *Proc. KDD'04.* 49–58.

[10] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-tree: an efficient and robust access method for points and rectangles. *Proc. SIGMOD'98* (1990), 322–331.

[11] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (Sept. 1975), 509–517.

[12] Kanishka Bhaduri, Qiang Zhu, Nikunj C. Oza, and Ashok N. Srivastava. 2010. Fast and Flexible Multivariate Time Series Subsequence Search. In *Proc. ICDM'10.* 48–57.

[13] Angela Bonifati, Francesco Del Buono, Francesco Guerra, Miki Lombardi, and Donato Tiano. 2023. Interpretable Clustering of Multivariate Time Series with Time2Feat. *Proc. VLDB Endow.* 16, 12 (2023), 3994–3997.

[14] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. *Proc. SIGMOD'00* (2000), 93–104.

[15] John S Bridle. 1990. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing: Algorithms, architectures and applications*. Springer, 227–236.

[16] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn Keogh. 2010. iSAX 2.0: Indexing and Mining One Billion Time Series. In *Proc. ICDM'10.* 58–67.

[17] 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*. 48–57.

[18] Matteo Ceccarello, Alexandra Levchenko, Ileana Ioana, and Themis Palpanas. 2025. Evaluating and Generating Query Workloads for High Dimensional Vector Similarity Search. In *Proc. KDD'25.*

[19] Georgios Chatzigeorgakidis, Dimitrios Skoutas, Kostas Patroumpas, Themis Palpanas, Spiros Athanasiou, and Spiros Skiadopoulos. 2023. Efficient Range and kNN Twin Subsequence Search in Time Series. *IEEE Transactions on Knowledge and Data Engineering* 35, 6 (2023), 5794–5807.

[20] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. 2001. Searching in metric spaces. *ACM Comput. Surv.* 33, 3 (Sept. 2001), 273–321.

[21] Paolo Ciaccia, Marco Patella, and Pavel Zezula. 1997. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proc. VLDB '97.* 426–435.

[22] Michele Dallachiesa, Themis Palpanas, and Ihab F. Ilyas. 2014. Top-k nearest neighbor search in uncertain data series. *Proc. VLDB'14* (2014), 13–24.

[23] Jens E. d'Hondt, Haojun Li, Fan Yang, Odysseas Papapetrou, and John Paparrizos. 2025. A Structured Study of Multivariate Time-Series Distance Measures. *Proc. SIGMOD'25* (2025).

[24] Rui Ding, Qiang Wang, Yingnong Dang, Qiang Fu, Haidong Zhang, and Dongmei Zhang. 2015. YADING: fast clustering of large-scale time series data. *Proc. VLDB'15* (2015), 473–484.

[25] Jens E. d'Hondt, Odysseas Papapetrou, and John Paparrizos. 2024. Beyond the Dimensions: A Structured Evaluation of Multivariate Time Series Distance Measures. In *Proc. ICDEW'24.* 107–112.

[26] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2018. The Lernaean Hydra of Data Series Similarity Search: An Experimental Evaluation of the State of the Art. *Proc. VLDB'18,* 112–127.

[27] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. 2019. Return of the Lernaean Hydra: experimental evaluation of data series approximate similarity search. In *Proc. VLDB'19.* 403–420.

[28] Ronald Fagin, Amnon Lotem, and Moni Naor. 2001. Optimal aggregation algorithms for middleware. In *Proc. PODS'01.* 102–113.

[29] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. 1994. Fast subsequence matching in time-series databases. In *Proc. SIGMOD '94.* 419–429.

[30] Kefeng Feng, Peng Wang, Jiaye Wu, and Wei Wang. 2020. L-Match: A Lightweight and Effective Subsequence Matching Approach. *IEEE Access* 8 (2020), 71572–71583.

[31] Hakan Ferhatosmanoglu, Ertem Tuncel, Divyakant Agrawal, and Amr El Abbadi. 2000. Vector approximation based indexing for non-uniform high dimensional data sets. In *Proc. CIKM'00.* 202–209.

[32] Ada Wai-chee Fu, Eamonn Keogh, Leo Yung Hang Lau, and Chotirat Ann Ratanamahatana. 2005. Scaling and time warping in time series querying. In *Proc. VLDB'05.* 649–660.

[33] Roger Grosse, Rajat Raina, Helen Kwong, and Andrew Y. Ng. 2012. Shift-Invariance Sparse Coding for Audio Classification. https://arxiv.org/abs/1206.5241

[34] Antonin Guttman. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching *(Proc. SIGMOD '84).* 47–57.

[35] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. 2014. Classification of time series by shapelet transformation. *Data Min. Knowl. Discov.* 28, 4 (jul 2014), 851–881.

[36] Gísli R. Hjaltason and Hanan Samet. 1999. Distance browsing in spatial databases. *ACM Trans. Database Syst.* 24, 2 (June 1999), 265–318.

[37] 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 (Aug. 2014), 27–39.

[38] K. Kalpakis, D. Gada, and V. Puttagunta. 2001. Distance measures for effective clustering of ARIMA time-series. In *Proc. ICDM'01.* 273–280.

[39] K. Kashino, G. Smith, and H. Murase. 1999. Time-series active search for quick retrieval of audio and video. In *Proc. ICASSP'99.* 2993–2996 vol.6.

[40] Steven M Kay. 1993. *Fundamentals of statistical signal processing: estimation theory.* Prentice-Hall, Inc.

[41] Eamonn Keogh, Themistoklis Palpanas, Victor B. Zordan, Dimitrios Gunopulos, and Marc Cardle. 2004. Indexing large human-motion databases. In *Proc. VLDB'04.* 780–791.

[42] S. Knieling, J. Niediek, E. Kutter, J. Bostroem, C.E. Elger, and F. Mormann. 2017. An online adaptive screening procedure for selective neuronal responses. *Journal of Neuroscience Methods* 291 (2017), 36–42.

[43] Haridimos Kondylakis, Niv Dayan, Kostas Zoumpatianos, and Themis Palpanas. 2018. Coconut: a scalable bottom-up approach for building data series indexes. *Proc. VLDB'18* (2018), 677–690.

[44] S.T. Leutenegger, M.A. Lopez, and J. Edgington. 1997. STR: a simple and efficient algorithm for R-tree packing. In *Proc. ICDE'97.* 497–506.

[45] Stefan Liess, Saurabh Agrawal, Snigdhansu Chatterjee, and Vipin Kumar. 2017. A Teleconnection between the West Siberian Plain and the ENSO Region. *Journal of Climate* 30, 1 (2017), 301 – 315.

[46] Michele Linardi and Themis Palpanas. 2018. Scalable, variable-length similarity search in data series: the ULISSE approach. *Proc. VLDB'18* (2018), 2236–2248.

[47] R. B. Marimont and M. B. Shapiro. 1979. Nearest Neighbour Searches and the Curse of Dimensionality. *IMA Journal of Applied Mathematics* 24, 1 (08 1979), 59–70.

[48] Yang-Sae Moon, Kyu-Young Whang, and Wook-Shin Han. 2002. General match: a subsequence matching method in time-series databases based on generalized windows. In *Proc. SIGMOD'02.* 382–393.

[49] Yang-Sae Moon, Kyu-Young Whang, and Wook-Shin Han. 2002. General match: a subsequence matching method in time-series databases based on generalized windows. In *Proc. SIGMOD'02.* 382–393.

[50] Abdullah Mueen, Suman Nath, and Jie Liu. [n.d.]. Fast Approximate Correlation for Massive Time-Series Data. In *Proc. SIGMOD'10.*

[51] Abdullah Mueen, Sheng Zhing, Yan Zhu, Michael Yeh, Kaveh Kamgar, Krishnamurthy Viswanathan, Chetan Gupta, and Eamonn Keogh. 2022. The Fastest Similarity Search Algorithm for Time Series Subsequences under Euclidean Distance. http://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html.

[52] Carlos Enrique Muniz-Cuza, Søren Kejser Jensen, Jonas Brusokas, Nguyen Ho, and Torben Bach Pedersen. 2024. Evaluating the Impact of Error-Bounded Lossy Compression on Time Series Forecasting. In *Proc. EDBT'24.*

[53] National Oceanic and Atmospheric Administration. [n.d.]. NOAA Integrated Surface Dataset. https://www.ncei.noaa.gov/access/search/dataset-search.

[54] Themis Palpanas and Volker Beckmann. 2019. Report on the First and Second Interdisciplinary Time Series Analysis Workshop (ITISA). *SIGMOD Rec.* 48, 3 (2019), 36–40.

[55] John Paparrizos and Michael J. Franklin. 2019. GRAIL: efficient time-series representation learning. *Proc. VLDB'19* (2019), 1762–1777.

[56] John Paparrizos and Luis Gravano. 2016. k-Shape: Efficient and Accurate Clustering of Time Series. *Proc. SIGMOD'16* (2016), 69–76.

[57] John Paparrizos, Chunwei Liu, Aaron Elmore, and Michael J. Franklin. 2023. Querying Time-Series Data: A Comprehensive Comparison of Distance Measures. *IEEE Data Engineering Bulletin (DEB 2023)* 47 (2023), 69–88.

[58] 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).

[59] Balazs Pelok and Jens E. d'Hondt. 2025. MULISSE: Variable-Length Similarity Search for Multivariate Time Series. In *Proc. ICDEW'25*.

[60] Davood Rafiei and Alberto O. Mendelzon. 1998. Efficient Retrieval of Similar Time Sequences Using DFT. *Proc. FODO'98* (1998).

[61] 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 *Proc. KDD'12*. 262–270.

[62] Chotirat Ann Ratanamahatana and Eamonn Keogh. 2005. Three Myths about Dynamic Time Warping Data Mining. In *Proc. SDM'05*. 506–510.

[63] John F. Roddick and Kathleen Stewart Hornsby. 2001. Temporal, Spatial, and Spatio-Temporal Data Mining. In *Lecture Notes in Computer Science*.

[64] Alejandro Pasos Ruiz, Michael Flynn, James Large, Matthew Middlehurst, and Anthony Bagnall. 2021. The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery* 35, 2 (01 Mar 2021), 401–449.

[65] A Salarpour and H Khotanlou. 2018. An Empirical Comparison of Distance Measures for Multivariate Time Series Clustering. *International Journal of Engineering* 31, 2 (2018), 250–262.

[66] Hanan Samet. 2005. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[67] Jin Shieh and Eamonn Keogh. 2008. iSAX: indexing and mining terabyte sized time series. In *Proc. KDD'08*. 623–631.

[68] Ahmed Shifaz, Charlotte Pelletier, François Petitjean, and Geoffrey I. Webb. 2023. Elastic similarity and distance measures for multivariate time series. *Knowledge and Information Systems* 65, 6 (01 Jun 2023), 2665–2698.

[69] Mohammad Shokoohi-Yekta, Bing Hu, Hongxia Jin, Jun Wang, and Eamonn J. Keogh. 2014. Generalizing Dynamic Time Warping to the Multi-Dimensional Case Requires an Adaptive Approach.

[70] Gilbert Strang. 1994. Wavelets. *American Scientist* 82, 3 (1994), 250–255.

[71] Shreshth Tuli, Giuliano Casale, and Nicholas R. Jennings. 2022. TranAD: Deep Transformer Networks for Anomaly Detection in Multivariate Time Series Data. *Proc. VLDB'22* (2022), 1201–1214.

[72] Michail Vlachos, Marios Hadjieleftheriou, Dimitrios Gunopulos, and Eamonn Keogh. 2003. Indexing Multi-Dimensional Time-Series with Support for Multiple Distance Measures. In *Proc. KDD '03*. 216–225.

[73] 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. *Proc. VLDB'13* (2013), 793–804.

[74] J. Wu, P. Wang, N. Pan, C. Wang, W. Wang, and J. Wang. 2019. KV-Match: A Subsequence Matching Approach Supporting Normalization and Time Warping. In *Proc. ICDE'19*. 866–877.

[75] Xiaopeng Xi, Eamonn Keogh, Christian Shelton, Li Wei, and Chotirat Ann Ratanamahatana. 2006. Fast time series classification using numerosity reduction. In *Proc. ICML'06*. 1033–1040.

[76] Fan Yang and John Paparrizos. 2025. SPARTAN: Data-Adaptive Symbolic Time-Series Approximation. In *Proc. SIGMOD'25*.

[77] Kiyoung Yang and Cyrus Shahabi. 2004. A PCA-Based Similarity Measure for Multivariate Time Series. In *Proc. MMDB'04*. 65–74.

[78] Haohan Zhu, George Kollios, and Vassilis Athitsos. 2012. A generic framework for efficient and effective subsequence retrieval. *Proc. VLDB'12* (2012), 1579–1590.

[79] Kostas Zoumpatianos, Yin Lou, Ioana Ileana, Themis Palpanas, and Johannes Gehrke. 2018. Generating data series query workloads. *The VLDB Journal* (2018).