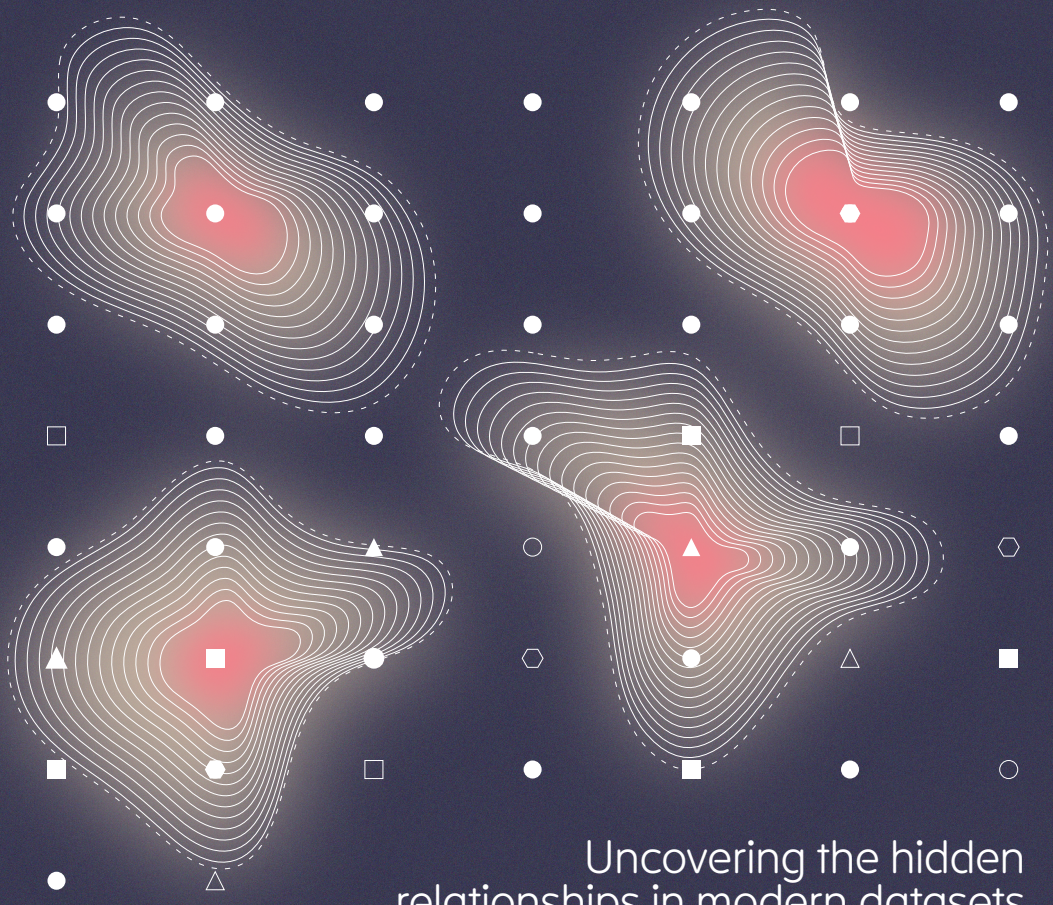
EFFECTIVE AND EFFICIENT MULTIVARIATE SIMILARITY SEARCH

Jens E. d'Hondt



Uncovering the hidden relationships in modern datasets

Effective and Efficient Multivariate Similarity Search

Uncovering the Hidden Relationships in Modern Datasets

Jens E. d'Hondt



Copyright © 2025 by Jens E. d'Hondt. All rights are reserved. Reproduction in whole or in part is prohibited without the written consent of the copyright owner.

A catalogue record is available from the Eindhoven University of Technology Library.

ISBN 978-90-386-6562-7
NUR 980
SIKS Dissertation Series No. 2025-64

Cover design by Sharon Coone.

Printed and bound by ADC Nederland.



The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Effective and Efficient Multivariate Similarity Search

Uncovering the Hidden Relationships in Modern Datasets

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van
de rector magnificus prof. dr. S.K. Lenaerts,
voor een commissie aangewezen door het College
voor Promoties, in het openbaar te verdedigen op
dinsdag 16 december 2025 om 13:30 uur

door

Jens Emiel d'Hondt

geboren te Sint-Niklaas, België

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

Voorzitter:	prof. dr. M.T. de Berg
Promotor:	prof. dr. G.H.L. Fletcher
Co-promotor:	dr. O. Papapetrou
Leden:	dr. V. Menkovski
	prof. dr. F. Naumann (Hasso Plattner Institute)
	prof. dr. T.B. Pedersen (Aalborg University)
	prof. dr. Y. Velegrakis (Universiteit Utrecht)

Het onderzoek dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

"Learn how to see.
Realize that everything connects to everything else."

– Leonardo da Vinci

Acknowledgments

I would like to express my sincerest thanks

to Odysseas,

for giving me the opportunity to pursue this PhD,

for your guidance and feedback throughout these four years,

for our overly passionate "5 minute" discussions about science, life, and everything in between;

to my doctoral committee members,

for your time and effort in reviewing my work and providing valuable feedback,

for traveling to Eindhoven to attend my defense and celebrate this milestone with me;

to my co-authors, Balazs, Christoph, Fan, Giorgos, Haojun, John, Koen, Odysseas, Teun, Themis, and Wieger,

for your collaboration and contributions to the research presented in this dissertation,

for bearing with me through late-night chats, tight deadlines, and last-minute changes;

to my colleagues at the Data and AI group,

in particular to Ashkey, Daan, Hilde, Koen, Prabhant, Tim, Thomas, Wieger, Wouter, and Ziwei, for having breaks together, and making the office a fun place to be;

to Koen,

for being a collaborator, colleague, sparring partner, and friend throughout my entire academic career,

for the countless hours spent discussing research and life while enjoying a coffee, lunch, or beer (not necessarily in that order);

to my friends,

for beers on Fridays, hangovers on Saturdays, and bike rides on Sundays,

for trips to snowy mountains, sunny hills, and vibrant cities,

for being there through thick and thin, making me happy, and reminding me of what truly matters in life;

to my family,

for your unconditional love and support,

for listening to my occasional rants,

for always believing in me, and encouraging me to pursue that which makes me happy;

to Rosella,

for bearing with me through the ups and downs of the past four years,

for being my safe haven in times of stress and uncertainty,

for your endless love; I could not have done this without you.

Jens E. d'Hondt
Eindhoven, September 2025

Contents

Acknowledgments **v**

1 Introduction **1**
1.1 Similarity Search . . . 1
1.2 Problem Statement . . . 2
1.3 Contributions & Outline . . . 4

2 Foundations of Multivariate Similarity Search **7**
2.1 Definitions and Notation . . . 7
2.1.1 Pairwise Similarity Search on Multivariate Time Series (Part I) . . . 10
2.1.2 Multivariate Similarity Search on Univariate Data (Part II) . . . 10
2.2 Time Series Approximation Methods . . . 13
2.3 Indexes and Data Structures . . . 14
2.3.1 R-Trees . . . 15
2.4 Related Work . . . 16
2.4.1 Pairwise Similarity Search on Multivariate Time Series (Part I) . . . 16
2.4.2 Multivariate Similarity Search on Univariate Data (Part II) . . . 17

I Pairwise Similarity Search on Multivariate Time Series **19**

3 A Structured Study of Distance Measures for Multivariate Time Series **21**
3.1 Introduction . . . 21
3.2 Related Work . . . 24
3.3 Primer on Comparing Multivariate Time Series . . . 25
3.3.1 Data Assumptions . . . 26
3.3.2 Axis 1: Normalization . . . 26
3.3.3 Axis 2: Temporal Models . . . 27
3.3.4 Axis 3: Channel-Dependency Models . . . 28
3.4 Multivariate Distance Measures . . . 30
3.5 Evaluation . . . 33
3.5.1 Task 1: Classification . . . 36
3.5.2 Task 2: Clustering . . . 46
3.5.3 Task 3: Anomaly Detection . . . 47
3.5.4 Runtime Analysis . . . 48
3.6 Summary of Observations and Guidelines . . . 51
3.7 Reflection and Conclusions . . . 52

4 Efficient Subsequence Search in Multivariate Time Series 55
4.1 Introduction . . . 55
4.2 Background . . . 57
4.2.1 Related Work . . . 58
4.3 MS-Index for Subsequence Search on Multivariate Time Series . . . 61
4.3.1 Summarizing All Subsequences with DFTs . . . 62
4.3.2 Indexing of the MTS . . . 64
4.3.3 Query Execution . . . 65
4.3.4 Optimizations . . . 66
4.4 Extending Existing Algorithms to the Multivariate Case . . . 69
4.5 Evaluation . . . 71
4.5.1 Tuning the Algorithms . . . 73
4.5.2 Evaluation Results . . . 74
4.6 Reflection and Conclusions . . . 80

II Multivariate Similarity Search on Univariate Data 83

5 Efficient Detection of Multivariate Correlations 85
5.1 Introduction . . . 86
5.2 Preliminaries . . . 88
5.2.1 Similarity Measures . . . 88
5.2.2 Additional Constraints . . . 89
5.2.3 Related Work . . . 90
5.3 Detection of Multivariate Correlations in Static Data . . . 92
5.3.1 Initialization and Clustering . . . 93
5.3.2 Threshold Queries . . . 94
5.3.3 Theoretical Bounds . . . 95
5.3.4 Empirical Pairwise Bounds . . . 97
5.3.5 Top-k Queries . . . 99
5.3.6 Progressive Queries . . . 101
5.4 Detection of Multivariate Correlations in Streaming Data . . . 102
5.4.1 Stream Processing Model . . . 102
5.4.2 Algorithm Core . . . 104
5.4.3 User Constraints and Top-k Queries . . . 107
5.4.4 Impact of Streaming Processing Model on CDStream . . . 108
5.4.5 CDHybrid: Combining CD and CDStream . . . 109
5.5 Evaluation . . . 110
5.5.1 Comparison to the Baselines . . . 112
5.5.2 CD on Static Data . . . 114
5.5.3 CDStream on Streaming Data . . . 118
5.6 Reflection and Conclusions . . . 122

6 Towards a General Multivariate Similarity Search Framework 125
6.1 Introduction . . . 125
6.2 Motivation . . . 127
6.2.1 Databases . . . 127

6.2.2 Signal Processing . . . 128
6.2.3 Data Mining and Machine Learning . . . 129
6.2.4 Biology, Genomics, and Medicine . . . 130
6.2.5 Finance . . . 130
6.2.6 Domain-agnostic Approaches . . . 130
6.3 Analysis of Existing Solutions . . . 131
6.4 The Similarity Detective (SD) Framework . . . 132
6.4.1 Types of Multivariate Similarity Measures . . . 133
6.4.2 Properties of Multivariate Similarity Measures . . . 133
6.4.3 Query Execution Workflow . . . 135
6.4.4 Bound Derivation . . . 139
6.4.5 Optimization Suite . . . 145
6.4.6 Measure Interface . . . 153
6.4.7 Examples . . . 156
6.4.8 Extending the Framework to Streaming Data . . . 160
6.5 Evaluation . . . 161
6.5.1 Effect of Optimizations . . . 163
6.5.2 Sensitivity Analysis . . . 164
6.5.3 Comparison to Baselines . . . 168
6.6 Vision . . . 170
6.7 Reflection and Conclusions . . . 172

7 Case Study: Sparse Representation 175
7.1 Introduction . . . 175
7.2 Background and Problem Definition . . . 176
7.3 Current Approaches . . . 177
7.3.1 Exact Methods . . . 177
7.3.2 Approximate Methods . . . 178
7.3.3 Comparison and Limitations . . . 178
7.4 Solving Sparse Representation with SD . . . 179
7.4.1 Methodology . . . 179
7.4.2 Reflection . . . 181
7.5 Evaluation . . . 182
7.5.1 Comparing Sparsity and Reconstruction Error . . . 185
7.5.2 Comparing Classification Accuracies . . . 187
7.5.3 Comparing Computational Efficiency . . . 189
7.5.4 Adversarial Examples . . . 190
7.6 Conclusions . . . 191

8 Conclusions and Future Directions 193
8.1 Summary of Contributions . . . 193
8.2 Reflection on Central Research Question . . . 194
8.3 Open Challenges and Future Directions . . . 195
8.3.1 The Grand Challenge: Multivariate Squared . . . 195
8.3.2 Directions in MTS Search . . . 196
8.3.3 Directions in High-Order Relationship Search . . . 197

8.3.4 Overarching Directions . . . 197
8.3.5 Towards an Overarching Framework . . . 199
8.4 Closing Remarks . . . 199

A Proofs to Chapter 4 **201**
A.1 Proof of correctness and completeness of MS-Index . . . 201

B Proofs to Chapter 5 **203**
B.1 Proof of Lemma 5.3.1 . . . 203
B.2 Proof of Theorem 5.3.2 . . . 204
B.3 Proof of Theorem 5.3.3 . . . 205

C Proofs to Chapter 6 **207**
C.1 Proof of Lemma 6.4.2 . . . 207
C.2 Proof of Lemma 6.4.3 . . . 208
C.3 Proof of Lemma 6.4.4 . . . 208

D Proofs to Chapter 7 **211**
D.1 Proof of Equivalence of Sparse Representation and Multipole Maximization . 211

E Distance Measure Definitions **213**

F Distribution of Contributions in Chapter 5 **219**
F.1 Distribution of Contributions . . . 219
F.2 Quantitative Comparison between old and new CD algorithms . . . 220

G Framework Interface **223**

H Implementations of Example Measures **227**
H.1 Multivariate Euclidean Distance (*mED*) . . . 227
H.2 Multivariate Total Correlation (*TC*) . . . 227
H.3 Multivariate Manhattan Distance (*mMD*) . . . 227

Bibliography **231**

SIKS dissertations **249**

Author's Publications **263**

Summary **265**

About the Author **267**

List of Acronyms

k-NN	k-Nearest Neighbors
CD	Correlation Detective
DCC	Decisive Cluster Combination
DFT	Discrete Fourier Transform
DTW	Dynamic Time Warping
FFT	Fast Fourier Transform
iMP	Inverse Multipole
MASS	Mueen's Algorithm for Similarity Search
MBR	Minimum Bounding Rectangle
mED	Multivariate Euclidean Distance
mMD	Multivariate Manhattan Distance
MP	Multipole
mPC	Multivariate Pearson Correlation
MTS	Multivariate Time Series
SBD	Shape-Based Distance
SD	Similarity Detective
TC	Total Correlation
UTS	Univariate Time Series

CHAPTER 1

Introduction

1.1 Similarity Search

In recent decades, the rapid development of data collection technologies and storage solutions has led to an unprecedented accumulation of data across a wide range of domains, from astrophysics and seismology to health care and finance [21, 118, 126, 136, 194]. Every day, billions of new data points are being generated, resulting in vast multidimensional datasets, where each observation consists of measurements on multiple variables [209]. To extract value from these complex datasets, effective and efficient analysis techniques are essential.

One of the most fundamental operations in data analysis is similarity search — the task of identifying similarities between data objects in a dataset [81, 82]. This fundamental operation not only serves as a standalone task [8, 38, 63, 68, 74, 88, 127, 134, 139, 140, 148, 152, 188, 203, 204, 217] but also forms the backbone of more high-level tasks such as outlier detection [35, 68, 227], classification [24, 115, 211, 218, 219, 243], clustering [25, 33, 75, 124, 213], motif discovery [29, 58, 152, 176, 245, 246], and pattern mining [125].

Research in similarity search over the past three decades has primarily focused on two fundamental aspects: (a) search *effectiveness* (the quality of results), and (b) search *efficiency* (computational performance). Both aspects are crucial for addressing the diverse requirements of modern applications [81, 82].

Effective search hinges on the selection of appropriate distance measures and data preprocessing techniques given the nature of the data and the downstream application of the search [24, 73, 179, 193, 213, 218, 219]. The core difficulty here lies in capturing the human ability to identify similarities while ignoring various *distortions* in the data, such as temporal misalignments, scaling differences, and noise, that are irrelevant to the comparison. These perceptual rules, though natural to us, are complex and deeply context-dependent, making them hard to formalize into mathematical expressions [86]. In response to this challenge, researchers have developed a wide range of similarity measures and preprocessing techniques, each with different approaches to handling distortions and capturing the underlying patterns in the data [17, 31, 49, 50, 55, 74, 80, 88, 94, 167, 175, 190, 191, 193, 212, 221, 230].

Efficient search is also crucial, as the high-dimensional nature of modern datasets makes it computationally expensive to perform similarity search on large datasets with millions of observations on hundreds of variables [81, 82]. This led researchers to propose a wide range of dimensionality reduction techniques [54, 84, 132, 147, 151, 156, 160, 164, 196, 216, 240, 247], indexing structures [8, 37, 38, 45, 51, 63, 74, 88, 127, 131, 140, 148, 188, 202], and search algorithms [87] to speed up the search under different measures and conditions. These conditions include (a) quering in a static, streaming or distributed setting, (b) potential additional constraints on the search space, (c) guarantees on the completeness of the results (i.e., exact or approximate search), and (d) the available computational resources (e.g., main memory, disk, or cloud, CPU, GPU, or FPGA), among others [81, 82].

These research efforts in both effectiveness and efficiency have been fruitful; similarity search algorithms now serve as fundamental building blocks in numerous data analysis pipelines across various domains. For instance, in industrial settings, similarity search enables predictive maintenance by identifying historical patterns similar to current machinery behavior [6]. In healthcare, it aids in diagnosis by finding similar patient trajectories [142] and detecting anomalies in vital signs [68]. More recently, similarity search has become integral to modern machine learning applications, particularly in Retrieval-Augmented Generation (RAG) systems, where it helps to "ground" large language models by retrieving relevant context from knowledge bases [144]. Other emerging applications include financial market analysis for detecting changes in the economic climate [181], smart city management through traffic pattern analysis [254], and environmental monitoring through the detection of similar weather patterns [176].

1.2 Problem Statement

Despite the maturity of similarity search as a research field, current approaches suffer from a fundamental limitation: they predominantly focus on pairwise relationships between individual objects, like correlations between two stock prices or the similarity between two audio recordings. However, as many domain scientists (e.g., physicists, biologists, social scientists) have observed, our world is inherently multivariate, with complex relationships between multiple objects with multiple attributes [9, 10, 43, 48, 111, 137, 149, 172, 183, 199, 215, 251]. For example, in a social network, the similarity between two users is not just based on their friendship status, but also on their shared interests, the groups they are part of, and the content they have interacted with. In a similar vein, the neurons in our brain do not just fire based on a single stimulus, but on a combination of multiple stimuli with complex interactions between them [10]. *Consequently, by only looking for pairwise relationships in univariate data, we are potentially missing crucial information that could lead to more accurate and meaningful results.*

To address this challenge, this thesis proposes to extend the traditional similarity search paradigm beyond its current univariate, pairwise focus. Specifically, as illustrated in Figure 1.1, we explore two fundamental extensions of the original similarity search problem:

1. **Data-level extension:** We develop methods for searching *pairwise relations* in *multivariate data*, offering a more complete view of the objects or phenomena that are be-

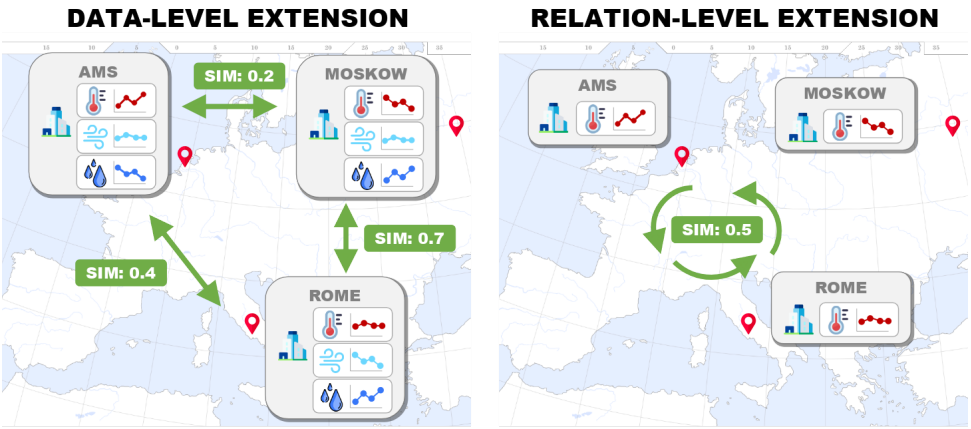


Figure 1.1: Visualization of the two extensions of the similarity search problem using weather sensor data between cities as an example. Under the data-level extension, we measure pairwise similarities between MTS (e.g., temperature, wind speed, and precipitation) of different cities. Under the relation-level extension, we measure the combined similarity between three UTS (e.g., temperature) of different cities.

ing compared. Where a univariate data object captures a single attribute or variable, a multivariate object captures multiple co-evolving variables, providing a richer representation of the underlying phenomena [243]. For instance, consider a sensor measuring weather conditions at a certain location, resulting in a *time series*; a sequence of data points collected over time. If the sensor records only temperature, the resulting data is a *univariate time series* (UTS). If the sensor records multiple attributes, such as temperature, humidity, and air pressure, the resulting data is a *multivariate time series* (MTS). This example is visualized in the left panel of Figure 1.1, where we measure pairwise similarities between MTS (e.g., temperature, wind speed, and precipitation) of different cities.

2. **Relation-level extension:** We introduce techniques for discovering *multivariate* or *high-order relations* in *univariate data*, enabling the detection of complex patterns involving multiple data objects simultaneously. This is visualized in the right panel of Figure 1.1, where we measure the combined similarity between three UTS (e.g., temperature) of different cities.

Note that their combination – searching for *multivariate relations* between *multivariate objects* – represents the most complete form of similarity search, and should ultimately be the end goal of the field to solve efficiently. However, as even these partial extensions have been largely unexplored in literature, it is important to first understand their unique challenges and potential solutions to form a solid foundation for tackling the complete extension. Therefore, in this thesis, we focus on advancing both partial extensions independently, while keeping in mind their eventual unification. This step-by-step approach allows us to methodically tackle the increasing complexity while establishing the necessary theoretical and practical groundwork.

Similar to decades of research on classic similarity search, we take a holistic approach to these extensions, focusing on both the quality of the results (i.e., *effectiveness*) and the *efficiency* of the algorithms. Particularly, we will be (a) performing large-scale evaluations that investigate the best distance measures and preprocessing steps, (b) developing novel algorithms that yield results within time frames appropriate for their applications, and (c) conducting case studies that evaluate the practical applicability of our methods. Through these contributions, we aim to advance the field of similarity search by developing effective and efficient methods for multivariate similarity search that better reflect the inherent complexity of real-world phenomena.

This leads us to the central research question of this thesis:

> *Is it possible to extend the traditional similarity search paradigm to the multivariate setting, enabling the discovery of better and more meaningful patterns in datasets?*

1.3 Contributions & Outline

The main contributions of this thesis are organized into two parts, following the data-level and relation-level extensions introduced earlier.

Chapter 2 lays the groundwork for multivariate similarity search. It introduces essential notation, formalizes the two problem settings, and reviews existing research to contextualize our contributions.

Part I: Pairwise Similarity Search on Multivariate Time Series. This part focuses on the data-level extension, developing methods for comparing multivariate data objects. As the title of this part suggests, we focus on time series as a representative type of multivariate data. The reasoning behind this scope is twofold: (a) time series are the most common and widely studied form of multivariate data in similarity search literature [21, 24, 73, 118, 126, 136, 194, 213, 218, 219], and (b) the techniques developed for time series can often be generalized to other types of multivariate data with appropriate adaptations [73, 78]. The contributions in this part are as follows:

Chapter 3 presents a comprehensive evaluation of distance measures for MTS. It introduces a novel taxonomy based on normalization, temporal modeling, and channel dependency, and provides practical guidelines for selecting appropriate measures based on extensive empirical evaluations.

Chapter 4 introduces MS-Index, a novel and efficient algorithm for subsequence search in MTS. It significantly outperforms existing approaches while guaranteeing complete results, making multivariate subsequence search practical for large-scale applications.

Part II: Multivariate Similarity Search on Univariate Data. This part explores the relation-level extension, developing methods for detecting high-order relationships in datasets composed of univariate data objects. In contrast to Part I, which focused on time series data, this part is not limited to a specific data types, applying to any univariate data representation, such as vectors, sequences, or single-channel images. The contributions in this part are as follows:

Chapter 5 introduces Correlation Detective (CD), an efficient algorithm for detecting multivariate correlations in large univariate datasets. We provide theoretical analysis of the problem and present pruning strategies that make the search practical. We then extend CD to support streaming data through an incremental maintenance scheme. Through comprehensive evaluation on real-world applications, we demonstrate the effectiveness of both the static and streaming versions of the algorithm.

Chapter 6 generalizes the CD algorithm into a comprehensive, measure-agnostic framework for efficient multivariate similarity search, naming it the Similarity Detective (SD) framework. This framework shows how to efficiently answer *any* multivariate similarity search query by decomposing the problem into key properties and providing an optimization suite that adapts to different distance measures and query types. Through extensive evaluation, we demonstrate the framework's effectiveness across various similarity measures and real-world applications.

Chapter 7 presents a case study applying the SD framework to the sparse representation problem in signal processing and machine learning. It demonstrates that SD achieves competitive or superior performance compared to established methods in terms of representation error, sparsity level, classification accuracy, and computational efficiency.

Finally, **Chapter 8** summarizes our contributions and discusses their impact on the field. It identifies promising directions for future research and discusses open challenges in multivariate similarity search.

CHAPTER 2

Foundations of Multivariate Similarity Search

Research in similarity search over the past three decades has resulted in an established set of foundational techniques, including a rich variety of distance measures, dimensionality reduction methods, and specialized indexing structures. This chapter provides an overview of these core concepts as they form the groundwork for our extension into the multivariate domain. It establishes the necessary background and terminology used throughout this thesis, from formal definitions of multivariate data to the key algorithms and data structures that enable efficient search. Additionally, we position our work within the broader research landscape by discussing related work in both similarity search on multivariate time series and multivariate similarity search on univariate data.

2.1 Definitions and Notation

We now introduce the notation and definitions used throughout this thesis. See Table 2.1 for an overview of the full nomenclature.

Data Objects. A data object $\boldsymbol{x}$ represents any piece of data that captures a specific entity or phenomenon over which we can define meaningful distance or similarity measure, meaning that we can reason about whether these entities are similar or not. A dataset of n data objects is denoted as $D = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$.

Data objects can be multivariate, meaning they are composed of c individual data objects (called *channels*), with each channel covering one variable or aspect of the measured entity. An object is univariate when $c = 1$, and multivariate when $c > 1$. For multivariate data objects, the j-th channel is denoted as $\boldsymbol{x}^{(j)}$.

Vector Data. The most common representation of a data object is through a real-valued vector $\boldsymbol{x} \in \mathbb{R}^m$, where m is the dimensionality of the vector, also denoted as $|\boldsymbol{x}|$. A mul-

tivariate vector is a collection of c vectors, represented as a matrix $\boldsymbol{x} \in \mathbb{R}^{c \times m}$. Common examples include images (where channels represent color components), text embeddings (where channels represent different semantic aspects), and audio signals (where channels represent different frequency bands).

Time Series. Time series are a special case of vector data where the values are ordered by time. For a time series $\boldsymbol{t}$, each element t_i represents an observation at time step i. A unique property of sequence data such as time series is that we can define meaningful subsequences: a subsequence of length l starting at timepoint o is denoted as $\boldsymbol{t}_{o,l} = [\boldsymbol{t}_o, \ldots, \boldsymbol{t}_{o+l-1}]$, where $1 \leq o \leq |\boldsymbol{t}|$ and $1 \leq l \leq |\boldsymbol{t}| - o + 1$. By definition of the general data object, a univariate time series (UTS) has $c = 1$, while a multivariate time series (MTS) has $c > 1$ channels, with each channel representing a different variable observed over time (e.g., temperature, humidity, and wind speed recorded by a weather station).

Distance Measures. The distance between two data objects $\boldsymbol{x}$ and $\boldsymbol{y}$ is denoted as $d(\boldsymbol{x}, \boldsymbol{y})$, with $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$, where $\mathcal{X}$ is the space of data objects. In contrast to distance measures, similarity measures $sim(\boldsymbol{x}, \boldsymbol{y})$ quantify how similar two objects are, with $sim : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_0^+$. While distance measures assign low values to similar objects, similarity measures assign high values to similar objects. Similarity measures can often be derived from distance measures through transformations, such as $sim(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{d(\boldsymbol{x}, \boldsymbol{y})+1}$ [163].

Similarity Search Queries. There are two types of similarity search queries: k-Nearest Neighbors (k-NN) queries and r-range queries. We will now present them in their classical form, meaning that they consider pairwise distances between univariate objects. Afterwards, we discuss how each of these definitions is extended to the two multivariate settings we consider in this work.

A k-NN query aims to identify the k data objects within a dataset D that exhibit the smallest distances from a query object $\boldsymbol{q}$ [81].

> **Definition 1 (k-Nearest-Neighbor Query).** Given an integer k and a query object $\boldsymbol{q}$, a k-NN query retrieves the set of objects $R \subseteq D$ s.t. any object $\boldsymbol{t} \notin R$ has a distance from the query $\boldsymbol{q}$ greater than the distance of $\boldsymbol{q}$ with any object in R. Formally, $R = \{\{\boldsymbol{t}_{R_1} \cdots \boldsymbol{t}_{R_k}\} \subseteq D \mid \forall_{\boldsymbol{t}_R \in R, \boldsymbol{t}_{R'} \notin R} d(\boldsymbol{q}, \boldsymbol{t}_R) \leq d(\boldsymbol{q}, \boldsymbol{t}_{R'})\}$.

An r-range query, alternatively, identifies all data objects in D whose distance from a query object $\boldsymbol{q}$ falls below a specified threshold r [81].

> **Definition 2 (r-Range Query).** Given a distance threshold r and a query object $\boldsymbol{q}$, an r-range query retrieves the set of data objects $R \subseteq D$ s.t. the distance between $\boldsymbol{q}$ and any object $\boldsymbol{t} \in R$ is less than or equal to r. Formally, $R = \{\boldsymbol{t} \in D \mid d(\boldsymbol{q}, \boldsymbol{t}) \leq r\}$.

Alternatively to the definitions above where the query includes a specific query object, the query can also be defined to find all high-similarity combinations (i.e., pairs in this case)

Table 2.1: Nomenclature

$\boldsymbol{x}$	A data object
$\boldsymbol{q}$	Query object
m	Dimensionality of data object (e.g., length of time series)
$\boldsymbol{x}^{(j)}$	Channel j of $\boldsymbol{x}$
$\boldsymbol{x}_i^{(j)}$	Value i of channel j of $\boldsymbol{x}$
$\boldsymbol{t}_{o,l}$	Subsequence of a time series $\boldsymbol{t}$ of length l starting at index o
$\tilde{\boldsymbol{t}}$	Approximation of $\boldsymbol{t}$
$\hat{\boldsymbol{t}}$	Normalized version of $\boldsymbol{t}$
D	A dataset of data objects
n	Number of objects in the dataset D
c_D	Total number of channels in the dataset
$c_{\boldsymbol{q}}$	The set of channels of an object $\boldsymbol{q}$
$d(\boldsymbol{x}, \boldsymbol{y})$	Distance between objects $\boldsymbol{x}$ and $\boldsymbol{y}$
$sim(\boldsymbol{x}, \boldsymbol{y})$	Similarity between objects $\boldsymbol{x}$ and $\boldsymbol{y}$

within a dataset. A good example of this is correlation analysis, where the goal is to find all pairs of objects with a correlation above a certain threshold, in order to create a correlation network [10]. As such, a k-NN query can be generalized to a top-k query, which retrieves the k most similar pairs of objects in a dataset.

Definition 3 (Top-k Query). Given an integer k and a query object $\boldsymbol{q}$, a top-k query retrieves the set R of pairs of objects in D s.t. any pair of objects $(\boldsymbol{x}, \boldsymbol{y}) \in R$ with $\boldsymbol{x} \neq \boldsymbol{y}$ has a distance smaller or equal to the distance of any pair of objects not in R. Formally, $R = \{\{(\boldsymbol{x}_{R_1}, \boldsymbol{y}_{R_1}) \cdots (\boldsymbol{x}_{R_k}, \boldsymbol{y}_{R_k})\} \subseteq (D \times D) \mid \forall_{(\boldsymbol{x},\boldsymbol{y}) \in R, (\boldsymbol{x}',\boldsymbol{y}') \notin R} d(\boldsymbol{x}, \boldsymbol{y}) \leq d(\boldsymbol{x}', \boldsymbol{y}')\}$.

Similarly, an r-range query can be generalized to a threshold query, which retrieves all pairs of objects in a dataset whose distance falls below a specified threshold τ.

Definition 4 (Threshold Query). Given a distance threshold τ and a query object $\boldsymbol{q}$, a threshold query retrieves the set R of pairs of objects from D s.t. the distance between any pair of objects $\boldsymbol{x}, \boldsymbol{y} \in R$ with $\boldsymbol{x} \neq \boldsymbol{y}$ is less than or equal to τ. Formally, $R = \{(\boldsymbol{x}, \boldsymbol{y}) \in D \times D \mid d(\boldsymbol{x}, \boldsymbol{y}) \leq \tau \wedge \boldsymbol{x} \neq \boldsymbol{y}\}$.

It is worth noting that while the presented definitions follow the literature convention of using *distance* functions, they can be trivially adapted to work with *similarity* functions instead. This adaptation involves substituting the distance function d with a similarity function s and reversing the inequality operators. Specifically, while k-NN and top-k queries seek minimal distances, their similarity-based counterparts search for maximal similarities. Likewise, r-range and threshold queries that identify distances below a threshold transform into queries that find similarities above a threshold.

Orthogonal to the distinction between top-k and threshold-based queries, similarity search on time series can also be categorized based on the type of matching performed between the

query and candidate time series; whole-search or subsequence search [81]. Whole-search evaluates the distance between complete time series, requiring both query and candidate series to have identical lengths [81].

Definition 5 (Whole-Search Query). A whole-search query retrieves the set R of time series in D similar to a query series $\boldsymbol{q}$, with $|\boldsymbol{t}| = |\boldsymbol{q}|$ for all $\boldsymbol{t} \in R$.

In contrast, subsequence search considers the distances between an entire query series and all $|\boldsymbol{q}|$-length subsequences of a candidate series [81]. This case allows the time series from which the candidate subsequences are extracted to be longer than the query.

Definition 6 (Subsequence Search Query). A subsequence search query retrieves the set R of subsequences $\boldsymbol{t}_{o,l}$ of time series $\boldsymbol{t} \in D$ similar to a query series $\boldsymbol{q}$, with $l = |\boldsymbol{q}|$ and $1 \leq o \leq |\boldsymbol{t}| - |\boldsymbol{q}| + 1$.

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 [81]. Below, we extend the definitions above to the two multivariate settings introduced in Section 1.2.

2.1.1 Pairwise Similarity Search on Multivariate Time Series (Part I)

The similarity search queries defined above can be directly extended to multivariate objects by using appropriate distance measures that handle multiple channels. When working with multivariate objects ($c > 1$), all definitions remain identical, with the only modification being that the distance function d must be capable of comparing those multivariate objects. For instance, when using Euclidean distance, a multivariate extension of the Euclidean distance would be applied instead of the standard univariate version. We discuss such multivariate distance measures for time series in detail in Chapter 3.

2.1.2 Multivariate Similarity Search on Univariate Data (Part II)

Multivariate similarities (also known as high-order similarities) are a generalization of pairwise similarities that consider relationships between multiple entities, in this case, multiple univariate data objects. While pairwise similarity considers how close two objects are to each other, multivariate similarity examines the more complex relationships and interdependencies between multiple objects that may not be visible when considering only pairs (e.g., in time series data, ice cream sales only go up when temperature is high *and* precipitation is low). The goal of multivariate similarity search is to identify groups of two or more objects that demonstrate strong collective relationships, which can reveal complex interactions in the underlying processes that generated the data.

This extension requires modifying the traditional similarity search definitions in two key aspects: (1) the similarity function must operate on sets of objects rather than individual objects, and (2) the query results must return sets instead of individual matches. Formally,

we denote the set of all subsets of D of size at most p as the power set $\mathcal{P}_{\leq p}(D) = \{G \subseteq D \mid 2 \leq |G| \leq p\}$.

To illustrate this concept, consider a multivariate similarity measure that first computes the element-wise average of two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, and then calculates the correlation between this average and a third vector $\boldsymbol{c}$. This measure can be expressed as $sim((\boldsymbol{a}, \boldsymbol{b}), \boldsymbol{c}) = \text{corr}(\frac{\boldsymbol{a}+\boldsymbol{b}}{2}, \boldsymbol{c})$. In this case, the group consists of all three vectors, but there is a specific order in which they are combined: the first two vectors are averaged before comparing them with the third. This ordered combination is essential for some multivariate similarity measures, while for others, such as the multipole measure [10], the order is irrelevant. The multipole measure, for instance, quantifies the linear dependence of a set of vectors based on the eigenvalues of their matrix representation, treating all vectors in the group uniformly.

We capture this difference in order sensitivity though distinguishing between two types of multivariate similarity measures: (1) *one-sided* measures $sim(\cdot)$, which take in a single set of objects and measures their collective relationship, and (2) *two-sided* measures $sim(\cdot, \cdot)$, which take in two sets of objects and measure how similar the sets are to each other (often by comparing some aggregate representation of each set). The example provided earlier can be classified as a two-sided measure, as it compared two sets of objects $\{\boldsymbol{a}, \boldsymbol{b}\}$ and $\{\boldsymbol{c}\}$.

We now present the definitions for multivariate top-k and threshold queries under both one-sided and two-sided measures, noting that the corresponding k-NN and r-range definitions can be derived by constraining the first object in the left-most set to be the query object $\boldsymbol{q}$. Note that, for these definitions, we use similarity measures rather than distance measures, as similarity measures naturally extend to multiple arguments, whereas distance measures are traditionally defined as binary functions which could lead to confusion.

Definition 7 (One-Sided Multivariate Top-k Query). Given an integer k and a maximum set size p, a one-sided multivariate top-k query retrieves the set R of sets of data objects G from the power set $\mathcal{P}_{\leq p}(D)$ such that any other set of objects $G' \notin R$ has a similarity smaller than the similarity of any set of objects in R. Formally, $R = \{\{G_1, \ldots, G_k\} \subseteq \mathcal{P}_{\leq p}(D) \mid \forall_{G \in R, G' \notin R} sim(G') \leq sim(G)\}$.

Definition 8 (One-Sided Multivariate Threshold Query). Given a similarity threshold τ and a maximum set size p, a one-sided multivariate threshold query retrieves the set R of sets of data objects G from the power set $\mathcal{P}_{\leq p}(D)$ such that the similarity in any set of objects $G \in R$ is at least τ. Formally, $R = \{G \in \mathcal{P}_{\leq p}(D) \mid sim(G) \geq \tau\}$.

Definition 9 (Two-Sided Multivariate Top-k Query). Given an integer k and maximum left set size p_l and right set size p_r, a two-sided multivariate top-k query retrieves the set R of pairs of sets of data objects (G, H) with $G \in \mathcal{P}_{\leq p_l}(D)$ and $H \in \mathcal{P}_{\leq p_r}(D)$ such that any other pair of sets of objects $(G', H') \notin R$ has a similarity smaller than the similarity of any pair of sets of objects in R. Formally, $R = \{\{(G_1, H_1), \ldots, (G_k, H_k)\} \subseteq \mathcal{P}_{\leq p_l}(D) \times \mathcal{P}_{\leq p_r}(D) \mid \forall_{(G,H) \in R, (G',H') \notin R} sim(G', H') \leq sim(G, H)\}$.

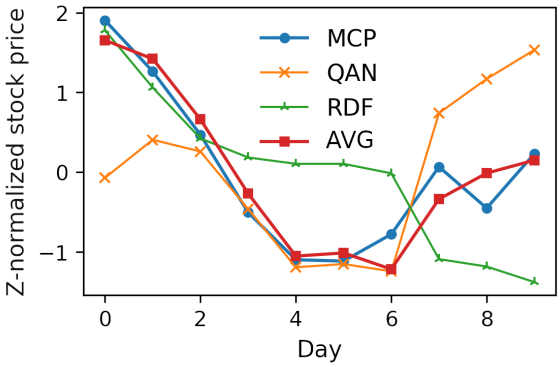


Figure 2.1: Normalized daily closing prices for stocks traded at the Australian Securities Exchange

Definition 10 (Two-Sided Multivariate Threshold Query). Given a similarity threshold τ and maximum left set size p_l and right set size p_r, a two-sided multivariate threshold query retrieves the set R of pairs of sets of data objects (G, H) with $G \in \mathcal{P}_{\leq p_l}(D)$ and $H \in \mathcal{P}_{\leq p_r}(D)$ such that the similarity in any pair of sets of objects $(G, H) \in R$ is at least τ. Formally, $R = \{(G, H) \in \mathcal{P}_{\leq p_l}(D) \times \mathcal{P}_{\leq p_r}(D) \mid sim(G, H) \geq \tau\}$.

Intuitively, one might assume that high multivariate similarities always require high pairwise similarities between the members of a set. This would suggest that sets could be identified post-hoc by analyzing the results of pairwise similarity search queries through clustering or community detection methods [91]. However, this assumption only holds when the multivariate similarity measure is merely a sum of pairwise similarities, which is often not the case in practice.

To illustrate this, let us return to the earlier example of a multivariate similarity measure that combines $\boldsymbol{a}$ and $\boldsymbol{b}$ through averaging before comparing with $\boldsymbol{c}$. In this case, even if $\boldsymbol{a}$ and $\boldsymbol{b}$ individually show no correlation with $\boldsymbol{c}$, their combined signal might strongly correlate with $\boldsymbol{c}$. A strong example of this is shown in Figure 2.1, which presents the stock prices (i.e., time series) of three companies traded at the Australian Securities Exchange. In this example, the pairwise Pearson correlation ρ between any two of the three time series is relatively low (i.e., $\rho(\text{MCP}, \text{QAN}) = 0.41$, $\rho(\text{MCP}, \text{RDF}) = 0.56$, $\rho(\text{QAN}, \text{RDF}) = -0.47$). Yet, when we average the time series of QAN and RDF, the resulting time series AVG shows a strong correlation with MCP ($\rho(\text{AVG}, \text{MCP}) = 0.95$).[1] This emergence of collective patterns that are not visible in pairwise comparisons exemplifies the non-linear nature of multivariate similarities and represents one of the fundamental challenges in multivariate similarity search, which we explore extensively in Part II of this thesis.

[1] Weighted averages of stock prices are commonly considered in risk management to evaluate portfolio performance, diversity, and volatility [186].

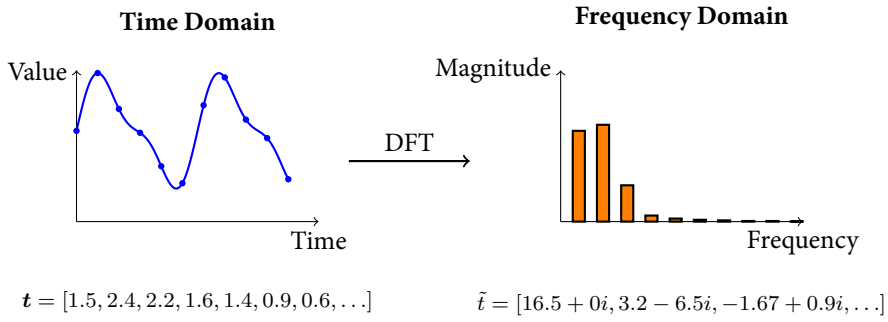


Figure 2.2: DFT transformation example: A time series (left) decomposed into its frequency components (right), showing the magnitude of the first 10 DFT coefficients.

2.2 Time Series Approximation Methods

The computational cost of similarity search grows rapidly with the dimensionality of the data objects, e.g., with the length of the vectors or time series [7, 18]. Time series, specifically, tend to be extremely high-dimensional as they often span long periods of time with data collected every few seconds (or even milliseconds in the case of some sensors) [45]. As distance measures typically operate over all these dimensions, computation can become prohibitively expensive. Furthermore, the high dimensionality of time series data can lead to the curse of dimensionality, where the volume of the space increases so much that the available data becomes sparse. This sparsity makes it difficult to find meaningful patterns and increases the likelihood of spurious results [18].

To address this challenge, researchers have developed various dimensionality reduction techniques that transform time series into compact, lower-dimensional representations while preserving essential characteristics for similarity comparison. These approximation methods generally fall into two categories: techniques that preserve time-domain characteristics (e.g., PAA [132], SAX [151]) and those that capture frequency-domain properties (e.g., DFT [88], SFA [216]). The choice of approximation method depends on the specific requirements of the application, such as the type of patterns to be preserved, the desired compression ratio, and characteristics of the original data.

In this section, we examine an approximation technique that has significantly influenced the field: Discrete Fourier Transform (DFT) approximation [88, 177, 203, 222]. This technique continues to serve as a building block for modern time series analysis systems (including our work), and has spawned numerous variants and optimizations [54, 84, 132, 147, 151, 156, 160, 164, 240, 247].

DFT Approximation. DFT approximation is a fundamental technique for dimensionality reduction in time series analysis [88, 177, 203, 222]. It transforms a time series from the time domain into the frequency domain by decomposing it into a sum of sinusoidal components, where each component represents a specific frequency in the original signal.

Given a UTS $\boldsymbol{t} \in \mathbb{R}^m$, the DFT produces a complex vector of equal length, where each coefficient represents both the amplitude and phase of a sinusoid at a particular frequency. These coefficients are ordered by increasing frequency, with the first coefficients capturing the low-frequency (slow-changing) components and later coefficients representing high-frequency (rapid) fluctuations. Figure 2.2 illustrates this transformation: the time domain signal (left) is decomposed into its constituent frequencies (right), where the magnitude of each DFT coefficient reveals the strength of that frequency component in the original signal.

The key insight behind DFT approximation is that most real-world time series can be accurately represented using only their low-frequency components [88]. By retaining only the first f DFT coefficients (typically $f \leq 3$ for many applications [88]), we can create a compact approximation $\tilde{\boldsymbol{t}} \in \mathbb{C}^f$ that preserves the essential characteristics of the original series, as demonstrated in Figure 2.3.

A crucial property that makes DFT approximation particularly valuable for similarity search is that it provides a lower bound on the Euclidean (L_2) distance between time series [177]. Specifically, for two UTS $\boldsymbol{t}, \boldsymbol{q} \in \mathbb{R}^m$ with DFT approximations $\tilde{\boldsymbol{t}}, \tilde{\boldsymbol{q}} \in \mathbb{C}^f$:

$$d(\boldsymbol{t}, \boldsymbol{q}) \geq \sqrt{\frac{\sum_{i=1}^{f} |\tilde{\boldsymbol{t}}_i - \tilde{\boldsymbol{q}}_i|^2}{m}} = \frac{d(\tilde{\boldsymbol{t}}, \tilde{\boldsymbol{q}})}{\sqrt{m}} \tag{2.1}$$

This lower bound becomes tighter as f increases, eventually converging to the exact distance when all coefficients are included. To illustrate, note that the green line in Figure 2.3 – composed of the first $f = 10$ DFT coefficients – more closely follows the original time series (blue) than the orange line, which uses only the first $f = 3$ coefficients. Consequently, lower-bounding the distance to the original time series with an approximation of $f = 10$ will be more accurate than with $f = 3$. This lower-bounding property enables efficient similarity search by first comparing the low-dimensional DFT approximations and only computing exact distances when the lower bound distance is below a certain threshold.

For MTS, the DFT approximation extends naturally by applying the transform independently to each channel, resulting in an $f \times c$ dimensional approximation. This channel-wise application preserves the lower-bounding property while reducing the dimensionality of each component series.

2.3 Indexes and Data Structures

Even with efficient approximation methods, searching through large collections of time series remains computationally challenging when comparing the query directly against every candidate sequence in the database. To address this challenge, specialized index structures organize the data in ways that enable efficient pruning of the search space. These structures often leverage lower bounds on distances to eliminate large portions of the dataset from consideration without computing exact distances, significantly reducing the computational cost of similarity queries while guaranteeing no false dismissals. Notice that the usage of lower bounds creates a clear interplay with the approximation methods discussed in Section 2.2; approximations that come with a lower bounding property are particularly well-suited to be

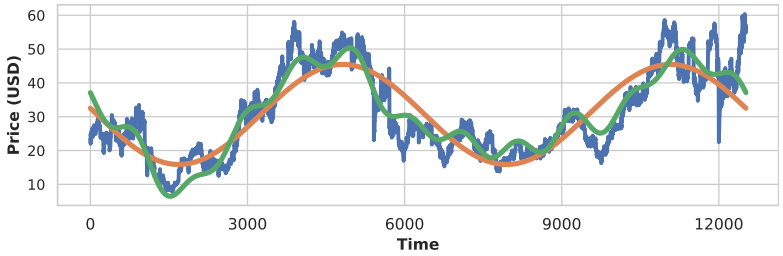


Figure 2.3: The price of a stock over time (blue), and its reconstruction through its first three DFT coefficients (orange), and first ten DFT coefficients (green).

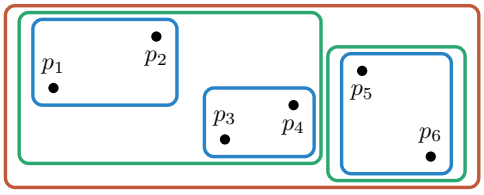


Figure 2.4: An example of an R-tree with 6 data points $p_1, \ldots, p_6$. The tree has 3 levels, with the root node containing all data points.

used in conjunction with index structures. This synergy between approximation methods and index structures is crucial: the tighter the lower bound provided by the approximation, the more effective the index can be at pruning the search space. In this section, we examine one particularly influential index structure: the R-tree.

2.3.1 R-Trees

An R-tree is a tree-based index structure that is commonly used for indexing spatial data [28, 107]. The key idea behind the R-tree is to group nearby objects into Minimum Bounding Rectangles (MBRs) that tightly enclose all objects in the group. The MBRs are then recursively grouped into smaller MBRs, forming a tree structure where each node represents a group of objects [107]. An example of an R-tree can be seen in Figure 2.4. R-trees can be constructed either top-down or bottom-up. Top-down construction involves starting with a single MBR that contains all the data points, and then recursively splitting the MBRs into smaller MBRs until a certain threshold is reached. The balance of the tree and the quality of the partitioning is determined by the choice of the split strategy, for which many were proposed in literature such as the Quadratic Split [107], the Linear Split [107], and the R*-tree topological split strategy [28]. Bottom-up construction (or *bulk loading*), on the other hand, starts with the data points as individual MBRs, and then merges them into larger MBRs based on a certain *leaf size* L and partitioning strategy. Bottom-up construction generally results in less overlap between MBRs, but it requires the data to be completely known in advance. A popular bulk loading algorithm is the *Sort-Tile-Recursive* (STR) algorithm [143]. This algorithm constructs nodes by sorting the children based on their coordinates in each dimension (using the middle in case the children are rectangles instead of

points), and then recursively partitioning 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 would partition 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

This section establishes the context of our work within the broader research landscape. We first examine the emerging field of MTS similarity search (i.e., Part I), followed by a discussion of multivariate similarity measures for univariate data (i.e., Part II). Note that here we focus on the overarching themes and approaches that shape the field; specific related work for each subproblem is addressed in their respective chapters.

2.4.1 Pairwise Similarity Search on Multivariate Time Series (Part I)

Similarity search on MTS has become an emerging topic in time series analysis throughout the last few years, as evidenced by several recent works in the field. The common motivation for these works is the recognition that the focus of time series research has been predominantly on UTS, with MTS receiving considerably less attention, despite their prevalence in real-world applications. Furthermore, they argue that the existing body of research on UTS is not directly applicable to MTS, and that the latter requires its own research efforts.

For instance, Bagnall et al. [24] noted that the field of MTS classification is in a position similar to where UTS classification was a decade ago, with evaluations typically limited to few datasets and lacking statistical rigor. To address this, they introduced the UEA MTS classification archive. Building on this foundation, Ruiz et al. [211] conducted the first large-scale experimental evaluation of MTS classification algorithms. Furthermore, in the domain of distance measures, Yekta et al. [219], Shifaz et al. [218], and Yang et al. [243] pointed out that virtually all attempts to improve time series methods in the last two decades focused on the univariate case, with most researchers incorrectly assuming that generalization to multivariate cases was trivial. In response, these works demonstrated that the generalization of univariate distance measures to the multivariate case is not straightforward, and subsequently proposed extensions of existing univariate distance measures as well as new distance measures specifically designed for MTS. In the domain of indexes, Vlachos et al. [230] and Yu et al. [247] extended usage of the R-tree and the Locality Sensitive Hashing (LSH) framework to MTS similarity search through modifications to the representation of the inserted data. Lastly, in the context of motif discovery, Yeh et al. [244] demonstrated that previous attempts to generalize motif discovery to multidimensional cases by finding patterns across all dimensions simultaneously were too constraining and would not produce meaningful results except in contrived situations.

While each of these works makes valuable contributions to their respective subdomains, their focused nature reveals a concerning fragmentation in the field of MTS research. Each study approaches its specific subdomain—be it individual distance measures, indexing solutions, or motif discovery—largely in isolation, with limited discussion of how their insights

might generalize to other problems or domains. This siloed approach risks creating incompatible solutions and duplicating efforts, potentially slowing the overall progress of the field.

In this thesis, we aim to take a more holistic approach to MTS similarity search by developing solutions in a more systematic and transferable manner. Specifically, by individually addressing the two core aspects of similarity search – effectiveness (normalization and distance measures) and efficiency (search algorithms) – we first conduct thorough literature reviews to identify the current state of the art and then systematically develop solutions that can be adapted across different subproblems and domains. Additionally, through careful analysis and reflection of each work, we aim to extract broader insights that can inform researchers and practitioners on how to adapt existing univariate solutions to the multivariate domain. In that, we aim to both advance the current state of the art in MTS similarity search and provide a foundation for future research that can build on our findings.

2.4.2 Multivariate Similarity Search on Univariate Data (Part II)

In contrast to similarity search on MTS, which has emerged as a distinct research area, multivariate similarity search—the search for high-order relationships between multiple univariate objects—remains largely unexplored. However, the concept has been extensively discussed across various scientific domains, albeit under different names and with domain-specific definitions.

For instance, in physics, Santoro et al. [215] explored what they term "high-order dependencies" in time series, focusing on characterizing group-wise temporal relationships. In neuroscience, Agrawal et al. [10] investigated "multivariate relationships" using correlation networks. The concept appears in biology and medicine as "epistasis," where Carlborg et al. [43] and Mitra et al. [172] studied gene-gene interactions in complex traits and autism spectrum disorders, respectively. In meteorology, Liess et al. [150] examined multivariate relationships under the term "teleconnections," while in finance, Chen et al. [48] and Perlin et al. [199] explored multi-pairs trading strategies. We further discuss these works in Chapter 6.

These domain-specific approaches share similar limitations to those we observed in MTS research. Each field has developed its own specialized solutions, working in isolation from others and often reinventing similar concepts under different names. Moreover, these solutions typically emerge from outside the computer science community and thus often fail to leverage decades of progress in efficient similarity search techniques. For example, Santoro et al.'s [215] approach to computing high-order correlations, while novel in its domain, proves computationally inefficient for large-scale applications.

This thesis aims to bridge this gap by providing a unified framework for multivariate similarity search. Rather than treating each domain's solution as a separate case, we develop a general definition that aligns with the established literature on similarity search while also covering the various domain-specific requirements. By elevating the discussion from domain-specific implementations to a general framework, we aim to offer more systematic and transferable solutions.

I

Pairwise Similarity Search on Multivariate Time Series

CHAPTER 3

A Structured Study of Distance Measures for Multivariate Time Series

How do we measure similarity between multivariate time series in a meaningful and effective way? This question, simple in concept, shows to be profoundly complex when dealing with time series that suffer from a variety of distortions, such as temporal misalignments, scale differences, and noise. Appropriately handling these distortions is crucial for achieving high-quality results in similarity search on multivariate data. This chapter confronts this challenge by systematically evaluating the tools we use to handle distortions and quantify similarities: distance measures. We move beyond simply listing measures and introduce a structured framework for understanding them, organized along three axes: how they normalize data, model time, and handle dependencies between channels. Through an extensive empirical evaluation, we test these measures to reveal what truly works, debunking long-standing misconceptions and culminating in a practical set of guidelines for researchers and practitioners working with multivariate time series. As we will see, the intuitions built on simple, univariate data do not always hold true in the multivariate setting.

The contents of this chapter have previously appeared in d'Hondt et al. [73, 80] and Paparrizos et al. [192].

3.1 Introduction

The quality of the results of a similarity search task is highly dependent on the distance measure used to compare the time series. The design of such measures is non-trivial though, as — in contrast to other data types (e.g., text) — time series data has a temporal aspect that plays a crucial role in effective comparison [81]. Furthermore, as noted in Chapter 1, the dif-

ficulty in formulating a suitable distance measure stems from the lack of a clear formulation of what actually makes up a meaningful similarity [193]. Namely, as humans, we can easily identify perceptually similar time series by ignoring a variety of *distortions* in the series, such as temporal misalignments, scaling differences, and noise irrelevant to the comparison. However, the subliminal concepts guiding this process are complex and context-dependent, making them difficult to mathematically formalize [86].

The difficulty of handling various distortions in time series has resulted in the development of dozens of distance measures over decades of research. Paparrizos et al. [193] introduced a taxonomy categorizing measures into five groups: (a) *lock-step* measures, which compare values at corresponding time points and aggregate the differences across time, (b) *sliding* measures, which find the optimal shift between two time series, (c) *elastic* measures, which allow one-to-many mappings of time steps to handle local temporal distortions, (d) *kernel* measures, which implicitly project time series into a higher-dimensional space using a kernel function, and (e) *embedding* measures, which explicitly transform time series to new representations that then serve as the new basis for comparison. Paparrizos demonstrates that no single measure can effectively address all types of distortions simultaneously. This becomes even more challenging in the context of MTS, as distortions may or may not be propagated across channels [219]. For instance, in motion capture data, a phase shift in one joint's movement might affect multiple related joints, while in medical time series, an artifact in one sensor reading might be independent of other measurements [228]. Moreover, channels can exhibit complex correlations that evolve over time, making it difficult to determine whether similarities should be assessed independently per channel or jointly across all channels [219]. These additional complexities have led to various approaches for handling channel dependencies in MTS measures, from treating channels independently to incorporating cross-channel correlations through sophisticated mathematical frameworks.

Despite extensive efforts in developing distance measures, misconceptions about their properties and the trade-offs between accuracy and efficiency persist, due to the limited focus on performing comprehensive evaluation studies [193]. Recent works managed to debunk several of these misconceptions by performing large-scale evaluations across a variety of datasets and measures [74, 193]. Unfortunately, these evaluations were restricted to UTS, leaving the challenges associated with MTS under-explored. The handful of existing studies on distances for MTS have key limitations [24, 218], including (a) mainly focusing on lock-step and elastic measures while disregarding competing families such as sliding and kernel measures, (b) considering only a single normalization, and (c) performing limited statistical analysis. With these limitations, the guidance these studies offer remains limited.

Motivated by these issues, in this chapter we aim to provide a rigorous comparison of MTS distance measures. Namely, to accurately map the current landscape of measures, we take a systematic approach that considers three key axes: (a) ***Normalization***, which involves the rescaling of time series as a preprocessing step, (b) the ***Temporal Model***, which determines how measures address temporal distortions between time series, and (c) the ***Channel-Dependency Model***, which determines the criteria to address correlations between channels, either by assuming independence across all channels (*channel-independent*) or incorporating interdependencies (*channel-dependent*). These three axes serve as the basis for categorizing

Table 3.1: Summary of our experimental evaluation of 30 standalone MTS measures (46 variants in total when considering channel-dependency models and ensembles) across two channel-dependency models (I: independent, D: dependent) and three downstream tasks (CLS: classification, CLU: clustering, AD: anomaly detection). The last two columns show the summary of prior evaluation studies [24] and [218].

Temp. Model	Measures	Dep. Models, #Norms	Downstream Tasks	[24]	[218]
Lock-step	11	(I, 13)	CLS, CLU, AD	1 | (I,1) | CLS	1 | (I,1) | CLS
Sliding	1	(I & D,13)	CLS, CLU, AD	✗	✗
Elastic	5	(I & D, 13)	CLS, CLU, AD	1 | (I&D,1) | CLS	5 | (I&D,1) | CLS
Kernel	4	(I & D, 13)	CLS	✗	✗
Feature	2	(I, 13)	CLS	✗	✗
Model	2	(I & D, 13)	CLS	✗	✗
Embedding	5	(I & D, 13)	CLS	✗	✗
Ensemble	2	(I & D, 13)	CLS	✗	✗

and comparing existing MTS distance measures, facilitating the selection and discovery of effective preprocessing techniques and new measure designs.

Guided by these three key axes, we present the most comprehensive study of MTS distance measures to date, where we implement, parameterize, and evaluate a total of 30 standalone MTS measures. Furthermore, we (a) conduct a meta-analysis on 13 normalizations, to determine whether existing methods outperform the case of not normalizing (Nonorm), (b) propose a taxonomy of eight categories (seven temporal models + measure ensembles), and (c) explore two channel-dependency models, providing guidelines on how to implement each model for each measure category and normalization method. Lastly, as a byproduct of the study, we provide an easy-to-use library with implementations of all evaluated measures [5].

To quantify the discriminative power of each measure, we evaluate their accuracy in the one-nearest-neighbor (1NN) classification task across all 30 datasets from the UEA archive [24], under both supervised and unsupervised parameter settings. We also perform two additional experiments on clustering and anomaly detection to extend our findings beyond classification. Alongside accuracy comparisons, we evaluate measures on their runtime performance across varying time series lengths and numbers of channels in MTS. We validate our results and findings using two statistical tests to assess the significance of differences, one for pairwise comparisons and the other for evaluating global performance across multiple measures.

In summary, the results reaffirm previous findings on distance measures in the context of MTS, confirming that: (a) Z-score normalization is not the best normalization technique, (b) various lock-step measures outperform Euclidean distance, and (c) newer elastic measures outperform Dynamic Time Warping (DTW) [212] in the supervised setting. Furthermore, our results show that: (a) sliding measures offer the best trade-off between accuracy and runtime efficiency on tasks where distortions obscure effective comparison (i.e., classification, clustering), (b) lock-step measures are the best choice for tasks where distortions are the main

concern (i.e., anomaly detection), (c) embedding measures, including deep-learning-based methods, are not superior to traditional measures, (d) current normalization techniques do not significantly improve accuracy on multivariate data, and (e) channel-dependent variants of measures generally outperform their channel-independent counterparts, with the exception of elastic measures. Table 3.1 compares statistics of our experimental evaluation against related studies [24, 218], highlighting the key observations that motivated this research and demonstrating the comprehensiveness of our work.

We start with the related work in the field of distance measure evaluations (Section 3.2). Then, we present our contributions:

- We present three axes that ensure the comprehensiveness of our evaluations on MTS measures: normalization, temporal model, and channel-dependency model (Section 3.3).
- We provide a complete overview of existing measures, structured by the introduced measure properties (Section 3.4).
- We conduct an evaluation of 30 standalone measures across 30 datasets with 13 normalization techniques, measuring their performance in terms of accuracy and runtime efficiency on three downstream tasks: classification, clustering, and anomaly detection (Section 3.5).
- We provide guidelines for MTS measures selection (Section 3.6).

Finally, we conclude with the implications of our work and a discussion of challenges and new directions (Section 3.7).

3.2 Related Work

In this section, we review recent experimental studies on time series distance measures, emphasizing the relevance of our work and illustrating how it aligns with, or challenges prior research. Additional details on individual distance measures within each category are presented in Section 3.4.

Recently, Paparrizos et al. [193] introduced a taxonomy that classifies univariate distance measures into five distinct categories and conducted a comprehensive comparison to challenge long-standing misconceptions in the field. The experimental results yielded four key conclusions: (a) alternative normalization methods can outperform Z-score normalization, (b) other lock-step distance measures surpass the widely used Euclidean distance, (c) sliding measures demonstrate competitive performance compared to elastic measures in supervised and unsupervised settings, and (d) DTW is not always the best-performing elastic measure, with newer measures like Move-Split-Merge (MSM) outperforming it. Inspired by this work, we extend this taxonomy to MTS, introduce three new categories, and propose a principled approach to extending UTS distance measures to MTS. Our study partly confirms the four main conclusions of the study of [193] in the multivariate case, but also uncovers several discrepancies and novel findings: (a) Not normalizing (Nonorm) is currently the best choice for MTS, (b) elastic measures for MTS *only* outperform sliding measures under supervised parameter tuning, and (c) state-of-the-art deep-learning-based measures are not superior

to traditional measures. Furthermore, our study additionally considers ensemble methods and includes two extra downstream tasks on clustering and outlier detection to extend our findings beyond classification.

Shokoohi-Yekta et al. [219] proposed two extension strategies for extending DTW to the multivariate case: *channel-dependent* (DTW-D) and *channel-independent* (DTW-I) strategies. In DTW-D, all channels are treated collectively and share a single warping path, whereas in DTW-I, each channel is treated independently with its own warping path. The authors demonstrate the relevance of each variant through real-world examples of RGB images (i.e., matrices with three channels). Namely, the authors address the case of uneven color fading in manual illustrations in sixteenth-century manuscripts, a case where DTW-I is more appropriate as it treats each color separately. Conversely, in the case of photos that are uniformly faded due to sun exposure or scanning artifacts, the authors show that DTW-D is more appropriate as it corrects for hue shifts across all channels.

Bagnall et al. [24] introduced the UEA archive, one of the largest collection of datasets for MTS classification. The work also includes an evaluation on Euclidean distance, DTW-D, and DTW-I, comparing the effects of a single normalization method as well as Nonorm. Unfortunately, due to the limited range of measures considered (only Euclidean distance and two DTW variants) and the lack of statistical testing (only accuracies are reported), it is challenging to draw general conclusions from this study about MTS distance measures as a whole. Shifaz et al. [218] argued that the channel-dependency extension strategies are also applicable to elastic measures besides DTW. They extended four additional measures to MTS and conducted evaluations on the UEA archive. Similar to [219], the authors of [218] concluded that there exists no superior measure or extension strategy that consistently outperforms others.

We revisit and extend these comparisons by exploring a broader range of measures, normalizations, and downstream tasks, and by considering parameter tuning to ensure comparison of measures in their best possible form. Particularly, our study (a) considers measures from seven categories rather than only lock-step and elastic measures, (b) considers 13 normalization methods (including 5 novel ones) rather than one, (c) evaluates measures on clustering, anomaly detection, and classification, and (d) analyzes results in a principled manner, with both pairwise and grouped comparisons, supported by statistical tests. As such, the study provides a robust evaluation with a more complete view of the full landscape of MTS distance measures (cf. Table 3.1).

3.3 Primer on Comparing Multivariate Time Series

We will first establish the necessary background and describe the three axes that will form a structured framework for our comparison of MTS distance measures; (a) normalization, (b) temporal models, and (c) channel-dependency models. The role of these axes in the processing pipeline is visualized in Figure 3.1. Normalization is a critical preprocessing step, which corrects for distortions in the form of scale imbalances and offsets. As we will show in Section 3.5.1, the choice of normalization methods is non-trivial, as it can have notable impact on the accuracy of distance measures. The temporal model and channel-dependency model,

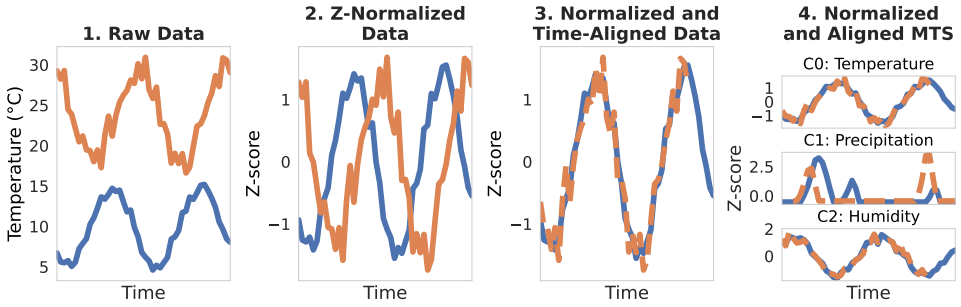


Figure 3.1: Visualization of the role of the three axes of the taxonomy in the processing pipeline of similarity search, using weather data with temperature, precipitation, and humidity as channels. Axis 1, normalization, is applied first to bring the data into a comparable scale, then axis 2, temporal models, is applied to handle temporal distortions, and finally axis 3, channel-dependency models, determine how the transformations of the previous two channels are applied to the different channels of the MTS.

as fundamental properties of MTS distance measures, serve as the basis for our measure categorization. By incorporating these two models, our study enables evaluation at both the level of individual measures and the level of models. As such, the results can ultimately aid the design of future measures and the choice between existing ones. We will now establish some key terminology, definitions, and data assumptions that will be used throughout the chapter. We then introduce the three axes and their role in the comparison of MTS.

3.3.1 Data Assumptions

Recall from Section 2.1 that we consider an MTS dataset D as a set of n two-dimensional arrays, i.e., $D = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{n \times c \times m}$. In this chapter, we consider each MTS to have the same number of channels, and each channel to have the same length, as not all distance measures support comparison of time series of varying length and number of channels. We reflect on this assumption in Section 3.6. Following standard practices [23, 24, 74, 193, 218, 231], we consider the sampling rates of all time series the same. To ensure equal lengths and no missing values for all MTS, we applied polyphase resampling [110] and linear interpolation to the data. As such, we create a standardized test bed for our evaluation, consistent with [24].

3.3.2 Axis 1: Normalization

Normalization is a common preprocessing step to mitigate distortions caused by differences in scale, variance, or offsets [133]. As for all distortions, these differences can hinder the identification of similar patterns between time series and are therefore crucial to address before further analysis. To illustrate the relevance of normalization, consider two examples: (a) two stocks following similar price patterns but trading at different volumes, and (b) two recordings of the same song but played at different octaves. Directly comparing the raw values in these cases could lead to key issues: the first case suffers from a difference in scale, while the second case has a difference in offset. Although time series samples share similar

patterns that are likely relevant for several tasks (e.g., portfolio management or song recognition), they can still produce large dissimilarity scores due to these distortion. This highlights the need for normalization.

Compared with the univariate case, the extra channels of MTS introduce new challenges. On the one hand, when considering channels generated by different devices or entities, the nature of the distortions can vary between channels, requiring different corrections to be applied to each channel independently. On the other hand, this independent normalization strategy overlooks the interdependencies between channels and may lead to suboptimal performance (further discussed in Section 3.3.4). Despite existing surveys on the normalization of UTS [193], multivariate-specific normalization techniques remain unexplored. Particularly, existing works on MTS distances have (at best) only considered applying Z-score normalization on each channel independently [23, 219], without regard for other methods such as Min-Max scaling or normalization based on the entire MTS (e.g., normalize with the global mean and standard deviation).

In this work, we consider a range of 8 normalization techniques, and extend them to the multivariate case with two strategies. We briefly introduce the techniques in the context of UTS here, and explain their extension to the multivariate case in Section 3.3.4. See [193] for a more detailed description of the techniques and their properties. There exist 8 common normalization methods: (a) **Z-score** normalization (denoted further as Z-score), which transforms a time series s.t. it has zero-mean and unit-variance, (b) **Min-Max** normalization (MinMax), which scales the time series to a fixed range, (c) **Mean** normalization (Mean), which combines part of Z-score and MinMax s.t. the data has zero-mean and is scaled to a fixed range, (d) **Median** normalization (Median), which divides the time series by its median, (e) **Unit-length** normalization (Unit), which scales the time series to unit length, (f) **Adaptive** scaling [60] (Adaptive), which works on pairs of time series and scales them by their normalized dot product, (g) **Sigmoid** normalization (Sigmoid), which uses the logistic function to scale or "activate" the time series, and (h) **Hyperbolic tangent** normalization (Tanh), which uses the hyperbolic tangent function to scale the time series. Additionally, to evaluate the effect of normalization in general, we introduce a ninth option: (i) **Nonorm**, which performs no normalization on the data.

3.3.3 Axis 2: Temporal Models

The temporal model of a time series distance measure defines how the time dimension is handled in the comparison. This involves the strategy of handling temporal distortions (e.g., by aligning the time steps in the original space or constructing a new representation to address the distortion). In contrast to resampling, imputation, and normalization, temporal misalignments are "pairwise distortions"; they only arise in the context of comparing two time series, and are therefore handled on the fly by the distance measure during the comparison. As such, the temporal model is a key property of a distance measure, and can be used as the basis for categorization. Furthermore, as these concepts only regard the time dimension, they are not specific to univariate or MTS distance measures. The authors of [193] categorize UTS distances into five families of measures: **lock-step**, **sliding**, **elastic**, **kernel**, and **embedding** measures. We adopt this categorization for MTS as well and extend the taxonomy with

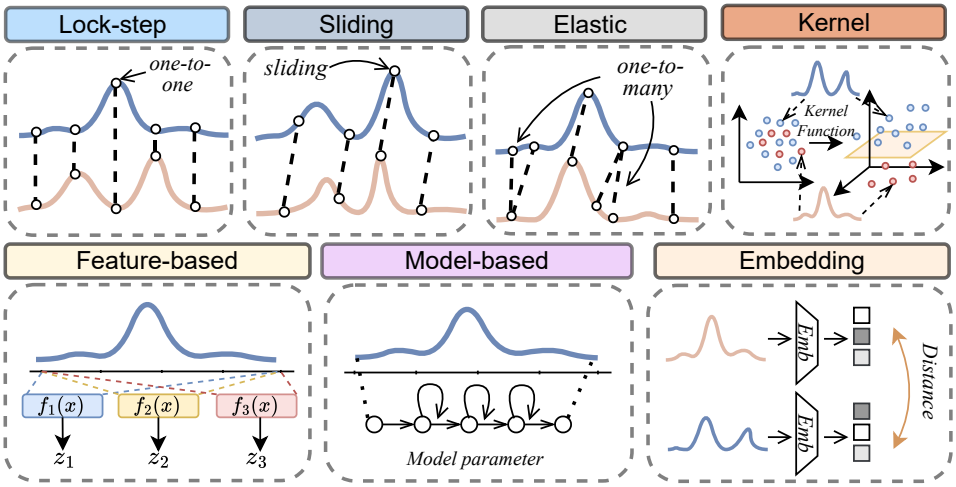


Figure 3.2: Visualization of the seven temporal models, where the horizontal axis represents time.

two additional families, **feature-based** and **model-based** measures, which reflect recent innovations in the field. Additionally, we introduce the concept of **ensemble** measures, which combine the distance scores from multiple base measures to improve the accuracy and robustness of the comparison. At a high level, lock-step, sliding, and elastic measures focus on handling temporal distortions in the original space, differing in their freedom to align time points. Kernel, feature-based, model-based, and embedding measures focus on implicitly or explicitly transforming the time series to handle distortions. We will review the detailed information in Section 3.4. The taxonomy of temporal models is visualized in Figure 3.2.

3.3.4 Axis 3: Channel-Dependency Models

In the multivariate case, questions may arise whether the transformations of normalizations and temporal models should be derived and performed on the whole MTS, or individually per channel. For example, in the case of a sliding measures, should the optimal shift be derived across channels or can each channel be shifted independently? This question holds for all normalizations and temporal models, and determines how measure-specific concepts like scale, optimal shift and warping paths should be derived and applied. The answer to this question is defined as a *channel-dependency model*, and serves as a property of distance measures and normalizations for MTS. We identify two channel-dependency models: *channel-independent* and *channel-dependent*, and provide their realizations for each normalization and temporal model below.

Channel-independent model. The channel-independent model involves treating the channels as independent UTS, normalizing and computing distances channel-by-channel. For normalization, this implies that statistics used in the normalization are computed for each channel independently. For the temporal model, this implies that temporal distortions are handled independently over channels, and that representations are derived for each channel. Then, the distances between channels are aggregated to form a final distance.

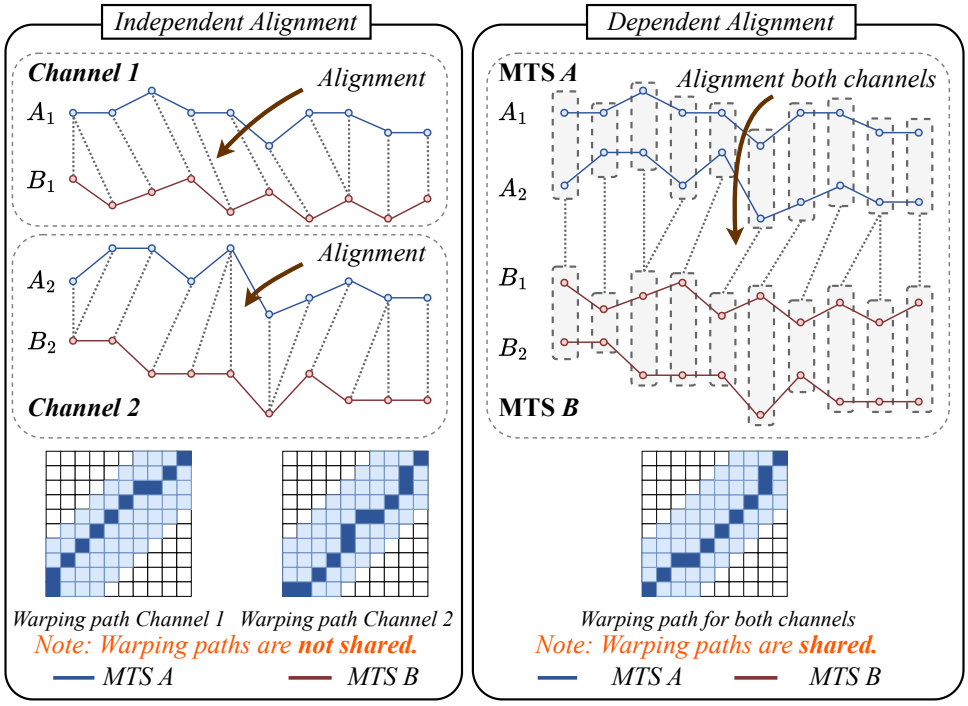


Figure 3.3: Visualization of the alignment and resulting warping paths when utilizing either a channel-independent (left) or channel-dependent (right) extension of DTW.

Channel-dependent model. The channel-dependent model involves treating the channels as a single entity, applying normalization and computing distances over all channels simultaneously. This essentially implies that distortions — either temporal or in scale — are assumed to be shared across all channels. For example, the dependent version of elastic measures aims to find one global warping path for an entire MTS, rather than individual warping path for each channel (Figure 3.3). In contrast to channel-independent extensions, the channel-dependent extension of distance measures is specific to its temporal model. In the next section, we will demonstrate how to extend a representative measure for each temporal model to be channel-dependent. For normalization methods, the channel-dependent extension involves rescaling time series based on global statistics rather than channel-specific ones. For example, channel-dependent Z-score normalization involves computing the mean and standard deviation over all channels, and subtracting and dividing each channel with these values. Note that this model is not applicable to all normalizations. For example, Sigmoid normalization includes no such statistics so it is inherently channel-independent. Only the following methods have a channel-dependent extension: Z-score, Min-Max, Mean, Median, and Unit-length.

Channel weighting. An additional challenge with multivariate data is controlling the influence of each channel. This can be relevant in two scenarios: (a) when channels differ

in scale, the distance score might be biased towards specific channels, and (b) when some channels are more important than others to the task at hand (e.g., only one channel is predictive of the class label). Case (a) is a matter of normalization; both channel-dependent and channel-independent normalizations can rescale channels to control their impact on the distance. Case (b) is a matter of the distance measure itself; all temporal models and channel-dependency models provide the conceptual freedom to weigh channels differently in the distance computation. This, however, always requires some form of supervision, either by learning weights through ground-truth training data as done by some embedding measures [92, 243, 248], or through the input of a domain expert. Naturally, optimal channel weightings are highly task and dataset-dependent. As the aim of this study is to obtain general guidelines on the selection of normalization methods and distance methods without prior knowledge of the specific task or dataset, we will not consider any form of supervised channel weighting in our experiments.

3.4 Multivariate Distance Measures

In this section, we provide more detailed definitions of each temporal model and present an overview of the measures to be evaluated in Section 3.5. Here, we only include the formal definitions of representative measures to illustrate the extension of univariate distance measures to a multivariate setting. A complete overview of all measures and their formal definitions can be found in Appendix E.

Furthermore, we only discuss temporal models in the context of the *channel-dependent* model. The channel-independent versions of measures can be derived by applying the univariate distance measure to each channel individually, and taking the sum of the resulting distances. We indicate the channel-dependency model of each measure by adding a suffix to the measure name. For example, $SBD\text{-}D$ and $SBD\text{-}I$ denote the channel-dependent and independent variants of multivariate SBD, respectively.

Lock-step measures compare values at corresponding time points between two UTS and aggregate these differences across time. The lack of temporal alignment makes lock-step measures applicable to cases where temporal distortions are unlikely to exist in the time series, making distortion corrections unnecessary and potentially harmful. Consequently, due to the lack of any form of temporal alignment or transformation, we deem lock-step measures inherently channel-independent, and therefore do not consider a channel-dependent variant. To illustrate the concept of multivariate lock-step measures, we provide an example formula for the L_p distance, defined as:

$$L_p(\boldsymbol{x}, \boldsymbol{y}) = \sqrt[p]{\sum_{i=1}^{c}\sum_{j=1}^{m} |\boldsymbol{x}_j^{(i)} - \boldsymbol{y}_j^{(i)}|^p}. \tag{3.1}$$

Notice that this distance is equivalent to the univariate L_p distance taken over the flattened time series, i.e., $L_p(\boldsymbol{x}, \boldsymbol{y}) = L_p([\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(c)}], [\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(c)}])$. In our evaluation, we selected representative measures that demonstrated strong performance in the univariate case [193], including Euclidean, L_1, Lorentzian, $Avg(L_1, L_\infty)$, Canberra, Chord, Clark, Emannon4, Jaccard, Soergel, and Topsoe, as detailed in [193].

Sliding measures shift one time series relative to another to find the alignment that minimizes distance. To extend sliding measures to the multivariate case in a channel-dependent manner, a single optimal shift is derived and applied uniformly across all channels. To illustrate this concept, we consider the Shape-Based Distance (SBD) [191]. The channel-dependent SBD (SBD-D) involves finding the optimal shift w only along the time dimension that maximizes the 2D Normalized Cross-Correlation (NCC2) between $\boldsymbol{x}$ and $\boldsymbol{y}$, and substracts that from 1:

$$NCC2_w(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{c} \frac{\sum_{j=1}^{m-w} \boldsymbol{x}_j^{(i)} \cdot \boldsymbol{y}_{j+w}^{(i)}}{\left\|\boldsymbol{x}^{(i)}\right\|_2 \cdot \left\|\boldsymbol{y}^{(i)}\right\|_2} \tag{3.2}$$

$$SBD\text{-}D(\boldsymbol{x}, \boldsymbol{y}) = 1 - \max_w(NCC2_w(\boldsymbol{x}, \boldsymbol{y})) \tag{3.3}$$

Based on the convolution theorem [15], the NCC2 can be computed efficiently ($O(m \log m)$ rather than $O(m^2)$) in the frequency domain using Fast Fourier Transform (FFT) and Inverse FFT (IFFT) as follows:

$$NCC2(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{c} \frac{IFFT(FFT(\boldsymbol{x}^{(i)}) * FFT(\boldsymbol{y}^{(i)}))}{\left\|\boldsymbol{x}^{(i)}\right\|_2 \cdot \left\|\boldsymbol{y}^{(i)}\right\|_2} \tag{3.4}$$

where the operator $*$ represents the complex conjugate in the frequency domain. Note that Equation 3.4 computes the Normalized Cross-Correlation for all possible time-windows (i.e., shifts) through one computation, outputting a *vector* of similarities (i.e., one for each shift) rather than a single value as with $NCC2_w$. Consequently, $\max_w(NCC2_w(\boldsymbol{x}, \boldsymbol{y}))$ is derived as the maximum value in this output vector.

Sliding measures are particularly useful in cases where time series can suffer from phase shifts. In our evaluation, we focus exclusively on SBD [191] as a sliding measure due to its highly competitive performance in the univariate case [193].

Elastic measures handle temporal distortions in time series by deriving a non-linear mapping (i.e., alignment) between time points, minimizing the distance between the aligned series. The mapping of each time point makes elastic measures very applicable to cases where the temporal distortions in time series can be complex, i.e., when individual readings can be delayed, repeated, or missing. With the channel-dependent extension, these mappings are constructed simultaneously for all channels at each time point, using three atomic operations: diagonal, vertical, and horizontal movements. Each atomic operation incurs a specific cost, denoted as c^D, c^V, and c^H for diagonal, vertical, and horizontal movements, respectively. The optimal alignment is determined using dynamic programming, with the total alignment cost (i.e., distance) being the sum of the selected movements. Formally, the *recursive* alignment cost up to time points i and j for two MTS is defined as:

$$Cost(i, j) = \min \begin{cases} Cost(i-1, j-1) + c^D & \textit{diagonal} \\ Cost(i-1, j) + c^V & \textit{vertical} \\ Cost(i, j-1) + c^H & \textit{horizontal} \end{cases} \tag{3.5}$$

where the border conditions are specific to the individual measures. Elastic measures vary in the cost functions assigned to each movement type and the constraints they impose on the alignment process. For instance, in DTW, the cost is calculated as the squared Euclidean distance between corresponding data points, and the difference between DTW-I and DTW-D is displayed in Figure 3.3. In our evaluation, based on elastic measures' performance in [193], we consider the five representative measures: DTW [212], Longest Common Subsequence (LCSS) [230], Edit Distance with Real Penalty (ERP) [49], Time Warping Edit Distance (TWE) [167], and Move-Split-Merge (MSM) [221].

Kernel measures implicitly map the raw time series data to a higher-dimensional space using a kernel function. Similar to the kernel trick in machine learning, this transformation allows for capturing non-linear relationships between time series without explicitly computing the high-dimensional representation. The intuition is that some patterns in time series that are difficult to compare in the original space become more easily distinguishable in this transformed space. For example, two time series that differ by a complex non-linear transformation (e.g., a phase shift, or non-linear scaling) might appear similar when projected into an appropriate kernel space, just as non-linearly separable data points become linearly separable in kernel-based classification. The extension of kernel measures differs according to the base measures on which the kernel functions are applied. For example, the dependent extension of the Shift-Invariant Kernel (SINK) relies on the channel-dependent version of NCC (NCC2). In this work, we consider four kernel measures: one using a lock-step base (Radial Basis Function (RBF) [65]), two using an elastic base (Global Alignment Kernel (GAK) [66], Kernel Dynamic Time Warping (KDTW) [166]), and one using a sliding base (SINK [190]).

Feature-based measures transform a raw time series to a feature vector based on pre-defined features, e.g., mean, variance, slope, entropy, etc. Feature-based measures differ in the features used to describe the MTS and the distance measure used to compare the feature vectors. In this work, we consider two widely adopted feature sets: (a) the CAnonical time series CHaracteristics set (catch22) [159], and (b) the TSFresh feature set [59]. Accordingly, we propose two feature-based methods, referred to as $D_{Catch22}$ and $D_{TSFresh}$, which both use the Euclidean distance to compare the feature vectors of two MTS. While conceptually possible, both methods were initially proposed for UTS, lacking a straightforward extension to incorporate multivariate features (e.g., capturing inter-channel correlations). Therefore, they are inherently channel-independent and require the development of new features to create a channel-dependent variant, which is beyond the scope of this work.

Model-based measures construct probabilistic models to represent time series — either a single model for the entire MTS in channel-dependent cases or separate models for each channel in channel-independent cases — which are then used as proxies for computing distances. Model-based measures differ in the model used to represent the MTS, and the distance measure for comparing the models. In this work, we consider two types of models: Multivariate Gaussian Models (Gauss) and Hidden Markov Models (HMM) [27], and compare them using the Kullback-Leibler (KL) divergence as a distance measure, which has a closed form for Gaussian models and can be approximated for HMMs [100].

Embedding measures project the time series into a latent space for capturing the most important characteristics, allowing for the calculation of dissimilarity. In constrast to feature-based measures, embedding measures use *implicit* features, meaning that the features are learned from the data rather than being pre-defined. Embedding measures differ in their choice of the embedding method. For example, the extension process of Generic Representation Learning (GRAIL) [190] directly depends on the multivariate SINK, its core kernel function for representation learning. In this work, we consider five embedding measures: TS2Vec [248] and TLoss [92], two deep learning-based embedding methods, GRAIL [190], the best-performing embedding method in [193], and Principal Component Analysis (PCA) similarity factor (D_{PCA}) [243] and Eros (D_{Eros}) [243], two embedding methods based on PCA projections that are inherently channel-dependent. TS2Vec and TLoss were transformed to distance functions by taking the Euclidean distance over the embeddings obtained by the encoder-based models.

Ensemble measures combine the distance scores of multiple measures to improve robustness. They cannot be classified as a temporal model, but rather as a *"meta-model"* that combines the strengths of individual measures. This is achieved by aggregating (e.g., averaging) the distance scores of the individual measures, optionally with weights assigned to each measure. As the distances scores of different measures can have varying ranges and distributions, distances are normalized across a set of training MTS before aggregation. In this work, we consider two ensembles: (i) SBD-D + DTW-I and (ii) SBD-D + MSM-I, with Minmax normalization applied to the distance scores before averaging.

Time complexities: The differences in the temporal models' approaches to handling temporality and distortions lead to varying computational complexities. Namely, lock-step measures, which process each value in the MTS only once, have a linear complexity with respect to the number of channels and time steps, $O(cm)$, making them the most efficient. Sliding measures utilize FFTs to compute distances efficiently, resulting in a complexity of $O(cm \log m)$ for SBD-I and $O(cm \log cm)$ for SBD-D. For elastic measures, computation of the optimal time-alignment worst-case requires to compare all pairs of time points, resulting in a quadratic complexity of $O(cm^2)$ for both channel-dependent and channel-independent versions. Kernel measures and ensembles inherit the complexities of their base measures. The complexities of feature-based, model-based, and embedding measures are highly dependent on the underlying methods used to extract features, fit models, or learn representations, respectively; and thus do not have a general complexity. Empirical efficiency is investigated in Section 3.5.4.

3.5 Evaluation

The purpose of our evaluation is threefold: (a) to evaluate the discriminative power of individual measures, (b) to analyze what normalization methods, temporal models, and channel dependency models are best suited for MTS, and (c) to determine whether previous findings on UTS also hold for the MTS case [193]. With this evaluation, we aim to lay the foundation for future research, paving the way for a practical handbook to guide the selection of MTS distance measures in similarity search.

Table 3.2: Parameter spaces for MTS distance measures. d denotes the number of input channels for MTS comparison. Specifically, $d = c$ for channel-dependent measures (using all channels), and $d = 1$ for channel-independent measures.

Distance Measure	Parameter Range
DTW	$\delta \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 100\}$
LCSS	$\delta \in \{5, 10, 20, 50, 100\}$ $\epsilon \in \{0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0, 2.0\} * \sqrt{d}$
MSM	$\mathbf{c} \in \{0.01, 0.1, 1, 10, 100, 0.5, 5, 50, 500\} * \sqrt{d}$
TWE	$\lambda \in \{0, 0.25, 0.5, 0.75, 1\} * \sqrt{d}$ $\nu \in \{0.00001, 0.0001, 0.001, 0.01, 0.1, 1.0\}$
SINK	$\gamma \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 100, 1000\}$
GAK	$\gamma \in \{0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 5, 10, 15, 20\}$
KDTW	$\gamma \in \{2^{-15}, 2^{-14}, 2^{-13}, 2^{-12}, 2^{-11}, 2^{-10}, 2^{-9}, 2^{-8}, 2^{-7}, 2^{-6}, 2^{-5}$ $2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}\}$
RBF	$\gamma \in \{-1, 1\}$
GRAIL	$\gamma \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 100, 1000\}$
TS2Vec	Embedding size $\in \{40, 80, 160, 320, 640\}$
TLoss	Embedding size $\in \{40, 80, 160, 320, 640\}$
D_{PCA}	$\sigma^2_{\text{covered}} \in \{50\%, 60\%, 70\%, 80\%, 95\%, 100\%\}$

Datasets: We use the *UEA archive* for experiments on classification and clustering. The UEA archive is the largest set of labeled datasets for MTS classification, comprising of 30 labeled real-world datasets from diverse domains, with varying number of channels and lengths [24]. We adopt the predefined train-test splits, which have been widely used in MTS tasks [23, 169, 211, 218, 219]. Following the creator's recommendation [24], we resample shorter time series to match the length of the longest for each dataset, and impute missing values using linear interpolation. Table 3.3 summarizes the datasets in the UEA archive, including the number of time series, channels, time points, and classes. For anomaly detection (AD), we use the TSB-AD-M archive [155], which is one of the largest collections of datasets for AD on MTS, consisting of 200 real-world time series with labeled anomalous time points (i.e., ground truth anomalies) and varying lengths and channels.

Evaluation framework: In line with prior works [24, 193, 218], we evaluate the performance of distance measures on classification, clustering, and anomaly detection using k-NN-based algorithms (i.e., 1-NN classifier for classification, k-means for clustering, and a 1-NN detector for AD), which are well-suited to this evaluation for three key reasons [74, 193]: (a) they reflect a similar workflow to time series similarity search tasks, the primary application of distance measures [81], (b) they are parameter-free and straightforward to implement, and (c) they solely depend on the discriminative power of distance measures. Our evaluation is structured into two phases: a *parameterization phase*, where we determine the best parameters for each measure and dataset, and a *performance phase*. The performance phase consists of six analyses, three of which focus on the axes of MTS comparison in the context of *classification* (Section 3.5.1), two focus on the validation of findings in *clustering* and *anomaly detection* (Section 3.5.2-3.5.3), and one is a runtime analysis, where we focus on the accuracy-to-runtime trade-off of the measures (Section 3.5.4).

Dataset	#MTS	#Channels	#Time-points	Classes
ArticularyWordRecognition (AWR)	575	9	144	25
AtrialFibrillation (AF)	30	2	640	3
BasicMotions (BM)	80	6	100	4
CharacterTrajectories (CT)	2858	3	182	20
Cricket (CR)	180	6	1197	12
DuckDuckGeese (DDG)	100	1345	270	5
EigenWorms (EW)	259	6	17984	5
Epilepsy (EPI)	275	3	206	4
EthanolConcentration (EC)	524	3	1751	4
ERing (ER)	60	4	65	6
FaceDetection (FD)	9414	144	62	2
FingerMovements (FM)	416	28	50	2
HandMovementDirection (HMD)	467	10	400	4
Handwriting (HW)	1000	3	152	26
Heartbeat (HB)	409	61	405	2
JapaneseVowels (JV)	640	12	29	9
Libras (LB)	360	2	45	15
LSST (LSST)	4925	6	36	14
InsectWingbeat (IW)	50000	200	78	10
MotorImagery (MI)	378	64	3000	2
NATOPS (NO)	360	24	51	6
PenDigits (PD)	10992	2	8	10
PEMS-SF (PSF)	440	963	144	7
Phoneme-Spectra (PS)	6668	11	217	39
RacketSports (RS)	303	6	30	4
SelfRegulationSCP1 (SR1)	561	6	896	2
SelfRegulationSCP2 (SR2)	380	7	1152	2
SpokenArabicDigits (SAD)	8798	13	93	10
StandWalkJump (SWJ)	27	4	2500	3
UWaveGestureLibrary (UWGL)	440	3	315	8

Table 3.3: A summary of the 30 datasets in the UEA Multivariate Time Series Classification archive, 2018 [24].

Parameterization: To ensure comparison of measures in their best possible form, we start by fine-tuning their parameters (where applicable) under two experimental settings: (i) supervised setting, where the optimal model parameters are selected on the training set of each dataset using Leave-One-Out Cross-Validation (LOOCV), (ii) unsupervised setting, where we derive a default parameter setting for each measure by analyzing which single set of values performed *generally* well across all datasets. Note that for clustering and anomaly detection only the unsupervised setting is considered as both methods are inherently unsupervised tasks. We employ a broad parameter range for each measure, and account for both channel-dependency models where applicable. Parameterization was performed across different normalizations, though optimal parameter values showed to be fairly similar across different normalizations. The parameter ranges for all measures are shown in Table 3.2. The default parameters (i.e., unsupervised setting) are presented alongside the results in the respective sections (e.g., Table 3.6).

Statistical analysis: We assess the significance of the differences in performance between measures, normalization methods, and channel-dependency models using two widely adopted statistical tests [23, 74, 193, 218]. Specifically, for pairwise comparisons, we employ one-sided Wilcoxon tests [235] with $\alpha = 0.05$, following the guidelines of [71]. For global comparisons, we apply the Friedman test [95] followed by a post-hoc Nemenyi test [182] with $\alpha = 0.1$ to determine the significance of differences (referred to as the Friedman-Nemenyi test). Such global comparisons are then visualized by critical diagrams (e.g., Figure 3.5), which show the average rankings of the compared methods, with horizontal lines connecting methods that exhibit insignificant differences.

Platform and implementations: We conduct our experiments on a server with a 2 GHz AMD EPYC 7713 processor and 1 TB of RAM, running Ubuntu 22.04.3 LTS. All measures are implemented in Python 3.8.5. We use the HMMLEARN library [141] for HMMs, the SCIPY library [229] for PCA, and the SKTIME library [157] for feature-based measures. Experiments for runtime analysis are evaluated in a single-threaded fashion without use of acceleration libraries such as numba or Cython. All reported times exclude data loading and preprocessing (i.e., resampling, imputation, and parameterization), and are averaged over 10 runs to account for variations due to system load. All code used in this work is publicly available [5].

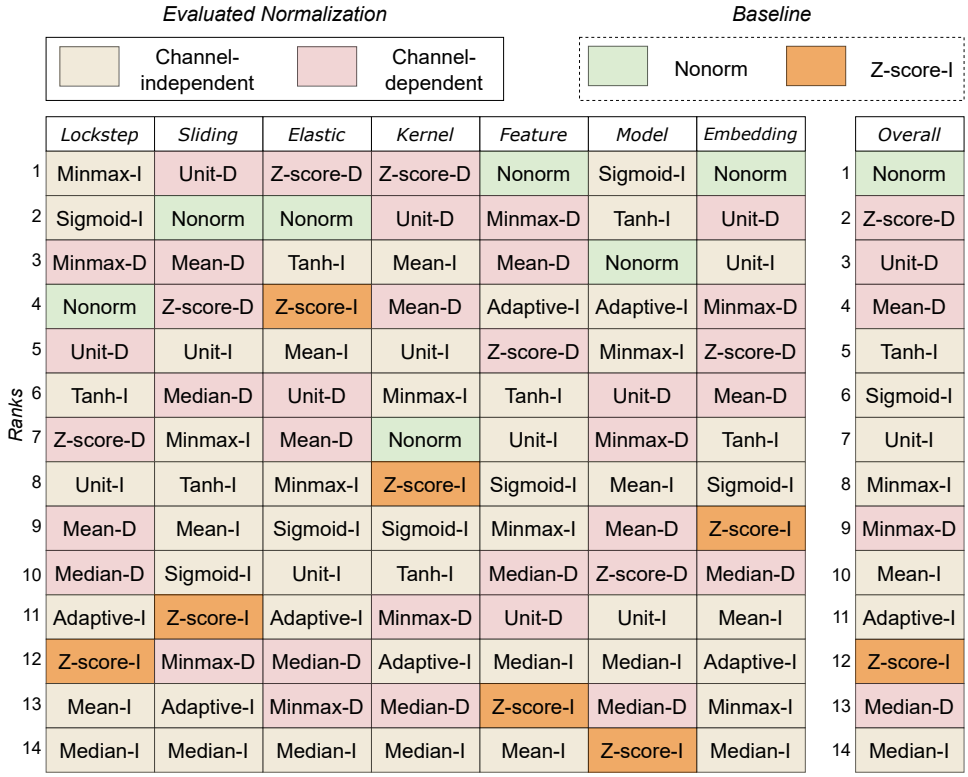
3.5.1 Task 1: Classification

The following presents the evaluation of distance measures on the task of classification. The analysis is organized into three subsections, each focusing on one of the taxonomy axes.

Normalization Methods

We consider 13 unique normalizations, along with Nonorm, and assess their effect on the classification accuracy of multivariate distance measures. In UTS literature, Z-score normalization has long been regarded as the optimal choice [8, 88, 102]. Paparrizos et al. [193] debunked this though based on a large-scale evaluation of univariate distance measures. In MTS literature, previous evaluations of multivariate distance measures have *not* systematically evaluated the impact of normalization methods, focusing only on Nonorm and Z-score-I, at best [211, 218]. To address this gap, we extend the analysis of [193] to the multivariate case, conducting a meta-analysis to evaluate whether their findings on normalization apply to MTS as well.

Figure 3.4 presents the average ranks of normalization methods based on the classification accuracies across measure-dataset combinations. The results indicate that Z-score-I — the natural extension of the popular UTS normalization — consistently underperforms compared to other normalization methods across the temporal models; it ranks 12th in the overall meta-ranking and ranks 8th or lower for all temporal models except elastic measures. Our findings confirm and extend the conclusion of [193] that Z-score is not the best normalization method, showing that this also holds true for MTS. Additionally, we observe that Nonorm consistently outperforms other methods; ranking first overall and placing in the top four across all temporal models except kernel measures. This indicates that normalizations do not always offer improvements over the natural baseline of no normalization, as



	Lockstep	Sliding	Elastic	Kernel	Feature	Model	Embedding		Overall
1	Minmax-I	Unit-D	Z-score-D	Z-score-D	Nonorm	Sigmoid-I	Nonorm	1	Nonorm
2	Sigmoid-I	Nonorm	Nonorm	Unit-D	Minmax-D	Tanh-I	Unit-D	2	Z-score-D
3	Minmax-D	Mean-D	Tanh-I	Mean-I	Mean-D	Nonorm	Unit-I	3	Unit-D
4	Nonorm	Z-score-D	Z-score-I	Mean-D	Adaptive-I	Adaptive-I	Minmax-D	4	Mean-D
5	Unit-D	Unit-I	Mean-I	Unit-I	Z-score-D	Minmax-I	Z-score-D	5	Tanh-I
6	Tanh-I	Median-D	Unit-D	Minmax-I	Tanh-I	Unit-D	Mean-D	6	Sigmoid-I
7	Z-score-D	Minmax-I	Mean-D	Nonorm	Unit-I	Minmax-D	Tanh-I	7	Unit-I
8	Unit-I	Tanh-I	Minmax-I	Z-score-I	Sigmoid-I	Mean-I	Sigmoid-I	8	Minmax-I
9	Mean-D	Mean-I	Sigmoid-I	Sigmoid-I	Minmax-I	Mean-D	Z-score-I	9	Minmax-D
10	Median-D	Sigmoid-I	Unit-I	Tanh-I	Median-D	Z-score-D	Median-D	10	Mean-I
11	Adaptive-I	Z-score-I	Adaptive-I	Minmax-D	Unit-D	Unit-I	Mean-I	11	Adaptive-I
12	Z-score-I	Minmax-D	Median-D	Adaptive-I	Median-I	Median-I	Adaptive-I	12	Z-score-I
13	Mean-I	Adaptive-I	Minmax-D	Median-D	Z-score-I	Median-D	Minmax-I	13	Median-D
14	Median-I	Median-I	Median-I	Median-I	Mean-I	Z-score-I	Median-I	14	Median-I

Figure 3.4: Meta ranking of normalization methods.

also observed in previous studies on the UEA archive [211, 218]. In those studies, this finding was interpreted as a sign that the scales and variances of channels serve as discriminatory factors in classification tasks, and that normalization removes this information. While we recognize the validity of this hypothesis, we now also show that channel-dependent normalizations — that should better preserve this information — also do not significantly improve upon Nonorm. This suggests that the loss of channel-specific information is not the only factor at play, and that the existing normalization methods themselves (i.e., designed for UTS) may not be well-suited for MTS data.

To gain deeper insight into the performance of normalization methods, we evaluate the statistical significance of the differences between the methods using the Friedman-Nemenyi test. The results (connecting lines in Fig. 3.5) reveal that no single method or group of methods performs statistically better than the others, neither on a global level, nor when considering specific temporal models. This, combined with the fact that Nonorm ranks first overall, suggests that current normalization methods, which are direct extensions of methods for UTS, do not significantly improve classification accuracy for most temporal models. This implies that existing normalization approaches do not generalize well to the multivariate

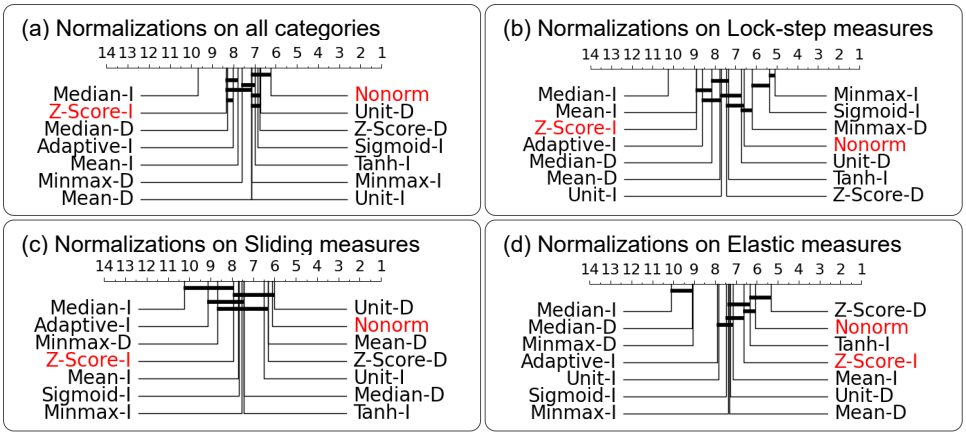


Figure 3.5: Ranking of normalization methods based on the average of their ranks across measures and datasets.

case, and *highlights the need for new MTS-specific normalizations* that better accommodate the multivariate nature of the data. In the current absence of those methods, we will use Nonorm as the default for the remainder of our analyses.

Temporal Models

The objective of the next round of experiments is (a) to identify the best-performing measure within each temporal model, but also (b) to compare the performance of these temporal models as a whole. To make such an evaluation feasible, we employ an approach first proposed in [193] that starts with lock-step measures, comparing each to a baseline, and based on the results, decide whether to update the baseline before proceeding to the next temporal model. This method enables a dynamic evaluation, continuously refining comparisons with a reasonable baseline to achieve accurate and reliable results. We begin with Euclidean distance as the baseline, since it is the most widely used measure in the literature [193].

Lock-step measures. Table 3.4 presents the pairwise comparison of lock-step measures under Nonorm based on the Wilcoxon test, using Euclidean distance as the baseline, and each measure ordered by its average rank across datasets. The results show that three lock-step measures, $Avg(L_1, L_\infty)$, L_1, and Lorentzian, significantly outperform the baseline. This confirms and extends a key finding from [193] that Euclidean distance is not necessarily the best lock-step measure in the multivariate case, and that $Avg(L_1, L_\infty)$, L_1, and Lorentzian offer superior alternatives. To identify the best-performing lock-step measure, we recompute the average ranks for the top-performing measures, and assess their significance using the Friedman-Nemenyi test. The results (Fig. 3.6a) show (a) additional evidence that $Avg(L_1, L_\infty)$ and Lorentzian significantly outperform Euclidean, and (b) that Lorentzian ranks the highest, making it the new baseline for subsequent comparisons.

Table 3.4: Pairwise comparison of lock-step measures with Euclidean under Nonorm, with measures ordered by their average rank across datasets. ✓, ✗, and ≈ indicate significantly better, worse, or equal performance compared to the baseline based on the Wilcoxon test ($\alpha = 0.05$). ">", "=", and "<" indicate how many datasets the given combination performs better, equal, or worse than the baseline, respectively.

Measure	Diff	Average Accuracy	>	=	<
$Avg(L_1, L_\infty)$	✓	0.6321	18	4	8
Lorentzian	✓	0.6334	20	2	8
Jaccard	≈	0.6507	15	7	8
L_1	✓	0.6321	21	2	7
Chord	≈	0.6261	15	1	14
Topsoe	✗	0.5589	11	0	19
Soergel	✗	0.4153	9	1	20
Clark	✗	0.4462	8	0	22
Canberra	✗	0.3104	8	1	21
Emanon4	✗	0.3246	6	0	24
Euclidean	≈	**0.6264**	**0**	**30**	**0**

Table 3.5: Pairwise comparison of sliding measures with Lorentzian (Nonorm). See Table 3.4 for column descriptions.

Measure	Diff	Average Accuracy	>	=	<
SBD-D	✓	0.6817	19	0	11
SBD-I	≈	0.6520	17	0	13
Lorentzian	≈	**0.6334**	**0**	**30**	**0**

Sliding measures. Table 3.5 presents the pairwise comparison of sliding measures with Lorentzian, all under Nonorm. We observe that only SBD-D significantly outperforms Lorentzian; SBD-I shows better performance on most datasets but without reaching statistical significance. These findings are validated through a Friedman-Nemenyi test (Fig. 3.6b), where both sliding measures rank higher than Lorentzian, with SBD-D showing statistical significance. These findings demonstrate that SBD-D's ability to address global misalignments can increase classification accuracy. Moreover, the superior performance of SBD-D over SBD-I suggests that the additional flexibility in temporal alignment provided by SBD-I does not always lead to better accuracy and may result in overly liberal alignments. We explore this difference between channel-dependency models further in Section 3.5.1. As SBD-D outperformed the previous baseline, Lorentzian, SBD-D will be used as the new baseline for the following experiment on elastic measures.

Elastic measures. We now present a comparison of elastic measures, both in supervised and unsupervised settings, with SBD-D as the baseline on a downsampled version of the UEA archive. The reason for downsampling the original archive is that variants of MSM and TWE were very time-consuming; the classification process exceeded server's time limit of 7 days on 7 different datasets. All datasets that had over 500 MTS in their training/testing size, 500 channels, or 500 time points per MTS, were downsampled in their respective dimensions, using stratified sampling to select MTS, random sampling to select channels, and polyphase resampling [110] to reduce the number of time points.

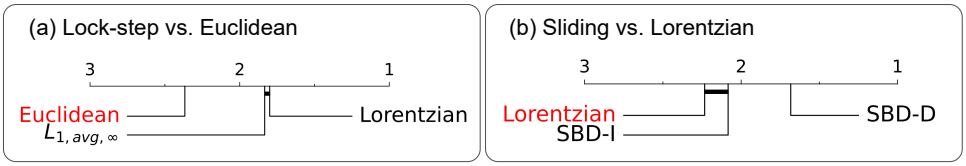


Figure 3.6: Ranking of lock-step and sliding measures under Nonorm based on the average of their ranks across datasets.

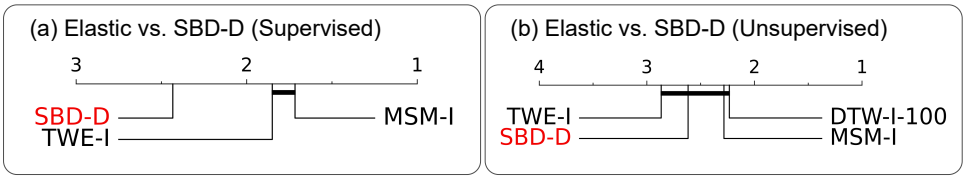


Figure 3.7: Ranking of elastic measures under Nonorm across UEA datasets, using (a) supervised and (b) unsupervised tuning for their parameters

The results in Table 3.6 demonstrate that only MSM-I and TWE-I significantly outperform SBD-D, but only in the supervised setting. This shows that SBD-D, being parameter-free, is a highly competitive baseline, with most elastic measures unable to outperform it in either the supervised or unsupervised setting. Furthermore, we conclude that (a) elastic measures are highly sensitive to parameter settings, with fixed values often not being optimal across all datasets, and that (b) the ability to handle local temporal distortions does not necessarily lead to improved accuracy. We also assess the significance of the differences when considering multiple elastic measures alongside the baseline. Figures 3.7c-d reinforce previous findings, showing that MSM-I and TWE-I outperform SBD-D significantly under supervised tuning, but not in the unsupervised setting. These findings align with those from the univariate case reported in [193], allowing us to both confirm and extend two key conclusions to the multivariate setting: (a) elastic measures are not necessarily superior to sliding measures and (b) alternative elastic methods can outperform DTW in the supervised setting. While tuning parameters is crucial for the performance of elastic measures, substantially increasing the computational load as it requires a grid search over the full parameter range. This cost comes on top of already high theoretical complexity of elastic measures, as discussed in Section 3.4. This computational load may hinder the application of elastic measures on large-scale datasets. Further details are provided in the runtime analysis in Section 3.5.4. In view of this observation, we retain SBD-D as the running baseline for the next comparison.

Kernel measures. Table 3.7 shows the performance of kernel measures compared to SBD-D, evaluated on the downsampled UEA archive discussed in Section 3.5.1. We observe here that no measure is able to significantly outperform SBD-D; only GAK-D and SINK-D are able to match the performance of SBD-D with and without tuning. KDTW-I and RBF perform significantly worse, even with supervised parameter tuning. We further observe no statistical significance in a grouped comparison (Fig. 3.8), where GAK-D ranks slightly higher than SBD-D in the supervised setting but not in the unsupervised setting. These findings can be

Table 3.6: Pairwise comparison of elastic measures under Nonorm against SBD-D. See Table 3.4 for column descriptions.

Measure	Parameter Tuning	Diff.	Average Accuracy	>	=	<
MSM-I	LOOCV	✓	0.6573	20	2	8
	$c = 0.5$	≈	0.6525	18	0	12
DTW-I	LOOCV	≈	0.6570	16	3	11
	$\delta = 100$	≈	0.6529	16	2	12
	$\delta = 10$	≈	0.6413	15	2	13
TWE-I	LOOCV	✓	0.6561	21	2	7
	$\lambda = 0.5, \nu = 0.01$	≈	0.6325	13	1	16
MSM-D	LOOCV	≈	0.6384	13	3	14
	$c = 0.5 * \sqrt{d}$	≈	0.6200	15	2	13
DTW-D	LOOCV	≈	0.6296	13	1	16
	$\delta = 100$	≈	0.6276	12	2	16
	$\delta = 10$	≈	0.6237	12	2	16
ERP-D	$\delta = 100$	≈	0.6301	14	2	14
LCSS-I	LOOCV	≈	0.6385	16	0	14
	$\delta = 5, \epsilon = 1.0$	✗	0.5456	6	0	24
TWE-D	LOOCV	≈	0.6298	13	2	15
	$\lambda = \sqrt{d}, \nu = 0.0001$	✗	0.6184	13	3	14
ERP-I	$\delta = 100$	≈	0.6400	12	3	15
LCSS-D	LOOCV	✗	0.5969	9	3	18
	$\delta = 10, \epsilon = 0.5 * \sqrt{d}$	✗	0.4981	5	3	22
SBD-D	—	—	**0.6496**	**0**	**30**	**0**

attributed to the sensitivity of kernel measures to their scaling parameter, which influences the similarity between data points in the kernel space and, consequently, impacts the classifier's decision boundaries. Naturally, the optimal scaling parameter is dataset-dependent, requiring tuning for good performance. This sensitivity also explains why kernel measures are generally paired with classifiers that adaptively learn these decision boundaries, such as Support Vector Machines (SVM) [64], rather than being used as a standalone measure with 1-NN classifiers. Still, what is particularly surprising is that none of the measures significantly outperform SBD-D; not even those with an elastic foundation like GAK-D and GAK-I. This comes in contrast to the UTS cases, which showed that kernel measures — when properly tuned — were able to improve upon the performance of their base measures [193], e.g., KDTW versus DTW. This discrepancy may hint that the temporal alignment happening in

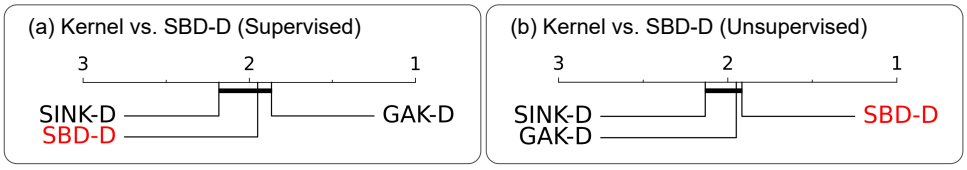


Figure 3.8: Ranking of kernel measures under Nonorm across UEA datasets, using (a) supervised and (b) unsupervised tuning for their parameters

Table 3.7: Pairwise comparison of kernel measures under Nonorm normalization with SBD-D as a baseline. See Table 3.4 for column descriptions.

Measure	Parameter Tuning	Diff.	Average Accuracy	>	=	<
GAK-D	LOOCV	≈	0.6302	16	2	12
	$\sigma = 0.5$	≈	0.6140	14	2	14
SINK-D	LOOCV	≈	0.6482	10	3	17
	$\gamma = 5$	≈	0.6404	11	3	16
GAK-I	LOOCV	≈	0.6440	12	0	18
	$\sigma = 0.05$	✗	0.5950	6	2	22
KDTW-D	LOOCV	≈	0.6256	12	2	16
	$\sigma = 64$	✗	0.5769	8	2	20
SINK-I	LOOCV	≈	0.6270	9	3	18
	$\gamma = 5$	✗	0.6166	7	1	22
KDTW-I	LOOCV	✗	0.6142	9	0	21
	$\sigma = 32$	✗	0.5372	2	2	26
RBF	LOOCV	✗	0.5845	4	0	26
	$\gamma = -1$	✗	0.5757	3	0	27
SBD-D	-	-	**0.6496**	**0**	**30**	**0**

these measures does not generalize well to the multivariate case, and either better temporal or channel-dependency models may be required for optimal performance. We leave this for future investigation.

Feature-based, model-based, and embedding measures. Moving on to the final three temporal models, we again observe that these measures perform significantly worse than SBD-D. To prevent only comparing against an overly strong baseline, we report comparison of these measures to Euclidean distance on the downsampled archive in Table 3.8. From the results we observe that only GRAIL-D, TS2Vec-D, and TS2Vec-I significantly outperform Euclidean distance in both the supervised and unsupervised settings; all other measures except TLoss, GRAIL-I, and KL_{Gauss}-D perform significantly worse in the comparison. These observations are reinforced by the global comparisons in Figure 3.9, where most methods rank lower than Euclidean distance (though not significantly), with GRAIL, TLoss, and TS2Vec being the only exceptions. Still, the results also show that these measures still do not outrank SBD-D.

For embedding measures, the unexpectedly poor performance from the PCA-based measures (D_{PCA} and D_{Eros}) may be due to the fact that current PCA-based approaches in the MTS context primarily capture correlations between channels. When these correlations are not strongly indicative of class labels, they may lead to suboptimal performance. Looking at the deep-learning-based embedding measures, we see that TS2Vec performs a lot better than TLoss. This ranking is in line with the results in [248], where TS2Vec outperformed TLoss on the UEA archive when paired with a supervised SVM classifier. However, it also shows that they are very dependent on the type of classifier used, as the accuracies reported here with 1-NN classifiers are significantly lower than those reported in [248] for both methods. Most importantly, TS2Vec, being the state-of-the-art encoder-based deep learning meth-

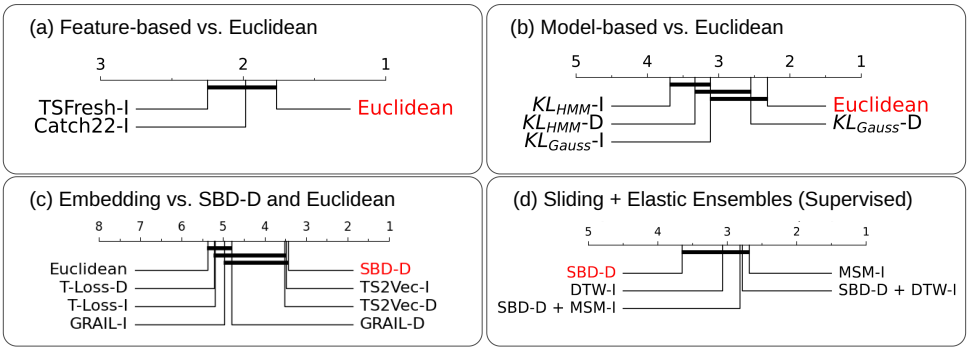


Figure 3.9: Ranking of different measure families under Nonorm based on the average of their ranks across datasets.

ods, still performs worse than SBD-D (Figure 3.9), demonstrating that deep-learning-based measures do not necessarily outperform traditional measures in the MTS domain.

As shown in Table 3.8 and Figure 3.9a, we observe the low performance of feature-based measures compared to Euclidean distance. This can be attributed to two key reasons: (a) $D_{Catch22}$ and $D_{TSFresh}$ primarily focus on *global* properties of the time series, overlooking critical local information that the considered embedding measures do capture in their features, and (b) the feature-based measures included in this study consider only channel-independent features, neglecting channel-dependent characteristics. Further exploration of these two directions represents a potential future work.

For model-based measures, we note that the considered measures employ models that assume the time series to be i.i.d samples from a distribution, overlooking the temporal structure critical to time series classification. Namely, while HMMs can conceptually capture temporal dependencies, the limited number of hidden states (fewer than 5) likely restricts their focus to long-term trends, missing short-term patterns. We therefore conclude that while model-based approaches are promising, simple probabilistic models like Gaussian distributions and HMMs are not yet effective as demonstrated from this study. Future work should therefore investigate utilizing models that can better capture temporal patterns in time series.

Ensemble measures. Finally, we evaluate the ability of ensemble measures to improve upon the performance of their base measures. Motivated by the findings up to now, we combine SBD-D with MSM-I and DTW-I to investigate if we can surpass the current state-of-the-art performance of supervised MSM-I and unsupervised DTW-I. Our results (Fig. 3.9d) show that this is not the case; while combining SBD-D with DTW-I seems to improve upon the performance of their individual components (i.e., DTW-I and SBD-D), the ensemble of SBD-D and MSM-I does not improve upon the performance of MSM-I alone. This shows that ensembles are not guaranteed to improve upon the performance of their components, and thus that careful selection of the ensemble members is crucial. Furthermore, we stress that there currently exist no principled approach to aggregating distance scores; the current

Table 3.8: Pairwise comparison of representation-based measures under Nonorm with Euclidean as a baseline. See Table 3.4 for column descriptions.

	Measure	Parameter Tuning	Diff.	Average Accuracy	>	=	<
Embedding	TS2Vec-I	LOOCV	✓	0.6538	21	2	7
		Embedding length = 320	✓	0.6409	19	3	8
	TS2Vec-D	LOOCV	✓	0.6559	23	0	7
		Embedding length = 320	✓	0.6454	19	1	10
	GRAIL-D	LOOCV	✓	0.6371	16	2	12
		$\gamma = 2$	✓	0.6337	18	0	12
	GRAIL-I	LOOCV	✓	0.6241	18	2	10
		$\gamma = 2$	≈	0.6144	15	6	9
	TLoss-I	LOOCV	≈	0.6159	13	1	16
		Embedding length = 320	≈	0.6088	12	0	18
	TLoss-D	LOOCV	≈	0.6129	13	0	17
		Embedding length = 320	≈	0.6038	12	1	17
	D_{Eros}	—	✗	0.4842	10	2	18
	D_{PCA}	LOOCV	✗	0.2793	7	0	23
		Covered Variance = 95%	✗	0.3031	10	1	19
Feat.	$D_{Catch22}$-I	—	✗	0.4977	10	4	16
	$D_{TSFresh}$-I	—	✗	0.4102	10	2	18
Model	KL_{Gauss}-D	—	≈	0.5191	10	1	19
	KL_{Gauss}-I	—	✗	0.4322	11	1	18
	KL_{HMM}-D	$h = 2$	✗	0.4164	9	1	20
	KL_{HMM}-I	$h = 2$	✗	0.4406	8	0	22
	Euclidean	—	—	**0.5846**	**0**	**30**	**0**

method of averaging and Minmax scaling is a simple example introduced for this study, though a lot of work remains to be done in this area to determine the optimal ensembling approach.

Channel-Dependency Models

We now investigate whether the channel-independent model delivers better performance than the channel-dependent model. Note that with the current results, it is not meaningful to analyze the impact of channel-dependency models on normalizations, as no normalization method shows significant impact on performance. Therefore, we limit ourselves to the comparison of channel-dependent vs. channel-independent *measures* in this section. This aspect should be reconsidered when new normalization methods are proposed that demonstrate significant improvements over Nonorm.

Previous studies have consistently suggested that the optimal channel-dependency model is data-dependent, as it reflects the nature of distortions inherent to the dataset [218, 219]. To validate this claim beyond elastic measures, we conduct a comprehensive analysis across all temporal models. Specifically, our methodology involves evaluating measures that have channel-dependent and channel-independent variants, and tallying the number of times each variant outperforms its counterpart for each dataset.

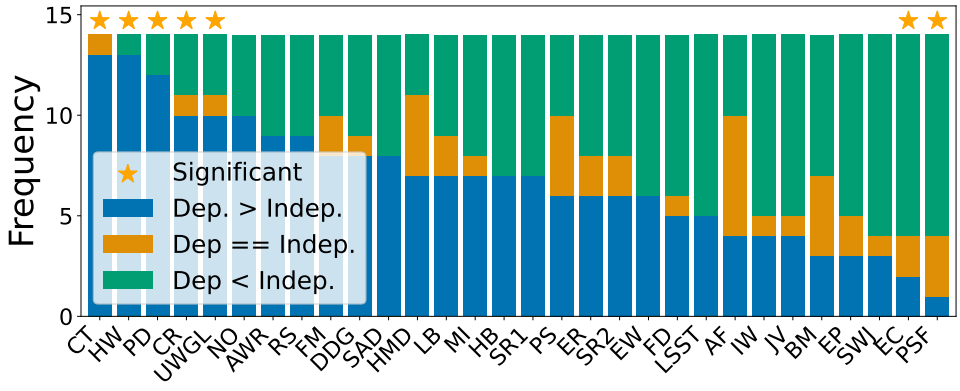


Figure 3.10: Histogram of the number of measures whose channel-dependent variant outperforms the channel-independent variant, and vice versa, for each dataset. Stars indicate significant differences between the models according to a two-sided Wilcoxon test ($p < 0.1$). More information for the datasets is included in Table 3.3.

Table 3.9: Pairwise comparison of channel-dependent vs. independent model for different temporal models.

Measures	Diff.	>	=	<
All dependent	≈	180	42	175
Sliding-D	✓	19	3	8
Elastic-D	✗	60	14	76
Kernel-D	✓	53	10	27
Model-based-D	≈	33	6	21
Embedding-D	≈	38	8	44
Independent measures	≈	0	360	0

The results in Fig. 3.10 reveal that there are indeed datasets where the channel-dependent variant outperforms its counterpart for most measures (i.e., the left-most bars), and vice versa (i.e., the right-most bars). Furthermore, the star-highlighted bars indicate that the channel-dependent model significantly outperforms the independent model on six datasets, while the opposite is true for two datasets. The remaining datasets show no significant difference between the models, which might be counterintuitive in some cases when only considering the ratio of wins and losses (i.e., Dep. > Indep., and Dep. < Indep.).

Notice, however, that the Wilcoxon signed-rank takes into consideration both the magnitude and direction (i.e., sign) of the differences in accuracy. For instance, on the StandWalkJump (SWJ) dataset, the channel-dependent model was better on 3/14 measures, tied on 1 measure, and was worse on 10/14 measures. Even though this distribution of wins and losses is similar to that of EthanolConcentration (EC), a Wilcoxon test on the raw accuracies of the two models yields a p-value of 0.322 in the case of SWJ, and 0.008 in the case of EC. Consequently, significant differences between the models can not be concluded for SWJ as — despite the skewed ratio in losses to wins — the differences in accuracies when Dep. < Indep. were relatively small.

Based on our findings, we validate the claim that the choice of the channel-dependency model is data-dependent. Therefore, the optimal choice of channel-dependency model should be determined through training and validation, as suggested by the introduction of adaptive DTW (DTW-A) in [219].

To gain deeper insights into the two channel-dependency models, we perform a pairwise comparison between them for each temporal model. Table 3.9 shows that there is no statistically significant difference between the two models in general. However, we do observe significant differences at the level of temporal models. Specifically, for sliding and kernel measures, the dependent model significantly outperforms the independent model, whereas for elastic measures, the channel-independent model is significantly better. Furthermore, we observe that although no statistical difference is observed for model-based and embedding measures, the channel-dependent model outperforms its counterpart in at least half of the comparisons. These findings indicate that, in general, independent consideration of time series channels is advantageous only for elastic measures. This conclusion also aligns with our previous findings in Section 3.5.1. This discrepancy between elastic measures and the other temporal models can be attributed to the types of distortions that they aim to correct for. Sliding measures, for example, correct for shifts between time series, which are typically caused by uncalibrated sensor arrays or different starting points of measurements. Consequently, those shifts are likely to be the same for all channels, favouring channel-dependent alignment in case such distortions are present. Elastic measures, on the other hand, correct for local distortions through time warping, which are generally caused by short-term events or measurement errors like sensor latency or jitter. The nature of such distortions make that they are more likely to happen only in one or a few channels, and not across all channels, which makes that channel-independent alignment is more effective in these cases.

3.5.2 Task 2: Clustering

To extend our findings on classification to other downstream applications, we perform a study on clustering. For this experiment, we use Partitioning Around Medoids (PAM) [128], a clustering algorithm that iteratively updates k medoids, which are actual time series from the dataset, and assigns each time series to its closest medoid until convergence. The method is parameter-free besides the choice of k, and performance directly depends on the distance matrix of the respective measure, making it an appropriate proxy to evaluate distance measures. In view of computational feasibility, we focus on the most prominent lock-step, sliding, and elastic measures under unsupervised parameter settings, as their comparisons constitute the main findings of our study so far and as clustering is inherently an unsupervised task. Furthermore, we conduct the evaluation on the UEA archive with Nonorm and report the average performance over 10 random initializations to mitigate variance.

Table 3.10 presents the pairwise comparison of Lorentzian and SBD variants with Euclidean distance, based on the Rand Index (RI) [206] which measures the similarity between the cluster assignment of PAM and the ground truth of the dataset, with a value of 1 indicating perfect agreement and 0 indicating no agreement. The results reconfirm our finding that sliding measures with SBD-D significantly outperform lock-step measures. Lorentzian again outperforms Euclidean among the lock-step measures, though lacking statistical significance in

Table 3.10: Pairwise comparison of lock-step and sliding measures with Euclidean and elastic measures with SBD-D on the task of clustering on the UEA archive (Nonorm). See Table 3.4 for column descriptions.

Clustering Method	Parameters	Diff	Avg. RI	>	=	<
SBD-D	—	✓	0.7522	20	0	10
SBD-I	—	✓	0.7408	18	0	12
Lorentzian	—	≈	0.6979	15	3	12
Euclidean	—	≈	**0.6647**	**0**	**30**	**0**
DTW-D	$\delta = 100$	≈	0.7048	17	2	11
DTW-I	$\delta = 100$	≈	0.7384	17	0	13
ERP-I	$\delta = 100$	≈	0.7296	17	0	13
MSM-D	$c = 0.5 * \sqrt{d}$	≈	0.6785	14	0	16
MSM-I	$c = 0.5$	≈	0.7117	16	0	14
ERP-D	$\delta = 100$	≈	0.7001	16	0	14
TWE-I	$\lambda = 0.5, \nu = 0.01$	≈	0.7119	17	1	12
TWE-D	$\lambda = \sqrt{d}, \nu = 0.0001$	✗	0.6582	11	0	19
LCSS-I	$\delta = 5, \epsilon = 1.0$	✗	0.6638	8	0	22
LCSS-D	$\delta = 10, \epsilon = 0.5 * \sqrt{d}$	✗	0.6205	6	0	24
SBD-D	-	≈	**0.7522**	**0**	**30**	**0**

this case. Furthermore, the comparison of elastic measures with SBD-D in Table 3.10 shows that SBD-D ranks first in terms of average performance, and that again no elastic measure significantly outperforms SBD-D with unsupervised parameters, which is consistent with the findings from Section 3.5.1. In conclusion, the study on clustering demonstrates the generalizability of our previous findings from the classification task.

3.5.3 Task 3: Anomaly Detection

We also perform a study on anomaly detection (AD), which involves computation of distances between MTS subsequences to identify anomalous periods in a time series. We consider the top-performing lock-step, sliding, and elastic measures under unsupervised parameter settings. MSM and TWE measures are excluded in this study for their extremely high computation cost (estimated 3-4 months of computation time). We employ 1-NN detection as our AD algorithm [205], which involves computing for each w-length subsequence in an MTS, the closest w-length subsequence from that same MTS. The distance between them indicates the anomaly score for each subsequence, where high values indicate greater dissimilarity from the rest of the time series, suggesting a higher likelihood of an anomaly. In our experiment we use $w = 100$ and a stride of 50 (the number of time points to shift between consecutive subsequences) to keep the experiment tractable for expensive measures.

Table 3.11 presents the average Volume Under Surface Precision-Recall (VUS-PR) [189] of the measures across datasets of the TSB-AD-M archive [155], which contains 200 MTS with

Table 3.11: Pairwise comparison of lock-step, sliding, and elastic measures on the TSB-AD-M archive, using a 1NN anomaly detector. See Table 3.4 for column descriptions.

AD Methods	Diff	Avg. VUS-PR	>	=	<
Lorentzian	✗	0.4238	62	8	130
SBD-D	✗	0.3720	69	6	125
DTW-D	✗	0.3909	55	5	140
DTW-I	✗	0.3694	46	4	150
SBD-I	✗	0.2469	58	2	140
Euclidean	≈	**0.4413**	**0**	**200**	**0**

annotated (i.e., ground truth) anomalies. The table also shows the number of datasets where each measure significantly outperforms, is equal to, or is outperformed by Euclidean distance, as well as statistical significance of the differences. We observe a clear discrepancy with the results on classification and clustering: here Euclidean distance significantly outperforms all other measures. The ranking of measures is almost perfectly inverse to the rankings observed for classification and clustering. While these results might seem counter-intuitive, they are actually in line with expectations for AD. Namely, where it is imperative for classification and clustering to *ignore or correct for distortions*, AD actually focuses on *identifying distortions*, as these frequently correspond to anomalies. As lock-step measures do not address distortions, their distance score will be more sensitive to them, making it a key indicator to identify an anomaly. Sliding and elastic measures, on the other hand, correct for distortions in their distance computation, thereby losing critical information. These observations show the importance of considering the downstream task in measure selection for similarity search, and particularly, to what case of similarity search this task belongs: the case where distortions should be *corrected* for, or the case where distortions are the *target*.

Additionally, the strong performance of Euclidean distance can be attributed to the subsequence-based nature of anomaly detection. When computing distances between subsequences, we inherently compare each subsequence with all possible windows of the reference time series, which naturally compensates for phase shifts that would otherwise disadvantage lock-step measures. This observation helps explain why Euclidean distance performs comparably in this context to how sliding measures performed in classification and clustering, as it is able to capture the distortions that sliding measures are designed to correct for. While this hypothesis aligns with our theoretical understanding, we note that it cannot be directly validated through classification experiments as current MTS classification archives like UEA only contain whole time series classification tasks, not subsequence classification tasks; there currently exist no open-source datasets for subsequence classification tasks.

3.5.4 Runtime Analysis

Up to this point, our analysis has focused exclusively on the discriminative power of the measures. In practical applications, the computational efficiency of these measures is equally critical. Figure 3.11a presents a comparative analysis of the accuracy-to-runtime performance of the top measures per category on the task of classification. The reported accura-

cies are averaged across all 30 datasets from the downsampled UEA archive [24] under the supervised setting, while the runtime represents the median inference time over five runs on the AtrialFibrillation dataset. We focus on this dataset as taking the average runtime over all datasets would be misleading due to the significant variation in dataset sizes — computing a meaningful average across datasets where runtimes range from seconds to days is impractical. The AtrialFibrillation (AF) dataset serves as a good representative case, being medium-sized in the number of time points (640), which is the main differentiating factor in the computational complexity of the measures (i.e., practically all temporal models scale linearly with the number of time series and channels, as shown in subsequent experiments). At the same time, AF is small enough to allow running the more computationally expensive methods multiple times, which is not possible on largest datasets in the archive as such measures would take would take multiple hours or even days to complete a single run. All reported times exclude data loading and preprocessing (i.e., resampling, imputation, and parameterization), and are averaged over 10 runs to account for variations.

In line with the theoretical analysis in Section 3.4, we observe that lock-step measures are generally the fastest, with the top-performing lock-step measure, Lorentzian, exhibiting a runtime of only 0.004 seconds on AtrialFibrillation dataset, albeit with a relatively modest accuracy of 0.60. SBD-D and SINK-D, both of which leverage FFT, also lie on the Pareto frontier, i.e., the sets of methods that achieve the highest accuracy at a given runtime. These measures outperform lockstep by a significant margin in downstream accuracy, with SBD-D reaching up to 0.65, while maintaining a relatively low runtime of 0.05 seconds. Elastic measures do achieve marginally higher accuracies (≈ 0.66), yet this improvement requires a high computational cost, with MSM-I taking 4982 seconds and DTW-D taking 297 seconds, making them 4-5 orders of magnitude slower than SBD-D. Again note though that elastic measures and their kernel variants require parameter tuning in the absence of domain experts, which further multiplies the runtime proportional to the parameter searching space (which is not included in the reported runtimes). For instance, given our parameter space for MSM, the total runtime would approximate 3 hours (including both parameterization and inference) on this relatively small dataset, assuming parallel execution of validation runs. Considering the quadratic complexity of elastic measures, this would result in weeks of runtime on the largest datasets in the UEA archive, as observed during the execution of our experiments. Lastly, feature-based, model-based, and embedding measures generally fall below the Pareto frontier with exception of GRAIL, which remains near the Pareto frontier by benefiting from high inference speed through dimensionality reduction.

Note that TS2Vec and TLoss were excluded from this analysis due to their dependency on GPU acceleration, making them incomparable to all other CPU-based measures. Regarding measure scalability, the results in Figure 3.12 reveal that lock-step, sliding, elastic, and kernel measures scale accordingly to their theoretical complexity w.r.t. the length of the time series (cf. Section 3.4). Furthermore, we see that current model-based measures scale linearly with the length of the time series, and that GRAIL scales identically to SBD due to their reliance on FFT. Regarding the number of channels c, Figure 3.13 reveals that empirical scaling w.r.t. c can deviate substantially from the theoretical worst-case complexity. Namely, we find for elastic measures that while both channel-dependency variants share the same theoretical complexity, the channel-dependent variants of measures often exhibit sub-

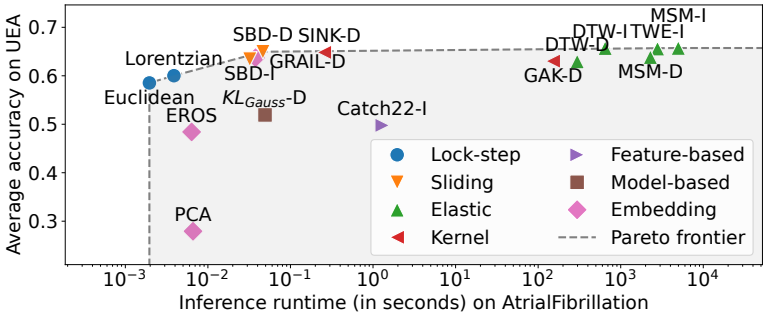


Figure 3.11: Accuracy-to-runtime comparison across all categories.

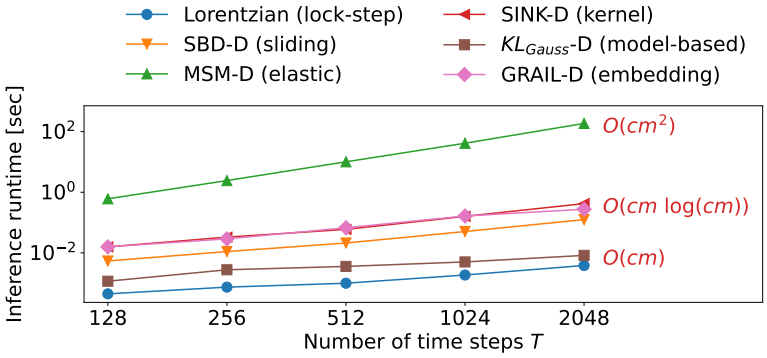


Figure 3.12: Effect of time series length on the runtime of the top measures per category.

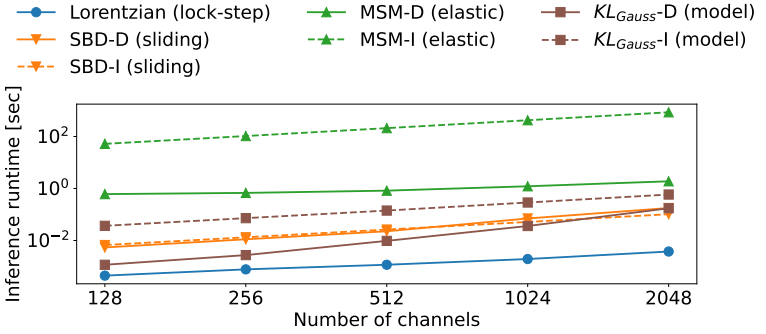


Figure 3.13: Effect of the number of channels on the runtime of the top measures per category.

stantially lower runtimes. This is possibly due to the construction of a single warping path, reducing the overhead from resursive method calls and enabling vectorization to compute the cost of alignments. In summary, these findings emphasize the importance of selecting temporal and channel-dependency models that balance accuracy and runtime in practical applications.

3.6 Summary of Observations and Guidelines

Our evaluation revealed several key observations about MTS distance measures:

1. *Normalization methods:* While MinMax and Mean normalization has been considered optimal for UTS [193], we find that existing normalization methods do not significantly improve classification accuracy compared to no normalization (Nonorm), suggesting a need for MTS-specific normalization approaches.

2. *Channel-dependency models:* Contrary to previous studies that only considered channel-dependency for elastic measures [218, 219], we show that the choice of channel-dependency model impacts all temporal models, with the optimal choice generally being the channel-dependent variant except for elastic measures, where channel-independent variants generally performs better.

3. *Deep-learning-based measures:* Unlike their success in other domains, deep learning-based measures do not necessarily outperform traditional measures in the MTS domain, even with supervised tuning.

4. *Feature-based measures:* underperform due to their focus on global properties and lack of channel-dependent features.

5. *Ensemble measures:* While ensemble methods have led to consistent improvement for UTS [153], we find their effectiveness for MTS heavily depends on the selection of components, with no guarantee of improvement. This hints towards the need for a further study on principled approaches for selecting and combining ensemble components in the MTS domain.

6. *Optimal measure choice:* In contrast to previous work that focused solely on classification as a downstream task [24, 193, 218], we show through evaluating on classification, clustering, and anomaly detection that the optimal choice of measure depends strongly on whether the downstream task requires correction or preservation of distortions.

Guidelines for Measure Selection.

Based on these observations, we provide the following guidelines for selecting a distance measure for MTS similarity search when the downstream task requires *correction of distortions* (e.g., classification, clustering, pattern matching):

- In the **general case**, we recommend SBD-D, a parameter-free sliding measure that handles global misalignment, offering highly competitive performance at low computational cost.

Table 3.12: Guidelines for measure selection.

Available time	Best temporal model	Best measure
Limited time (< ms)	Lock-step	Lorentzian
General case (ms — sec)	**Sliding**	**SBD-D**
Unlimited time (days)	Elastic	MSM-I + tuning

- In the case of **runtime performance** being the primary concern, lock-step measures are the optimal choice. Among them, *Lorentzian* outperforms the commonly used Euclidean distance, making it the recommended choice for performance.

- In the case **maximizing accuracy** being the primary concern and runtime is not a constraint, a channel-independent elastic measure like *MSM-I* with supervised tuning is the best option. However, it is noted that this measure requires significant runtime on large datasets for both parameterization and inference.

- In the case of selecting a **channel-dependency model** for a given distance measure, we recommend only employing the channel-independent variant for elastic measures. For all other measures, the channel-dependent variant is generally preferred.

We summarize these guidelines in Table 3.12.

When the downstream task requires *preservation of distortions* (e.g., anomaly detection), we provide the following guideline:

- Choose a measure family that **does not correct for distortions**. Lock-step measures are the most efficient and effective choice here, with the recommended choice being Euclidean distance.

Note that these guidelines were obtained by studying distance measures in the context of MTS of *equal length* and *equal number of channels*. Appropriate preprocessing steps are expected to be performed (prior to analysis) to ensure the applicability of these guidelines.

3.7 Reflection and Conclusions

In this chapter, we addressed the data-level extension of similarity search by conducting a structured evaluation of MTS distance measures. Our goal was to understand how to effectively compare MTS, a fundamental prerequisite for building efficient search algorithms.

Chapter summary. We benchmarked 30 standalone measures across 8 categories and 2 channel-dependency models, utilizing 13 normalization methods on 30 datasets, and evaluating over 3 downstream tasks with proper parameter tuning. This evaluation was structured along the three key axes of MTS comparison: normalization, temporal model, and channel-dependency model. Our results extend findings of prior works to the multivariate case but also provide insights specific to MTS distances: (a) SBD-D offers the best accuracy-to-runtime trade-off across all temporal models, delivering performance comparable to elastic measures while being significantly faster and parameter-free, (b) no existing normalization

method provides significant benefit over not normalizing, highlighting the need for MTS-specific normalizations to be developed, and (c) channel-independent variants of measures prove beneficial only for elastic measures.

Transferable insights. These findings have broader implications for the field of multivariate similarity search. First, they demonstrate that the principles that work well for UTS do not translate directly to the multivariate setting – a key insight that motivates the design of future distance measures. Second, our comprehensive evaluation framework, considering normalization, temporal models, and channel dependencies, establishes a methodology for evaluating future developments in MTS distance measures.

Limitations and path forward. While our evaluation provides a comprehensive overview, it is not exhaustive. The performance of distance measures can be application-dependent, and our findings are based on a specific set of datasets and tasks. Furthermore, this chapter focused on the effectiveness of distance measures, leaving the challenge of their efficient computation largely unaddressed. Many of the best-performing measures, particularly elastic ones, are computationally expensive, making them impractical for large-scale search without specialized indexing techniques. These insights are particularly valuable as we move forward to tackle the computational challenges of multivariate similarity search in the next chapter.

Reflection on research question. Reflecting on our central research question of extending similarity search to the multivariate setting, this chapter has made significant progress on the data-level extension to multivariate time series. Our evaluation has demonstrated that it is indeed possible to effectively compare multivariate time series, albeit with important caveats. The strong performance of SBD-D shows that we can achieve meaningful multivariate similarity search without sacrificing computational efficiency. However, our findings about normalization and channel dependencies reveal that simply extending univariate approaches is insufficient – truly effective multivariate similarity search requires techniques specifically designed for the multivariate nature of the data. This validates our thesis's premise that extending similarity search to the multivariate setting requires fundamental rethinking of existing approaches.

CHAPTER 4

Efficient Subsequence Search in Multivariate Time Series

Having established *what* constitutes an effective comparison for multivariate time series, we now turn to the critical question of how to perform that comparison efficiently. Specifically, we focus on the problem of subsequence search, which involves finding occurrences of a query pattern within longer time series. This problem is fundamental to many applications, such as anomaly detection and pattern recognition, but it poses significant computational challenges due to the high dimensionality and complexity of multivariate time series data. In this chapter, we introduce the MS-Index, a novel algorithm designed to efficiently perform k-NN subsequence search in multivariate time series. Our approach combines a unique indexing strategy with the efficiency of accelerated distance computation to reduce search times to mere milliseconds, even for large datasets. Specifically, a comprehensive experimental evaluation across 34 datasets demonstrates that the MS-Index outperforms state-of-the-art methods by orders of magnitude, achieving speedups of up to 1000x.

The contents of this chapter have previously appeared in d'Hondt et al. [78] and Pelok & d'Hondt [198].

4.1 Introduction

As introduced in Chapter 1, this thesis explores two fundamental extensions to similarity search: the data-level extension for comparing multivariate data, and the relation-level extension for detecting multivariate patterns. This chapter focuses on the data-level extension, specifically addressing the challenge of efficient subsequence search in MTS.

While Chapter 3 established guidelines for selecting appropriate distance measures for MTS, the computational challenges of efficiently searching for similar MTS remain unresolved. This is particularly true for subsequence search (cf. Section 2.1.1), where we need to find

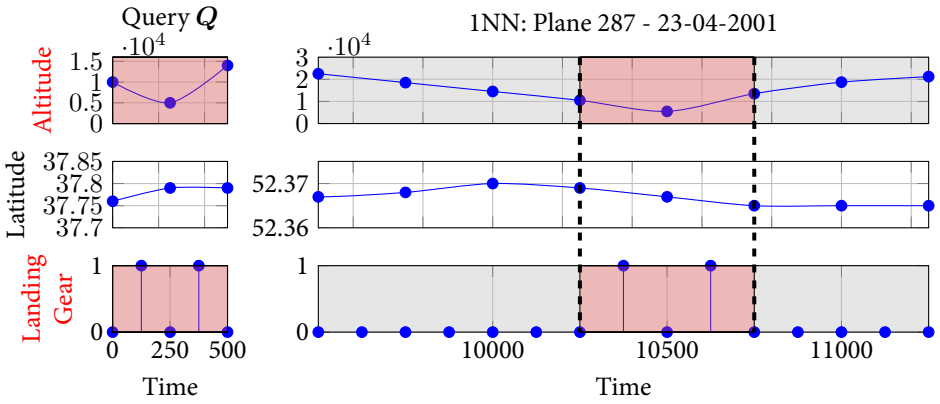


Figure 4.1: Example query and 1NN for MTS of synthetic airplane data. Altitude and landing gear are the query channels. The highlighted boxes (red) are the considered subsequences.

4

similar occurrences of a query pattern within longer time series. The need for such flexible search capabilities is evident in many real-world scenarios. 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 (e.g., heart rate and blood pressure) to understand the root cause of their condition. This requires three key capabilities that current solutions lack: (a) native support for MTS, (b) comparison of subsequences rather than whole time series, and (c) the flexibility to select relevant channels at query time (e.g., only the sensors relating to the patient's relevant vital signs). Another notable example requiring such capabilities is the analysis of sensor data from airplanes [32], 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 Fig. 4.1) 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 create scalable search algorithms for subsequence search on univariate datasets, the work on MTS search is still rudimentary. Current solutions only support searching for whole time series (whole-search) instead of subsequences, and only on a fixed, pre-determined set of channels. When these solutions are extended for subsequences, their performance suffers significantly, and handling multiple channels only exacerbates the problem. 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 restrictive assumptions of the user query, such as user-defined thresholds on each channel.

In this chapter, 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 subse-

quences, allowing the user to choose whatever is most appropriate for their use case.[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) [15, 178]. 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 [88, 178], 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 index and algorithm for k-NN subsequence search on MTS under Euclidean distance (Section 4.3).
- We propose a general set of optimizations to further improve any search solution using R-trees and/or DFT approximations (Section 4.3.4). These are shown to improve the performance of MS-Index by a factor of 4.
- We provide a general approach to extending current solutions for UTS to the multivariate case, which provide baselines for our evaluation (Section 4.4).
- We conduct a thorough evaluation of MS-Index across 33 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 4.5).

4.2 Background

This section provides the necessary background for understanding our contribution. We build upon the foundational concepts introduced in Chapter 2, which covered key techniques for time series representation (such as DFT approximation) and efficient indexing structures (such as R-trees). Rather than repeating these preliminaries, we focus here on defining the distance measure used throughout this chapter and reviewing related work specific to our subsequence search problem.

[1] Supporting normalized subsequences is more challenging than indexing normalized time series, as it cannot be done through preprocessing the data. Furthermore, while Chapter 3 established that not normalizing is actually the best option for MTS, in view of the wide usage of z-normalization for UTS [8, 88, 102, 133], we still add support for it here, so users have the option to use it if they want to.

Table 4.1: Overview of related work

		Name	Ref	Type	Notes
UTS	Whole	VA+ file	[90]	Index	
		ISAX2+	[39]	Index	
		DSTree	[232]	Index	
	Sub.	ST-index	[88]	Index	
		KV-Match	[238]	Sequential	Custom definition of normalized subsequences
		MASS	[178]	Sequential	
MTS	Whole	Vlachos	[230]	Index	Does not support normalized subsequences
	Sub.	Bhaduri	[32]	Index	Only supports r-range queries
		MS-Index (ours)		Index	

Distance. In line with previous studies on MTS similarity search [24, 213, 218, 219, 230, 243], we use Euclidean distance (ED) to measure the distance between time series, which is defined over MTS as:

$$d(\boldsymbol{x}, \boldsymbol{y}) = ||\boldsymbol{x} - \boldsymbol{y}||_2 = \sqrt{\sum_{i \in c_{\boldsymbol{xy}}} \sum_{j=1}^{t} (\boldsymbol{x}_j^{(i)} - \boldsymbol{y}_j^{(i)})^2} \tag{4.1}$$

with $c_{\boldsymbol{xy}} = c_{\boldsymbol{x}} \cap c_{\boldsymbol{y}}$ the common channels of $\boldsymbol{x}$ and $\boldsymbol{y}$, and $t = \min(|\boldsymbol{x}|, |\boldsymbol{y}|)$ the length of the shortest time series. Note that this definition corresponds to the multivariate L_p definition in Chapter 3, with $p = 2$. 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 [24, 32, 230].

It is important to note that while our findings in Chapter 3 showed that lock-step measures like ED are generally outperformed by other temporal models (particularly sliding measures like SBD) in MTS classification tasks, the subsequence search setting fundamentally changes the dynamics of distance measures. In subsequence search, we inherently compute the distance between the query and all possible windows over the reference time series, which naturally compensates for temporal misalignments that would otherwise disadvantage lock-step measures. This window-based comparison effectively mirrors the core principle behind sliding measures like SBD, which finds the optimal alignment between time series by considering all possible shifts. The key difference is that in subsequence search, this alignment is built into the problem definition rather than the distance measure itself. This relationship between subsequence search and sliding measures combined with the findings of Chapter 3 provides theoretical backing for why simpler lock-step measures can be effective in this specific context.

4.2.1 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 4.1.

UTS whole-matching. To the best of our knowledge, the most complete comparison of similarity search algorithms for UTS whole-matching is that of Echihabi, et al. [81]. In their work, the authors compare 10 solutions, including 7 indices, 2 sequential-scan algorithms, and 1 hybrid method. The results show that the best solution depends on the number of time series in the dataset and their length. Index-based algorithms generally outperform the other algorithms for UTS whole-matching. For small datasets that fit in memory, VA+ file [90] and iSAX2+ [39] are the best, with VA+ file dominating on long time series. For larger datasets, the DSTree index [232] outperforms all others. DSTree is a tree-based index that uses the Extended Adaptive Piecewise Constant Approximation (EAPCA) summarization technique to estimate the distance of the query to groups of time series. The tree differentiates time series at each node based on the mean and variance in their summarization, and dynamically increases the resolution of the summarizations at each level to obtain increasingly tighter bounds on distances during search. Whole-matching indices can be naively extended to support subsequence search by treating each subsequence as an individual time series, and adding it in the index. However, given the large number of subsequences per time series if $|\boldsymbol{q}| \ll m_T$ this approach overwhelms the indices and degrades their performance. We verify this hypothesis in Section 4.5.

UTS subsequence search indices. Multiple indices have been proposed for UTS subsequence search. The ST-index [88] for r-range queries first extracts a DFT approximation of each subsequence of length $|\boldsymbol{q}|$ in a time series. These $2f$-dimensional vectors form a trail in the $2f$-dimensional space, which is segmented into sub-trails using MBRs.[2] The MBRs are then indexed in an R*-tree [28], to enable efficient search. More recents improvements through Dual Match and General Match [96, 173] use 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 [238], 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.

Another notable UTS subsequence search index is KV-Match [238], which focuses on querying for similar subsequences within *a single time series*. KV-Match works by building an index that stores disjoint windows over the reference time series based on their mean and variance. At query time, subsequences are queried that have similar means and variances to parts of the query time series. The index supports both normalized and raw subsequences under Euclidean distance and DTW, and arbitrary length queries. However, when querying

[2]The dimensionality of the DFT approximation is $2f$ because each of the f Fourier coefficients has a real and an imaginary part.

for normalized subsequences, KV-Match limits the search space by only considering subsequences with similar means and variances to the query *before normalization*, to better accomodate their indexing approach. Still, the algorithm can be adapted to support querying a database of time series building one KV-Match per time series and iteratively querying each, and dropping the filter on means and variances before querying normalized subsequences. Accordingly, we adapt KV-Match and compare it to our method in Section 4.5.

Other UTS subsequence indices include L-match [89] and TS-index [46], 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 4.5, our work outperforms both ST-index and 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) [178] 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 raw time series is equivalent to the point-wise multiplication of their Fourier transforms [15]. 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:

$$[\langle \boldsymbol{q}, \boldsymbol{t}_{1,|\boldsymbol{q}|} \rangle, \dots \langle \boldsymbol{q}, \boldsymbol{t}_{|\boldsymbol{t}|-|\boldsymbol{q}|+1,|\boldsymbol{q}|} \rangle] = F^{-1}(F(\boldsymbol{q}) \otimes F(\boldsymbol{t})) \tag{4.2}$$

where F, F^{-1}, and $\otimes$ are the Fourier transform, inverse Fourier transform, and point-wise multiplication, respectively. Then, as the L_2 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 Eq. 4.2 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 L_2 distance. We will be using MASS in this work, as a subroutine in our algorithm.

MTS whole-matching. The first index for MTS whole-matching was proposed by Vlachos et al. [230], focusing on r-range queries under DTW and LCSS distance, besides L_2 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 [28]. 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 L_2 distance essentially reduces to a version of Dual Match [96] (without the use of DFTs), which – as discussed earlier – prevents the index from supporting normalized subsequences.

MTS subsequence search. Recently, we proposed a novel algorithm for *variable-length* MTS subsequence search named MULISSE [198], as a multivariate extension of the state-of-the-art UTS subsequence search algorithm ULISSE [152]. MULISSE extends iSAX2+ [39] 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 4.5. However, note that the lightweight design of MULISSE is specifically tailored for variable-length search, where the index must store information about subsequences of all possible lengths. However, for fixed-length search, this focus on minimal index size is less critical, meaning that a more specialized index could achieve better query performance through tighter bounds and more effective pruning strategies while still keeping the index size manageable. This is exactly the approach taken by the MS-index, and shown to be effective in our experiments.

To the best of our knowledge, the only other solution for MTS subsequence search is by Bhaduri, et al [32]. 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 querying on normalized subsequences [230], or (b) optimize for index size rather than query performance [198] (*MULISSE*). There are UTS solutions that could be extended to MTS, though at the cost of query performance and index efficiency. This is either because (a) they are not build for subsequence search (*DSTree*), (b) they are sequential scan algorithms (*MASS*), or (c) they are simply designed for UTS (*ST-index*). Therefore, in this work we propose a method that is custom-built for our problem, and we evaluate it against alternative solutions based on the most promising methods from related work.

4.3 MS-Index for Subsequence Search on Multivariate Time Series

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 t, an exhaustive search would require computing the L_2 distance between all $n * (t - |\boldsymbol{q}| + 1)$ subsequences and the query $\boldsymbol{q}$, with each computation costing $|c_{\boldsymbol{q}}| * |\boldsymbol{q}|$ time. Second, the requirement to support ad-hoc selection of query channels precludes the use of approximations that involve merging channels, such as PCA [243].

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. The solution involves three key steps (cf. Figure 4.2): (a) summarizing the MTS subsequences through DFT approximations, (b) building an R-tree index on these approximations, 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 these steps, 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 to compared to the state-

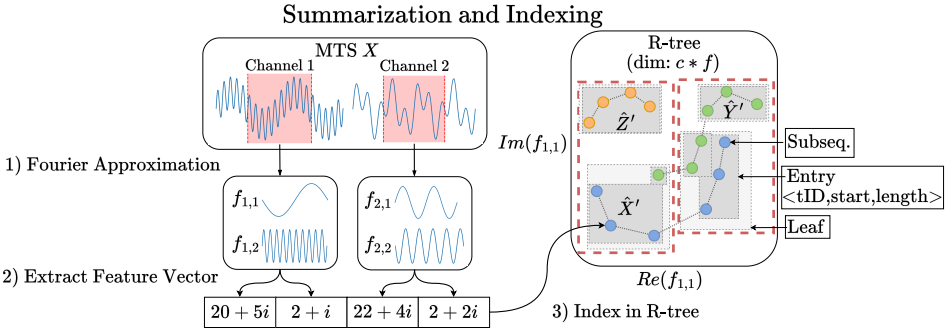


Figure 4.2: Summarization and indexing of MTS subsequences. We represent the indexed subsequences of three MTS in the feature space with blue, green, and orange nodes. Time-neighbouring subsequences are connected with lines.

4

of-the-art. We also propose a number of optimizations to further improve the efficiency of the search process (Section 4.3.4). 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 [61, 88, 90, 177, 203, 230].

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 Appendix A.

We now present the key components of MS-Index, starting with time series summarization (Section 4.3.1) and indexing (Section 4.3.2), followed by the query execution algorithm (Section 4.3.3), and the optimizations (Section 4.3.4).

4.3.1 Summarizing All Subsequences with DFTs

As discussed in Section 2.2, a key 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 [88, 90, 177]. 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 4.3,

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 4.5 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 [105, 190, 241]; 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 4.3, which shows that the cumulative contribution to distance increases much faster than the contribution to energy, implying that one can already cover ∼80% 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 small uniformly random sample S of subsequences of length $|\boldsymbol{q}|$ 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 4.3). 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* $|\boldsymbol{q}|$-length subsequences in the dataset. Notice that the summation in Equation 2.1 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 $\tilde{\boldsymbol{t}}$ we will denote the matrix of size $c \times f$ that stores the DFT approximations for subsequence $\boldsymbol{t}$. Also, $\tilde{\boldsymbol{t}}'$ 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 Fig. 4.2).

[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, showing non-noticeable differences in performance with larger sample sizes.

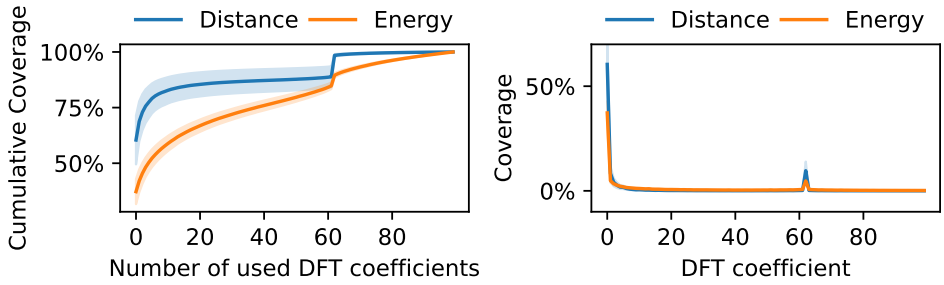


Figure 4.3: Cumulative and absolute percentage of total distance and energy across DFT coefficients on the temperature channel of a weather dataset.

4.3.2 Indexing of the MTS

The next step involves indexing the generated feature vectors in an R-tree (step 3 in Fig. 4.2), in order to enable efficient query execution.[4] Notice, however, that the total number of feature vectors is $O(n * |\boldsymbol{t}|)$, which can lead to a large and inefficient R-tree when the number of time series n is large, negatively impacting the query execution performance.

We mitigate this problem as follows. First, we build the R-tree in a bottom-up fashion with leafs containing more than one entry (i.e., leaf size $L > 1$), to reduce the number of nodes in the tree. Once the leafs 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 feature vectors* 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. We represent these groups with a single entry that contains: (a) the start and end position of the continuous subsequence, and, (b) a Minimum Bounding Rectangle (MBR) that encloses all feature vectors in the group. 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 [88]. 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

[4]We also experimented with other spatial indexes like KD-trees [30], but found that the recall stage of our algorithm (i.e., Line 14 in Algorithm 1) 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 [88].

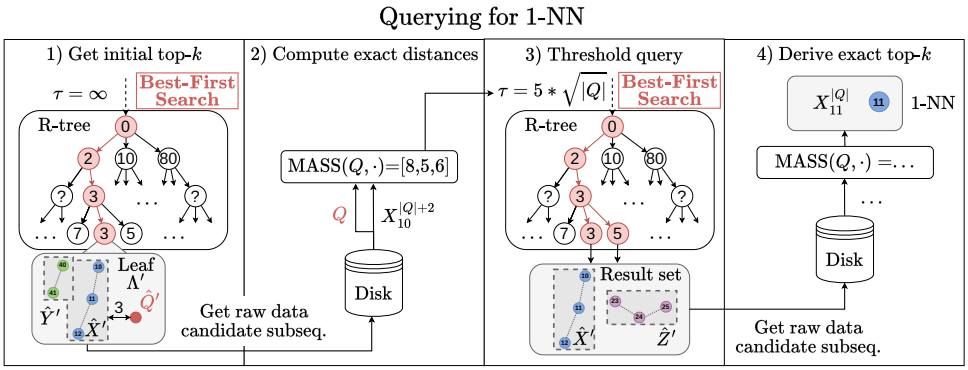


Figure 4.4: Query execution in MS-Index. Numbers in tree nodes indicate the lower bound distance of the respective MBR to the query.

and enables efficient distance computation with MASS. In our experiments, we observed that time-neighboring subsequences are typically grouped into entries containing between 8 and 50 subsequences, depending on the characteristics of the dataset.

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 4.2, where the nodes represent the indexed subsequences of different MTS (differentiated by color), and the edges connect time-neighbouring subsequences of the same MTS.

4.3.3 Query Execution

We want to facilitate efficient execution of k-NN queries on arbitrary subsets of channels, decided at query time. Our algorithm probes the index twice. The first probe is for finding an upper bound on the distance to the k-th nearest neighbor, whereas the second probe retrieves all entries within this distance to guarantee correctness of the answer, and computes the exact result. The second probe is necessary to guarantee correctness of the answer.

Queries are executed as follows (cf., Alg. 1 and Figure 4.4, correctness proof follows). First, we extract a feature vector $\tilde{\boldsymbol{Q}}'$ of the query $\boldsymbol{Q}$ (Alg. 1, line 1). 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 $\tilde{\boldsymbol{Q}}'$ (lines 2-7). 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 (lines 8-11). 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 the final k-NN (lines 12-15).[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 $\boldsymbol{Q}$ does not necessarily contain all channels (i.e., $c_{\boldsymbol{Q}} \subseteq \{1, \ldots, c\}$), the R-tree is only queried on the dimensions of the feature space that correspond to the channels in $\boldsymbol{Q}$. This is natively handled by the R-tree algorithm [214].

To illustrate this process, consider a query $\boldsymbol{Q}$ for which the 1-NN is the subsequence $\boldsymbol{X}_{11}^{|\boldsymbol{q}|}$ in leaf Λ' from the previous section. The algorithm will return the entry $\boldsymbol{X}_{10}^{|\boldsymbol{q}|+2}$ in the first probe, as it has the smallest lower bounding distance to $\boldsymbol{Q}$ (Step 1 in Figure 4.4). It then computes MASS($\boldsymbol{Q}$, $\boldsymbol{X}_{10}^{|\boldsymbol{q}|+2}$) on the original subsequences, which outputs the distances $[8, 5, 6]$, and adds $\boldsymbol{X}_{11}^{|\boldsymbol{q}|}$ to the running 1-NN set as it corresponds to the smallest distance of 5 (Step 2). Accordingly, the threshold is set to $5 * \sqrt{|\boldsymbol{q}|}$ and the algorithm performs the second probe, retrieving the entries $\boldsymbol{Z}_{23}^{|\boldsymbol{q}|+2}$ and $\boldsymbol{X}_{10}^{|\boldsymbol{q}|+2}$ (Step 3). After computing the exact distances for these entries with MASS, the algorithm returns $\boldsymbol{X}_{11}^{|\boldsymbol{q}|}$ as the 1-NN to $\boldsymbol{Q}$ (Step 4).

We provide a correctness proof of the algorithm in Appendix A.

4.3.4 Optimizations

We now present four 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 4.3.1, in most real-world datasets, around 60%-80% of the distance between time series is covered by the top 2-5 DFT coefficients (which are generally also the first coefficients [88]). Formally, this concept is expressed as:

$$d^2(\boldsymbol{t}, \boldsymbol{q}) = (\underbrace{d^2(\tilde{\boldsymbol{t}}_f, \tilde{\boldsymbol{q}}_f)}_{\sim 80\%} + \underbrace{d^2(\tilde{\boldsymbol{t}}_{f+}, \tilde{\boldsymbol{q}}_{f+})}_{\sim 20\%})/|\boldsymbol{q}| \tag{4.3}$$

where $\tilde{\boldsymbol{t}}_f$ denotes the DFT approximation an MTS $\boldsymbol{t}$ using the top f coefficients, and $\tilde{\boldsymbol{t}}_{f+}$ denotes the set of c vectors composed of the remaining coefficients. In the DFT bound of Eqn. A.1, 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.

[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.

Algorithm 1: QUERY$(I, \boldsymbol{q}, k)$
Input : An R-tree index I, a query MTS $\boldsymbol{q}$, result set size k
Output: The top-k subsequences in I with the lowest L_2 distance to $\boldsymbol{q}$.

```
1  q̃ ← DFT(q, f)
2  R̂ ← PriorityQueue()
3  τ_k ← ∞
4  for t̃ ∈ BeFS(I, q̃) do                                   // First probe
5  |  if LB(q̃, t̃) > τ_k then break;                       // Early abort
6  |  R̂.push(S, LB(q̃, t̃))
7  |  if R̂.size() > k then τ_k ← R̂.peek().distance;
8  D ← {}
9  for S ∈ R̂ do                                           // Derive threshold
10 |  D ← D ∪ MASS(q, S)
11 τ_k ← KthSmallest(D, k)
12 R ← {}
13 for S ∈ ThresholdQuery(I, q, τ_k * √|q|) do           // Second probe
14 |  R ← R ∪ MASS(q, S)
15 return TopK(R, k)
```

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 $(|\boldsymbol{q}| - f)$ 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 substantially improves the pruning power of the index.

Namely, during index construction, after computing the DFT approximation for each subsequence $\boldsymbol{t}$, we also derive the part of $\boldsymbol{t}$ that is not covered by the top f-coefficients, called its *remainder* $\boldsymbol{t}_{f+} = [\boldsymbol{t}^{(1)} - \text{IDFT}(\tilde{\boldsymbol{t}}^{(1)}), \ldots, \boldsymbol{t}^{(c)} - \text{IDFT}(\tilde{\boldsymbol{t}}^{(c)})]$ with $\tilde{\boldsymbol{t}}^{(i)}$ being the approximation of channel i using the top f coefficients. Note that the remainder can be computed solely based on the time series and its top f coefficients, and does not require knowing all other $|\boldsymbol{q}| - f$ 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 Eq. 4.3. Then, Eqn. 4.3 can be rewritten to utilize the remainders:

$$d^2(\boldsymbol{t}, \boldsymbol{q}) = \frac{d^2(\tilde{\boldsymbol{t}}, \tilde{\boldsymbol{q}})}{|\boldsymbol{q}|} + d^2(\boldsymbol{t}_{f+}, \boldsymbol{q}_{f+}) \tag{4.4}$$

The remainders are of length $|\boldsymbol{q}|$, meaning that the cost is as much as computing the full distance on the original data (i.e., $O(|c_{\boldsymbol{q}}| * |\boldsymbol{q}|)$). 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 $\boldsymbol{q}_{f+}$, and use the reverse triangle inequality to lower-bound the distance between $\boldsymbol{q}_{f+}$

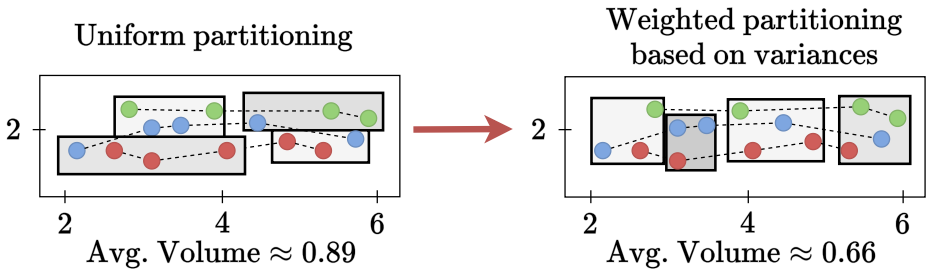


Figure 4.5: Different partitioning strategies for a 2-dimensional feature space; the STR algorithm (left) and the proposed weighted partitioning (right), that leads to smaller MBRs and to tighter bounds. The diagrams represent the feature space, and the points of different colors represent the DFT approximations for subsequences of three different MTS (one color per MTS). For illustration, time-neighboring subsequences are connected with a line.

and the remainders of the indexed subsequences through their distances to pivots. We refer to this latter bound as the *correction term*. We apply this correction to Eqn. 4.4 as follows:

$$\begin{aligned} 0 \quad &\leq (|d(\boldsymbol{t}_{f+}, \boldsymbol{p}) - d(\boldsymbol{q}_{f+}, \boldsymbol{p})|)^2 \leq d^2(\boldsymbol{t}_{f+}, \boldsymbol{q}_{f+}) \Rightarrow \\ d^2(\boldsymbol{t}, \boldsymbol{q}) \quad &\geq \underbrace{d^2(\tilde{\boldsymbol{t}}_f, \tilde{\boldsymbol{q}}_f)/|\boldsymbol{q}|}_{\text{DFT distance}} + \underbrace{(|d(\boldsymbol{t}_{f+}, \boldsymbol{p}) - d(\boldsymbol{q}_{f+}, \boldsymbol{p})|)^2}_{\text{Correction term}} \end{aligned} \tag{4.5}$$

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 pivot closest to the query's remainder, and uses this pivot $\boldsymbol{p}$ to compute the bound, according to Eqn. 4.5. In Section 4.5 we show that this optimization leads to a 2x speedup in query execution.

Tightening the MBRs. The pruning efficiency of the R-tree can be further enhanced by reducing the volume of the MBRs for the R-tree nodes. The R-tree is constructed in a bottom-up fashion using the STR algorithm [143], briefly presented in Section 2.3.1. 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 Fig. 4.5). The first dimension will likely be the main contributor of the L_2 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. The example in Fig. 4.5 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 4.3.1. We leverage this observation by weighing the dimensions (i.e., the number of coefficients per dimension) based on the variance of their values, which is a good proxy for the contribution of the dimension to the

distance between points, as follows. Namely, before constructing the R-tree, we use the sample S of subsequences extracted during the summarization step (Section 4.3.1) 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 [36]. 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 re-distributes the number of partitions across the dimensions, such that it is proportional to the contribution of each dimension to the distance. In Section 4.5 we will show that this optimization leads to a 2-4x speedup in query execution, due to more aggressive pruning at the index.

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* [116].

Caching the DFT twiddle factors. MS-Index computes many Fourier transforms, both during index construction and query execution. We optimize these computations by caching the so-called *twiddle factors* of the Fourier transforms [98]. These twiddle factors are data-independent constants that are used in the computation. In particular, computation of the k'th DFT coefficient requires the k'th twiddle factor for each position j of the subsequence, which is $e^{-2\pi i \cdot k \cdot j/|\boldsymbol{t}|}$, with $|\boldsymbol{t}|$ corresponding to the length of the time series. We therefore precompute these twiddle factors, for $|\boldsymbol{t}| = |\boldsymbol{q}|$ and save them in an array. Notice however that for running MASS on subsequences of length greater than $|\boldsymbol{q}|$ — which becomes necessary because of the grouping at the leafs — we need up to $|\boldsymbol{q}| + L$ twiddle factors per DFT coefficient, with L denoting the maximum leaf size. We therefore restrict the size of subsequences passed to MASS to powers of two, which allows us to maintain a cache of reasonable size, with twiddle factors for lengths in the range $[|\boldsymbol{q}|, |\boldsymbol{q}| + 1, |\boldsymbol{q}| + 2, |\boldsymbol{q}| + 4, \ldots, |\boldsymbol{q}| + 2^{\lceil \log_2 L \rceil}]$. All subsequences are then zero-padded to the next allowed length. Finally, during distance computation with MASS, we limit the output of MASS to only the distances of the subsequences that do not include padded values.

4.4 Extending Existing Algorithms to the Multivariate Case

As mentioned in Section 4.2.1, 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 index-based 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 4.5. Still, it is a useful approach to enable comparison of MS-Index with out-of-the-box extensions of UTS algorithms such as DSTree [232], ST-index [88], and KV-Match [238] on MTS data.

Algorithm 2: UTSBaseline($I, \boldsymbol{q}, k$)
Input : A set of channel-level indices I, a query MTS $\boldsymbol{q}$, a result set size k.
Output: The top-k subsequences in D 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 ← ExhTopK(R̂, q)                               // Comp. distances
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);
```


The general approach is based on the well-known Threshold algorithm [87], 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 on which the top-k is based. In our case, the objects are the MTS subsequences in the dataset, the values are the squared L_2 distances to the query on a certain channel, and the aggregation function is the rooted sum of these distances (cf. Eq. 4.1). The approach works as follows (cf., Alg. 2): (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 $i \in c_{\boldsymbol{q}}$ by querying the corresponding index (lines 2-3), where $c_{\boldsymbol{q}}$ is the set of channel ids included in the query, (c) for each MTS in the union of the local estimates, we go over all subsequences in the MTS and 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 $i \in c_{\boldsymbol{q}}$ to the largest distance in $\hat{R}$ on that channel (lines 5-6), (e) we re-query the channel-level indices with their respective thresholds τ_i, 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). Notice that, since any subsequence that belongs to the global top-k must have a distance at most τ_i on at least one channel $i \in c_{\boldsymbol{q}}$, the result of this algorithm is guaranteed to be correct. Lastly, MASS is used to speed up the exact distance computations (lines 4 and 10).

We implemented this baseline approach using several state-of-the-art UTS algorithms, including DSTree [232], ST-Index [88], and KV-Match [238]. As we will show in Section 4.5, while this approach enables these algorithms to handle MTS queries, it exhibits significantly lower pruning power compared to MS-Index, particularly when the channels in the dataset are uncorrelated. This is mainly because each channel-level index operates independently, without leveraging the relationships between channels.

4.5 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 4.3.4. Since our method and all baselines are exact, our evaluation solely focuses on efficiency; experiments on accuracy would always yield 100%.

Compared methods. First of all, we compare MS-Index to **MULISSE** [198], applying it to fixed-length search by restricting the supported range of query lengths to $|\boldsymbol{q}|$. 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 2.[6] (a) **ST-Index*** [88], a well-known index for subsequence search [173, 256], (b) **KV-Match*** [238], a SOTA index for subsequence search on single long series, extended with per-MTS indices, and (c) **DSTree*** [232], an index for whole-matching [81], adapted by indexing all subsequences. We also include two sequential scan baselines: (a) **MASS** [178] applied to all MTS, and (b) **Brute-force** exhaustive comparison.

DSTree*, ST-Index*, and KV-Match* were extended to handle MTS by using the algorithm described in Section 4.4. Since KV-Match* was originally designed to query a single time series, it was extended to support a dataset of MTS by building an index for each channel and each MTS in the dataset, and by iteratively performing the threshold query in lines 7-11 of Algorithm 2 over all time series, updating the channel-level thresholds along the way. Furthermore, since DSTree* supports only whole-matching, we used it by reducing our problem to whole-matching by indexing all possible subsequences in the dataset. The MASS baseline also required a small modification for handling MTS efficiently. Precisely, as a preprocessing step, the DFTs of the full MTS were computed per channel, and were stored in RAM for later use. At query time, the DFTs of the query were computed, again per channel. Then, we executed MASS to compute the per-channel L_2 distances of all MTS with the query, and summed up those per-channel distances to get the final distance for each MTS subsequence. Finally, we derived the k-NN from the final distances. Unless mentioned otherwise, all algorithms are compared at their best possible configurations and with tuned parameters.

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 [198], 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 [232], and was called as a subroutine in Alg. 2. Due to the lack of publicly available code, all other algorithms were implemented from scratch in Java-11. To prevent implementation bias in parallelism and memory management, all algorithms were run *single-threaded*, and were implemented to operate fully in *main memory*. Our code is available on Github [4].

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

[6] We add an asterisk to their names to differentiate them from their original versions.

Table 4.2: Default query configurations.

	Stocks	**Weather**	**Synthetic**	**Wind**	**UEA (30x)**
n	2000	1300	1600	1	all, see [24]
Avg. m	5590	8692	4096	432,000	see [24]
$\|\boldsymbol{q}\|$	730 (2 y.)	1488 (2 mth.)	1024	1800 (1 h.)	20%
#Channels	5	4	64	10	see [24]

28678 stocks over the period Jan. 2, 1987 to Feb. 26, 2021 leading to an average of 5590 observations per MTS. (b) **Weather.** Segment of the ISD weather dataset [185] containing hourly readings of wind speed, sea level pressure, atmospheric temperature, dew point temperature of 13545 sensors taken between Jan. 1, 2020 and Dec. 31, 2021, leading to an average of 8692 observations per MTS. (c) **Wind.** Sensor output of an active wind turbine sampled every 2 seconds for 10 days [180], 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) **UEA archive.** Popular benchmark archive for MTS classification consisting of 30 labeled real-world datasets from different domains [24], with varying number of MTS, channels, and observations per MTS. The train and test splits for each dataset were merged, to form a single dataset. (e) **Synthetic.** Random walk datasets with 1000 MTS, of 1024 channels and 4096 observations each. These datasets were generated similar to the synthetic dataset used in [81, 82]. The data was drawn 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 $[0, \ldots, 100]$.

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 4.2 summarizes the default query confirms for each dataset. We include more details about the datasets in our code repository [4].

Queries and evaluation metrics. As per standard practice [81], we generated the query workloads for each dataset by randomly selecting $|\boldsymbol{q}|$-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 4.5.2. Unless otherwise stated, we query on all channels of the dataset to ensure that the multivariate aspect of the problem is well represented. When querying on a subset of channels (as in Section 4.5.2), 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.

Outline. Our experimental evaluation begins with a detailed analysis of parameter tuning for the different algorithms (Section 4.5.1). We then examine index construction costs and memory requirements (Section 4.5.2), followed by an extensive analysis of query performance under various conditions including dataset size, query length, and number of channels. Finally, we evaluate the effectiveness of our proposed optimizations for improving query performance (Section 4.5.2-4.5.2).

Table 4.3: Average query time (ms) of MS-Index for different distance coverage (raw datasets).

d_{target}	Insectwingbeat	Stocks	Synthetic	Weather
20%	**10.93**	9.33	3.06	5.91
40%	11.82	9.44	3.12	5.85
60%	13.98	9.37	**3.04**	**4.69**
80%	13.83	**9.16**	3.06	4.89
100%	42.06	9.72	3.28	5.13

Table 4.4: 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
0.001 %	8.63	8.41	**2.97**	4.68
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
10 %	31.63	28.57	4.00	6.54
100 %	38.12	158.10	4.59	100.87

4.5.1 Tuning the Algorithms

Several algorithms in our evaluation require careful parameter tuning to achieve optimal performance. In this section, we analyze the impact of these parameters and justify our choices for the main evaluation.

Number of DFT coefficients for MS-Index and ST-Index* Recall from Section 4.3.1 that the number of DFT coefficients f – used to approximate each MTS channel in MS-Index and ST-Index*– 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%, …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.

Leaf size DSTree*, ST-Index*, and MS-Index support tuning of the leaf size L, i.e., the *maximum* number of entries per leaf. Similar to the number of DFT coefficients, we tuned the leaf size for query time, ranging its value from 0.0001% to 100% (i.e., no constraints on the leaf size) of the total number of entries to index. Looking at the results in Table 4.4, 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. [81], 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. We stress that the described

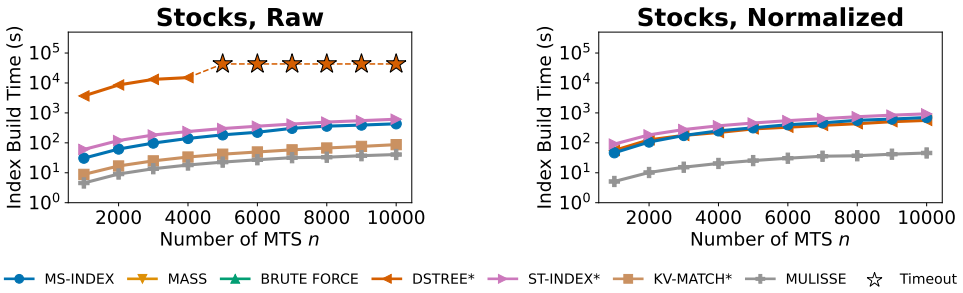


Figure 4.6: Scalability over number of MTS n on *Stocks*. (a) Initialization time for raw subsequences, and (b) normalized subsequences. The y axes are presented in log scale.

configurations for both parameters have no impact on the correctness and completeness of the results — they only impact the efficiency of the respective algorithms.

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 for both raw and normalized subsequences to be 1 for all datasets. For raw subsequences, this implies that the index reduces to a single lookup table for each channel and time series, with each subsequence indexed by its overall mean and variance. For normalized subsequences, this implies that all subsequences end up in the same bucket as they are all guaranteed zero mean and unit variance. This renders the index useless, and it implies that KV-Match* has no pruning power in the normalized case. Instead, it reduces to MASS with a small overhead.

4.5.2 Evaluation Results

Our experimental evaluation consists of a comprehensive analysis of all aspects of MS-Index against competing methods. We begin by examining initialization costs and index sizes, and then continue by analyzing query performance under various configurations and optimizations.

Initialization time Figs. 4.6a-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*

Table 4.5: 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	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%

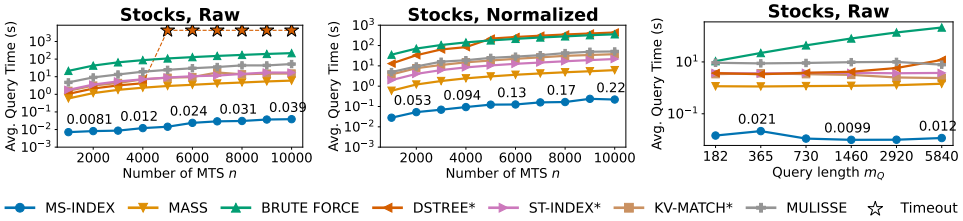


Figure 4.7: Scalability over dataset size on *Stocks*. (a) Query time for raw subsequences over number of time series n, and (b) normalized subsequences. (c) Query time for raw subsequences with varying query length. The y axes are presented in log scale.

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.

Size of the data structures for each algorithm Table 4.5 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 [152, 204].

Query time Figs. 4.7a-b 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 an efficient pruning stage

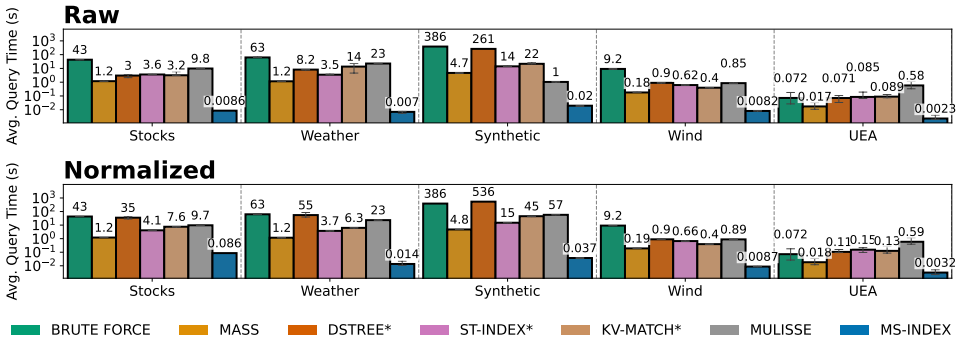


Figure 4.8: Query time of algorithms on different datasets. The y axes are presented in log scale.

prior to MASS, which in itself 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 pre-filters 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. 2), 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 design achieves the smallest index size among all methods, this comes at the cost of loose distance bounds that result in many exhaustive comparisons.

Query length Since query length must be specified before index construction, we evaluate its impact on performance. Figure 4.7c shows that all algorithms except Brute-force are invariant to query length, as they use MASS which scales with subsequence length ($O(|\boldsymbol{t}| \log |\boldsymbol{t}|)$ with $|\boldsymbol{t}| \geq |\boldsymbol{q}|$) rather than query length. This suggests MS-Index's performance is largely independent of $|\boldsymbol{q}|$, with only minor overhead from computing the query's DFT, allowing users to choose lengths that best fit their use case.

Different datasets Figure 4.8 presents the query execution performance across all datasets using default query parameters from Table 4.2. For the 30 UEA datasets, we present average results as individual dataset results showed similar patterns. MS-Index consistently outperforms all competitors with roughly two orders of magnitude advantage over MASS and three orders over other methods. The exception is the UEA datasets, where MS-Index outperforms MASS by only 5-7 times versus the 140-170x speedups on Stocks and Weather (raw subsequences). 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

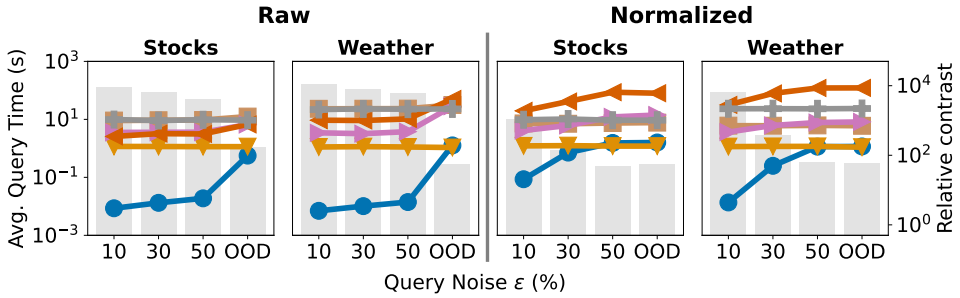


Figure 4.9: Query time of algorithms with varying levels of query noise with "HO" referring to a hold-out query. The y axes are presented in log scale.

in the Stocks or Weather datasets. This creates more challenging queries for index-based methods, further discussed in Section 4.5.2. 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 [180]. All in all, we can conclude that MS-Index is significantly faster than all competitors across various datasets with different characteristics and from different domains, confirming its robustness and general applicability.

Effect of query difficulty Pruning-based algorithms exploit the assumption that queries are selective, i.e., that only a small fraction of the dataset is similar to the query, also known as the "left-tail" assumption [18]. Their pruning power – and thus performance – depends on the contrast between $\boldsymbol{q}$ and the indexed data; queries that are similar to either all or none of the indexed entities will be more difficult to answer. This difficulty can be quantified by the *relative contrast* [7] of a query – the ratio between distances to the closest and farthest indexed entities, with lower values indicating more difficult queries. While the meaningfulness of queries with low relative contrast have been questioned [7, 44], they can be useful to show the limits of the algorithms. Following common practice [19, 259], we test algorithm sensitivity to query difficulty by adding varying noise levels and using out-of-distribution queries (OOD). OOD queries were generated by splitting each time series into a head (80%) and tail (20%), indexing only head subsequences, and sampling queries from tails. Figure 4.9 shows results for Stocks and Weather, with query noise ϵ representing the standard deviation of added noise as a percentage of the original subsequence's standard deviation. We observe that the query times of index-based algorithms like MS-Index are inversely proportional to query relative contrast (grey bars on secondary y-axis), consistent with theory and previous work [19, 259]. Furthermore, normalized subsequence queries have lower relative contrast than raw ones, confirming our hypothesis from the previous experiment that they are more challenging for index-based algorithms. For high-noise or OOD queries, MS-Index's pruning power weakens, and it performs similarly to MASS (with a small overhead due to the tree traversal). In these cases, sequential scan approaches are more appropriate as they don't rely on pruning. This suggests a future research direction: developing a "hybrid" approach that chooses between MS-Index and MASS based on estimated query difficulty.

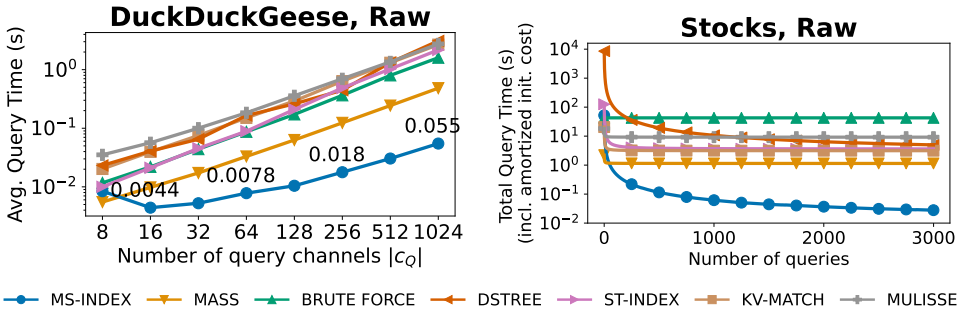


Figure 4.10: Impact of query on algorithms. (a) Query time over queried channels; (b) Total query time (including amortized initialization time) over queries. The y axes are presented in log scale.

Table 4.6: 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 $\|c_Q\|$		**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%

Number of query channels We investigate how the number of query channels influences performance using the *DuckDuckGeese* dataset from UEA, which contains 1024 channels. Figure 4.10a shows the query times for different methods on the DuckDuckGeese datasets, indexing all 1024 channels but querying on progressively larger subsets of channels (using uniform sampling to select which channels to query). The results show that MS-Index's query time scales sublinearly with the number of query 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 channel selectivity diminishes. Table 4.6 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 information.[7] In contrast, DSTree* and ST-Index* 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 [28, 165]. 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 4.6 confirm this: increasing with $|c_Q|$ for raw subsequences while slightly decreasing for normalized ones.

[7]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.

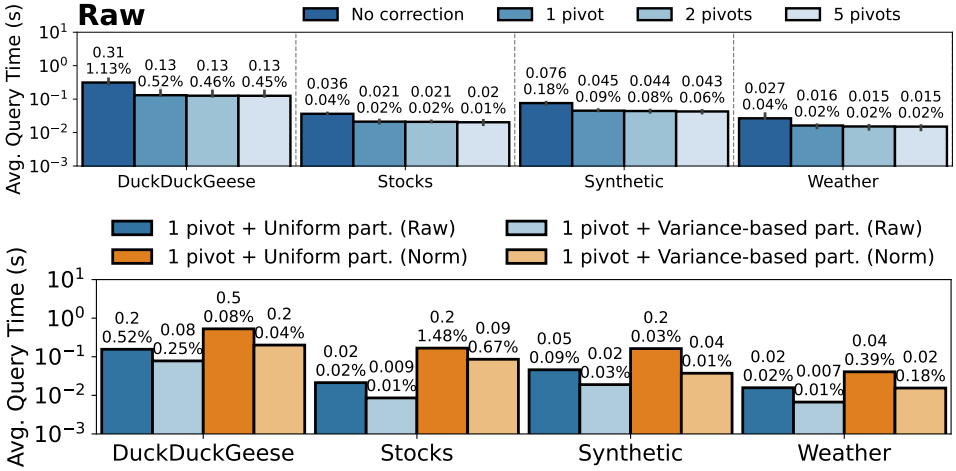


Figure 4.11: Impact analysis of the proposed optimizations: (a) Effect of the number of pivots in DFT-bound correction on the query time of MS-Index; (b) Effect of weighted partitioning on the query time of MS-Index. Note that each y-axis is in log scale.

Amortizing the initialization cost with the query execution cost The compared algorithms come with different initialization costs. Therefore, the best algorithm for each use case depends on the expected number of queries. For example, if we need to execute only a few queries on a large MTS dataset, algorithms that need to pre-process the dataset (such as MS-Index and ST-Index*) will be slower than MASS, which has a lower initialization cost. To compare the overall performance of all algorithms, we computed the amortized total cost of each algorithm for different numbers of queries (i.e., the initialization and total query execution cost, amortized over all queries). Fig. 4.10b plots the amortized cost, for up to 3000 queries. As expected, for very few queries, MASS outperforms all others. However, MS-Index becomes more efficient already after 45 queries. This difference becomes starker for more queries, indicating that the initialization time of MS-Index quickly pays off by enabling significantly faster query times compared to all competing algorithms.

Effect of DFT bound correction optimization Recall from Section 4.3.4 that we can tighten the distance bound using a triangle-inequality-based correction term, enabling more aggressive pruning at the cost of additional initialization overhead. We evaluate this correction's impact on both initialization and query times with varying number of pivots. Figure 4.11a shows that adding just a single pivot improves query performance by a factor of 2, while additional pivots yield 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 [22, 47]. 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 [129]. 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.

Effect of the optimization for tightening the MBRs We now evaluate the impact of the optimization for tightening the MBRs via weighted R-tree partitioning (Section 4.3.4). As a baseline we use MS-Index with the R-tree constructed with uniform partitioning across all channels. The results in Fig. 4.11b show that the weighted partitioning strategy generally improves the query time by a factor of 1.5-3. This indicates that the weighted partitioning strategy is effective in improving the pruning power of MS-Index, as it allows the R-tree to better adapt to the characteristics of the dataset. This is particularly important for datasets with a larger number of channels, such as DuckDuckGeese and Synthetic, where we can also see that the performance gap between the two partitioning strategies is larger. To summarize, from these results we conclude that the weighted partitioning strategy is beneficial.

4.6 Reflection and Conclusions

In this chapter, we addressed the efficiency challenge of similarity search on multivariate time series (MTS) through MS-Index, an algorithm for efficient subsequence search in multivariate time series. Building on our understanding of MTS distance measures from Chapter 3, MS-Index bridges the gap between theoretical understanding and practical implementation of MTS search.

Chapter summary. MS-Index represents a fundamentally different approach to multivariate time series search compared to existing solutions. Rather than treating each channel as an independent time series, MS-Index constructs a unified representation that captures the joint information across all channels. This is accomplished through a two-level architecture: first, applying DFT-based approximations to create feature representations of subsequences, and second, indexing these representations in a single R-tree structure that considers all channels simultaneously. By adaptively selecting DFT coefficients based on their distance contribution and employing weighted partitioning strategies, MS-Index achieves both high pruning power and efficient distance computation across multivariate data.

Our experimental evaluation across 33 diverse datasets demonstrated that MS-Index consistently outperforms existing solutions by up to two orders of magnitude. The algorithm shows robust performance across various dataset characteristics and query parameters, including different query lengths and varying numbers of channels. Notably, MS-Index's query time scales sublinearly with the number of query channels, indicating that it effectively leverages the additional information provided by multiple channels to improve pruning power. This contrasts sharply with baseline approaches whose performance degrades linearly as the number of channels increases.

Transferability of findings. The design principles uncovered through MS-Index are transferable to other similarity search contexts beyond subsequence search. First, combining the summarizations of all channels into a single index works better than having separate indices per channel. Second, when using such unified representations, it is important to apply different weights to different dimensions to account for the varying impact of channels on the

overall distance, as we showed with our weighted partitioning strategy. Third, for subsequence search specifically, it is better to perform aggregation over time-neighboring subsequences after indexing rather than before, which preserves important information that the index can use for effective pruning. Fourth, the technical improvements we developed for DFT-based bounds – like pivot-based correction – can improve any indexing system that uses DFT approximations, including those for UTS. Together, these principles provide a useful template for extending similarity search algorithms to handle multiple channels while maintaining speed and efficiency.

Limitations and opportunities for improvement. While MS-Index represents significant progress, important limitations remain. The current algorithm requires the query length to be known at index construction time, which constrains its flexibility in applications where variable-length queries are needed. Our evaluation also revealed that MS-Index's performance significantly degrades for normalized queries with low relative contrast, such as when querying with many channels or on highly similar subsequences. This limitation stems from the curse of dimensionality making distance-based pruning less effective as distances between points become more uniform - a fundamental challenge affecting all spatial indexes. This limitation could be addressed through hybrid approaches that adaptively switch between index-based search and sequential scanning based on the estimated query difficulty.

In conclusion, MS-Index shows how to effectively extend similarity search to handle multiple channels. Our results clearly show that treating all channels together works much better than handling them separately. The key lessons from our work - using unified data structures, combining information across channels, and adapting to different channel importance - help create a more complete approach to multivariate similarity search. This avoids the fragmented solutions we identified in Section 2.4.1.

Reflection on research question. This chapter directly advances our central research question by demonstrating that efficient multivariate similarity search is not only possible but can be achieved with acceptable performance. Compared to naive extensions, MS-Index proves that by properly accounting for the multivariate nature of the data in the index design itself, we can overcome the computational challenges that initially motivated this thesis. Together with our findings from Chapter 3, where we identified effective distance measures for MTS comparison, we have now established both the theoretical and practical foundations for multivariate similarity search: we can not only meaningfully compare multivariate time series but also do so efficiently at scale.

II

Multivariate Similarity Search on Univariate Data

CHAPTER 5

Efficient Detection of Multivariate Correlations

Real-world phenomena are rarely governed by simple pairwise interactions; more often, they arise from the complex interplay of multiple entities or processes. This chapter moves beyond traditional pairwise similarity to address the discovery of these high-order similarities, specifically high-order correlations. Tackling this problem is computationally daunting due to a combinatorial explosion in the search space. To overcome this, we introduce Correlation Detective (CD), an efficient algorithm that leverages novel bounding and pruning strategies to find meaningful multivariate relationships in univariate datasets. We then extend this approach to dynamic environments with CDStream, a variant designed for streaming data. Our evaluation shows these methods consistently outperform state-of-the-art approaches by orders of magnitude, making the discovery of complex, high-order patterns computationally tractable, opening new avenues for analysis in various scientific domains.

Note: Part of the work presented in this chapter is the result of a joint work with other authors, conducted as part of a previous degree [72, 170, 171] and is therefore not considered a novel contribution in this thesis. Still, it is included as it is the basis for the current work, and thus provides important context for the reader. We provide an overview of all contributions in this chapter in Appendix F, where each contribution is explicitly mapped to the relevant degree. In short, the novel PhD contributions include (a) significant algorithmic improvements to CD (a new top-k query algorithm with novel optimizations), (b) a new streaming model for CDStream that handles non-synchronized streams, and (c) a substantially expanded evaluation with more datasets, query types, and baselines. Collectively, these changes improved query performance by up to 77x on static data and enabled support for asynchronous and non-aligned streams in the case of CDStream.

The contents of this chapter have previously appeared in d'Hondt et al. [79] and Minartz et al. [171].

5.1 Introduction

Multivariate similarities, also known as high-order similarities, extend the concept of pairwise similarities to relationships among three or more variables. These variables may represent various forms of data, such as time series or other high-dimensional data stored as vectors.[1]

As introduced in Chapter 2, similarity search can be formulated in different ways. In contrast to the previous chapters where we focused on query-centric definitions (k-NN and r-range queries), in this chapter we focus on the "all-to-all" formulation (top-k and threshold queries as defined in Section 2.1). The combinatorial nature of multivariate similarity search – examining relationships between multiple variables simultaneously – favors this broader search perspective. Nevertheless, all methods presented in this chapter can be extended to nearest neighbors and r-range definitions by constraining the combinations to always include a specific query object.

Applications of multivariate similarities. In the last few years, search for high-order similarities has emerged across various scientific domains, albeit under different names and with domain-specific formulations. In neuroscience, researchers have identified "high-order interactions" in fMRI time series that revealed how different brain regions work in cohort for executing tasks [9, 10]. For instance, the activity of the left middle frontal region was found to have a high correlation with the combined activity of the right superior frontal and left inferior frontal regions during audiovisual stimulus processing. This insight suggests that the left middle frontal has an integrative role of assimilating information from the other two regions, which was not possible to find by looking only at pairwise correlations. In climate science, detection of higher-order relationships (termed "teleconnections") led to the characterization of new weather phenomena and improved climate models [150]. In machine learning, multivariate information-theoretic measures have increasingly served as learning objectives or regularizers for training of neural networks aimed at optimizing the correlation among multiple variables. Usage of such regularizers lead to improved robustness, generalizability, and interpretability of the models [11, 53, 56].

Existing algorithms and measures. To address the challenges in these diverse application domains, researchers have developed a variety of measures and algorithms for detecting multivariate relationships, including Tripoles [9], Multipoles [10], Canonical Correlation Analysis (CCA) [119], and Total Correlation (TC) [183, 184, 234, 251]. While these domain-specific approaches have yielded valuable insights, they fundamentally fail to address the critical bottleneck in discovering strong multivariate similarities: the exponentially growing search space comprising all possible time series combinations. The reason that this growing search space is difficult to avoid is that pairwise similarities alone do not suffice to identify all significant multivariate relationships in the data, as established in Chapter 2 and demonstrated through the example in Figure 2.1. From a computational perspective, this implies that apriori-like pruning techniques cannot be applied to the general case of multivariate similarities.

[1] Although all presented methods in this chapter are applicable to both time series and vectors, we will mostly refer to time series in this chapter, in line with the rest of the thesis.

```
SELECT v1.vid, v2.vid, v3.vid
FROM vectors v1, vectors v2, vectors v3
WHERE v1.vid < v2.vid AND v1.vid != v3.vid AND v2.vid != v3.vid
AND pearson(avg(v1.vec, v2.vec), v3.vec) > 0.8
```

Figure 5.1: Example SQL query for a multivariate threshold query (cf. Definition 8) using Pearson correlation and a threshold of 0.8.

Computational challenges. At the same time, using an exhaustive approach that evaluates all possible combinations leads to combinatorial complexity, making it impractical for even moderately sized datasets. To illustrate this complexity, consider the SQL query shown in Figure 5.1, which implements a basic multivariate threshold search according to Definition 8. The query requires three joins (cartesian products) just to generate all possible triplet combinations, resulting in cubic computational complexity. For perspective, with just 100 time series, finding all strong ternary relationships would involve examining over 1 million combinations, while identifying quaternary relationships among 1000 time series involves 1 trillion possibilities. Even generating and enumerating these combinations becomes prohibitively expensive. This underscores the need for intelligent algorithms that can substantially reduce the search space and computational burden.

Limitations of existing methods. To tackle this computational challenge, existing algorithms typically take one of three paths: (a) they use restricted definitions of multivariate similarities that make apriori-like filtering possible [10, 183, 251], (b) they rely on handcrafted assumptions of the user query, which may be too constraining for other application scenarios [9, 10, 251] or, (c) they provide approximate results that come with no guarantees of accuracy [9, 10]. While these algorithms work well for their intended uses, they lack the versatility needed for broader applications.

Beyond these computational limitations, a common concern when working with multivariate similarities is the risk of discovering spurious correlations – relationships that appear statistically significant but lack meaningful causal or practical importance [62]. While this is a valid concern, the current lack of efficient computational methods actually hinders our ability to address it effectively. Without systematic ways to discover and analyze multivariate relationships at scale, data scientists are often forced to rely on ad-hoc approaches or examine only a small subset of possible relationships, potentially missing important patterns while still being susceptible to spurious ones. A comprehensive solution should enable a data-driven approach that can systematically explore the space of multivariate relationships, allowing domain experts to apply their knowledge and additional validation techniques to distinguish meaningful patterns from spurious ones. However, this systematic exploration is currently impossible due to the computational limitations of existing approaches.

Our approach. To address these challenges, we take a more general approach in this chapter that aligns with the overall direction of this thesis. First, we also consider similarity measures that are not suitable for apriori-like pruning, expanding beyond the more constrained approaches in existing literature. Second, we abide by Ockham's razor: we prioritize discovery

of simpler multivariate similarities – those involving fewer time series – as these tend to be more meaningful and easier to interpret than complex relationships over many time series. This approach also helps to prevent overfitting, as relationships involving more variables are more likely to produce spurious similarities that do not generalize well, a phenomenon that was validated in the context of sparse representations in Chapter 7. Third, we develop multiple algorithm variants to suit different needs: an exact threshold algorithm that finds all similarities above a specified value τ, and an exact top-k algorithm that identifies the top-k highest similarities, as defined in Section 2.1.2. We also explore progressive result generation and extend our methods to handle streaming data scenarios.

We evaluate our algorithms on 7 datasets and compare them to the state-of-the-art. Our evaluation demonstrates that we outperform the existing methods, frequently by several orders of magnitude. Finally, we show that the progressive version of the algorithm produces around 90% of the answers in 10% of the time.

The remainder of the chapter is structured as follows. In the next section we formalize the problem, and discuss the preliminaries and related work. We then propose the algorithmic variants for the case of static data (Section 5.3), and the streaming extension of the algorithm (Section 5.4). Section 5.5 summarizes the experimental results. We conclude in Section 5.6.

5.2 Preliminaries

We start with a discussion of the multivariate similarity measures that will be considered in this work. We then formalize two additional constraints on the similarity search problem, and discuss prior work on algorithms for multivariate similarity search.

5.2.1 Similarity Measures

In this chapter we will be focusing on two multivariate similarity measures: the two-sided Multivariate Pearson Correlation (*mPC*) and the one-sided Multipole (*MP*). Note that as this part of the thesis considers similarities over UTS, we define a time series $\boldsymbol{v} \in \mathbb{R}^m$ as a 1-dimensional time series rather than the 2-dimensional representation (i.e., $\boldsymbol{v} \in \mathbb{R}^{c \times m}$) used in the previous chapters. We now define the two measures.

Definition 11 (Multivariate Pearson Correlation (*mPC*)). Given two sets of time series X and Y, the Multivariate Pearson Correlation $mPC(X, Y)$ is defined as:

$$mPC(X, Y) = \rho\left(\frac{\sum_{\boldsymbol{x} \in X} \hat{\boldsymbol{x}}}{|X|}, \frac{\sum_{\boldsymbol{y} \in Y} \hat{\boldsymbol{y}}}{|Y|}\right) \tag{5.1}$$

where ρ denotes the Pearson correlation coefficient and $\hat{\boldsymbol{x}}$ denotes $\boldsymbol{x}$ after z-normalization, i.e., $\hat{\boldsymbol{x}}_i = \frac{\boldsymbol{x}_i - \mu_{\boldsymbol{x}}}{\sigma_{\boldsymbol{x}}}$.

Note that both this definition and our proposed algorithms can be easily extended to weighted linear aggregates (with predefined weights), instead of averaging, as shown in Chapter 6.

Definition 12 (Multipole (MP)). The Multipole measure $MP(X)$ measures the linear dependence of an input set of time series X [10]. Specifically, let $\hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_n$ denote n z-normalized input time series, and $\boldsymbol{X} = [\hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_n]$ the matrix formed by concatenating the time series. Then:

$$MP(X) = 1 - \min_{||\boldsymbol{v}||_2=1} \text{var}(\boldsymbol{X} \cdot \boldsymbol{v}) \tag{5.2}$$

The value of $MP(X)$ lies between 0 and 1. The measure takes its maximum value when there exists perfect linear dependence, i.e., there exists a time series $\boldsymbol{v}$ with norm 1, such that $\text{var}(\boldsymbol{X} \cdot \boldsymbol{v}) = 0$, and its minimum value when the time series are orthogonal.

Notice that the Multipole measure is not equivalent to, nor a generalization of, the *mPC* measure. By definition, *MP* assumes optimal weights (time series $\boldsymbol{v}$ is such that the variance is minimized), whereas for *mPC*, the aggregation function for the time series (e.g., avg) is determined at the definition of the measure.

As both measures capture linear dependencies (i.e., correlations) between time series, we use the term **correlations** rather than the more general term of **similarities** to refer to them in the remainder of this chapter. Furthermore, for conciseness, we will use $Corr(p_l)$ and $Corr(p_l, p_r)$ to denote the combination of the correlation measure, and the user-chosen values of p_l and p_r. This is referred to as the *correlation pattern*. For example, *mPC*$(2, 1)$ will identify the combinations of sets of time series of *at most* size 2 and 1 with high Multivariate Pearson correlation (i.e., including *mPC*$(1, 1)$), whereas *MP*(4) will identify the combinations of *at most* 4 time series with high Multipole correlation (i.e., including *MP*(2), and *MP*(3)).

5.2.2 Additional Constraints

Complementary to top-k and threshold queries, defined in Chapter 2, users may also want to specify additional constraints, relating to the targeted diversity and significance of the answers. These constraints serve to filter the result set to include only the most interesting and informative patterns, avoiding redundancy and ensuring that each returned pattern provides substantial value. We consider two different constraints, but other constraints (e.g., the weak-correlated feature subset constraint of [251]) can also be integrated in the algorithm in a similar manner:

Definition 13 (Irreducibility constraint). For each (X, Y) in the result set, there exists no (X', Y') in the result set such that $X' \subseteq X$, $Y' \subseteq Y$, and $(X', Y') \neq (X, Y)$. Intuitively, if $Corr(X', Y') \geq \tau$, then no supersets of X' and Y' should be considered together. This constraint prioritizes simpler answers and prevents result redundancy by favoring minimal sets that exhibit high multivariate correlation.

Example: Consider a dataset containing stock market time series, where ($X = \{$AAPL, MSFT$\}$, $Y = \{$GOOG$\}$) has a *mPC* value of 0.85, and ($X' = \{$AAPL, MSFT, AMZN$\}$,

$Y' = \{\text{GOOG}\}$) has a *mPC* value of 0.87. With the irreducibility constraint applied over a threshold query with $\tau = 0.8$, only (X, Y) would be included in the result set, since it represents a simpler mode. The additional time series AMZN is considered redundant in this context.

Definition 14 (Minimum jump constraint). For each (X, Y) in the result set, there exists no (X', Y') such that $X' \subseteq X$, $Y' \subseteq Y$, $(X', Y') \neq (X, Y)$, and $Corr(X, Y) - Corr(X', Y') < \delta$. This constraint, which was first proposed in [9], discards solutions where a time series in $X \cup Y$ contributes less than δ to the increase of the *mPC* value, ensuring that each included time series provides significant additional explanatory power.

Example: Suppose we have sensor readings from a manufacturing process, where ($X = \{\text{Temperature, Pressure}\}$, $Y = \{\text{ProductQuality}\}$) has a *mPC* value of 0.75, and ($X' = \{\text{Temperature, Pressure, Humidity}\}$, $Y' = \{\text{ProductQuality}\}$) has a *mPC* value of 0.76. With the minimimum jump constraint with $\delta = 0.05$ applied over a threshold query with $\tau = 0.7$, then (X', Y') would be excluded from the result set. This is because adding the Humidity sensor increases the *mPC* value by only 0.01, which is less than the minimum required jump of 0.05. This indicates that Humidity provides minimal additional correlation beyond what Temperature and Pressure already provide, and thus does not contribute significantly to the multivariate relationship.

Note that for top-k queries, the irreducibility constraint is ill-defined. For example, consider two sets of time series (X, Y) and (X', Y'), where $X' \subset X$ and $Y' \subset Y$, and $Corr(X, Y) = 0.9$, and $Corr(X', Y') = 0.8$. In this case, the definition of top-k does not dictate which of (X, Y) or (X', Y') should be in the answer set. We therefore refrain from considering irreducibility as an option for top-k queries.

5.2.3 Related Work

As discussed in Chapter 1 and Chapter 4, numerous algorithms have been developed for efficiently searching for high pairwise similarities in datasets, leveraging techniques such as dimensionality reduction, indexing structures, and optimized distance computations. Two notable contributions focusing on mining high pairwise correlations are StatStream [257] and the algorithm of Mueen et al. [177]. These methods map pairwise correlations to Euclidean distances and then utilize Discrete Fourier Transforms, grid-based indexing, and dynamic programming to effectively reduce the search space. However, these approaches do not extend to multivariate correlations, as they cannot capture cases where two time series with low individual correlations to a third time series may produce a highly correlated aggregate (as illustrated in Fig. 2.1).

Existing work addressing multivariate similarities propose algorithms that rely on additional constraints to achieve effective pruning. For instance, Agrawal et al. investigated the problem of finding highly-correlated tripoles [9]. The Tripole measure represents a specific case of the *mPC* measure where $|X| = 2$ and $|Y| = 1$ (i.e., *mPC*$(2, 1)$). Their algorithm, CoMEt, relies on the minimum jump constraint for pruning efficiency. In contrast to CoMEt, our

	Completeness	Require Constraints	Correlation Measures	Query Types	Data formats
[9]	Yes	Yes	$mPC(1,2)$	Threshold	Static
[10]	No	Yes	$MP(\cdot)$	Threshold	Static
[183]	No	No	$TC(\cdot)$	Threshold	Static
[251]	Yes	Yes	$TC(\cdot)$ (binary data)	Threshold	Static
Ours	Yes	No	$mPC(\cdot,\cdot)$, $MP(\cdot)$	Threshold, Top-k,Progressive	Static, Streaming

Table 5.1: Comparison to the most relevant related work for multivariate similarity search.

algorithm handles the generalized definition of Multivariate Pearson correlation over aggregated time series, supporting multiple time series on both the left- and right-hand side. Furthermore, our approach leverages novel theoretical results for search space pruning and demonstrates scalability to larger datasets without requiring additional constraints such as minimum jump or irreducibility.

For the Multipole measure (Eqn. 5.2), algorithms were first introduced in [10] with the CoMEtExtended algorithm. Both CoMEt and CoMEtExtended provide approximate solutions and employ clique enumeration to navigate the search space efficiently. Their performance hinges on a parameter ρ that balances result completeness against computational efficiency. The minimum jump constraint becomes essential for reducing computational complexity. While these algorithms produce more comprehensive results compared to techniques like l_1-regularization and structure learning approaches at reasonable computation times, they lack completeness and accuracy guarantees. In contrast, our proposed algorithm delivers exact results – guaranteeing complete answer retrieval – while outperforming both existing algorithms.

With respect to other measures, Total Correlation (TC) is a non-linear information-theoretic metric quantifying shared information across variables [234]. Nguyen et al. [183] introduced a closely related similarity measure and an algorithm for identifying column sets with high TC to minimize data redundancy in databases. Their method first identifies pairs with high Mutual-Information – the pairwise equivalent of TC – and uses these to establish a lower bound on a group's TC. The algorithm then identifies quasi-cliques where most pairwise similarities are high, resulting in high TC values. However, this approach fails to detect strongly related groups with low pairwise similarities, which often represent the most interesting patterns. Similarly, Zhang et al. [251] developed an algorithm for discovering binary time series sets with high TC. This method is approximate, limited to binary data, and relies on a restrictive weak-correlated subset constraint.

In supervised learning contexts, subset regression bears similarities to multivariate similarity search. This feature selection challenge aims to identify the optimal p predictors from n candidate features [69]. However, our multivariate similarity search definition offers greater versatility by not requiring a predefined dependent variable. Additionally, rather than identifying only the time series set with maximum similarity, our approach (per Definitions 7 and 8) combined with the additional constraints enables discovery of diverse result sets.

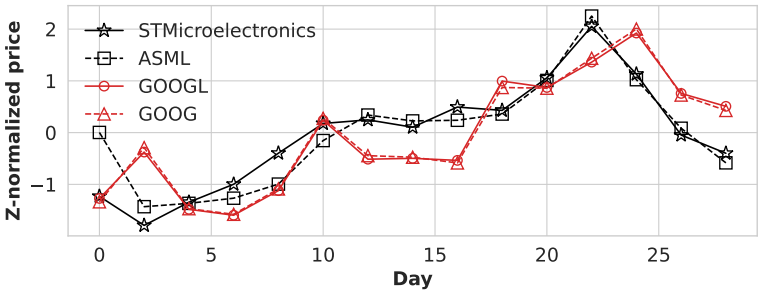


Figure 5.2: Two groups of closely related stocks: ASML and STMicroelectronics are exposed to similar risks, while GOOG and GOOGL participate in the same conglomeration

This facilitates domain experts' qualitative assessment of results, yielding more meaningful insights.

5.3 Detection of Multivariate Correlations in Static Data

The main challenge in detecting high multivariate correlations stems from the combinatorial explosion of the number of candidates that need to be examined. In a dataset of n time series, there exist $O\left(\sum_{p=2}^{p_l+p_r}\binom{n}{p}\right)$ possible combinations for a correlation pattern $Corr(p_l, p_r)$.[2] Even if each combination can be checked in constant time, the enumeration of all combinations still requires significant computational effort.

Our algorithm – Correlation Detective (CD) – exploits the insight that time series often exhibit (weak) correlations between each other. For example, securities of companies that participate in the same conglomeration (e.g., Fig. 5.2, GOOGL and GOOG) or are exposed to similar risks and opportunities (e.g., STMicroelectronics and ASML) typically exhibit a high correlation between their stock prices. CD exploits such correlations, even if they are weak, to drastically reduce the search space.

CD works as follows: rather than iterating over all possible time series combinations that correspond to the correlation pattern, CD clusters the time series based on their correlation, and enumerates the combinations of only the cluster centroids. For each of these combinations, CD computes upper and lower bounds on the correlations of all time series combinations in the Cartesian product of the clusters. Based on these bounds, CD decides whether or not the combination of clusters (i.e., all combinations of time series derived from these clusters) should be added to the result set, can safely be discarded, or, finally, if the clusters should be split into smaller subclusters for deriving tighter bounds. This approach effectively reduces the number of combinations that need to be considered, making CD at least an order of magnitude faster than existing methods.

[2]The same holds for one-sided patterns, e.g., the complexity of MP $(p_{\max})$ is $O\left(\sum_{p=1}^{p_{\max}}\binom{n}{p}\right)$.

In the remainder of this section, we will present the key elements of CD, explaining how the two types of queries presented in Section 2.1 are handled. We will start with a brief description of the initialization phase, which includes data preprocessing and clustering. In Sections 5.3.2 and 5.3.5 we will describe how CD answers threshold and top-k queries respectively.

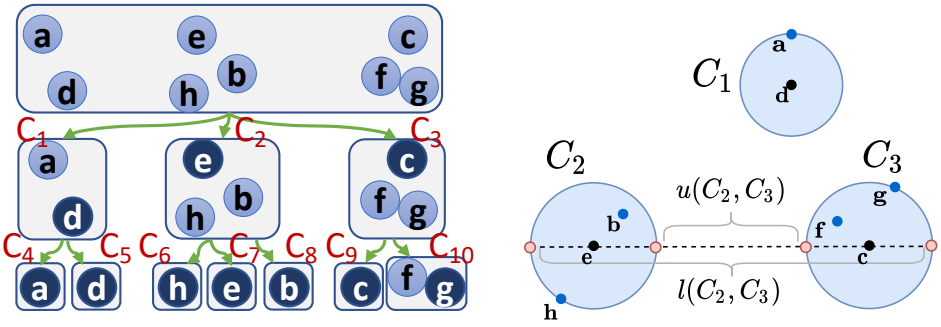
5.3.1 Initialization and Clustering

First, as both *mPC* and *MP* implicitly normalize the time series, we pre-process the dataset by z-normalizing all time series, i.e., shifting and scaling them such that they have zero mean and a standard deviation equal to 1. From here on, the algorithm operates only on these z-normalized time series. When clear from the context, we will denote time series as $\boldsymbol{x}$ instead of $\hat{\boldsymbol{x}}$.

Next, we construct a hierarchical clustering of all time series. The clustering algorithm operates in top-down fashion. A root cluster containing all time series is first created to initialize the hierarchy. The algorithm then consists of three steps. First, κ time series are picked from the root cluster and used as the initial top-level centroids in the hierarchy. These time series are picked using the seeding strategy of κ-means^{++} [16].[3] The use of κ-means^{++} (as opposed to sampling κ random time series) ensures that these initial centroids are well-distributed over the metric space, and not very close to each other. In the second step, we run the standard κ-means algorithm for at most r_1 iterations, or until convergence using the average function to recompute the cluster centroids after each iteration. The clustering is evaluated using the Within-Cluster Sum of Squares (WCSS) (the sum of the variances within all clusters). In the third step, steps one and two are repeated r_2 times (i.e., with different centroids), and the clustering with the lowest WCSS is kept as the final clustering assignment for the first level of the hierarchy. These three steps are executed recursively on each individual cluster with non-zero radius, to construct the second, third, etc. levels of the hierarchy, until all leaf nodes contain only one time series.

There is a clear tradeoff between the cost of the clustering algorithm and the clustering quality. Increasing the values of r_1 and r_2 will generally result in a higher clustering quality (lower WCSS), but will take longer to compute. However, the quality of the clustering does *not* affect the correctness of CD – in fact, regardless of the employed hierarchical clustering algorithm, CD always returns the same correct result set. A poor clustering only affects the computational efficiency of CD. Still, our experiments show that as long as the clustering is reasonable, a suboptimal clustering is not detrimental to CD's efficiency. More precisely, we found that the value of r_1 (max. iterations of κ-means, after the initial centroids were decided) had no observable effect on CD's efficiency. Therefore, we simply set $r_1 = 1$. The same generally holds for r_2, although to prevent ruinous effects due to coincidentally very poorly chosen initial centroids, we set $r_2 = 50$. Still, the clustering takes at most a few seconds in our experiments, which is negligible compared to the total execution time of the algorithm.

[3] Note that we use κ here rather than the commonly used k to avoid confusion with the k in top-k.



(a) Running example in 2 dimensions: the centroids of each cluster are depicted with darker background. All clusters are labeled for easy reference

(b) Illustration of pessimistic pairwise bounds of Lemma 5.3.1

Figure 5.3: Example of a hierarchical clustering and the bounds used in CD.

5.3.2 Threshold Queries

We now describe how CD answers threshold queries, i.e., queries of the form Corr$(p_l, p_r) \geq \tau$, where τ is a user-defined threshold. Note that we will be using the two-sided *mPC* measure as an example, but the same principles apply to one-sided patterns and the *MP* measure; the only difference is in the construction of the root combinations, as we will explain later.

CD receives as input (a) the z-normalized time series, (b) the cluster tree produced by the hierarchical clustering algorithm, (c) a correlation pattern Corr(p_l, p_r), and (d) a correlation threshold τ. The algorithm then proceeds to construct cluster combinations of increasing cardinalities, consisting of only root clusters C_{root}, referred to as *root combinations*, until the maximum cardinality as specified by the correlation pattern is reached. For example, if the correlation pattern is $mPC(1, 3)$, CD will consider the root combinations (C_{root}, C_{root}), $(C_{root}, C_{root}, C_{root})$ and $(C_{root}, C_{root}, C_{root}, C_{root})$ in order.

A combination of clusters compactly represents the combinations created by the Cartesian product of the time series inside the clusters. To illustrate this, we introduce the following running example with three clusters C_1, C_2, and C_3 (Figure 5.3b) with 2, 3, and 3 time series respectively. In the example, the cluster combination (C_1, C_2) represents a set of 6 time series combinations, which we will refer to as its *materializations*. For each root combination, CD computes the lower and upper bounds on the correlation of all its materializations, denoted as LB and UB respectively (line 1 of Algorithm 3). These bounds guarantee that any possible materialization of the cluster combination, i.e., replacing each cluster with any one of the time series in that cluster, will always have a correlation between LB and UB. We will present how these bounds are computed in the following sections.

The next step is to compare the bounds with the user-chosen threshold τ (lines 2, 4, 6). If $UB < \tau$, the combination is *decisive negative* – the upper bound is lower than the threshold, meaning that no materialization of this cluster combination can yield a correlation higher than τ. Therefore, this cluster combination does not need to be examined further. If $LB \geq \tau$, the combination is *decisive positive*, guaranteeing that all possible materializations

Algorithm 3: ThresholdQuery(S_l, S_r, $Corr$, τ)
Input: Sets of clusters S_l and S_r that adhere to the user-defined correlation pattern including a correlation measure $Corr$, correlation threshold τ.

```
1  (LB, UB) ← CalcBounds(S_l, S_r, Corr)
2  if LB ≥ τ then
3  |  Add (S_l, S_r) to the result set
4  else if UB < τ then
5  |  Discard (S_l, S_r)
6  else
   |  /* Replace largest cluster with subclusters and recurse */
7  |  C_max ← arg max_{C ∈ S_l ∪ S_r} {C.radius}
8  |  Set SC ← C_max.subclusters
9  |  for S ∈ SC do
10 |  |  (S'_l, S'_r) ← (S_l, S_r) with C_max replaced by S
11 |  |  ThresholdQuery((S'_l, S'_r), Corr, τ)
```

of this cluster combination will have a correlation of at least τ. Therefore, all materializations are inserted in the result. Finally, when $LB < \tau$ and $UB \geq \tau$, the combination is *indecisive*. In this case, the algorithm (lines 7-11) chooses the cluster C_{max} with the largest radius,[4] and recursively checks all combinations where C_{max} is replaced by one of its sub-clusters. In our running example in Figure 5.3a, assume that the algorithm examined an indecisive combination of clusters C_1, C_2, C_3, and C_2 is the cluster with the largest radius. The algorithm will drill down to consider the three children of C_2, and examine their combinations with C_1 and C_3. The recursion continues until each combination is decisive.

We will refer to this process as *traversing the comparison tree*. Decisive combinations are typically found at high levels of the cluster tree, thereby saving many comparisons. In the following, we will discuss two different approaches for deriving LB and UB for arbitrary correlation patterns. The first approach (theoretical bounds) has constant complexity in the number of materializations a cluster combination covers. The second approach (empirical bounds) extends the theoretical bounds with additional information. It has a slightly higher cost, but typically leads to much tighter bounds.

5.3.3 Theoretical Bounds

We first present a lemma for bounding the Pearson correlation between only two clusters, which serves as a stepping stone for bounding *mPC* and *MP* correlations.

Lemma 5.3.1. Let $\rho(\boldsymbol{x}, \boldsymbol{y})$ denote the Pearson correlation between two z-normalized vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, and $\theta_{\boldsymbol{x},\boldsymbol{y}}$ the angle formed by these vectors. Consider four z-normalized vectors $\boldsymbol{u_1}$, $\boldsymbol{u_2}$, $\boldsymbol{v_1}$, and $\boldsymbol{v_2}$, such that $\theta_{\boldsymbol{v_1},\boldsymbol{u_1}} \leq \theta_1$ and $\theta_{\boldsymbol{v_2},\boldsymbol{u_2}} \leq \theta_2$. Then, correlation

[4] Radii are defined as the Euclidean distance between the centroid and the furthest point in the cluster.

$\rho(\boldsymbol{u_1}, \boldsymbol{u_2})$ can be bounded as follows:

$$\cos(\theta^{\max}_{\boldsymbol{v_1},\boldsymbol{v_2}}) \leq \rho(\boldsymbol{u_1}, \boldsymbol{u_2}) \leq \cos(\theta^{\min}_{\boldsymbol{v_1},\boldsymbol{v_2}})$$

where $\theta^{\min}_{\boldsymbol{v_1},\boldsymbol{v_2}} = \max\left(0, \theta_{\boldsymbol{v_1},\boldsymbol{v_2}} - \theta_1 - \theta_2\right)$, $\theta^{\max}_{\boldsymbol{v_1},\boldsymbol{v_2}} = \min\left(\pi, \theta_{\boldsymbol{v_1},\boldsymbol{v_2}} + \theta_1 + \theta_2\right)$.

Proof. All proofs for this chapter can be found in Appendix B.

Lemma 5.3.1 bounds the correlation between two time series $\boldsymbol{u_1}$ and $\boldsymbol{u_2}$ that belong to two clusters with centroids $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$ respectively, by using: (a) the angle between the two centroids, and, (b) upper bounds on the angles between $\boldsymbol{u_1}$ and $\boldsymbol{v_1}$, and between $\boldsymbol{u_2}$ and $\boldsymbol{v_2}$. For instance, in the running example (Fig. 5.3a), we can bound the correlation between $\boldsymbol{a}$ and $\boldsymbol{b}$ if we have the cosine of the two cluster centroids $\boldsymbol{d}$ and $\boldsymbol{e}$, the cosines of $\boldsymbol{a}$ with $\boldsymbol{d}$, and $\boldsymbol{h}$ with $\boldsymbol{e}$ (as $\boldsymbol{h}$ is the furthest point in C_2 from the centroid $\boldsymbol{e}$). The bounds are tightened if the maximum angle formed by each centroid with its corresponding cluster time series is reduced. We now extend our discussion to cover multivariate correlations, which involve three or more clusters. We first derive bounds for *mPC* (Theorem 5.3.2), and then for *MP* (Theorem 5.3.3).

Theorem 5.3.2 (Bounds for *mPC*). For any pair of clusters C_i, C_j, let $l(C_i, C_j)$ and $u(C_i, C_j)$ denote lower/upper bounds on the pairwise correlations ρ between the cluster pair's materializations, i.e., $l(C_i, C_j) \leq \min_{\boldsymbol{x}\in C_i, \boldsymbol{y}\in C_j} \rho(\boldsymbol{x}, \boldsymbol{y})$ and $u(C_i, C_j) \geq \max_{\boldsymbol{x}\in C_i, \boldsymbol{y}\in C_j} \rho(\boldsymbol{x}, \boldsymbol{y})$. Consider sets of clusters $S_l = \{C_i^l\}_{i=1}^{p_l}$ and $S_r = \{C_j^r\}_{j=1}^{p_r}$. Let $L(S_1, S_2) = \sum_{C_i\in S_1, C_j\in S_2} l(C_i, C_j)$, and $U(S_1, S_2) = \sum_{C_i\in S_1, C_j\in S_2} u(C_i, C_j)$. Then, for any two sets of z-normalized time series $X = \{\hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_{p_l}\}$, $Y = \{\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_{p_r}\}$ such that $\hat{\boldsymbol{x}}_i \in C_i^l$, $\hat{\boldsymbol{y}}_i \in C_i^r$, multivariate correlation *mPC*(X, Y), can be bounded as follows:

1. if $L(S_l, S_r) \geq 0$:

$$\frac{L(S_l, S_r)}{\sqrt{U(S_l, S_l)}\sqrt{U(S_r, S_r)}} \leq \textit{mPC}(X, Y) \leq \frac{U(S_l, S_r)}{\sqrt{L(S_l, S_l)}\sqrt{L(S_r, S_r)}}$$

2. if $U(S_l, S_r) \leq 0$:

$$\frac{L(S_l, S_r)}{\sqrt{L(S_l, S_l)}\sqrt{L(S_r, S_r)}} \leq \textit{mPC}(X, Y) \leq \frac{U(S_l, S_r)}{\sqrt{U(S_l, S_l)}\sqrt{U(S_r, S_r)}}$$

3. else:

$$\frac{L(S_l, S_r)}{\sqrt{L(S_l, S_l)}\sqrt{L(S_r, S_r)}} \leq \textit{mPC}(X, Y) \leq \frac{U(S_l, S_r)}{\sqrt{L(S_l, S_l)}\sqrt{L(S_r, S_r)}}$$

Combined with Lemma 5.3.1, Theorem 5.3.2 enables bounding the Multivariate Pearson correlation of any cluster combination that satisfies the correlation pattern, without testing all

its possible materializations. For example, for combination $((C_1, C_2), C_3)$ from our running example, we first use Lemma 5.3.1 to calculate bounds for all cluster pairs in $O(1)$ per pair, which leads to values for $L(\cdot,\cdot)$ and $U(\cdot,\cdot)$. The bounds on *mPC*$((C_1, C_2), C_3)$ then follow directly from Theorem 5.3.2.

Also, observe that by tightening the bounds for the pairwise correlations in Lemma 5.3.1, we can reduce $L(\cdot,\cdot)$ and $U(\cdot,\cdot)$, which will in turn tighten the bounds for *mPC*. This is further exploited in Section 5.3.4.

Theorem 5.3.3 (Bounds for *MP*). For any pair of clusters C_i, C_j, let $l(C_i, C_j)$ and $u(C_i, C_j)$ denote lower / upper bounds on the pairwise correlations between the cluster's materializations, i.e., $l(C_i, C_j) \leq \min_{\boldsymbol{x}\in C_i, \boldsymbol{y}\in C_j} \rho(\boldsymbol{x}, \boldsymbol{y})$ and $u(C_i, C_j) \geq \max_{\boldsymbol{x}\in C_i, \boldsymbol{y}\in C_j} \rho(\boldsymbol{x}, \boldsymbol{y})$. Consider the set of clusters $S = \{C_i\}_{i=1}^{p}$. Furthermore, let $\mathbf{L}$ and $\mathbf{U}$ be symmetric matrices such that $\boldsymbol{L}_{ij} = l(C_i, C_j)$ and $\boldsymbol{U}_{ij} = u(C_i, C_j)\ \forall 1 \leq i, j \leq p$. For any set of **z-normalized** time series $X = \{\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2, \ldots, \hat{\boldsymbol{x}}_p\}$ such that $\hat{\boldsymbol{x}}_i \in C_i$, multipole correlation $MP(X)$ can be bounded as follows:

$$MP(X) \in 1 - \lambda_{\min}\left(\frac{\boldsymbol{L}+\boldsymbol{U}}{2}\right) \pm \frac{1}{2}||\boldsymbol{U}-\boldsymbol{L}||_2 \tag{5.3}$$

where $\lambda_{\min}\left(\frac{\boldsymbol{L}+\boldsymbol{U}}{2}\right)$ is the smallest eigenvalue of matrix $\left(\frac{\boldsymbol{L}+\boldsymbol{U}}{2}\right)$.

Similar to Theorem 5.3.2 for *mPC*, we can use Lemma 5.3.1 to compute the bounds on the pairwise correlations between any pair of clusters, which allows us to compute the bounds of Theorem 5.3.3, and to analyze the *MP* values of all materializations of the cluster combination in one go. Proofs for both theorems can be found in Appendix B.

5.3.4 Empirical Pairwise Bounds

The bounds of Lemma 5.3.1 – which are used for deriving the bounds of Theorems 5.3.2, 5.3.3 – tend to be pessimistic, as they always account for the worst theoretical case. Namely, in the example of Fig. 5.3b, the theoretical lower bound (resp. upper bound) accounts for the case that hypothetical time series (depicted in pink) are located on the clusters' edges, resulting in the smallest (resp. largest) possible distance between any pair of points in the clusters.

Tightening the bounds on pairwise correlations will in turn tighten the bounds on *mPC* and *MP*, which will lead to more aggressive pruning power of the algorithm described earlier in this section. The *empirical bounds* approach builds on the observation that the pairwise correlations of any pair of time series $\boldsymbol{x_i}, \boldsymbol{x_j}$ drawn from a pair of clusters C_i, C_j respectively is typically strongly concentrated around $(l(C_i, C_j) + u(C_i, C_j))/2$, especially for high-dimensional vectors such as time series. The approach works as follows. At initialization, we compute all pairwise correlations and store these in an upper-triangular matrix. Then,

during execution of Alg. 3, we lazily compute $l(C_i, C_j)$ and $u(C_i, C_j)$, as follows:

$$l(C_i, C_j) = \min_{\boldsymbol{x} \in C_i, \boldsymbol{y} \in C_j} \rho(\boldsymbol{x}, \boldsymbol{y}), \qquad u(C_i, C_j) = \max_{\boldsymbol{x} \in C_i, \boldsymbol{y} \in C_j} \rho(\boldsymbol{x}, \boldsymbol{y})$$

with $\rho(\boldsymbol{x}, \boldsymbol{y})$ retrieved from the upper-triangular matrix. The computed $l(C_i, C_j)$ and $u(C_i, C_j)$ are also cached and reused whenever (C_i, C_j) is encountered in another cluster combination.

It is important to note that the empirical bounds do not induce errors, since they trivially satisfy the requirements of Theorems 5.3.2 and 5.3.3 that $l(C_i, C_j) \leq \min_{\boldsymbol{x} \in C_i, \boldsymbol{y} \in C_j} \rho(\boldsymbol{x}, \boldsymbol{y})$ and $u(C_i, C_j) \geq \max_{\boldsymbol{x} \in C_i, \boldsymbol{y} \in C_j} \rho(\boldsymbol{x}, \boldsymbol{y})$. Therefore, bounds on *mPC* and *MP* derived using empirical bounds are still correct. Finally, they are at least as tight as the bounds of Lemma 5.3.1, since they account only the vectors that are actually present in the clusters and not the hypothetical worst case.

There is a clear tradeoff between the cost of computing the empirical pairwise bounds (worst case, quadratic to the number of time series), and the performance improvement of CD from the tighter bounds. Indicatively, in our experiments, the theoretical pairwise bounds computed from Lemma 5.3.1 were typically between two to eight times wider compared to the empirical pairwise bounds. Exploiting the tighter empirical bounds led to a reduction of the width of the bounds of Theorem 5.3.2 by 50% to 90% (for $mPC(1, 2)$), which empowered CD to reach to decisive combinations faster. As a result, total execution time of the algorithm with empirical bounds was typically an order of magnitude less than the time with the theoretical bounds. Therefore, all reported results will be using the empirical bounds. Lastly, note that the empirically-bounded versions of Theorem 5.3.2 and 5.3.3 do not require z-normalization. Still, it is performed in both cases to optimize pairwise cache computation and to ensure that $MP \in [0, 1]$, as suggested in [10]. However, z-normalization does not impact relative distances and therefore the top-k query answers are identical.

Handling of additional constraints

CD supports both the irreducibility and minimum jump constraints, as described in Section 5.2. For irreducibility, the process of identifying whether a simpler combination exists requires testing whether a combination of any of the subsets of S_l and S_r is already contained in the answers. To avoid the cost of enumerating all $O(2^{|S_l|+|S_r|})$ subsets during the execution of Alg. 3, only the pairwise correlations between any two clusters $C_l \in S_l$ and $C_r \in S_r$ are examined. Precisely, we use $l(C_l, C_r)$, which is already computed for Theorems 5.3.2 and 5.3.3. If there exist C_l, C_r s.t. $l(C_l, C_r) \geq \tau$, then any solution that can be derived by further examining the combination (S_l, S_r) cannot satisfy the irreducibility constraint. Therefore, (S_l, S_r) can be discarded. The case of minimum jump is analogous: if any $l(C_l, C_r) \geq UB - \delta$, where UB is calculated as in line 1 of Alg. 3, then the combination is discarded.

Only considering the pairwise correlations during the pruning process may lead to inclusion of answers that do not satisfy the constraints. Such combinations are filtered from the query

result before returning it to the user. Since the number of answers is typically in the order of a few tens to thousands, this final pass takes negligible time.

MP has the additional property that the correlation can only increase when adding an extra variable (i.e., $TC(X \cup \{y\}) \geq TC(X)$). We refer to this property as the *monotonicity over increasing pattern length*. This reduces the relevance of *MP* threshold queries without any constraints, as for any $MP(X) \geq \tau$ with $X \subset V$, all supersets of X will be in the result set, making it more cluttered. Therefore, we disallow such queries for *MP*, defaulting to the addition of the irreducibility constraint.

5.3.5 Top-k Queries

When exploring new datasets, it may be difficult to decide on a threshold τ. Setting the threshold too high for the dataset may lead to no answers, whereas a very low τ can result in millions of answers, and a decrease in performance. Top-k queries address this issue by allowing users to set the desired number of results, instead of τ. The answer then includes the k combinations of time series with the highest correlation that satisfy the correlation pattern.

If we had access to an oracle that could predict the value of τ that would yield exactly k results, we could transform top-k queries to threshold queries, and answer them with the standard CD algorithm for threshold queries (Algorithm 3) with the predicted τ. Since such an oracle is impossible, many top-k algorithms (e.g., Fagin's threshold algorithm [87]) start with a low estimate for τ, and progressively increase it, by observing the intermediate answers. The top-k variant of CD (see Alg. 4) follows the same idea; it starts with a low estimate of τ, and progressively increases it while examining cluster combinations (i.e., traversing the comparison tree) and maintaining an intermediate top-k result set R_k. Specifically, it initializes τ to the smallest possible correlation value (i.e., -1 for *mPC* and 0 for *MP*), and starts the recursion with the root clusters through Algorithm 3 as usual. However, now every time a decisive positive combination is added to the running result set R_k, the algorithm sorts R_k, limits it to the k highest correlations, and updates τ to the lowest correlation in the new R_k. Once the algorithm terminates, R_k will be guaranteed to contain the k highest correlations that satisfy the correlation pattern, and the correct result is returned.

Optimizing Top-k Queries

Notice that, as the running threshold τ increases during the execution of top-k queries, the pruning power of the algorithm also increases, as candidates have higher chances of being decisive negative. Therefore, to maximize the pruning power, it is beneficial to (a) initialize τ to the highest possible value without missing any answers, and (b) increase τ as quickly as possible. We now describe three techniques that CD employs to achieve this. Note that none of these techniques impact the completeness of the results, i.e., they do not miss any answers that would be returned by the standard top-k algorithm described above – they only improve the efficiency of the algorithm.

Technique 1: Initialize τ from pairwise correlations The first technique is a simple optimization that allows the algorithm to initialize τ to the highest possible value without missing any answers. Particularly, τ is initialized to the k'th highest pairwise correlation, which can be derived at minimal additional cost as the pairwise correlations are already computed for the empirical bounds before the algorithm starts the recursion.

Technique 2: Exploiting (soft) monotonicity The second technique aims to cheaply increase the threshold τ between different traversals of the comparison tree (i.e., before running Alg. 3 with a larger root combination). The technique is inspired by the property of monotonicity of *MP*, which implies that multivariate correlations can only increase when adding an additional variable (i.e., time series) to the set (i.e., correlation pattern). Thereby, given a set of combinations of size s, R, one can guarantee that any combination of size $s+1$ that is a superset of a combination in R will have a correlation greater than the lowest correlation in R. In other words, any superset of a combination in the top-k on *MP*(s) will have a higher multipole value than the lowest multipole value in that top-k set, and will thus be guaranteed to increase the running threshold τ when considered.

This observation is exploited as follows. Once the algorithm finishes traversing the comparison tree for combinations of size s, it does not immediately start traversing the tree for combinations of size $s+1$. Instead, it first iterates over all combinations $\{C_1, \ldots, C_s\}$ in the running top-k set R_k, adds a root cluster to that combination (i.e., $\{C_1, \ldots, C_s\} \cup C_{root}$), and initializes a recursion with Alg. 3 to consider all supersets of that combination of size $s+1$. This will increase the running threshold such that the next traversal of the comparison tree will start with a higher threshold, and thus prune more combinations.

While this technique is inspired by the monotonicity of *MP*, it is also applied to *mPC*; even when higher correlations for supersets are not guaranteed, supersets of highly similar combinations are still likely candidates for the top-k set. Note that using this technique does not jeopardize the exactness of the algorithm for either *MP* or *mPC*; it only impacts the degree at which the running threshold is increased during the query execution. At any time, this running threshold will be a lower bound on the threshold that yields the exact top-k results, thereby ensuring that no answers are missed.

Technique 3: Prioritization of candidates The last technique is an optimistic refinement of the upper bound, aiming to prioritize the combinations with the highest correlations, and thus the highest chances of increasing the running threshold. This technique splits the traversal of the comparison tree into two phases (see Alg. 4 for pseudocode). In the first phase (Alg. 4 lines 1-11), similar to Alg. 3, the algorithm traverses the comparison tree in a Breadth-First manner (BFS), and computes the upper and lower bound per cluster combination. However, instead of comparing the *exact* bounds to τ, it now artificially tightens the bounds by decreasing the value of the upper bound as follows;

$$UB_{\text{shrunk}} = (1-\gamma)\frac{UB+LB}{2} + \gamma UB$$

where $\gamma \in [-1, 1]$ is a parameter named the *shrink factor* with a default value of 0. Now, decisiveness of cluster combinations is determined based on (LB, UB_{shrunk}) analogous to

Algorithm 4: TOP-k-QUERY(S_l, S_r, $Corr$, τ, k, γ)
Input: Sets of clusters S_l and S_r that adhere to the user-defined correlation pattern. correlation measure $Corr$, starting threshold τ, desired output set size k, shrinkfactor γ.

1 $(LB, UB_{shrunk}) \leftarrow$ CALCBOUNDS($S_l, S_r, Corr, \gamma$)
2 $B \leftarrow$ new priority queue
3 **if** $LB \geq \tau$ **then**
4 | Add the contents of (S_l, S_r) to the result set R
5 | $R \leftarrow$ SORT(R)[1:k]
6 | $\tau \leftarrow \min_{(X,Y)\in R} Corr(X, Y)$
7 **else if** $UB_{shrunk} \geq \tau$ **then**
8 | Replace largest cluster with subclusters and recurse with TOP-k-QUERY (similar to lines 7-11 of Alg. 3)
9 **else**
10 | $\gamma^* = \frac{\tau-\mu}{UB-\mu}$ *// Compute the critical shrink factor*
11 | B.ADD $((S_l, S_r), \gamma^*)$ *// Add to the priority queue with priority γ^**
/ Phase 2 – starts when Phase 1 is completed */*
12 **for** $(S_l, S_r) \in B$ **do**
13 | THRESHOLDQUERY($S_l, S_r, Corr, \tau$)
14 $R \leftarrow$ SORT(R)[1:k]
15 $\tau \leftarrow \min_{(X,Y)\in R} Corr(X, Y)$

Alg. 3, with an exception of the case where $UB_{\text{shrunk}} \leq \tau < UB$ (Alg. 4 lines 3,7,12). In this case, the cluster combination is *postponed* for further inspection, and placed in a priority queue based on the combination's *critical shrink factor* γ^* – the minimum value of γ for which UB_{shrunk} surpasses τ (lines 9-11). Intuitively, a small γ^* means that the combination (i.e., branch in the comparison tree) is more promising to lead to higher correlation values as a large portion of its bound range ($UB - LB$) exceeds τ.

In the second phase (lines 12-15), postponed branches are traversed in a Depth-First manner (DFS) by invoking Alg. 3 on each combination sequentially. Since τ continuously increases, and the first branches are likely to contain the highest correlation values, most lower-priority branches do not need many cluster splits to reach decisive combinations. Similar to the previous optimizations, the value of γ only impacts efficiency of the algorithm, and not completeness of the results.

5.3.6 Progressive Queries

The prioritization technique of Alg. 4 can also be used as a basis for a progressive threshold algorithm. Precisely, Alg. 4 can be initialized with a user-chosen τ and with $k \to \infty$. This will prioritize the combinations that will yield the strongest correlations, and thus also the majority of correlations larger than τ. Progressive queries are particularly useful when the user is interested in the most similar combinations, but does not want to wait for the algo-

rithm to finish. A common application scenario is exploratory data analytics, where the user may choose to let the algorithm run until completion, which will yield results identical to Alg. 3, or interrupt the algorithm after receiving sufficient answers.

5.4 Detection of Multivariate Correlations in Streaming Data

Data is frequently observed as a live stream. For example, in finance, asset prices may need to be monitored in real-time for detecting strong correlations in a market, for portfolio diversification [210]. In weather monitoring, real-time detection of correlations may reveal interesting short-term weather events, whereas in server monitoring, detection of unexpected correlations, e.g., on server requests originating from many different IP addresses, may reveal attempts of attacks [224]. Similarly, in neuroscience, real-time analysis of fMRI streams to detect correlations brings novel exploitation opportunities, e.g., for neurofeedback training [114, 168, 258].

Our streaming algorithm, called CDStream, builds on top of CD such that it maintains CD's solution over a sliding window as new data arrives. CDStream does this efficiently by storing the decisive cluster combinations in a custom index, which can subsequently be used after each streaming update to quickly identify the potential changes to the result set. Clearly, the main challenge is to construct, maintain, and utilize this index efficiently, for processing streams with high update rates. CDStream currently supports threshold and top-k queries, though only on the *mPC* correlation measure. In the remainder of this section, we will explain the underlying stream processing model and CDStream algorithm in detail. We will also present an extension to CDStream named CDHybrid, which dynamically switches between CDStream and repeated execution of CD in order to adapt to sudden events and concept drift, and improve robustness.

5.4.1 Stream Processing Model

CDStream builds on the basic windows model, which is widely used for processing of data streams, e.g., in [97, 121, 249, 257]. The model works as follows: the sliding window, of length w, is partitioned to a set of smaller, fixed-length sub-windows (often called *basic windows*), each of length b. All stream updates received within a basic window are processed (typically aggregated), to generate a single value for that basic window. In other words, the basic windows define the time resolution handled by the algorithm.

The introduction of basic windows offers several benefits: (a) it makes the results robust to outliers, noise in the data, and time series with small-period oscillations, e.g., stocks with high trading volumes, (b) it allows for handling time-misaligned and out-of-order arrivals, which are fairly common in real-life data streams (e.g., stock ticks, sensors with variable measurement intervals, weak/slow network connections), and (c) it allows efficient handling of streams with high update rates. The downside of traditional basic windows it that they introduce a – potentially significant – delay on the results, which can be as large as b time units. The latter constraint becomes limiting when processing periods of high activity (e.g., in high-volatility periods of a stock market, or when a network is under a DDoS attack), where it is critical that the user observes intermediary results as soon as possible.

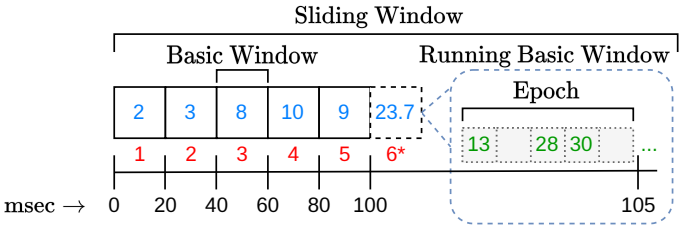


Figure 5.4: Example of a stream representation with the BW$^+$ model with $w = 100$, $b = 20$, $E = 5$. With red we denote the index/position of the basic window. The blue numbers correspond to the values of the corresponding windows. The updates in the running basic window and running epoch are shown in green color.

CDStream alleviates this limitation by disentangling the period of recomputing the results (the key reason behind the stale results) with the length of the basic window b. The model, called BW$^+$ hereafter, offers an extra knob to the user, called the *epoch size* E, which controls the acceptable delay/lag for the algorithm to account for new data. When the epoch size E is set to be equal to b, BW$^+$ degenerates to the standard basic windows model, e.g., as used in [257]. However, by setting $E < b$, the algorithm is instructed to recompute the results more than once within the period of a basic window, accounting also for the new arrivals in the incomplete basic window. The aggregation unit remains unchanged, i.e., the basic window of size b, which allows meaningful handling of time misalignment, noise and outliers. All completed basic windows are not impacted by the epoch size – hence their aggregate values are not recomputed. However, whenever an epoch is completed, the algorithm updates the aggregate value for the incomplete basic window and updates the correlations, to include these new values.

As an example, consider the time series depicted in Fig. 5.4. Assume that E is set to 5 msec, and the basic and sliding window lengths, b and w, are set to 20 and 100 msec respectively. Then, at time 100, BW$^+$ will have identical results to the standard basic windows model. At time 105, BW$^+$ will recompute the results, accounting for the values that arrived in basic windows 1 to 5, and within the first five seconds of the (still incomplete) basic window 6. Therefore, if in the period between time 100 and 105, there were drastic changes that led to updates of the results, these will be detected by BW$^+$. The same process will be repeated at times 110 and 115, whereas at time 120, basic window 1 will expire and the results of BW$^+$ will again become identical to the output of the standard basic windows model (not shown in figure). It is important to note that BW$^+$ with $E < b$ is not equivalent to running the standard basic windows algorithm with $b = E$. BW$^+$ keeps the completed basic windows intact – it does not change their boundaries when an epoch is complete. As we will explain in the following section, this is leveraged by CDStream to optimize performance by avoiding to store, or recompute, fine-grained partial results. We will come back to the discussion about the properties of BW$^+$, and its impact in terms of computational efficiency and accuracy/-completeness of the results of the algorithm in Section 5.4.4.

Algorithm 5: HANDLEEPOCH(D, A, C, I, τ)
Input: Set of time series D, set of arrivals A , pairwise correlations cache C, DCC Index I, correlation threshold τ

```
1 for (i, v) ∈ A do                                   // (i: time series id, v: value)
2 |   Recompute D_i's last basic window's aggregate
3 |   for j = 1 to n do                               // Update pairwise cache C
4 |   |   C[i, j] ←Corr(S_i, S_j)
5 for (i, v) ∈ A do                                   // Check for violations
6 |   V ← QUERYINDEX(i, I)                           // Query violations
7 |   for (S_l, S_r) ∈ V do                          // Recompute and re-index
8 |   |   THRESHOLDQUERY(S_l, S_r, Corr, τ)
9 |   |   UPDATEINDEX(S_l, S_r, Corr, τ, I)
```

Time-based vs arrival-based epoch. Even though our previous discussion assumed that epochs are defined in time units (seconds, minutes, etc.), this does not constitute a requirement of the model. Epochs can also be defined in number of arrivals (e.g., every 10 arrivals). A definition based on number of arrivals may be preferred in use cases where the arrival rate of the time series changes abruptly, e.g., during a market crash.

5

5.4.2 Algorithm Core

We start with a high-level description of CDStream before going over the details of the underlying index, which is instrumental for increasing the throughput of the algorithm.

CDStream receives as an input the set of streaming time series D, and the configuration parameters of the algorithm – the length of the sliding window w, basic window b, epoch size E, and query threshold τ. The algorithm starts by executing CD on the last w arrivals in the given time series, and prints the initial results to the user. A byproduct of CD is an upper-diagonal matrix that stores the pairwise correlations between all pairs of time series. We will refer to this as the pairwise correlations cache. Then, CDStream enters the monitoring phase.

In this phase, whenever an epoch is completed, the algorithm (shown in Alg. 5) first detects all time series that have at least one update and recomputes the corresponding aggregate for the last (potentially still incomplete) basic window (line 2). It then refreshes the cache of pairwise correlations, to account for the new arrivals (lines 3-4). Notice that this step does not recompute the correlations from scratch; it updates them from the previous correlation values and the change in aggregate value for the running basic window. Following, the algorithm goes through all updates within the epoch, and checks whether these could lead to changes in the result set (either new additions in the result or removals). This process is supported by a custom-build index, which returns all decisive cluster combinations with bounds impacted by the newly arrived updates. These impacted bounds are then reassessed using Algorithm 3, in order to detect the potential changes in the result set, and to update the index (Alg. 5 lines 5-9). The described steps are repeated for $\lfloor E/b \rfloor$ epochs, after which a basic window is completed. In that case, CDStream will additionally remove the expired

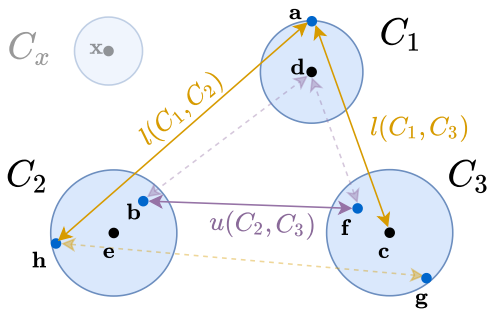


Figure 5.5: Illustration of extrema pairs $\langle \boldsymbol{a}, \boldsymbol{h} \rangle$, $\langle \boldsymbol{a}, \boldsymbol{c} \rangle$, and $\langle \boldsymbol{b}, \boldsymbol{f} \rangle$ for a positive DCC $(\{C_1\}, \{C_2, C_3\})$.

basic window, add the newly-completed basic window, and keep repeating the above process (not depicted in Alg. 5). In the remainder of this section we will look at the custom index, and how this is maintained and utilized by CDStream.

The DCC Index

The index is used for storing a collection of thresholds, that, when fired, signify a potential change in the answer set.[5] The core idea is to store Decisive Cluster Combinations (DCC) for all clusters, and enable re-validating only these after every time series update. Recall that each time series $\boldsymbol{t}$ belongs to a hierarchy of clusters. For example, time series $\boldsymbol{e}$ in Fig. 5.3a belongs to C_2 and C_7. For a time series $\boldsymbol{t}$, we denote the set of these clusters as $C(\boldsymbol{t})$. By construction, the algorithm takes a decision concerning any time series $\boldsymbol{t}$ based solely on the decisive combinations including any cluster in $C(\boldsymbol{t})$ (see the theoretical results in Section 5.3.3). As long as those decisive combinations are still valid, the final result will remain correct and complete.

A naive approach would be to construct an inverted index that maps each cluster to the decisive cluster combinations it participates in. Then, after any update of a time series $\boldsymbol{t}$, we would look at all clusters in $C(\boldsymbol{t})$, and find and re-validate all their decisive combinations from the index. The use of this index could become too slow for some use cases, particularly for large correlation patterns, due to the potentially large number of decisive combinations associated with each cluster that need to be checked. Two key observations can be exploited to optimize the use of this index: (a) the empirical bounds described in Section 5.3.4 do not depend on all time series contained in the cluster, but are determined solely by $l(C_i, C_j)$ and $u(C_i, C_j)$, the minimum and maximum *pairwise* correlations between all involved clusters in the combination, and (b) the previous applies independent of the number of clusters (and time series) contained in the left and right side of the cluster combination. Therefore, the DCC index is designed around these minimum and maximum pairwise correlations, named *extrema pairs*. We now start by introducing the intuition behind extrema pairs, and then explain the index structure and how it is used by CDStream.

[5] Similar indices were used in earlier works, e.g., [174], but for bounding the values of pairwise correlations.

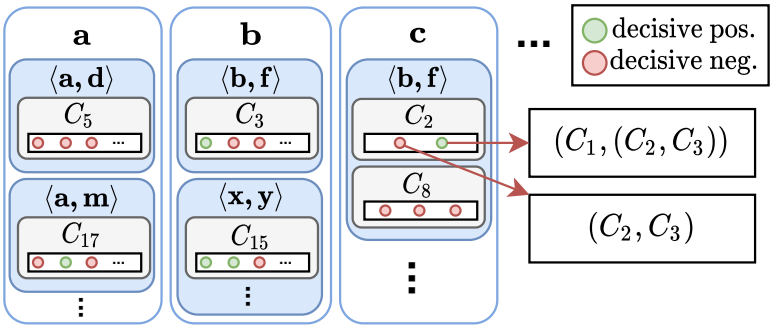


Figure 5.6: Visualization of the decisive combination index.

Extrema Pairs. Consider a *positive* DCC $(\{C_1\}, \{C_2, C_3\})$ composed of clusters from the running example in Fig. 5.3a. Figure 5.5 illustrates the extrema pairs for this combination.[6] In this example, the minimum and maximum extrema pairs for (C_2, C_3) are $\langle \boldsymbol{h}, \boldsymbol{g} \rangle$ and $\langle \boldsymbol{b}, \boldsymbol{f} \rangle$ respectively. From the definitions in Theorem 5.3.2 we can conclude that the dominant bound of this DCC (i.e., the bound closest to the threshold, so LB here) is made up of the minimum extrema pairs of (C_1, C_2) and (C_1, C_3), and the maximum extrema pair of (C_2, C_3). We call these the *active extrema pairs*. Naturally, the DCCs dominant bound will only decrease if any of these active extrema pairs changes undesirably, either by becoming more extreme (i.e., $\langle \boldsymbol{b}, \boldsymbol{f} \rangle$ increases) or by being surpassed by another pair (e.g., $\rho(\boldsymbol{e}, \boldsymbol{f})_{t+1} > \rho(\boldsymbol{b}, \boldsymbol{f})_{t+1}$). This is the key insight behind the DCC index: if we store and index DCCs based on their active extrema pairs, after each update we can quickly retrieve all potential "state changers" (i.e., DCCs that may change the result set) that involve the updated time series $\boldsymbol{t}$, and have a potential change in bounds. We refer to these DCCs as *violations*, since an update may cause their previous bounds to be violated.

Index Structure. Fig. 5.6 depicts an example of the internal organization of the DCC index. At the outer layer, the index is an inverted index that maps each time series $\boldsymbol{t}$ to a list of active extrema pairs. At the inner layer, for each extremum pair $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ we keep a list of all *opposite clusters*, i.e., the clusters that do not include $\boldsymbol{t}$, and participate in at least one decisive cluster combination having $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ as an active extremum pair. For example, focusing at $\boldsymbol{c}$ in Figure 5.6, we see that one of its extrema pairs is $\langle \boldsymbol{b}, \boldsymbol{f} \rangle$, which is reused by both clusters C_2 and C_8. The clusters are stored in decreasing size, i.e., the cluster at position $i+1$ will be a sub-cluster of the cluster at position i. For each cluster, we store all decisive combinations, and whether these are positive or negative. In our example in Fig. 5.6, for cluster C_2 we have a negative combination (C_2, C_3) and a positive combination $(C_1, (C_2, C_3))$. This way of indexing and querying ensures that we only re-validate DCCs with an actual change in bounds, and that this set is complete (i.e., we do not miss any violations).

Using the Index. When an update is observed at time series $\boldsymbol{t}$, the first step is to use the index for retrieving all extrema pairs that involve a cluster in $C(\boldsymbol{t})$. For each extremum pair,

[6] Note that Euclidean distance is inversely related with correlation, meaning that time series (i.e., points) close to one other have relatively high correlations [177].

we check the pairwise cache whether the pair has changed as a result of the last update. This will happen, e.g., if the update of $\boldsymbol{t}$ has caused $\boldsymbol{t}$ to form a new extremum pair with another time series, replacing an older pair. If the extremum pair has not changed, we can skip all contents grouped under this pair altogether. In our example DCC in Fig. 5.5, if $\boldsymbol{c}$ has been updated, but $\langle \boldsymbol{b}, \boldsymbol{f} \rangle$ is still a valid extremum pair for cluster C_2 (i.e., $\boldsymbol{c}$ did not move closer to $\boldsymbol{b}$ than $\boldsymbol{f}$), then no further validations are needed for any of the combinations involving C_2. Furthermore, considering the index structure in Fig. 5.6, no validations are required for the combinations involving C_8 (and any other clusters following C_2 with the same extremum pair), since C_8 is a strict subset of C_2 (recall that the clusters are ordered based on their size). If, on the other hand, the update has invalidated an extremum pair (e.g., $\boldsymbol{c}$ has moved closer to $\boldsymbol{b}$ than $\boldsymbol{f}$), the algorithm drills into the contents of the inner layer, and goes over the clusters sharing this extremum pair (i.e., C_2 and C_8 in our example). If, e.g., $\boldsymbol{c}$ was updated and $\langle \boldsymbol{b}, \boldsymbol{f} \rangle$ is no longer an extremum pair for C_2, we need to check and adjust all combinations stored for C_2 (in this example, (C_2, C_3) and $(C_1, (C_2, C_3))$). This is done by adjusting the extrema pairs and bounds using Theorem 5.3.2, re-validating whether the combination is still decisive – positive or negative, and updating the solution accordingly. In this step, the algorithm may even need to break a cluster to two or more sub-clusters, until it again reaches to decisive combinations. However, again, as soon as we find a cluster for which the extremum pair does not change after the update, we can move to the next extremum pair.

5.4.3 User Constraints and Top-k Queries

To support the minimum jump and irreducibility constraints, additional triggering functionalities, further described below, are added to the index of CDStream.

Irreducibility constraint. Let X, Y, X', Y' denote sets of clusters. Consider combinations (X, Y), and $(X' \subseteq X, Y' \subseteq Y)$, with $|X \cup Y| > |X' \cup Y'|$, i.e., irreducibility excludes (X, Y) from the results if (X', Y') is in. We need to detect two additional cases: (1) (X, Y) needs to be removed from the result set because (X', Y') just surpassed τ, and, (2) (X, Y) needs to be added in the result set, because (X', Y') was just removed from the result set because its correlation dropped below τ. Both cases can be triggered by an update of a time series from X or Y (hence, also from X' and Y').

Without the irreducibility constraint, the index contains the following extrema pairs: (a) for the negative decisive combinations, the pairs required for upper bounding the correlation, (b) for the positive decisive combinations, all pairs required for lower-bounding the correlation. The irreducibility constraint requires also monitoring of the upper bounds of positive decisive combinations (e.g., for case (1), when an increase of $Corr(X', Y')$ will cause the following condition to hold: $Corr(X', Y') > \tau$ which will mean that (X, Y) need to be removed from the result set) and the lower bounds of negative decisive combinations with any $Corr(X', Y') > \tau$. These decisive combinations are also added in the index, under the extrema pairs, and checked accordingly.

Minimum jump constraint. Monitoring for the minimum jump constraint is analogous to the irreducibility contraint. The following cases need to be considered: (1) (X, Y) needs to be removed from the result set because $Corr(X', Y') + \delta > Corr(X, Y)$, and (2) (X, Y)

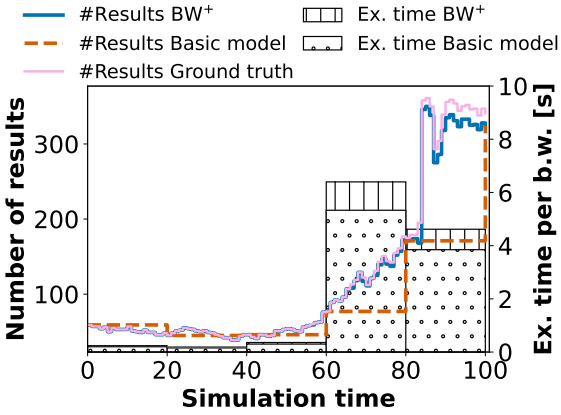


Figure 5.7: Number of results and execution time per basic window, with BW$^+$ and the standard basic window model. BW$^+$ is configured with minute granularity updates. The results correspond to the Stocks dataset, with $n = 1000$, $w = 120000$, and $b = 20$.

needs to be added in the result set because $Corr(X', Y') + \delta < Corr(X, Y)$. Both cases are identified using the discussed method for monitoring the irreducibility constraint.

Top-k queries. Recall that CDStream is initialized with the result of CD. For a top-k query, CDStream queries CD for a slightly larger number of results $k' = b_k * k$, where b_k is a small integer, greater than 1. CDStream finds the minimum correlation in these results, and uses it as a threshold τ in the streaming algorithm. As long as the size of the result set is at least k, the true top-k results will always have a correlation higher than τ and will be contained in the top-k' results maintained by the algorithm. Therefore, the top-k out of the detected top-k' correlations are returned to the user.

Scaling factor b_k controls the tradeoff between the robustness of the streaming algorithm for top-k queries, and its efficiency. Setting $b_k = 1$ may lead to the situation that, due to an update, fewer than k results exist with correlation greater than or equal to τ. CDStream then fails to retrieve enough results, and resorts to CD for computing the correct answer, and updating its index. Conversely, a large b_k will lead to a larger number of intermediary results, and to more effort for computing the exact correlations of these results, which is necessary for retaining the top-k results. Our experiments with a variety of datasets have shown that $b_k = 2$ is already sufficient to provide good performance without compromising the robustness of CDStream. We evaluate CDStream in Section 5.5.3.

5.4.4 Impact of Streaming Processing Model on CDStream

Recall that CDStream leverages the proposed extended basic window stream processing model (abbrev. as BW$^+$) in order to identify updates on the result set earlier. By construction, BW$^+$ is *at least as good* as the standard basic windows model BW in terms of completeness of the result set, since it replicates its behavior every time a basic window is completed. The further improvement that we can expect from BW$^+$ – compared to the stan-

dard basic windows model BW– depends on the volatility of the input streams. In periods where the streams contain negligible changes, BW$^+$ will detect very few additional correlations (if any), compared to the standard model. In periods of high volatility, such as market crashes, BW$^+$ will detect updates and new correlations faster.

To examine the importance of BW$^+$ and evaluate its impact on the computational efficiency of CDStream, we compared the results of CDStream using either the standard basic windows model BW or the new BW$^+$ model. The standard basic windows model was simulated by running BW$^+$ with $E = b$, i.e., the epoch size is equal to the basic window size. In the extended model case, we set $E = 1$, causing the algorithm to update the results every time a new arrival is received. Figure 5.7 presents the number of results (left axis) and runtime (right axis) of CDStream of the two models. The results correspond to processing of a streaming time series with minute-granularity stock prices of 1000 stocks on 16 March 2020 (the dataset is further described in Section 5.5). This day was selected because it was the day of the largest price drop in the 2020 Covid crash [1]. As ground truth, we used the results of CD on the same input dataset (without basic windows), recomputed at the end of each epoch.

We see that BW$^+$ is able to identify jumps in the number of results significantly earlier than BW. In fact the results of BW$^+$ closely resemble the ground truth. Comparison with the ground truth revealed that BW$^+$ maintained a recall of 97.8% during this period while BW recall decreased to 69.0%. From epoch 0 to 60 (prior the crash), the recall of BW$^+$ was 100%. It is also interesting to consider execution time per basic window. Since the new model subsumes the basic window model, it is slightly more expensive to maintain. However, extra computation is only around 10%, for the more-detailed epoch. This extra computation can of course be further reduced, by increasing the epoch size E. Therefore, all experiments hereafter will only focus on the BW$^+$ model.

5.4.5 CDHybrid: Combining CD and CDStream

Recall that CDStream handles the time series updates in epochs. The algorithm exhibits high performance when the updates do not drastically change the results set. In time series where the answer changes abruptly, it may be more efficient to simply run CD after the completion of each epoch and recompute the solution from scratch, instead of maintaining CDStream's index and the result through time. CDHybrid is an algorithm that orchestrates CD and CDStream, transparently managing the switch between the two based on the properties of the input time series.

To decide between CD and CDStream, CDHybrid needs to estimate the cost of both approaches for handling an epoch. Since switching needs to happen quickly without introducing noticeable overhead, the estimation of execution times needs to be extremely lightweight. A good predictor for the execution time of CDStream is the number of arrivals in the epoch – more arrivals tend to cause more changes in the result, which takes longer for CDStream to handle. Therefore, CDHybrid starts with a brief training period, where it collects statistics on the observed number of arrivals in each epoch, and the execution time of the two algorithms (i.e., it runs both CD and CDStream at every epoch, and measures the execution time). Simple (online) linear regression is then used to model the relationship between exe-

cution time and the observed number of arrivals. Note that the coefficients of a simple linear regression model can be maintained in constant time and space. Therefore, the regression model is continuously updated, even after the training phase. Switching from one algorithm to the other works as follows.

Switching from CDStream to CD. We cache the current results of CDStream (we will refer to these as R_{CDStream}) and stop maintaining the index. When an epoch is completed, the time series are updated and passed to CD for computing the result.

Switching from CD to CDStream. Since the index was not updated for some time, we need to update it before we can use it again. We compute the symmetric difference Δ of the current results of CD (denoted as R_{CD}) with the last results of CDStream R_{CDStream}. Any result r contained in $\Delta \cap R_{\text{CDStream}}$ is due to a positive decisive combination that has now become negative, whereas any r contained in $\Delta \cap R_{\text{CD}}$ leads to a new positive decisive combination. In both cases, the algorithm updates the index accordingly. There is also the case that a decisive combination becomes indecisive. In this case, the algorithm recursively breaks the combination further, as shown in Alg. 3. We evaluate CDHybrid in Section 5.5.3.

5.5 Evaluation

The purpose of our evaluation is twofold: (a) to assess the scalability and efficiency of our methods, and (b) to compare them to a series of baselines. The baselines include state-of-the-art algorithms for multivariate correlation discovery [9, 10] and two variants of an exhaustive search algorithm that iterate over all possible combinations. Our evaluation focuses on efficiency rather than practical significance of multivariate correlations, as the latter was already extensively demonstrated in earlier works and case studies from different domains, e.g., [9, 10, 150] (see Section 5.1 for more examples). Still, to ensure our evaluation is conducted on data where detection of multivariate correlations has practical relevance, we also compare our methods with datasets used in these past case studies (or similar datasets, where the original datasets were inaccessible).

Compared methods. For experiments on static data, we compare our method **CD** against four baselines: **UNOPT** and **OPT**, two variants of an exhaustive search that iterate over all possible combinations of time series. UNOPT is a naive implementation that recomputes all correlations for each combination, while OPT reuses cached pairwise correlations based on our theoretical findings (Section 5.3.2). Both provide perfect precision and recall by examining every possible combination. **CoMEtExtended** [10]: a state-of-the-art approximate algorithm for the Multipole measure that uses clique enumeration and relies on a parameter ρ to balance completeness against efficiency, and **CONTRa** [9]: an algorithm specifically designed for finding tripoles (i.e., $mPC(2, 1)$) that requires the minimum jump constraint. See Section 5.2.3 for more details on CONTRa and CoMEtExtended.

For streaming data experiments, we compare three approaches: (a) **CDStream**, our streaming-specific algorithm, (b) **CD**, which runs our static CD algorithm repeatedly on each window and, (c) **CDHybrid**, a hybrid approach that combines features from both CDStream and CD.

Hardware and implementations. All experiments were executed on a server equipped with two Xeon E5-2697v2 12-Core 2.70 GHz processors and 500GB RAM. For CoMEtExtended

	Static					Streaming					
Dataset	n	m	k	γ	κ	n	**Epoch size**	**Basic window size**	w **(hours)**	**Aggr. Method**	τ
Stocks	1440	1309	100	0	10	1000	1 min	2 hours	2000	sum	0.95
fMRI	1440	5470	100	0	10	1440	1 sec	1 sec	0.5	last	0.9
SLP	1440	1827	100	0	10	1000	1 hour	6 hours	2160	avg	0.99
TMP	1440	1827	100	0	10	1000	1 hour	6 hours	2160	avg	0.99
Crypto	708	2660	100	0	10	1000	1 min	1 hour	216	sum	0.95
Deep	96	10,000	100	0	10	-	-	-	-	-	-

Table 5.2: Default parameters for the experiments with static and streaming data

and CONTRa, we used the original implementations provided by the authors [9, 10]. We configured the implementation of CoMEtExtended to utilize all available cores of our server. Consequently, all implementations except CONTRa were multi-threaded. Algorithm performance comparisons are exclusively made under matching execution styles (e.g., comparing single-threaded CD only to CONTRa). All implementations, except the UNOPT exhaustive search baseline, cached and reused pairwise correlation computations using our results from Section 5.3.2. This caching consistently improved performance across all methods. The reported runtime for CD and CDStream corresponds to the total execution cost, including normalization, clustering, and pairwise correlation calculation. All reported results are medians from 10 repetitions.

Datasets. We conducted extensive evaluations on seven datasets from distinct domains (neuroscience, finance, crypto trading, climate science, and machine learning). Download links, preprocessing steps, and code for processing the data are available at [2].

- **Stocks:** Daily closing prices of 28678 stocks over the period Jan. 2, 2016 to Dec. 31, 2020 (1309 observations). For streaming experiments, we used minute-level closing prices.
- **fMRI:** Functional MRI data of a person watching a movie.[7] We extracted five datasets by mean-pooling with kernels of different sizes, resulting in 237, 509, 1440, 3152, and 9700 time series respectively, each with 5470 observations. A similar dataset was used in the case study of [9].
- **SLP & TMP:** Segment of the ISD weather dataset [185] containing sea level pressure (**SLP**) and atmospheric temperature (**TMP**) readings from 3222 sensors. For static evaluation, we used daily averages between January 1, 2016 and December 31, 2020 (2927 readings per time series). Streaming experiments used hourly measurements.
- **SLP-small:** Sea Level Pressure data [208] used in the case study of [10], containing 171 time series with 108 observations each.
- **Crypto:** 3-hour closing prices of 7075 crypto-currencies with 713 observations each, covering April 14, 2021 to July 13, 2021. Streaming experiments used minute-level prices.

[7] Available at https://openneuro.org/datasets/ds002837/versions/2.0.0. We used file "sub-1_task-500daysofsummer_bold_blur_censor", which includes recommended preprocessing for voxel-based analytics.

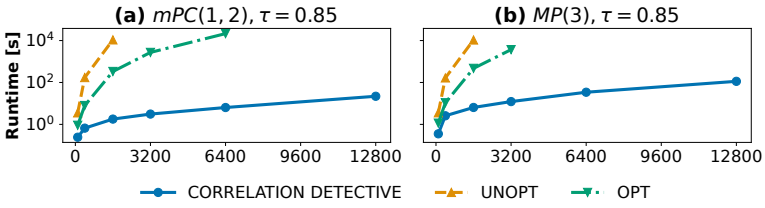


Figure 5.8: Scalability of CD and exhaustive baselines for threshold queries on subsets of Stocks. Notice that the Y axis is in logarithmic scale.

- **Deep:** A billion time series of length 96, obtained by extracting embeddings from the final layers of a convolutional neural network [3].

When needed, we obtained subsets of these datasets through random sampling. To avoid repetition, we only mention experimental configurations when they deviate from the default parameters in Table 5.2. The remainder of this section is organized as follows: We first compare our methods to the baselines (Section 5.5.1), then conduct sensitivity analyses of CD (Section 5.5.2) and CDStream (Section 5.5.3).

5.5.1 Comparison to the Baselines

We begin by comparing CD to: (a) two algorithms based on exhaustive search, (b) commercial and open-source database management systems, (c) CoMEtExtended [10], and (d) CONTRa [9]. Our experiments compare both efficiency and recall for threshold queries.

Comparison to exhaustive search baselines. Since no existing solution covers CD's range of queries and correlations, we constructed two baselines (UNOPT and OPT) that exhaustively compute all multivariate correlations by iterating over all possible combinations of time series. This comparison focuses solely on runtime, as all methods provide perfect precision and recall.

Figure 5.8 shows the time required by CD, UNOPT, and OPT to execute threshold queries on Stocks datasets of increasing size (up to 12,800 time series). All algorithms were given a maximum of 8 hours to complete. The thresholds were selected to ensure all correlation measures return approximately the same number of results on each dataset. Our results demonstrate that CD's runtime grows at a significantly slower rate than both baselines across all correlation measures. The efficiency gap widens as dataset size increases, highlighting the importance of an efficient solution like CD for handling large datasets.

Comparing OPT to UNOPT reveals that caching pairwise correlations improves performance for both correlation patterns. However, even OPT times out on larger datasets. CD's superior scaling compared to OPT indicates that its main performance advantage comes from effective utilization of cluster bounds rather than just correlation caching.

Comparison to CoMEtExtended. Our next experiment compared CD with CoMEtExtended [10]. CoMEtExtended's objective differs slightly from our problem statement. First CoMEtExtended only supports the Multipole measure, while CD supports both *mPC* and

(τ, δ)	CoMEtExtended						Correlation Detective			
	$\rho_{CE} = 0$		$\rho_{CE} = 0.01$		$\rho_{CE} = 0.02$		$MP(4)$		$MP(5)$	
	time	#res.	time	#res.	time	#res.	time	#res.	time	#res.
(0.4, 0.1)	604	62663	1318	67110	3530	70921	7	71083	1132	88305
(0.4, 0.15)	511	7244	1218	7300	3393	7343	5	7559	579	7562
(0.4, 0.2)	501	2166	1210	2171	3327	2174	4	2183	248	2183
(0.5, 0.1)	459	30632	1099	33718	2836	36457	5	34592	635	51391
(0.5, 0.15)	398	3646	1006	3702	2760	3745	4	3961	355	3964
(0.5, 0.2)	390	1434	1006	1439	2701	1442	3	1451	193	1451
(0.6, 0.1)	246	7823	598	8892	1592	9859	3	9204	289	17349
(0.6, 0.15)	223	1569	577	1606	1559	1635	3	1840	177	1843
(0.6, 0.2)	219	771	568	776	1532	779	2	788	112	788

Table 5.3: Comparison of CD with CoMEtExtended on SLP-small dataset: runtime (seconds) and number of retrieved results.

MP. Second, CoMEtExtended is approximate without guarantees, though its recall can be tuned via parameter ρ_{CE} (range: -1 to 1). Values around 0 provide a reasonable efficiency-recall tradeoff [10]. In contrast, CD delivers complete answers, making both runtime and recall relevant in our comparison. Trird, CoMEtExtended focuses on *maximal* strongly correlated sets, whereas CD finds *all* such sets (up to a specified cardinality). To ensure fair comparison, we also considered all subsets of CoMEtExtended's returned sets. When a subset satisfied the query, we added it to CoMEtExtended's results, increasing its recall. This post-processing was not included in CoMETExtended's runtime to avoid penalizing its performance. Table 5.3 presents the results and runtime of both algorithms on the SLP-small dataset using configuration parameters from [10].

The results show that CD not only executes at least an order of magnitude faster than CoMEtExtended, but also achieves substantially higher recall since it finds all answers, whereas CoMEtExtended, being approximate, misses a significant portion of the results. For $MP(4)$, CoMEtExtended with $\rho_{CE} = 0$ (resp. $\rho_{CE} = 0.02$) is one to two (resp. two to three) orders of magnitude slower than CD. For queries with $\delta = 0.1$, $\rho_{CE} = 0.02$ and $\tau = 0.4$, CoMEtExtended found 281 results with 6 time series, and one with 7. These account for approximately 0.3% of the total discovered results. CD, executed with $p_l = 5$, did not discover these as it prioritized simpler, more interpretable results. Nevertheless, CD still found 25% more results than COMEtExtended in one-third of the time. Moreover, case studies in [9, 10] demonstrate the usefulness of relatively simple relationships involving at most four time series on this dataset. Other works on multivariate correlations also emphasize discovering relationships without too many time series [57]. For these cases with fixed l_{max}, CD is guaranteed to find a superset of COMEtExtended's results at a fraction of its cost.

Comparison to CONTRa. We also compared CD to CONTRa [9] for tripole discovery (i.e., $mPC(1, 2) \geq \tau$). To ensure fair comparison, CD was configured to find the same results as CONTRa and to use equivalent hardware: (a) CD was executed with $\tau = 0$, with pruning solely due to the minimum jump constraint, and (b) CD was limited to a single-threaded execution since CONTRa's implementation was also single-threaded. CONTRa was configured to return exact results.

	CONTRa		Correlation Detective					
			$\tau = 0$		$\tau = 0.5$		$\tau = 0.9$	
δ	time	#res.	time	#res.	time	#res.	time	#res.
0.1	>24hrs	23e6	11510	23e6	1908	21e6	401	432
0.15	11160	73e4	4927	73e4	1569	73e4	391	102
0.2	5324	21e3	1983	21e3	1281	21e3	441	24

Table 5.4: Comparison of CD with CONTRa on fMRI dataset ($n = 9700$): runtime (seconds) and number of retrieved results.

Table 5.4 presents runtime and result counts for each method.[8] CD proves more efficient than CONTRa when detecting identical results, even with $\tau = 0$. However, $\tau = 0$ produces an impractically large result set. Thus, we also evaluated CD with $\tau = 0.5$ (corresponding to the lowest correlation reported in [9]'s case studies), and with $\tau = 0.9$, which yields a more reasonable result set. These settings further decrease CD's runtime by one to two orders of magnitude while preventing overwhelming result quantities.

Summary. Comparison of CD with two state-of-the-art algorithms and two exhaustive baselines demonstrates that CD consistently outperforms all competitors, requiring at least an order of magnitude less time. This enables CD to find more complex query patterns on larger datasets.

5

5.5.2 CD on Static Data

The following experiments evaluate CD's efficiency under various conditions (configurations, datasets, and queries). We first examine the impact of CD's configuration parameters (shrink factor and clustering distance) on efficiency. Since CD is exact, we focus solely on runtime rather than recall. Then, we evaluate CD's performance for top-k and threshold queries.

Parameterization

We tested the impact of the values of the shrink factor γ and the number of sub-clusters per cluster κ on CD's efficiency across different configurations. The results showed that both very small ($\gamma = -0.8$) and very large ($\gamma = 0.8$) shrink factor values lead to suboptimal performance (roughly 38%-72% slower than with optimal γ), as they delay increasing the running threshold τ. Similarly, extreme κ values led to suboptimal performance, with executions up to 2x slower than with optimal values. Detailed results appear in Table 5.5. Importantly, setting $\gamma = 0$ and $\kappa = 10$ consistently delivered near-optimal performance across all configurations — at most 17% worse than the optimal performance in each case. Therefore, we use $\gamma = 0$ and $\kappa = 10$ for subsequent experiments.

[8] For this experiment, the minimum jump parameter δ follows the definition in [9], representing the minimum difference between the *squared* correlations.

		fMRI					Stocks				
	$\tau \setminus \kappa$	2	5	10	25	50	2	5	10	25	50
$mPC(1,3)$	0.8	446	**108**	121	131	157	722	106	106	142	**104**
	0.9	140	**35**	40	49	63	281	**23**	27	48	36
$mPC(2,2)$	0.8	883	**174**	188	177	197	715	78	**75**	92	80
	0.9	264	**57**	64	68	94	179	**22**	**22**	35	30
$MP(4)$	0.8	4799	1037	**1014**	1061	1149	10547	**1366**	1369	1809	1424
	0.9	2451	**592**	606	641	706	6808	1020	**981**	1497	1200

Table 5.5: Runtime times (in seconds) for varying values of κ-means and τ on fMRI and Stocks datasets, with $\delta = 0.05$. Bold values indicate the optimal κ for each configuration.

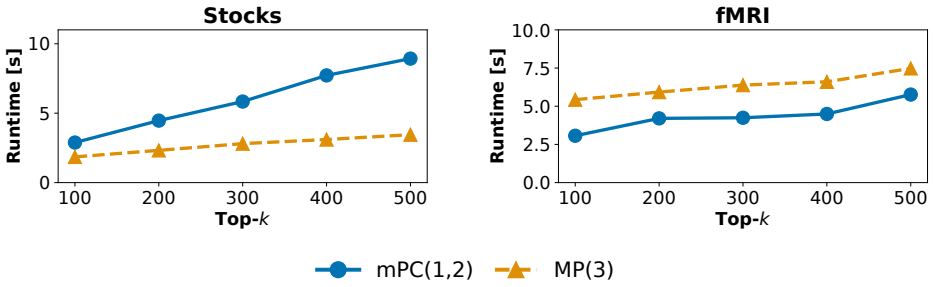


Figure 5.9: Effect of k values and dataset on runtime.

Top-k queries

Effect of k. Fig. 5.9 shows CD's runtime for different k values on Stocks and fMRI datasets. We observe that decreasing k generally improves efficiency. A smaller k enables a rapid increase in the running threshold τ, leading to more aggressive pruning in Alg. 3, line 4. Interestingly, this effect varies across correlation measures. For example, reducing k significantly improves performance for *mPC*, but provides a smaller boost for *MP*. This discrepancy stems from differences in result set correlation values and bound tightness. On the Stocks dataset, the lowest *MP* value in the result set only decreases from 0.998 (top-100) to 0.997 (top-500). In contrast, the lowest *mPC* value decreases from 0.898 (top-100) to 0.862 (top-500) on the same dataset.

Effect of the correlation pattern. Table 5.6 presents CD's runtime for different correlation patterns. As expected, increasing pattern complexity increases computational time. However, despite the search space complexity following $O\left(\binom{n}{p_l+p_r}\right)$, CD's runtime grows at a much slower rate. For the fMRI dataset, the search space size grows by 5 orders of magnitude between $mPC(1,2)$ and $mPC(1,4)$, whereas CD's runtime increases by only three orders of magnitude, demonstrating efficient search space pruning.

Experiments with different datasets. Fig. 5.10 shows CD's runtime across all correlation measures on different datasets. We observe that efficiency for *mPC* queries remains relatively stable across datasets. Performance for *MP* queries, on the other hand, fluctuates significantly. This results from the inherent characteristics of the datasets: analysis of the distributions of multivariate correlation values revealed that correlations in each dataset fol-

	mPC					*MP*	
	(1,2)	**(1,3)**	**(1,4)**	**(2,2)**	**(2,3)**	**(3)**	**(4)**
fMRI	2	12	3008	11	683	2	3
Stocks	1	5	564	4	1071	3	61

Table 5.6: Runtimes of CD with different correlation patterns on top-k queries (seconds)

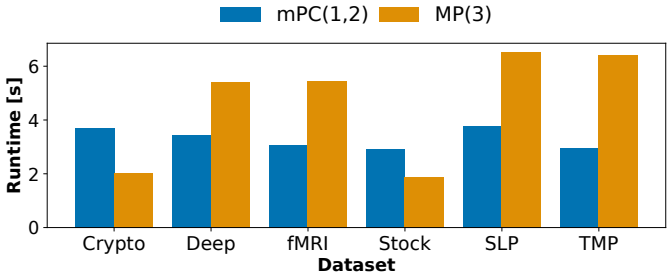


Figure 5.10: Effect of dataset on top-k runtime.

lowed gamma-like distributions. For *MP*, the mean of this distribution is sometimes very close to the minimum correlation in the answer set, i.e., the correlation of the top-k'th answer. In other words, the Multipole measure exhibits reduced discriminative power on these specific datasets. This behavior is expected since Multipole is particularly designed to capture higher-order relationships that may not be prominent in all domains. For instance, while financial time series often exhibit simpler pairwise correlations, datasets from fields like neuroscience or climate science frequently contain complex higher-order relationships that Multipole excels at detecting [10]. These situations could be prevented by performing exploratory analysis on the correlation value distribution of a small sample of the dataset. If this analysis does not indicate exceptionally high correlations, the data analyst could opt for an alternative correlation measure.

Threshold queries

Effect of threshold. Figure 5.11 shows the effect of threshold τ on CD's runtime for the Stocks (left Y-axis) and fMRI dataset (right Y-axis) for each correlation measure, and for different constraints. Our first observation is that increasing the threshold leads to higher efficiency for both correlation measures and both datasets. This is expected, since a higher threshold enables more aggressive pruning of candidate comparisons: the upper bounds derived by Theorems 5.3.2 and 5.3.3 will be below τ more often, leading to fewer recursions. For similar reasons, stronger constraints (i.e., higher δ or introduction of the irreducibility constraint) generally lead to better performance due to increased pruning power. Furthermore, CD is noticeably faster for *mPC* compared to *MP* for the same τ values. This is due to two reasons: (a) the high complexity of eigenvalue computation (cubic to p_l), required for *MP* bounds (Theorem 5.3.3), and (b) *MP* typically results in higher correlation values and more answers for the same τ compared to *mPC*.

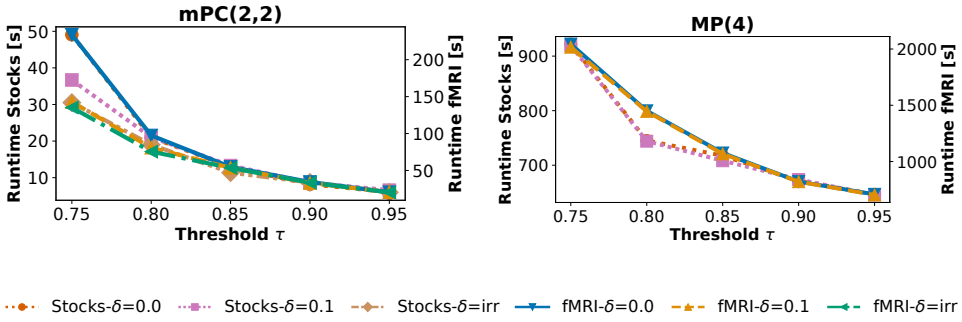


Figure 5.11: Effect of constraint and τ on query performance (Stocks and fMRI)

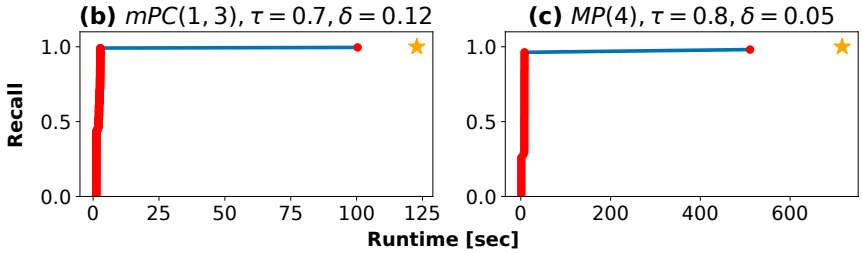


Figure 5.12: Number of retrieved results over runtime, for progressive execution of queries on Stocks.

Progressive variant of CD. Progressive algorithms should quickly collect the majority of results to provide early insights and enable query adjustments. To evaluate this characteristic of progressive CD (Section 5.3.6), we modified our code to save the retrieved results at different time points, and compared these intermediary results with the ground truth to compute recall. We focused on queries with significant runtime, as these benefit most from a progressive algorithm.

Figure 5.12 plots the number of results returned by progressive CD at different time points for all correlation measures on the Stocks dataset. For all correlation measures, CD retrieves all but one result within the first few seconds — less than 1% of the total runtime. This property is particularly appealing for cases where approximate results suffice.

Summary. The default configuration parameters for CD (number of clusters and shrink factor) provide near-optimal performance. Complexity of CD grows at a much slower rate compared to the search space size, and CD is more efficient when the chosen correlation measure and threshold are discriminating for the dataset. Finally, progressive CD retrieves more than 99% of results within the first few seconds.

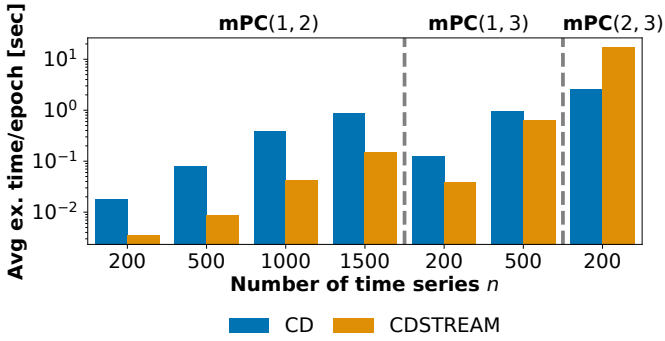


Figure 5.13: Effect of dataset size and correlation pattern, with $\tau = 0.95$, Stocks

5.5.3 CDStream on Streaming Data

The third set of experiments was designed to evaluate CDStream's performance. We used timestamps in all datasets (except Deep, which lacked time) to generate streaming time series. We present detailed results for the Stocks dataset and include other datasets only when they provide additional insights. We start with time-based epoch experiments, then investigate CDStream's performance using arrival-based epochs.

5

Experiments with time-based epochs

Effect of number of time series. Figure 5.13a presents CDStream's average processing time per epoch for different numbers of time series (See Table 5.2 for the epoch sizes). Since there is no streaming baseline for CDStream, the plot includes CD's average runtime per epoch using the same sliding window data (repeated CD executions maintain results with streaming updates). CDStream is more efficient than CD for small correlation patterns, requiring only a few milliseconds per epoch — an order of magnitude less than CD for both correlation measures. Despite the search space growing combinatorially with the number of time series, CDStream's runtime increases much more slowly. This is due to the internal hierarchy of the DCC Index, which reduces work for each update by grouping DCC by time series, extrema pairs, and clusters, in that order. However, CD outperforms CDStream on more complex correlation patterns due to CDStream's index maintenance cost: as pattern complexity increases, the number of combinations in the index grows, eventually surpassing the performance boost from the index. This highlights the importance of an automated algorithm (like the hybrid algorithm in Section 5.4.5) that dynamically switches between CD and CDStream for optimal performance.

Effect of the query parameters. Table 5.7 presents the effect of τ and additional constraint values (minimum jump and irreducibility) on CDStream's performance. CDStream's efficiency is robust to constraints — a constraint only slightly affects the number of decisive combinations to monitor. In contrast, increasing τ improves performance by allowing earlier decisive combinations, similar to CD.

		CD					CDSTREAM				
	$\tau \setminus \delta$	**0.0**	**0.05**	**0.1**	**0.15**	**irr**	**0.0**	**0.05**	**0.1**	**0.15**	**irr**
$mPC(1,2)$	**0.80**	1.025	0.903	0.772	0.756	0.723	0.234	0.226	0.214	0.206	0.206
	0.90	0.441	0.426	0.412	0.412	0.402	0.085	0.079	0.069	0.069	0.070
	0.95	0.370	0.366	0.365	0.346	0.342	0.051	0.045	0.045	0.046	0.045

Table 5.7: Effect of τ and δ on CD and CDStream's average runtime per epoch (in seconds) with streaming data, Stocks

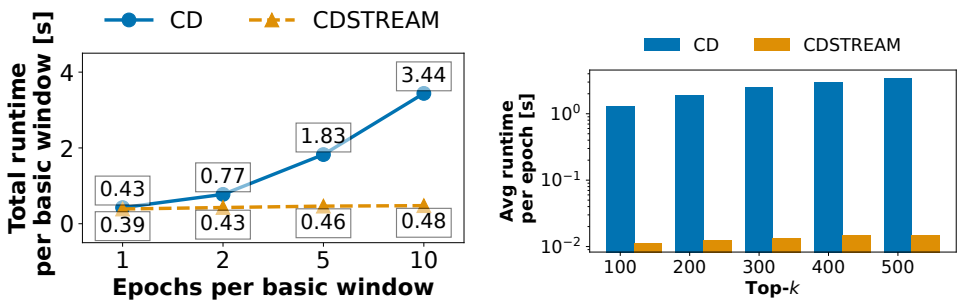


Figure 5.14: (a) Effect of epoch size E (time-based), $mPC(1,2)$ with $\tau = 0.95$, Stocks, (b) Effect of value of k in top-k queries, $mPC(1,2)$, Stocks

Effect of epoch size E. For the next experiment, we fixed the basic window size b to 10 minutes and measured the total processing time per basic window for different epoch sizes. Since the basic window size is fixed, E also determines the number of epochs per basic window. Figure 5.14a shows that CDStream exhibits significantly better scaling behavior compared to CD when increasing the number of epochs per basic window. For a single epoch, both algorithms show comparable performance (0.39 seconds for CDStream vs 0.43 seconds for CD). However, as the number of epochs increases, CD's runtime grows linearly since it must reconstruct the complete result set for each epoch. In contrast, CDStream's runtime increases only marginally (from 0.39 to 0.48 seconds when moving from 1 to 10 epochs per basic window) because it performs only the minimal necessary work by focusing on DCCs affected by updates. This efficiency difference has important implications for result freshness: with the same computational budget, CDStream can provide more frequent updates than CD. For instance, within a time span of 0.5 seconds, CDStream could update the results 10 times while CD would only manage a single update.

Top-k queries. Fig. 5.14b plots the average processing time per epoch for top-k queries under $mPC(1,2)$ for different k values. The results correspond to the Stocks dataset with 1000 stocks. Processing time for both algorithms increases with k, but at a much slower rate for CDStream compared to CD. In CD, runtime grows almost linearly with k (from 1.31 seconds to 3.41 seconds), whereas for CDStream the time increases only from 0.011 seconds to 0.015 seconds for the same queries. This notable difference in efficiency arises because CDStream maintains the top-k solutions, already having a good estimate for the top-k threshold from previous runs, whereas CD starts each run from scratch. Thus, CDStream's

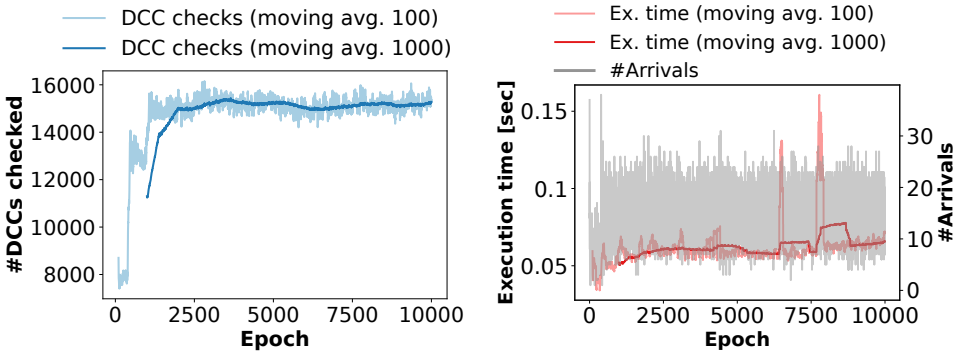


Figure 5.15: Long-term development of DCCs and execution time for CDStream, Stocks, $mPC(1, 2)$

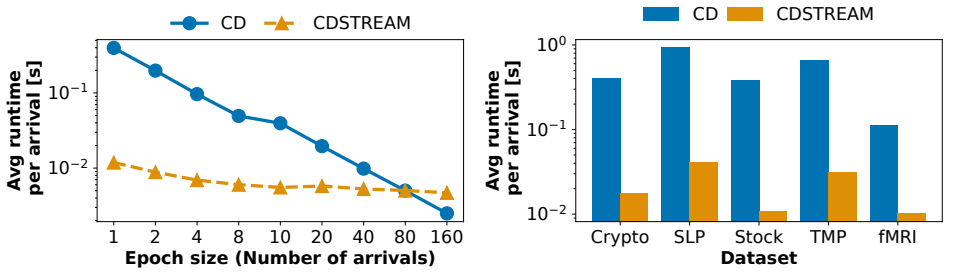


Figure 5.16: Effect of query parameters on CDStream's performance with an arrival-based epoch. (a) Effect of epoch size E, with $\tau = 0.95$, Stocks; (b) Effect of dataset

runtime increase for higher k values comes only from updating and sorting a slightly larger result set and buffer set.

Long-term CDStream performance To evaluate CDStream's index sustainability and robustness, we executed CDStream for an extended period (up to 10,000 epochs, detecting $mPC(1, 2)$) and measured processing time and key DCC index statistics. Figure 5.15 plots the number of DCCs checked after each epoch and the execution time per epoch (moving averages over 100 and 1000 epochs). After a few hundred epochs, the number of DCCs to validate and update per epoch stabilizes. Execution time remains stable throughout the time series, with small spikes in the 100-epoch moving average. These spikes occur when updates invalidate many DCCs, causing major result set changes. However, the spikes are smoothed out in the 1000-epoch moving average, indicating that a small buffer for queuing time series updates would suffice for CDStream to catch up.

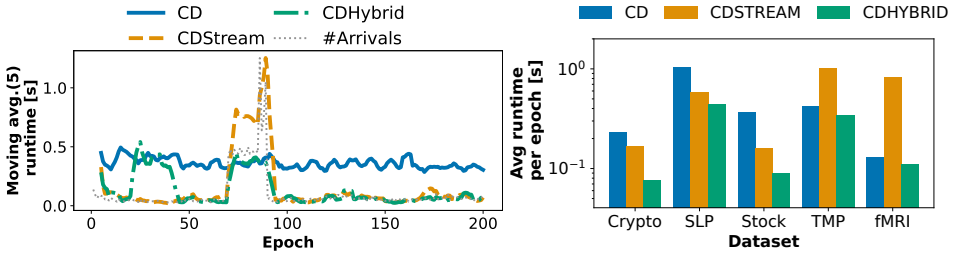


Figure 5.17: Analysis of CDHybrid with $mPC(1, 2)$ (a) Efficiency of over time on Stocks; (b) Impact of dataset on CDHybrid efficiency.

Experiments with arrival-based epochs

Effect of epoch size E. Figure 5.16a plots the average processing time per arrival for varying epoch sizes. As a reference, the plot includes the average processing time for periodic CD re-execution at the end of each epoch (amortized over the epoch's arrivals). Increasing E also increases CDStream's performance. This is expected, as larger epochs provide more opportunities for CDStream to reduce the number of DCCs to check. Thus, similar to time-based epochs, E allows users to balance CDStream's throughput with result freshness. The runtime per arrival for CD approaches that of CDStream as E increases, with processing times crossing at $E = 80$.

Effect of dataset. Figure 5.16b presents the average runtime per arrival ($E = 1$ arrival) for $mPC(1, 2)$ threshold queries on all datasets. The cost of periodic CD execution at the end of each epoch is also included as a reference. Arrivals are processed in at most 50 msec, but processing cost is noticeably higher for the two weather sensor datasets (SLP and TMP) compared to others. This is due to the lower time resolution in these datasets (minimum arrival rate is 1 hour, compared to seconds/minutes for others), leading to higher result set volatility and more frequent DCC index updates.

Evaluation of CDHybrid

For the final set of experiments, we tested CDHybrid's ability to switch seamlessly and efficiently between CD and CDStream to minimize processing cost during stream bursts. Since our time series lacked significant bursts to cause noticeable differences in CDStream's runtime, we introduced an artificial burst in all time series between epochs 70 and 90 by temporarily increasing the arrival rate by a factor of 30. CDHybrid had a small warmup period of 40 epochs, processing updates while switching between CD and CDStream to collect initial measurements and train the cost regression model.

Algorithm effectiveness. Figure 5.17a depicts the processing time per epoch (moving averages over 5 epochs) for processing Stocks with CD, CDStream, and CDHybrid. The figure also includes the number of arrivals within each epoch (right Y axis). When the burst starts (around epoch 70), CDStream becomes significantly slower than CD, while CD's performance remains unaffected. CDHybrid immediately recognizes the burst and switches to CD, maintaining peak performance. After the burst ends (shortly after epoch 90), CD-

Hybrid switches back to CDStream. This switch includes a small overhead for updating the DCC index, but this is negligible compared to the overall savings from using a more efficient algorithm at each epoch.

Effect of dataset. Figure 5.17b shows the average processing time per epoch for CD, CDStream, and CDHybrid on all datasets (excluding warm-up time). CDHybrid consistently outperforms both CD and CDStream, indicating that neither CD nor CDStream is optimal for processing the entire time series. CDHybrid efficiently switches between the two in response to varying arrival rates, providing near-optimal performance for each epoch.

Summary. CDStream outperforms CD in most streaming scenarios. Epoch size E provides a useful knob for balancing CDStream's throughput with result freshness. Finally, CDHybrid seamlessly combines CD and CDStream, offering consistently better performance than both.

5.6 Reflection and Conclusions

In this chapter, we addressed the challenge of detecting high-order multivariate correlations through a family of algorithms that outperform existing methods in both accuracy and computational efficiency. Our evaluation demonstrated that our solutions consistently outperform state-of-the-art approaches, often by several orders of magnitude, allowing them to handle datasets on which previous approaches failed to complete. Furthermore, we extended our methods to support streaming data scenarios, enabling efficient updates as new data arrives. Building on the foundations established in Chapter 2, our work tackles the relation-level extension of similarity search by efficiently identifying meaningful relationships among multiple time series.

Chapter summary. We introduced Correlation Detective (CD), a novel algorithm for efficiently discovering multivariate correlations in static data. Unlike previous approaches that either rely on restrictive similarity definitions, make constraining assumptions, or provide approximate results without guarantees, CD offers a more general solution that can handle two similarity measures, two query types, and two optional additional constraints on the results. By developing bounds for multivariate correlations and using these bounds to aggressively prune the search space, CD makes the combinatorial nature of the problem tractable even for large datasets.

We extended CD to streaming scenarios through CDStream, which maintains an index to efficiently update results as new data arrives. Furthermore, we developed CDHybrid, which automatically switches between CD and CDStream based on data characteristics and arrival patterns to optimize performance. Collectively, these algorithms provide a comprehensive solution for multivariate similarity detection across both static and streaming contexts.

Our extensive evaluation on seven real-world datasets demonstrated that our algorithms consistently outperform state-of-the-art approaches, often by several orders of magnitude. CD proved to be significantly faster than both exhaustive search baselines and contemporary database management systems. When compared with specialized algorithms like CoMEtEx-

tended and CONTRa, CD maintained superior performance while providing exact results rather than approximations. For streaming data, CDStream showed remarkable efficiency for frequent updates, while CDHybrid successfully used the strengths of both algorithms to maintain optimal performance under varying conditions.

Transferable insights. The design principles underlying our algorithms offer valuable insights for fast detection of multivariate similarities in general. First, the combination of hierarchical clustering with recursive bounding showed to heavily prune the search space of possible time series combinations, leading to significantly better scaling with the size of the dataset and the complexity of the queried patterns. Second, by rewriting the similarity measure as a linear combination of pairwise similarities, we can cache intermediate results to avoid repeating the same computations when evaluating different combinations of time series.

Limitations and path forward. While our algorithms represent significant progress, several limitations remain. The current implementation supports only two similarity measures (*mPC* and *MP*), and each measure requires custom-derived bounds. However, as identified in our literature review in Sections 1.1 and 5.1, the problem of multivariate similarity search includes many custom definitions and measures in different domains, each with its own unique characteristics. To enable efficient search to all these domains, we need a more general framework that can systematically handle a broader range of similarity measures, and query definitions, while maintaining the efficiency gains demonstrated in this chapter.

In the next chapter, we build exactly that. We address these limitations by generalizing our techniques into a comprehensive framework for multivariate similarity search. This framework decomposes the methods behind CD into modular components that can be combined for efficient search of any query definition and similarity measure. By doing so, we aim to provide a versatile and efficient solution for multivariate similarity search that can be applied across various domains and applications.

Reflection on research question. This chapter makes substantial progress on our central research question by addressing the relation-level extension of similarity search. Through CD and its variants, we demonstrate that detecting multivariate relationships is not only theoretically possible but also practically feasible at scale. Where Chapters 3 and 4 showed how to handle multiple variables within individual objects, this chapter complements that work by enabling the discovery of relationships between multiple objects simultaneously. This advances us toward the ultimate goal of complete multivariate similarity search, where we can efficiently find complex patterns in rich, multivariate data.

CHAPTER 6

Towards a General Multivariate Similarity Search Framework

Across diverse scientific fields, researchers have independently developed methods to find multivariate relationships, leading to a fragmented landscape of specialized, and often incompatible, solutions. This chapter addresses this fragmentation by generalizing the principles from the previous chapter into the Similarity Detective (SD) framework – a unified, measure-agnostic system for multivariate similarity search. The framework is built around a standardized measure interface, a core query execution strategy, and a modular suite of optimizations that are automatically applied based on the formal properties of the user-defined similarity measure. By abstracting the core computational challenges, SD provides a principled and reusable solution for any data type, aiming to serve as a foundational tool for any domain requiring the discovery of high-order patterns.

Part of the contents of this chapter have previously appeared in d'Hondt et al. [79].

6.1 Introduction

As mentioned in Chapter 2, multivariate similarity search represents a common problem across disciplines. However, despite its widespread relevance, the field is fragmented; each scientific or industrial domain has developed its own variant of the problem definition in isolation, with customized distance or similarity measures, query types, and result constraints suited to their specific needs [9–11, 119, 150, 183]. Consequently, there is a lack of shared knowledge and understanding of the problem and its solutions, leading to many specialized systems that are often incompatible with one another [187]. This is unfortunate, as the core computational challenge – efficiently navigating a combinatorial search space to find the most similar combinations of time series – is inherently the same across these variations,

meaning that the same optimizations could be applied to different similarity measures or query types with little or no modification. From a theoretical perspective, the lack of a unified definition of multivariate similarities also obscures the underlying principles that govern the problem, making it difficult compare and contrast different approaches.

Domain scientists across these disciplines would benefit substantially from a generic solution capable of accommodating different variants of the problem, allowing them to focus on their specific requirements without concern for underlying implementation details. More specifically, a formal, measure-agnostic problem definition would permit reuse of optimizations and code bases across different measure families. Similar to how relational algebra enabled the explosion of database technologies, such a formal theoretical basis could revolutionize multivariate similarity search by allowing ad-hoc, domain-specific optimizations while maintaining correctness guarantees. Furthermore, it would centralize research efforts on this topic, enabling more efficient use of resources.

In this chapter, we take a significant step towards realizing such a system by proposing a measure-agnostic framework for multivariate similarity search. This framework generalizes the principles underlying the efficient search strategies used for answering top-k and threshold queries under Pearson and Multipole correlations in the previous chapter, extending them to accommodate arbitrary aggregation and similarity measures. By identifying the fundamental properties of similarity measures and query types that enable efficient pruning and search, we establish a set of rules that guide the selection and application of appropriate optimization strategies based on the characteristics of a given problem instance. To distinguish this framework from the previously introduced CD algorithm, we refer to it as **Similarity Detective (SD)**, reflecting its expanded scope to handle any similarity or distance measure.

This framework is implemented as a software library where users can specify their multivariate similarity search problem through a defined interface. The library then automatically applies relevant, provably correct optimizations based on the problem's formal properties, rather than relying on heuristics. The key strength of our approach lies in this solid theoretical foundation, which ensures the correctness of the applied optimizations for diverse problem configurations. Through examples (Section 6.4.7) and a case study in Chapter 7, we demonstrate that this library can efficiently solve a wide range of problems. More so, it often achieves performance competitive with specialized, hand-tuned algorithms from respective domains. Note that while our current work focuses on the theoretical foundations, automated optimizations, and library implementation, we also discuss in Section 6.6 how this library could evolve into a more mature and user-friendly system where users can specify queries in a declarative language (e.g., SQL).

Notice that by generalizing CD to a system that supports any multivariate similarity measure (not just correlations), we also extend our problem scope from high-dimensional vectors to any data type over which a meaningful similarity or distance measure can be defined, such as sets, strings, or even more complex structures like graphs (i.e., *data objects* as per the definition in Chapter 2). Therefore, in this chapter we will refer to the input data as *objects*, rather than time series or vectors unless the context implies otherwise.

The remainder of this chapter is structured as follows. In Section 6.2, we motivate our work by highlighting the different works across domains that can be considered as multivariate similarity search problems, illustrating the fragmented nature of the field. In Section 6.3, we analyze the commonalities and differences between the specialized solutions to these problems, identifying the key properties and methods that can be used to generalize the problem definition and different solution strategies. Section 6.4 presents our framework and demonstrates how it can efficiently execute queries across a wide range of problems. In Section 6.5, we evaluate the framework's performance and versatility through experiments. Section 6.6 outlines our vision for extending the library into a full system with a declarative query interface. Finally, in Section 6.7, we summarize our findings and discuss the implications of our work.

6.2 Motivation

Multivariate similarity search exists across numerous scientific domains, though solutions are typically developed with domain-specific terminology and methodologies. To build an effective generic framework, we need to understand the full landscape of approaches and identify common patterns. In this section, we systematically discuss and categorize existing works that investigate multivariate similarity search problems, highlighting their similarities and differences. The works are categorized along several dimensions: their application domain, problem definition, similarity or distance measure employed, computational methods used for efficiency, and any specific constraints imposed on the results. By recognizing underlying commonalities, we can design a framework capable of addressing various domain-specific requirements while leveraging shared optimization principles. Table 6.1 provides an overview of these approaches, which we discuss in detail below.

6.2.1 Databases

In database research, multivariate similarity search manifests as the discovery of relationships between columns (attributes) in database tables. These problems can be naturally framed as multivariate similarity queries where the objects are database columns and the goal is to find sets meeting specific similarity criteria.

For example, Nguyen et al. [183] address what they term "detection of column correlations" using Total Correlation [234] to measure shared information across columns. From our perspective, this is a top-k query for sets of columns with maximum Total Correlation. However, due to the nature of their method, they also introduce an additional constraint to their problem definition: the pairwise correlations between the columns in a set must be high. This is because their algorithm builds a correlation graph based on Mutual Information – the pairwise equivalent of Total Correlation – and search for quasi-cliques, effectively restricting results to only column sets with high pairwise correlations.

Similarly, works on soft functional dependencies [14], composite functional dependencies [112], inclusion dependencies [77], composite keys [220], and index recommendation [135] can also be reformulated as multivariate similarity queries. Each seeks sets of columns maximizing different similarity measures: Jensen-Shannon divergence [14], set containment metrics [77, 112, 220], or set similarity measures similar to Jaccard [135].

Table 6.1: Overview of different multivariate similarity search approaches across domains.

Domain	Problem	Citation	(Dis-)Similarity Measure	Methods	Constraints
Databases	Subspace search	[183]	Total correlation	Clique enumeration	High pairwise
	Soft FDs	[14]	JS divergence	Clustering	
	Composite FDs	[112]	Functional dependency	Apriori, Graph coloring	
	Inclusion dependencies	[77]	Set containment	Apriori, Joins, Sketches	Maximality
	Composite keys	[220]	Functional dependency	Apriori	
	Index recommendation	[135]	Set similarity	Clustering, LP	
Signal Processing	Sparse approximation	[52]	Euclidean distance	LP	Irreducibility
Data Mining/ ML	Subset regression	[69]	Euclidean distance	Branch-and-bound	
	Subspace search	[26]	Euclidean distance	Locality sensitive hashing	
	Subspace search	[158]	Euclidean distance	Locality sensitive hashing	
	Subspace Search	[239]	Euclidean distance	Locality sensitive hashing	
	Outlier detection	[130]	Subspace contrast	Ordered exhaustive search	High conditional prob.
	Pattern discovery	[111]	Scaled joint entropy	Apriori	
	Feature selection	[120]	Interaction information	Exhaustive search	
	Feature selection	[253]	C-contribution	Index-based search	
	Freq. itemset mining	[137]	Joint entropy	Apriori	
Biology/ Medicine	Epistasis detection	[43, 149] [172]	Statistical interaction	Exhaustive search	
Finance	Multi pairs trading	[48, 199]	Pearson correlation	Exhaustive search	
	Test of independence	[207]	Correntropy	Exhaustive search	
Domain-agnostic	Tripole mining	[9]	Tripole	Bounding	Minimum jump
	Multipole mining	[10]	Multipole	Clique enumeration	Minimum jump
	High-order correlations	[251]	Total correlation	Apriori, Bounding	Non-redundancy
	Network classification	[215]	Product of values	Exhaustive search	Coherence

Unlike Nguyen's work, these approaches do utilize Apriori-style methods since their similarity measures generally possess (anti-)monotonicity properties. Particularly, column dependencies like functional dependencies and inclusion dependencies exhibit anti-monotonicity properties: if a certain combination of columns fails to satisfy the dependency, none of its supersets will satisfy it either. This anti-monotonicity is not an additional constraint like irreducibility, but an inherent property of the similarity measure – methods that do not exploit this property will still find the same results, but with potentially higher computational cost. These works also employ various other optimization techniques – graph methods [112], clustering [14], and linear programming (LP) [135] – but the fundamental query remains the same: find column sets with maximum similarity under the chosen measure.

6.2.2 Signal Processing

In signal processing, a well-studied problem related to multivariate similarity search is *sparse approximation*, where the goal is to represent a signal as a linear combination of a small number of basis functions from a larger dictionary. A common solution is Basis Pursuit [52, 237], which reformulates the problem as a linear programming task, minimizing both the Euclidean distance between the signal and the subspace spanned by the chosen basis functions, and the ℓ_1-norm of the coefficients. Linear programming techniques are then employed to efficiently find the optimal solution. This task can be viewed as a multivariate similarity search problem where the goal is to find a set of basis functions (i.e., vectors) aggregated in such a way that the resulting vector is as close as possible to the query signal in Euclidean

distance. The dataset in this case consists of the basis functions, and the query is a signal vector. Furthermore, in terms of constraints, one can argue that minimizing the ℓ_1-norm of the coefficients to promote sparsity is similar to applying an irreducibility constraint, as it forces the algorithm to prioritize sets of minimal size. Sparse approximation is further discussed as an application of our framework in Chapter 7.

6.2.3 Data Mining and Machine Learning

In data mining and machine learning, multivariate similarity search appears in several distinct problem settings. One setting is *nearest subspace search*, addressed by works like Basri et al. [26], Lu et al. [158] and Xu et al. [239], which focus on finding the subspace (e.g., line, plane, hyperplane, etc.) in a dataset closest to a query point or subspace, often using Euclidean or angular distance. Unlike sparse approximation where the closest vector to the query can be constructed from all database objects (i.e., the basis functions) – effectively forming one global subspace – here the data objects are subspaces themselves composed of a fixed number of basis vectors. In other words, subspace search can be seen as a constrained version of sparse approximation, where the query can only be reconstructed from a set of predefined subspaces. Due to scalability challenges in high dimensions, methods for subspace search typically employ approximate techniques like Locality Sensitive Hashing (LSH).

Another application of multivariate similarity search in data mining is *feature selection for interaction*, where the goal objective is to find feature sets in a dataset that interact meaningfully with a target variable to improve the predictive performance of models. Works by Jakulin et al. [120] and Zhao et al. [253] frame this as a top-k query using measures like interaction information or C-contribution to quantify the strength of feature interactions. The database objects in this case are individual features, and the query includes the target variable. No additional constraints are imposed, though efficient index structures are sometimes used to speed up the search with the cost of potentially missing results.

Furthermore, *Subset regression*, as studied by Das et al. [69], can be reformulated as a nearest neighbor query where the database consists of features, and the goal is to find the feature set whose spanned subspace is closest to a query vector (the target variable) under Euclidean distance. Like sparse approximation, this is effectively a subspace construction problem, though typically solved through exact methods like branch-and-bound.

In *outlier detection*, Keller et al. [130] search for subspaces where outliers exhibit high contrast relative to their neighbors. Particularly, given a dataset of points, they aim to identify linear combinations of points (i.e., subspaces) with high "conditional dependence" and a high "contrast", with both measures defined as functions of the conditional, marginal, and joint probability density functions of the points in the subspace. Through these measures, the authors quantify how well a subspace separates outliers from normal points, with high contrast indicating that the outliers are distinctly different from the normal points. Considering the contrast as a similarity measure and the high conditional dependence as a constraint, this can be considered a top-k multivariate similarity search problem.

Finally, *pattern discovery* by Heikinhemo et al. [111] (low-entropy sets) and Knobbe et al. [137] (maximally informative k-itemsets) represent top-k queries searching for column sets with minimum joint entropy. The goal of these works is to find sets of items (i.e., columns) in a dataset that exhibit low joint entropy, being an information-theoretical measure with similar properties to Total Correlation. The monotonicity of joint entropy enables efficient Apriori-style processing without additional constraints.

6.2.4 Biology, Genomics, and Medicine

In biology, genomics, and medicine, a key application related to multivariate similarity is epistasis detection. This involves the identification of interactions between multiple genes that influence a particular trait or disease risk [43, 149, 172]. The underlying similarity measure is typically an implicit statistical interaction measure, quantifying how the combined effect of a set of genes deviates from the sum of their individual effects. Due to the complexity of biological interactions and often the lack of clear monotonicity properties, exhaustive search over combinations of genes is a common (albeit computationally intensive) approach to identify these interactions. The focus of these works is usually qualitative, researching appropriate measures and constraints to identify meaningful results, rather than optimizing for computational speed.

6.2.5 Finance

In finance, multivariate similarity concepts are applied in areas like *multi-pairs trading*. Multi-pairs trading strategies [48, 199] extend traditional pairs trading by searching for combinations of assets whose price movements exhibit strong correlation (often Pearson correlation) or cointegration, aiming to identify arbitrage opportunities. In this context, the problem becomes a top-k query to find sets of assets with maximum correlation (Pearson) or cointegration, with the database objects being the financial instruments.

6.2.6 Domain-agnostic Approaches

Besides the domain-specific approaches above, several works propose generic solutions for multivariate similarity search that are applicable across domains. These approaches are particularly interesting as they explicitly frame their problems as similarity search tasks with clearly defined query types and constraints.

For instance, the CONTRa and COMEtExtended algorithms for Tripole [9] and Multipole [10] mining discussed in Chapter 5 demonstrate their domain-agnosticity through successful applications across diverse domains. The CONTRa algorithm has been effectively applied to both climate data, identifying relationships between different atmospheric variables, and fMRI data, discovering coordinated activity patterns in brain regions. Similarly, the COMEtExtended algorithm extends these applications to traffic data, identifying groups of road segments with correlated congestion patterns.

Zhang et al. [251], also discussed in Chapter 5, showcase the versatility of their approach through its application to gene expression data. While originally developed for binary data,

their method naturally extends to any domain where variables can be binarized, making it broadly applicable across fields where discrete relationships are of interest.

A different perspective on multivariate similarity search is provided by Santoro et al. [215] in their study of network dynamics. Rather than measuring similarity over an entire time series, they compute the product of node values (called the *discord* of the nodes) at each timestep t, and identify triplets that are *coherent*. Coherent triplets are defined as sets of three nodes where (a) the discord exceeds all pairwise products between the nodes, and (b) all three values in the triplet share the same sign (i.e., all positive or all negative). Then, they define the *hyper-coherence* of the network at time t as the fraction of coherent triplets among all possible triplets in the network. Through the coherence constraint, they effectively perform a constrained range query at each timestep, and compute the hyper-coherence based on the number of retrieved results. Through experiments, they demonstrate with data from brain functional activity (fMRI), financial markets (stock prices), and epidemics (disease spreak in the US) that the hyper-coherence metric can help classify the underlying ordered regime of the network, such as whether it is in a synchronized or chaotic state. As such, it provides insights into the temporal evolution of the dynamics of a network, which can be relevant for understanding certain events or phenomena in a system.

6.3 Analysis of Existing Solutions

The discussion in the previous section reveals that diverse domain-specific problems can indeed be reformulated as multivariate similarity search queries. By mapping these problems to our context, several recurring patterns and challenges emerge across domains. First, the query types consistently fall into three categories: top-k queries (as seen in database column correlation and feature selection), range queries (in Tripole mining and network classification), and nearest neighbor queries (in sparse approximation and subspace search). Second, these measures, while all designed to capture higher-order relationships, exhibit significant diversity – ranging from geometric distances to information-theoretic measures to domain-specific functions. This variety stems from the distinct requirements of each domain. For example, Total Correlation in databases and statistical interaction in biological systems each quantify different types of higher-order relationships fundamental to their respective fields.

Computational Methods A critical insight from our analysis is that computational methods across domains are fundamentally determined by the mathematical properties of the similarity measure, not the application domain itself. This pattern is consistent: when measures exhibit monotonicity properties (as seen with functional dependencies in [112], inclusion dependencies in [77], and joint entropy in [137]), Apriori-style algorithms consistently emerge as the optimal choice. For geometric distances, particularly in high-dimensional spaces, LSH and approximate methods dominate the landscape, as demonstrated by multiple subspace search approaches [26, 158, 239]. When the measure can be expressed as an optimization problem (exemplified by sparse approximation in [52]), linear programming techniques naturally become the method of choice. For measures that can be efficiently bounded, like Tripole [9] and Total Correlation [251], approaches combining bounding with pruning strategies are typically employed. For measures lacking these properties, like statistical interaction in epistasis detection [43, 172] or the discord measure in network classification [215], exhaustive search often remains the primary approach despite its computational cost. This

consistency strongly suggests that optimization strategies should be systematically selected based on measure properties rather than reinvented for each domain.

Role of Constraints Furthermore, constraints appear across domains serving two distinct purposes: making the search computationally tractable, e.g., by allowing the use of apriori filtering, and encoding certain domain-specific requirements. While many constraints, particularly in efficiency-focused works, are primarily computational artifacts, they remain a crucial component of the problem definition alongside the similarity measure. For instance, the coherence constraint in network classification [215] and the irreducibility constraint in sparse approximation [52] directly encode domain knowledge about what constitutes a meaningful result. This dual nature of constraints – as both computational aids and semantic filters – suggests that a unified framework must provide sufficient flexibility in constraint definition while still leveraging them for computational efficiency where possible.

These patterns highlight the potential impact of a unified multivariate similarity search framework: by abstracting away domain-specific terminology and focusing on the underlying similarity search operation, we can systematically apply optimization techniques across domains while preserving the flexibility to incorporate domain-specific requirements through constraints and similarity measures when needed. This is precisely the goal of our SD framework, which we present in the following section.

6.4 The Similarity Detective (SD) Framework

In this section, we present Similarity Detective, our unified framework for multivariate similarity search. Building upon the insights from the related work (Section 6.2), where we observed common patterns across diverse domain-specific problems, Similarity Detective aims to provide a generic solution capable of accommodating different variants of the problem. This is achieved through a well-defined similarity measure interface, a modular query execution pipeline, and a suite of twelve optimizations that can be automatically applied based on the properties of the similarity measure and query type, in order to increase the efficiency of query execution.

We begin by categorizing multivariate similarity measures and discussing the properties that form the basis of our optimizations for query execution (Section 6.4.1 & Section 6.4.2). Next, we outline the query execution workflow, detailing the steps involved to efficiently prune and traverse the search space and compute the results (Section 6.4.3). Following this, we present a theoretical framework for bounding the similarity between sets of clusters, which is a core component of efficient query execution (Section 6.4.4). Then, we introduce a modular suite of 12 distinct optimizations (Section 6.4.5) that can be applied to the query execution pipeline, enhancing its efficiency based on the properties of the similarity measure and query type. After that, we describe the similarity measure interface, which serves as a contract for implementing similarity measures within our framework (Section 6.4.6). Finally, we provide three examples of multivariate similarity measures and how they can be implemented within our framework (Section 6.4.7), and discuss how the framework can be extended to handle streaming data (Section 6.4.8).

Notes on terminology. We present the framework mainly in the context of *similarity measures*. However, it also applies to *dissimilarity measures* (i.e., distance functions); the only difference is that query execution now looks for combinations *under* a threshold rather than combinations *over* a threshold, or the *bottom-k* instead of the *top-k*. For the sake of generality, we will also extend the notion of *correlation patterns* of Chapter 5 to *query patterns* $sim(pl, pr), dist(pl, pr)$ in the following sections, which will refer to a combination of a similarity or distance measure, and the maximum allowed cardinalities of the vector sets in the combination. Furthermore, recall that we focus on general *data objects* in this chapter, rather than specifically time series or vectors.

6.4.1 Types of Multivariate Similarity Measures

Our framework supports two primary categories of multivariate similarity measures, each with distinct characteristics that influence how they are processed:

Bivariate Functions over Aggregated Data (Two-sided) Given two sets of objects X and Y, these measures compute a similarity score based on the aggregation of the objects in each set. Formally, it is defined as:

$$sim(X, Y) = f(\text{agg}(X), \text{agg}(Y)) \tag{6.1}$$

where $\text{agg}(X)$ and $\text{agg}(Y)$ are aggregations of the objects in X and Y, respectively, and f is a function that operates on the aggregated objects. In the case of vectors, the aggregation can be a simple sum, average, or any other function that combines the vectors in X and Y into a single vector. For simplicity, we will refer to these measures as **two-sided** measures, as they operate on two sets of objects. A prominent example of this type of measure is the *Multivariate Pearson correlation mPC* from Chapter 5, which computes the Pearson correlation element-wise averages of the vectors in X and Y.

Multivariate Functions over Raw Data (One-sided) Unlike two-sided measures, one-sided measures operate directly on the raw, unaggregated data points. They inherently consider all data points simultaneously without requiring intermediate aggregation. Formally, they are defined as:

$$sim(X) = f(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) \tag{6.2}$$

where $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ is a set of objects and f is a function that operates on the objects in X. We refer to these measures as **one-sided** measures, as they operate on a single set of objects. An example is the *Multipole* correlation measure (cf. Chapter 5), which measures the linear dependence of the vectors in X.

Throughout the remainder of this chapter, we adopt the notation $sim(X, Y)$ to represent both types of similarity measures. In the case of one-sided measures, we consider Y to be the empty set $\emptyset$, such that $sim(X, Y)$ naturally reduces to $sim(X)$. This notational convention enables a unified treatment of both measure types without loss of generality.

6.4.2 Properties of Multivariate Similarity Measures

The applicability of certain optimization strategies in our framework depends on the mathematical properties of the distance or similarity measure being used. We classify measures

based on the following key properties:

- **Monotonicity**: A multivariate similarity measure can be monotonically increasing or decreasing (or neither) with respect to the number of objects in the input sets. When a measure is *monotonically increasing*, adding more objects to the input sets will never decrease the similarity score. Conversely, if it is *monotonically decreasing*, adding more objects will never increase the score. This property is crucial for optimizations such as Apriori filtering or result expansion, discussed in Section 6.4.5. Formally, a measure is monotonically increasing if:

$$f(X, Y) \leq f(X', Y) \quad \text{if } X \subseteq X' \tag{6.3}$$

  and monotonically decreasing if:

$$f(X, Y) \geq f(X', Y) \quad \text{if } X \subseteq X' \tag{6.4}$$

  For example, the *Multipole* measure is monotonically increasing, as the linear dependence between vectors can only increase with the addition of more vectors.

- **Distributivity**: A multivariate similarity measure is distributive if the multivariate similarity score can be expressed as a function of pairwise similarities, e.g., $f(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) = f(\boldsymbol{a}, \boldsymbol{b}) + f(\boldsymbol{a}, \boldsymbol{c}) + f(\boldsymbol{b}, \boldsymbol{c})$. This property is particularly useful for decomposing multivariate computations into simpler pairwise calculations. As seen in Chapter 5, *Pearson correlation* is an example of a distributive measure when applied to z-normalized vectors, since the correlation between aggregate vectors can be expressed as the sum of their pairwise correlations.

- **Soft-Distributivity**: This property is a weaker form of distributivity, where the measure can be expressed as a function of univariate and pairwise *statistics* (e.g., means, norms, angles, dot products) rather than just pairwise similarities. Soft-distributivity is a crucial property for deriving empirical bounds on the measure, further discussed in Section 6.4.4. For example, the *Euclidean distance* over element-wise sums of two sets of vectors X, Y can be expressed as a function of dot products between the vectors in X and Y (shown in Section 6.4.4), making it soft-distributive.

- **Satisfying triangle inequality**: A measure satisfies the triangle inequality if, for any three vectors $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$, the following holds: $f(\boldsymbol{a}, \boldsymbol{b}) \leq f(\boldsymbol{a}, \boldsymbol{c}) + f(\boldsymbol{b}, \boldsymbol{c})$. This property is essential for establishing theoretical bounds on the similarity or distance scores. For example, the *Euclidean distance* satisfies the triangle inequality, which allows us to derive theoretical bounds on the Euclidean distance between two aggregate vectors, while the *Jaccard distance* does not.

- **Operating over vector spaces**: When a measure operates over a vector space, vectors in that space can be added together and/or scaled, which means that one can define a *norm* over the space. Norms are a function $\|\cdot\|$ that satisfy (1) non negativity $\|\boldsymbol{x}\| \geq 0$, (2) absolute homogeneity $\|\alpha \cdot \boldsymbol{x}\| = |\alpha| \cdot \|\boldsymbol{x}\|$, and (3) triangle inequality $\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$. This property is relevant for the derivation of theoretical bounds. For instance, the

Jaccard distance, which operates on binary vectors, does not satisfy this property, as the sum of two binary vectors is not necessarily a binary vector. In contrast, the *Euclidean distance* operates in a vector space.

- **Permutation invariance**: When a measure is permutation invariant, the order of the objects in the input sets does not affect the similarity score, i.e., $f(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) = f(\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{a})$. For two-sided measures, this also applies to which object set is designated as 'left' and which as 'right', i.e., $f(X, Y) = f(Y, X)$, and $f(X, Y) = f(X', Y')$ if X and X' are permutations of each other. This property allows pruning of redundant combinations during the search process by only considering unique combinations.

- **Preprocessing requirements**: Some measures assume specific data preprocessing steps are applied to the input data before computing the similarity score. For example, z-normalization is required for *mPC*, while quantization may be necessary for information-theoretic measures on continuous data. The framework accommodates these preprocessing needs by allowing users to specify the required preprocessing steps as part of the measure implementation.

By formally capturing these properties, the framework can automatically select and apply appropriate optimization strategies.

6.4.3 Query Execution Workflow

This section details the algorithmic workflow for executing multivariate similarity search queries within the SD framework. The core strategy of the query execution process is to efficiently prune the combinatorial search space by examining candidate n-ary combinations in a level-wise manner, where each level corresponds to an increasing combination cardinality (e.g., pairs at level 1, triplets at level 2, and so on). This iterative exploration continues until the query's objectives (such as identifying k results for a progressive query) are met, or a predefined maximum combination cardinality is reached.

A key component of this strategy is the initial organization of the dataset via hierarchical clustering. This structure, combined with recursive bounding techniques applied to combinations of clusters, facilitates the aggressive pruning of irrelevant parts of the search space. While several concepts such as hierarchical clustering and recursive bounding were introduced in the context of CD (Chapter 5), we reiterate them here for clarity and completeness, as part of a more generalized workflow. Figure 6.1 provides a high-level schematic of this process.

It is important to recognize that this workflow serves as a foundational framework; various optimizations, which will be discussed in detail in Section 6.4.5, can be integrated at different stages to enhance performance. The purpose of this section is to provide a clear overview that allows for the subsequent positioning and understanding of these optimizations.

Running example: To provide more context to each step in the workflow, we will work through the query execution of an example top-k query, *without the use of any optimiza-*

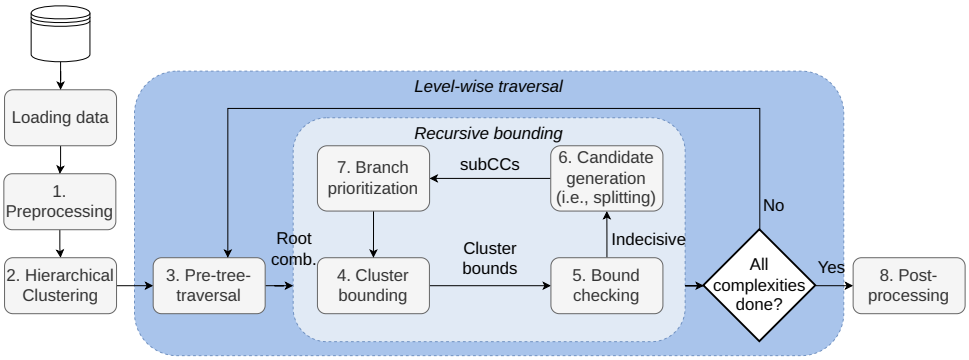


Figure 6.1: High-level overview of the query execution pipeline.

tions. Namely, consider the following scenario: we have a dataset $D = \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}, \boldsymbol{f}\}$ with six vectors, and we want to find the top-3 combinations of at most three vectors (i.e., triplets) with the highest *MP* similarity score. As an additional constraint, we only want to include combinations with *unique* vectors, meaning that no vector can appear more than once in a combination (e.g., $\{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{a}\}$ is not allowed).

Step 1: Preprocessing. Any data preprocessing required by the specific distance/similarity measure, as defined in the measure implementation, is performed here. Optimizations like dimensionality reduction (optimization O1 in Section 6.4.5) or pairwise matrix precomputation (O2) may also be applied at this stage, depending on the measure's properties.

Example: For the *MP* measure, we need to z-normalize the vectors in D before computing the similarity scores (cf. Chapter 5). We therefore perform this transformation on the whole dataset D to obtain the z-normalized vectors $\{\hat{\boldsymbol{a}}, \hat{\boldsymbol{b}}, \ldots, \hat{\boldsymbol{f}}\}$. Additionally, as standard for top-k queries, we set the initial similarity threshold τ to 0. Note that this is not the minimum similarity score for *MP* (which is -1), but rather a natural starting point for the search as it indicates no correlation between the vectors.

Step 2: Hierarchical Clustering. The dataset is clustered hierarchically using the K-Means^{++} algorithm described in Chapter 5. This hierarchical structure enables bounding over groups of objects (clusters), to efficiently traverse the search space of a given cardinality (i.e., level). The default distance measure used for clustering is Euclidean distance for continuous data, and Hamming distance for categorical and binary data. However, the framework allows for customization of the clustering distance measure via the measure interface to best suit the query's similarity measure (O3).

Example: We apply K-Means^{++} clustering to the z-normalized vectors $\{\hat{\boldsymbol{a}}, \hat{\boldsymbol{b}}, \ldots, \hat{\boldsymbol{f}}\}$, resulting in the hierarchy of clusters visualized in Figure 6.2.

Step 3-7: Complexity Climb and Recursive Bounding. The framework then explores combinations of increasing complexity (cardinality) in a level-wise manner (e.g., level 1: pairs, level 2: triplets). The exploration within each level involves a recursive bounding process, containing the steps outlined below.

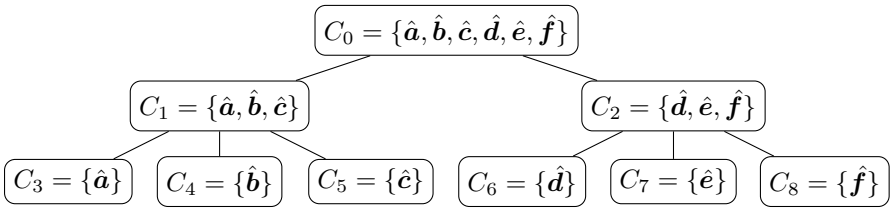


Figure 6.2: Example of hierarchical clustering result showing the cluster hierarchy from root (C_0) to individual objects.

Step 3: Pre-traversal. In this step, initial root candidate cluster combinations for the upcoming complexity level are generated. These are the initial cluster combinations that will be explored recursively in the next steps (i.e., they are the root of the search tree for the current complexity level). In lack of any optimizations, this is simply done by forming the query pattern of that complexity level (i.e., a pair, triplet, etc.) from the root cluster C_0 in the cluster hierarchy created in Step 2.

This generation process can be optimized though in several ways to reduce the search space early. For instance, with anti-monotonic measures, rather than initializing the search through the root combinations, one can generate candidates solely from the current result set as – by anti-monotonicity – supersets of non-answers are guaranteed to be non-answers as well (O6). Furthermore, for top-k queries, promising candidate combinations (e.g., supersets of high-scoring combinations from the previous level) can be generated here before generating and exploring the root candidates, as these are likely to yield high scores and can be used to quickly raise the running similarity threshold τ (O5).

Example: For our example, we generate at this step the root candidate combination (C_0, C_0) at the first complexity level, and the combination (C_0, C_0, C_0) at the second complexity level.

Step 4-7: Recursive Bounding Loop. The core of the search at a given complexity level is a recursive bounding loop, where the framework iteratively explores and processes combinations of clusters. The loop initiates with the root candidate cluster combinations generated in the pre-traversal step, and continues until all cluster combinations are either processed, pruned, or refined to the level of individual objects if necessary. The framework then proceeds to the next complexity level. Each recursion iteration involves the steps outlined below.

Step 4: Cluster Bounding. For a given combination of clusters, compute the lower and upper bounds on the similarity values achievable by any object combination drawn from these clusters (also known as *materizalizations*). The specific bounding method (empirical, theoretical) is determined by the measure's properties as explained in Section 6.4.4. At this step, bound-related optimizations are applied such as centroid and radius caching (07) and cluster pair caching (08).

Example: After initialization of the root candidate combination (C_0, C_0), we compute the bounds for this combination, which will yield a lower bound of -0.3 and an upper bound of 1.0 for the *MP* similarity measure, indicating that any vector combination in the cartesian product of from C_0 and C_0 will have a *MP* value between these bounds.

Step 5: Bound Checking. We compare the computed bounds against the query threshold (or the k-th best score found so far for top-k queries). If the entire range defined by the bounds is above the threshold for similarity measures (resp. below for distance measures), the combination can be definitively added to the result set. If the entire range is below the threshold for similarity measures (resp. above for distance measures), the combination and all its potential sub-combinations can be pruned. If the bounds are indecisive (i.e., they span the threshold), the combination is marked as inconclusive and will be further processed in the next step. In terms of optimizations, bound discounting (09) is applied here to increase the probability of finding decisive bounds. This step also provides an opportunity to check for additional constraints like minimum jump and irreducibility.

Example: For our example, the bounds of the root candidate combination (C_0, C_0) are indecisive, as they span the threshold of 0. If this were not the case and the bounds were decisive positive, the combination would be added to the result set.

Step 6: Candidate Generation (Splitting Clusters). This step generates more refined candidates from a cluster combination with indecisive bounds. This is done by splitting the largest cluster in the combination into their constituent sub-clusters from the hierarchy. This generates new, smaller candidate combinations that are then passed to the next iteration of the recursive bounding loop for further processing. Various optimizations are also applied at this stage to minimize the number of candidate combinations, including canonical candidate generation (O10) for measures that are permutation-invariant, and duplication checking (O11) when the query requires unique objects in the result set.

Example: For our example, the root candidate combination (C_0, C_0) is split into the combinations (C_1, C_1), (C_1, C_2), (C_2, C_1), and (C_2, C_2).

Step 7: Branch Prioritization. For top-k queries, branch prioritization (O12) determines the order in which inconclusive branches are explored, focusing on those most likely to yield high-scoring results first. This is implemented using a priority queue and involves depth-first traversal within prioritized branches, further discussed in Section 6.4.5.

Example: As no optimizations are applied in our example, we simply pass the next candidate combination (C_1, C_1) on to the next iteration of the recursive bounding loop.

Step 8: Post-processing. Once the search across all relevant complexity levels is complete, several post-processing steps are performed:

- *Result Unpacking*: Cluster combinations in the result set are unpacked into the actual object combinations they represent.
- *Final Constraint Checks*: A final verification ensures all reported combinations satisfy any additional constraints defined in the query.
- *False Positive Filtering*: If approximate techniques like dimensionality reduction were used, this step verifies the results using the original data to remove any false positives introduced by the approximation.
- *Formatting*: Results are formatted for user readability.

Example: For our example, the result set will contain the cluster combinations (C_3, C_5, C_6), (C_3, C_5, C_7), and (C_3, C_5, C_8), which are unpacked to the vector combinations $\{\boldsymbol{a}, \boldsymbol{c}, \boldsymbol{d}\}$, $\{\boldsymbol{a}, \boldsymbol{c}, \boldsymbol{e}\}$, and $\{\boldsymbol{a}, \boldsymbol{c}, \boldsymbol{f}\}$, respectively. For reader convenience, the result set is then formatted to show the top-3 combinations with their corresponding *MP* scores, which are 0.8, 0.75, and 0.7 for the three combinations, respectively, and (if provided) any metadata associated with the vectors (e.g., labels, timestamps, etc.).

6.4.4 Bound Derivation

The core of our recursive bounding approach involves determining whether object combinations belong in the result set without exhaustively evaluating each possibility. We accomplish this by replacing individual objects in similarity measures (e.g., $f(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$) with clusters of objects (e.g., $f(C_a, C_b, C_c)$, where $\boldsymbol{a} \in C_a$, $\boldsymbol{b} \in C_b$, $\boldsymbol{c} \in C_c$), and then computing bounds on the possible similarity scores for any object combination drawn from these clusters. The efficiency of this technique depends on both the ability to derive bounds for a given similarity measure, and the tightness of those bounds.

We present two types of bounds applicable to different similarity measures, each with distinct trade-offs between computational cost and bound tightness. Below, we detail these approaches and specify the conditions under which each can be effectively applied.

Empirical Bounds

Recall that a multivariate similarity measure is **soft-distributive** when it can be expressed as a function of univariate and pairwise statistics. This property enables us to derive empirical bounds by computing the minimum and maximum values of these statistics between clusters.

For example, consider the *Multivariate Euclidean distance mED*, which measures the Euclidean distance between the element-wise sums of two vector sets X and Y, expressed in terms of dot products $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$:

$$
\begin{aligned}
mED(X, Y) &= \left\| \left(\sum_{\boldsymbol{x} \in X} \boldsymbol{x}\right) - \left(\sum_{\boldsymbol{y} \in Y} \boldsymbol{y}\right) \right\|_2 \\
&= \sqrt{\left\| \sum_{\boldsymbol{x} \in X} \boldsymbol{x} \right\|_2^2 + \left\| \sum_{\boldsymbol{y} \in Y} \boldsymbol{y} \right\|_2^2 - 2 \sum_{\boldsymbol{x} \in X} \sum_{\boldsymbol{y} \in Y} \langle \boldsymbol{x}, \boldsymbol{y} \rangle} \\
&= \sqrt{\sum_{\boldsymbol{x} \in X} \sum_{\boldsymbol{x}' \in X} \langle \boldsymbol{x}, \boldsymbol{x}' \rangle + \sum_{\boldsymbol{y} \in Y} \sum_{\boldsymbol{y}' \in Y} \langle \boldsymbol{y}, \boldsymbol{y}' \rangle - 2 \sum_{\boldsymbol{x} \in X} \sum_{\boldsymbol{y} \in Y} \langle \boldsymbol{x}, \boldsymbol{y} \rangle}
\end{aligned}
$$

By replacing sets of vectors X, Y with sets of clusters S_X, S_Y, we can bound the multivariate similarity between any combination of vectors from these clusters through the minimum and

maximum values of dot products between vectors in the respective clusters:

$$mED(S_X, S_Y) \geq \sqrt{\sum_{C_X \in S_X} \langle C_X, C_X \rangle_{\min} + \sum_{C_Y \in S_Y} \langle C_Y, C_Y \rangle_{\min} - 2 \sum_{C_X \in S_X} \sum_{C_Y \in S_Y} \langle C_X, C_Y \rangle^{+}_{\max}} \tag{6.5}$$

$$mED(S_X, S_Y) \leq \sqrt{\sum_{C_X \in S_X} \langle C_X, C_X \rangle_{\max} + \sum_{C_Y \in S_Y} \langle C_Y, C_Y \rangle_{\max} - 2 \sum_{C_X \in S_X} \sum_{C_Y \in S_Y} \langle C_X, C_Y \rangle^{+}_{\min}} \tag{6.6}$$

with

$$\begin{aligned}
\langle C_X, C_Y \rangle^{+}_{\min} &= \min(|\langle C_X, C_Y \rangle_{\min}|, |\langle C_X, C_Y \rangle_{\max}|) \\
\langle C_X, C_Y \rangle^{+}_{\max} &= \max(|\langle C_X, C_Y \rangle_{\min}|, |\langle C_X, C_Y \rangle_{\max}|) \\
\langle C_X, C_Y \rangle_{\min} &= \min_{\boldsymbol{x} \in C_X, \boldsymbol{y} \in C_Y} \langle \boldsymbol{x}, \boldsymbol{y} \rangle \\
\langle C_X, C_Y \rangle_{\max} &= \max_{\boldsymbol{x} \in C_X, \boldsymbol{y} \in C_Y} \langle \boldsymbol{x}, \boldsymbol{y} \rangle
\end{aligned}$$

This bounding method is computationally efficient, as any pair of clusters will participate in potentially many higher-order combinations (i.e., triplets, quadruplets, etc.), meaning that when cached, the minimum and maximum values of pairwise statistics can be reused across multiple combinations.

For instance, given a query pattern $mED(2, 2)$ and the cluster hierarchy shown in Figure 6.2, the minimum and maximum dot products between clusters C_1 and C_2 can be reused for all combinations including C_1 and C_2 at opposite sides of the pattern, such as:

$$\begin{array}{lll}
(\{C_1\}, \{C_2\}), & (\{C_0, C_1\}, \{C_2\}), & (\{C_1, C_1\}, \{C_2\}) \\
(\{C_2, C_1\}, \{C_2\}) & (\{C_1\}, \{C_0, C_2\}), & (\{C_1\}, \{C_1, C_2\}) \\
(\{C_1\}, \{C_2, C_2\}), & (\{C_0, C_1\}, \{C_0, C_2\}) & (\{C_1, C_1\}, \{C_0, C_2\}) \\
(\{C_2, C_1\}, \{C_0, C_2\}), & (\{C_1, C_1\}, \{C_1, C_2\}), & \ldots
\end{array}$$

Specifically, $\langle C_1, C_2 \rangle_{\min}$ and $\langle C_1, C_2 \rangle_{\max}$ can be reused for a total of 1458 combinations in the search tree for $mED(2, 2)$. Given this simple example with a relatively small cluster hierarchy (3 levels, 6 clusters), the computational benefits of reusing pairwise statistics are already evident. These benefits become significantly more pronounced with larger hierarchies, where the number of possible combinations grows exponentially with both the pattern complexity and the number of clusters in the tree.

Theorem 6.4.1 (General Empirical Bounds). Let s be a soft-distributive multivariate similarity function over two sets of objects X and Y (if s is one-sided, $Y = \emptyset$). Then, through its soft-distributivity property, s can be expressed as a function of univariate and pairwise statistics:

$$s(X, Y) = g(U_X, U_Y, P_{X,Y}) \tag{6.7}$$

where $U_X = \{u_1(X), u_2(X), \ldots, u_m(X)\}$ and $U_Y = \{u_1(Y), u_2(Y), \ldots, u_n(Y)\}$ are sets of univariate statistics for X and Y respectively, and $P_{X,Y} = \{p_1(X,Y), p_2(X,Y), \ldots, p_l(X,Y)\}$ is a set of l pairwise statistics between elements of X and Y.
Then, the empirical bounds on s can be expressed as:

$$s_{\min}(S_X, S_Y) = g_{\min}(U_{S_X,\min}, U_{S_Y,\min}, P_{S_X,S_Y,\min}, P_{S_X,S_Y,\max}) \tag{6.8}$$
$$s_{\max}(S_X, S_Y) = g_{\max}(U_{S_X,\min}, U_{S_Y,\min}, P_{S_X,S_Y,\min}, P_{S_X,S_Y,\max}) \tag{6.9}$$

where:

- S_X and S_Y are sets of clusters.
- $U_{S_X,\min}$ and $U_{S_X,\max}$ represent the minimum and maximum values of each univariate statistic across all objects in the clusters in S_X (e.g., the minimum and maximum norms of the objects in each cluster in S_X).
- $U_{S_Y,\min}$ and $U_{S_Y,\max}$ represent the minimum and maximum values of each univariate statistic across all objects in the clusters in S_Y.
- $P_{S_X,S_Y,\min}$ and $P_{S_X,S_Y,\max}$ represent the minimum and maximum values of each pairwise statistic between clusters in the cartesian product of S_X and S_Y (e.g., $\langle C_X, C_Y \rangle_{\min}$ and $\langle C_X, C_Y \rangle_{\max}$ for *mED* in Equation 6.5 and 6.6).
- $g_{\min}$ and $g_{\max}$ are constructed from g by using the minimum and/or maximum values of each term based on whether the term contributes positively or negatively to the overall value of g (e.g., the bounds for *mED* in Equation 6.5 and 6.6 above).

Note: The above is formulated in the most general manner, allowing for (1) any number of univariate and pairwise statistics, and (2) different univariate statistics for each side. In practice, however, most multivariate similarity measures can be expressed with only one or two statistics which are shared across both sides, such as the dot product in the case of the *mED* measure above.

Note that neither this method nor the definition of soft-distributivity requires the measure to be metric, symmetric, or be expressable as a function of only a single type of statistic; as long as no functions over more than two objects are involved, the method can be applied. For instance, the *mPC* measure from Chapter 5 is soft-distributive but not a metric, as it does not satisfy the triangle inequality. Furthermore, the method is also not limited to only one-sided measure or two-sided measures; it can be applied to both. For instance, the Multipole measure can be expressed as a function over the smallest eigenvalue of the correlation matrix of a set of vectors, which is merely constructed from the pairwise correlations between vectors, allowing empirical bounds (cf. Chapter 5).

Theoretical Bounds

While empirical bounds use actual minimum and maximum values computed from the data, theoretical bounds derive guarantees based on geometric properties of the clusters, particularly their centroids and radii. As the theoretical bounds are based on geometric properties, they are **limited to vector data** and distance measures that satisfy the triangle inequality. Theoretical bounds are especially valuable for measures that are not soft-distributive. We

take a structured approach to the derivation of these bounds, starting with distance bounds between two vectors, extending to distances between clusters, and finally generalizing to multivariate measures.

We first consider a pair of vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^m$ and a distance measure d that satisfies the triangle inequality. When these vectors belong to clusters C_X and C_Y, we can bound their distance as follows:

Lemma 6.4.2 (Theoretic Bivariate Distance Bounds). Let d be a bivariate distance measure that satisfies the triangle inequality. Then, given two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^m$, each included in a cluster C_X and C_Y, respectively, one can bound $d(\boldsymbol{x}, \boldsymbol{y})$ as follows using the triangle inequality:

$$d(\boldsymbol{c}_X, \boldsymbol{c}_Y) - r_X - r_Y \leq d(\boldsymbol{x}, \boldsymbol{y}) \leq d(\boldsymbol{c}_X, \boldsymbol{c}_Y) + r_X + r_Y \tag{6.10}$$

with $\boldsymbol{c}_X$ and $\boldsymbol{c}_Y$ being the centroids of C_X and C_Y, respectively, and r_X and r_Y being the radii of C_X and C_Y. The radius of a cluster is the maximum distance between the centroid and any point in the cluster, according to d, i.e., $r_X = \max_{\boldsymbol{v} \in C_X} d(\boldsymbol{c}_X, \boldsymbol{v})$. The proof is provided in Appendix C.

Extending the bounds in Lemma 6.4.2 to multivariate measures requires us to consider the two types of multivariate measures separately:

For one-sided measures, Lemma 6.4.2 applies only when the measure can be expressed as a function of pairwise distances. However, this would make the measure soft-distributive, meaning that empirical bounds can be applied. Considering that $d(\boldsymbol{x}, \boldsymbol{y}) \leq \max_{\boldsymbol{v} \in C_X, \boldsymbol{w} \in C_Y} d(\boldsymbol{v}, \boldsymbol{w}) \leq d(\boldsymbol{c}_X, \boldsymbol{c}_Y) + r_X + r_Y$, empirical bounds are guaranteed to be tighter than theoretical bounds, so there is no benefit applying the theoretical bounds for one-sided measures.

For two-sided multivariate measures, we take a geometric approach to extend the bounds in Lemma 6.4.2 to the multivariate case. Recall that a two-sided multivariate measure is defined as $d(\text{agg}(\boldsymbol{x}_1, ..., \boldsymbol{x}_{pl}), \text{agg}(\boldsymbol{y}_1, ..., \boldsymbol{y}_{pr}))$, where $\text{agg}(\cdot)$ is an aggregation function (e.g., sum, average) and pl and pr are the number of vectors at each side. Through the aggregation, we obtain a single aggregate vector for each side, $\boldsymbol{x}_{\text{agg}}$ and $\boldsymbol{y}_{\text{agg}}$. Now when we replace the vectors with clusters, i.e., $d(\text{agg}(C_1, ..., C_{pl}), \text{agg}(C_1, ..., C_{pr}))$, these aggregate vectors are no longer fixed at a single point in space; they can be any point in a space defined by all the possible vector combinations in the cartesian product of the clusters, i.e., $\boldsymbol{x}_{\text{agg}} \in \{\text{agg}(\boldsymbol{x}_1, ..., \boldsymbol{x}_{pl}) | \boldsymbol{x}_i \in C_i\}$. Geometrically, this space is defined by the weighted Minkowski sum of the clusters at that respective side of the query pattern [108], **assuming the aggregation function is linear**, i.e., $\text{agg}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \sum_{i=1}^{n} \omega_i \boldsymbol{x}_i$ for some weight vector ω.

The concept of the Minkowski sum between two clusters C_1 and C_2 is illustrated in Figure 6.3, where the aggregate cluster C_S (orange) is formed by the weighted Minkowski sum of the two original clusters C_1 (red) and C_2 (green) with aggregation weights $\omega_1 = \omega_2 = 1/2$,

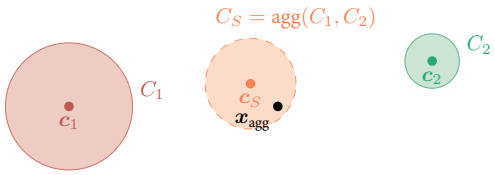


Figure 6.3: Illustration of an aggregate cluster C_S formed by the weighted Minkowski sum of two original clusters C_1 and C_2 (respective centroids $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$) with weights $\omega_1 = \omega_2 = 1/2$. An aggregate vector, such as $\boldsymbol{x}_{\text{agg}} = \text{agg}(\boldsymbol{x}_1, \boldsymbol{x}_2)$ where $\boldsymbol{x}_1 \in C_1, \boldsymbol{x}_2 \in C_2$, is a point within this aggregate cluster C_S. The centroid $\boldsymbol{c}_S$ and radius r_S of C_S are derived from the centroids and radii of C_1, C_2 and the aggregation weights (Lemmas 6.4.3 and 6.4.4).

i.e., simple averaging. This orange area represents the closed area where the aggregate vectors $\boldsymbol{x}_{\text{agg}}$ formed from any combination of vectors in C_1 and C_2 are located.

A key insight is that when the measure operates in a **vector space**, the weighted Minkowski sum of spherical clusters remains a spherical cluster [108] We refer to these clusters as *aggregate clusters*. The properties of these aggregate clusters can be derived from the constituent clusters, specifically their centroids and radii.

Lemma 6.4.3 (Centroid of an Aggregate Hypersphere Cluster). Given a set of spherical clusters $S = \{C_i\}_{i=1}^p$ with each cluster covering a set of vectors $C_i \subseteq \mathcal{V}$ and a linear aggregation $\text{agg}(X) = \omega^T X$, the centroid of the hypersphere C_S covering all points in the Minkowski aggregation of S is as follows;

$$\boldsymbol{c}_S = \omega^T [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_p] \tag{6.11}$$

with $\boldsymbol{c}_i$ being the centroid of the cluster C_i. The proof is provided in Appendix C.

Lemma 6.4.4 (Radius of an Aggregate Hypersphere Cluster). Given a set of spherical clusters $S = \{C_i\}_{i=1}^p$ with each cluster covering a set of vectors $C_i \subseteq \mathcal{V}$ and radius defined using a distance metric d on a *vector space* S with norm $\|\boldsymbol{x}\|_d = d(\boldsymbol{x}, \boldsymbol{0})$, and a linear aggregation $\text{agg}(X) = \omega^T X$, the radius of the hypersphere C_S covering all points in the Minkowski aggregation of S is bounded as follows;

$$r_S = \omega^T [r_1, \ldots, r_p] \leq \sum_{i=1}^{p} |\omega_i| r_i \tag{6.12}$$

with $\boldsymbol{c}_i$ being the centroid of the cluster C_i and r_i being the radius of the cluster C_i. The proof is provided in Appendix C.

Lemma 6.4.3 and Lemma 6.4.4 show that the centroid and radius of an aggregate cluster can be computed simply by aggregating the centroids and radii of the constituent clusters, respectively, using the same aggregation weights as used to form the aggregate vectors. Given

the well-defined properties of an aggregate cluster, we can now derive bounds on the distance between any two aggregate vectors $\boldsymbol{x}_{\text{agg}}$ and $\boldsymbol{y}_{\text{agg}}$ formed from clusters at each side of the query pattern (i.e., C_X and C_Y):

Theorem 6.4.5 (Theoretic Multivariate Distance Bounds). Let d be a bivariate distance measure that satisfies the triangle inequality and operates on vectors in a vector space, and two linear aggregations $\text{agg}_X(X) = \omega_X^T X$ and $\text{agg}_Y(Y) = \omega_Y^T Y$. Then, given two sets of vectors $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{pl}\}$ and $Y = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{pr}\}$, with each vector included in a cluster constructing the cluster sets $C_X = \{C_{x,1}, \ldots, C_{x,pl}\}$ and $C_Y = \{C_{y,1}, \ldots, C_{y,pr}\}$, respectively, one can bound $d(X, Y)$ as follows:

$$d(\boldsymbol{c}_X, \boldsymbol{c}_Y) - r_X - r_Y \leq d(X, Y) \leq d(\boldsymbol{c}_X, \boldsymbol{c}_Y) + r_X + r_Y \tag{6.13}$$

with centroids and radii of aggregate clusters C_X and C_Y defined as in Lemma 6.4.3 and Lemma 6.4.4.

Theorem 6.4.5 allows us to compute bounds for cluster combinations under any multivariate distance measure that satisfies the triangle inequality and operates on a vector space. Unfortunately, similarity measures, unlike distance measures, often do not satisfy the triangle inequality, and therefore, the above bounds cannot be applied directly. However, if such a similarity measure can be expressed as a function of a distance measure that does satisfies these properties, bounds are still feasible. For instance, *cosine similarity* $\cos(\theta)$ does not satisfy the triangle inequality directly, but it does operate on *angles* between vectors in a vector space $\theta = \angle(\boldsymbol{x}, \boldsymbol{y})$, which do satisfy the triangle inequality, i.e., $\angle(\boldsymbol{x}, \boldsymbol{y}) \leq \angle(\boldsymbol{x}, \boldsymbol{z}) + \angle(\boldsymbol{y}, \boldsymbol{z})$. As such, by considering that the cosine function is a monotonically decreasing function between 0 and π, we can still derive bounds on the cosine similarity between sets of clusters C_X and C_Y (respective centroids $\boldsymbol{c}_X$ and $\boldsymbol{c}_Y$) using the bounds on the angles between vectors in the clusters:[1]

$$\cos(\angle^{\max}_{X,Y}) \leq \cos(\angle(X, Y)) \leq \cos(\angle^{\min}_{X,Y}) \tag{6.14}$$

with:

$$\cos(\angle^{\min}_{X,Y}) = \max(0, \angle(\boldsymbol{c}_X, \boldsymbol{c}_Y) - r_X - r_Y), \qquad r_X = \max_{\boldsymbol{x} \in C_X} \angle(\boldsymbol{c}_X, \boldsymbol{x}),$$
$$\cos(\angle^{\max}_{X,Y}) = \min(\pi, \angle(\boldsymbol{c}_X, \boldsymbol{c}_Y) + r_X + r_Y), \qquad r_Y = \max_{\boldsymbol{y} \in C_Y} \angle(\boldsymbol{c}_Y, \boldsymbol{y})$$

This principle generalizes to any similarity measure expressible as a monotonic function of a triangle-inequality-satisfying distance:

Theorem 6.4.6 (General Theoretical Bounds). Let d be a bivariate distance measure that satisfies the triangle inequality and operates on vectors in a vector space, and s be a similarity measure that can be expressed as a monotonic function over d, i.e.,

[1] Under the convention that all angles between two vectors are in the range $[0, \pi]$, i.e., if $\angle(\boldsymbol{x}, \boldsymbol{y}) > \pi$, then $\angle(\boldsymbol{x}, \boldsymbol{y}) = 2\pi - \angle(\boldsymbol{x}, \boldsymbol{y})$.

$s(X,Y) = f(d(X,Y))$. Then, given two sets of vectors $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{pl}\}$ and $Y = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{pr}\}$, with each vector included in a cluster constructing the cluster sets $C_X = \{C_{x,1}, \ldots, C_{x,pl}\}$ and $C_Y = \{C_{y,1}, \ldots, C_{y,pr}\}$, respectively, one can bound $s(X,Y)$ as follows:
If f is monotonically *decreasing*:

$$f(d(\boldsymbol{c}_X, \boldsymbol{c}_Y) + r_X + r_Y) \leq s(X,Y) \leq f(d(\boldsymbol{c}_X, \boldsymbol{c}_Y) - r_X - r_Y) \tag{6.15}$$

If f is monotonically *increasing*:

$$f(d(\boldsymbol{c}_X, \boldsymbol{c}_Y) - r_X - r_Y) \leq s(X,Y) \leq f(d(\boldsymbol{c}_X, \boldsymbol{c}_Y) + r_X + r_Y) \tag{6.16}$$

with centroids and radii of aggregate clusters C_X and C_Y defined as in Lemma 6.4.3 and Lemma 6.4.4.

Theorems 6.4.5 and 6.4.6 thus provide a general method for bounding multivariate similarity measures that either operate in a vector space with the triangle inequality or can be expressed as monotonic functions of such measures.

No Bounds

When a measure lacks properties for either empirical or theoretical bounds (non-distributive, non-metric, not expressible as a monotonic function of a metric), the framework must resort to exhaustive evaluation. This requires recursively splitting cluster combinations down to individual objects and computing exact measure values. While this approach lacks the pruning efficiency of the bounding methods, it can still benefit from two types of optimizations: dimensionality reduction (discussed in Section 6.4.5), and caching of intermediate results that can be reused across different object combinations.

Bound Selection The framework systematically selects the most appropriate bounding method for each measure (Figure 6.4). As further discussed in Section 6.4.6, measure properties are defined in the measure interface, allowing the framework to check for soft-distributivity, metric properties, and monotonicity based on predefined measure attributes. The framework evaluates measure properties in a hierarchical manner, first checking for soft-distributivity, then for metric properties in vector spaces, and finally defaulting to exhaustive search if no bounding method is applicable.

6.4.5 Optimization Suite

To maximize performance across diverse scenarios, the framework incorporates a suite of optimization techniques that can be selectively applied based on the properties of the distance/similarity measure and the query parameters. Table 6.5 summarizes the key optimizations, their location in the pipeline, and the properties they require. These optimizations are largely independent of each other and *can be applied in any order*, with the sole exception that optimization O4 requires O2 to be applied first, as it relies on the precomputed pairwise matrix. We evaluate the impact of each of these optimizations on the execution time of the algorithm in Section 6.5.

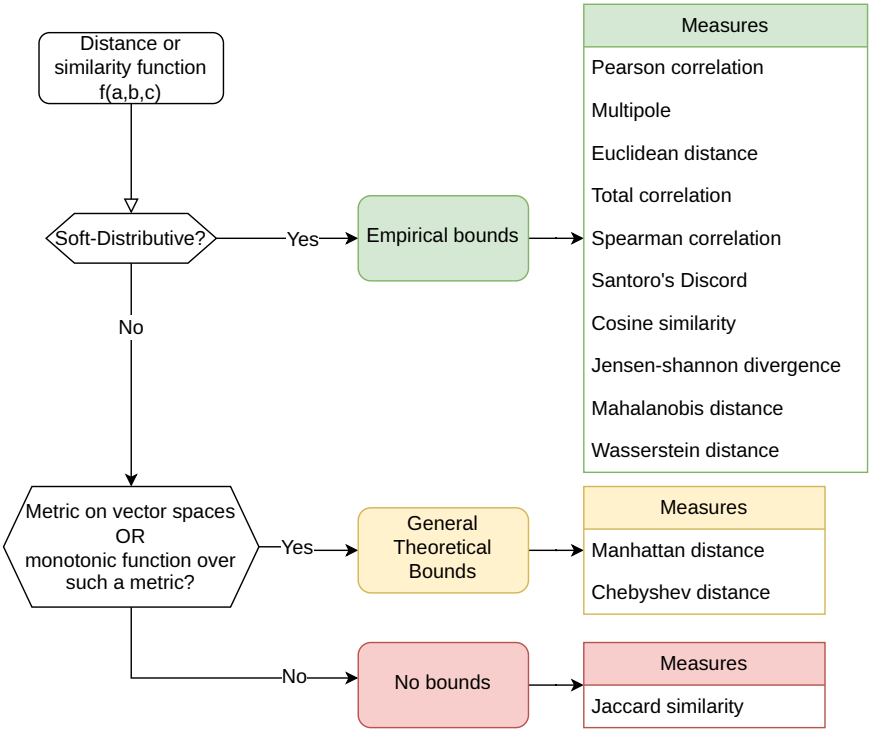


Figure 6.4: Decision tree for selecting the appropriate bounding method based on the properties of the similarity measure.

Note that several of these optimizations, such as dimensionality reduction (O1), Apriori filtering (O6), and empirical bound discounting (O9), were not part of the CD algorithm described in Chapter 5 and are thus novel contributions in this context. Moreover, to the best of our knowledge, the concept of empirical bound discounting is introduced here for the first time.

Running example: To provide more context, we revisit the running example from Section 6.4.3, which considered a top-k query with a query pattern mPC (2,1) on a dataset D consisting of 6 vectors. For simplicity, we will also re-use the example cluster hierarchy of the example dataset shown in Figure 6.2.

O1: Dimensionality Reduction (Preprocessing): Applies techniques like random projections to reduce data dimensionality. This is particularly beneficial for non-distributive measures relying on theoretical bounds or the "no bounds" case, where distances have to be computed over potentially very many high-dimensional objects. It reduces the computational burden of these distance computations by working in a lower-dimensional space, at the cost of introducing approximation errors that require filtering (FP Filtering) in post-processing or potentially missing results (i.e., false negatives). Ideally, dimensionality reduction techniques are used that come with probabilistic guarantees of preserving or lower-bounding the

Table 6.2: The framework's optimization suite.

ID	Optimization	Pipeline Location	Required Properties
O1	Dimensionality Reduction*	1. Preprocessing	Theoretical bounds
O2	Precompute Pairwise Matrix	1. Preprocessing	Empirical bounds
O3	Distance Measure Optimization	2. Clustering	
O4	Skip First Level	3. Pre-tree traversal	O2
O5	Result Expanding	3. Pre-tree traversal	Top-k query
O6	Apriori Filtering	3. Pre-tree traversal	Monotonically decreasing
O7	Centroid & Radius Caching	4. Cluster bounding	Theoretical bounds, Two-sided
O8	Cluster Pair Caching	4. Cluster bounding	Empirical bounds
O9	Empirical Bound Discounting	5. Bound checking	Empirical bounds
O10	Canonical Candidate Generation	6. Candidate generation	Permutation invariant/ Commutative aggregation
O11	Duplicate Objects Check	6. Candidate generation	No side overlap
O12	Branch Prioritization	7. Branch prioritization	Top-k query

* This optimization may introduce approximation errors, requiring filtering (FP Filtering) in post-processing or potentially missing results (i.e., false negatives).

distances in the original space, such as Johnson-Lindenstrauss projections [93, 122] or DFT approximations (cf. Section 2.2). This way, one can control the expected number of false negatives introduced by the approximation, as the probabilistic guarantees allow for a theoretical upper bound on the error of the similarity/distance computed over the approximated objects. In practice, however, the actual number of false negatives is often much lower than the theoretical upper bound, as the approximation effects bounds (i.e., the distances between aggregate clusters) rather than exact distances. In other words, even if an approximation would increase the measured similarity, the lower bound over a cluster combination might still be valid.

Example: When applied to z-normalized vectors, there is a one-to-one mapping between the Pearson correlation and Euclidean distance, namely $\rho(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) = 1 - \frac{d(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})^2}{2m}$, where d is the Euclidean distance. This allows us to use DFT approximation (cf. Section 2.2) to reduce the dimensionality of the vectors, as distances over the approximated vectors lower-bound the Euclidean distance in the original space, and therefore also upper-bound the Pearson correlation. Therefore, in case one decides to use theoretical bounds to answer our example query, we can apply DFT approximation to the 6 vectors in D to reduce their dimensionality, which would significantly speed up the correlation computations in the bounding process. It would, however, due to the lower-bounding nature of the approximation, also introduce false positives, which would have to be filtered out when finding a positive decisive cluster combination in phase 5 (Bound Checking) of the query execution pipeline.

O2: Precompute Pairwise Matrix (Preprocessing): For measures with empirical bounds, this optimization involves precomputing and storing the matrix of all pairwise statistics (e.g., dot products) used in the empirical bounds. This avoids redundant computations of the

same pairwise statistics across different cluster combinations, when deriving the minimum and maximum statistics for pairs of clusters (cf. Theorem 6.4.1).

Example: Considering our measure *mPC*, we would precompute the pairwise correlation matrix for all vectors in D at the preprocessing stage, storing the results in a 6×6 upper-triangular matrix C_2 that can be quickly accessed during the bounding process.

O3: Distance Measure Optimization (Clustering): Allows the user to specify a distance function appropriate for the hierarchical clustering step, overriding the default measure. As clusters form the core of our pruning mechanism, the choice of distance measure can significantly impact the tightness of the bounds derived over the clusters, which in turn affects the pruning power. Ideally, the distance function is highly correlated with the query's multivariate similarity function, meaning that objects that are close according to the distance function are likely to result in similar similarity scores. Note that this optimization does not require the multivariate measure to be expressable as a function of the distance measure; they merely need to be correlated.

Example: In our example, we could use cosine distance (i.e., angular) as measure for clustering, rather than the default Euclidean distance. However, in this case, the results would be equivalent, as for z-normalized vectors, cosine similarity is equivalent to Pearson correlation, which has a one-to-one relationship with the Euclidean distance, i.e., $\cos(\angle(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})) = \rho(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) = 1 - \frac{d(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})^2}{2m}$ so $\angle(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) = \arccos(1 - \frac{d(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})^2}{2m}) \Rightarrow \angle(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) \propto d(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}})$.

O4: Skip First Level (Pre-traversal): For measures with empirical bounds, this optimization utilizes the precomputed pairwise matrix (cf. O2) to skip the first complexity level (i.e., pairs) and directly derive the top-k pairs from the pairwise similarities. This way, we can avoid the overhead of the recursive bounding process for pairs, which can be substantial for large datasets.

Example: In our example, we would simply derive the top-3 pairwise correlations from the precomputed pairwise matrix C_2, update the running threshold, and proceed directly to the second complexity level (i.e., triplets).

O5: Result Expanding for Top-k Queries (Pre-traversal): After completing a complexity level c, this optimization proactively checks supersets (at level $c + 1$) of the high-scoring combinations found at level c. Since supersets of combinations with high scores are likely to yield high scores themselves, this optimization can help identify promising combinations early, increasing the running top-k threshold before the next level's traversal. As more extreme thresholds allow for more aggressive pruning, this can significantly reduce the number of combinations that need to be evaluated at the next level.

Example: In our example, consider that the top-3 pairs found at level 1 are $R = \{(\boldsymbol{a}, \boldsymbol{b}), (\boldsymbol{c}, \boldsymbol{d}), (\boldsymbol{e}, \boldsymbol{f})\}$. Then, rather than directly starting with the second level by considering the root combination (C_0, C_0, C_0), we would first check the combinations (C_0, C_3, C_4), (C_0, C_5, C_6), and (C_0, C_7, C_8), which compactly represent all the supersets of size 3 of the pairs in R.

Once finished with these combinations, we would then proceed with the root combination (C_0, C_0, C_0), making sure to not add duplicates to the result set during the traversal.

O6: Apriori Filtering (Pre-traversal): If the similarity measure is known to be monotonically decreasing with respect to set size, this optimization prunes candidate combinations that are supersets of already-processed combinations whose similarity was below the threshold. This technique can yield a very high impact on query performance, as it significantly reduces the search space by eliminating combinations that are guaranteed to yield low similarity scores. Namely, if the measure is monotonically decreasing, any combination that is a superset of a combination known to be below the threshold will by definition also be below the threshold, and can be pruned.

For instance, if clusters C_A and C_B have insufficient similarity with cluster C_C (i.e., below the threshold), and the measure is monotonically decreasing, then adding any additional objects to either side will only decrease the similarity further. Therefore, any combination containing both C_A and C_B can be safely pruned from consideration with C_C. This observation is exploited at the pre-traversal step for each level by constructing root combinations only from cluster combinations that were added to the result set at the previous level.

Similarly, when a measure is monotonically increasing, any combination that is a superset of a combination known to be above the threshold will also be above the threshold, and can be added directly to the result set without further checks. This optimization is effective for any query type, including threshold queries, as it directly reduces the search space to be traversed at each level. This saves the overhead of computing and checking the bounds for these combinations.

Example: For the sake of the example, imagine that the *mPC* measure is monotonically decreasing, and that the top-3 pairs found at level 1 are $R = \{(\boldsymbol{a}, \boldsymbol{b}), (\boldsymbol{c}, \boldsymbol{d}), (\boldsymbol{e}, \boldsymbol{f})\}$. Then, similar to the result expanding optimization, we would subsequently check the combinations (C_0, C_3, C_4), (C_0, C_5, C_6), and (C_0, C_7, C_8). This time, however, we would not have to consider the root combination (C_0, C_0, C_0) afterwards as no vector combination outside of the expanded combinations can yield a higher score than the ones already found.

O7: Centroid & Radius Caching (Cluster Bounding): For two-sided measures that rely on theoretical bounds (cf. Theorem 6.4.5 and Theorem 6.4.6), the aggregate centroid and radius of a set of clusters on one side of the query pattern might be needed multiple times. For instance, the aggregate properties of $\{C_A, C_B\}$ would be computed when evaluating $s(\{C_A, C_B\}, \{C_C, C_D\})$ and again for $s(\{C_A, C_B\}, \{C_E, C_F\})$. This optimization caches the aggregate centroid and radius (derived as per Lemma 6.4.3 and Lemma 6.4.4) for such recurring combinations of clusters on either side of the measure, preventing redundant calculations.

Example: In our example, the aggregate centroid and radius of the clusters $\{C_1, C_2\}$ would be computed once when evaluating $(\{C_1, C_2\}, \{C_0\})$ and reused when evaluating $(\{C_1, C_2\}, \{C_1\})$, $(\{C_1, C_2\}, \{C_2\})$, and so on.

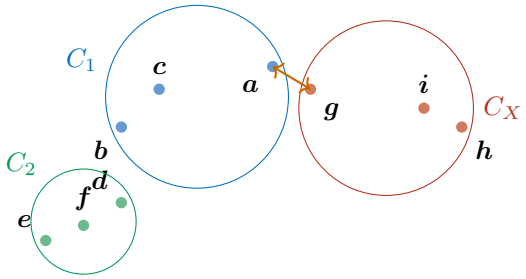


Figure 6.5: Illustration of empirical bound discounting. The maximum correlation between C_1 and C_X is dominated by an outlier pair $(\boldsymbol{a}, \boldsymbol{g})$, leading to an indecisive bound even though most materializations in the cluster combination would yield low similarity scores.

O8: Cluster Pair Caching (Cluster Bounding): Empirical bounds are functions over the minimum and maximum values of pairwise statistics between pairs of clusters (cf. Theorem 6.4.1). As pairs of clusters typically appear in a wide variety of combinations, this optimization caches these pairwise statistics for each pair of clusters, allowing for quick retrieval when needed and avoiding many redundant calculations at higher complexity levels.

Example: In our example, the pairwise statistics between clusters C_1 and C_2 would be computed once when evaluating the combination $(\{C_1, C_2\})$ and reused for all subsequent combinations involving these clusters, such as $(\{C_0, C_1\}, C_2)$, $(\{C_0, C_2\}, C_1)$, and so on.

O9: Empirical Bound Discounting (Bound Checking): Empirical bounds rely on bounds over pairwise statistics between clusters. While these bounds are typically tight for well-formed clusters, they can become inconclusive when a few outlier object pairs dominate the bounds. This can lead to excessive iterations of cluster splitting even when the majority of materializations would be prunable.

Bound discounting addresses this inefficiency by identifying these outlier pairs – corresponding to *extrema pairs* in the CDStream context (cf. Chapter 5) – and temporarily excluding them from the bound calculations, and recomputing the bounds on the remaining materializations. Specifically, the optimization involves tracking the k largest and smallest pairwise statistics (with non-overlapping object pairs) for each cluster pair, rather than merely the overall minimum and maximum values. Then, when a cluster combination has inconclusive bounds, we temporarily exclude the outlier pairs (i.e., the top k largest and smallest statistics) from the bound calculations, and recompute the bounds on the remaining materializations. If these adjusted bounds become decisive (e.g., allowing pruning), we handle the outlier pairs separately as distinct cluster combinations.

The process is parameterized by k (number of tracked pairs) and τ_{discount} (minimum critical shrink factor threshold, explained in O12), limiting discounting to combinations unlikely to contain many high-scoring results. While this optimization can significantly reduce unnecessary splitting iterations, it comes at the cost of additional bookkeeping in the bounding of cluster pairs and a slightly more complex bound checking process. This addition overhead in memory and computation can be controlled though through the choice of k and τ_{discount}.

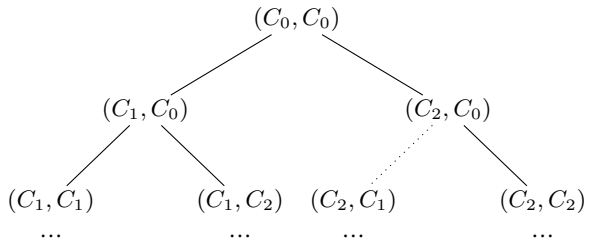


Figure 6.6: Canonical candidate generation from (C_0, C_0). Assuming C_0 splits into C_1, C_2, candidates are formed by splitting the leftmost largest cluster, maintaining an ordered representation to prevent duplicates like (C_2, C_1) if (C_1, C_2) has already been generated. The hidden edge to (C_2, C_1) indicates its generation is conditional on these ordering rules.

Example: For the sake of the example, consider cluster C_X, additional to the clusters in Figure 6.2, which contains the vectors $\{\boldsymbol{g}, \boldsymbol{h}, \boldsymbol{i}\}$. Then, Figure 6.5 illustrates a cluster combination $(\{C_1, C_2\}, C_X)$ with the *mPC* upper bound $UB = \frac{\rho_{\max}(C_1, C_X) + \rho_{\max}(C_2, C_X)}{\sqrt{2 + 2 * \rho_{\min}(C_1, C_2)}}$. As shown, the maximum correlation between C_1 and C_X is dominated by an outlier pair $(\boldsymbol{a}, \boldsymbol{g})$, leading to an indecisive bound even though most materializations in the cluster combination would yield low similarity scores. Given $k = 2$ for the top-k pairs, the algorithm would identify $[(\boldsymbol{a}, \boldsymbol{g}), (\boldsymbol{c}, \boldsymbol{i})]$ as the top-2 largest statistics between C_1 and C_X. After finding the bounds inconclusive, we compute the differences ("skips") between consecutive pairs in the top-k lists, i.e., $\rho(\boldsymbol{a}, \boldsymbol{g}) - \rho(\boldsymbol{c}, \boldsymbol{i})$ for (C_1, C_X), and rank them across all cluster pairs. Starting with the largest skip, we check if ignoring that vector pair makes the bounds decisive. If so, we prune the combination and handle the outlier pair separately as a new cluster combination, for instance, by computing bounds on $(\{\boldsymbol{a}, C_2\}, \boldsymbol{g}\})$ with $\boldsymbol{a}$ and $\boldsymbol{g}$ as singletons.

O10: Canonical Candidate Generation (Candidate Generation): If a multivariate measure is permutation invariant or if the aggregation function in a two-sided measure is commutative (cf. Section 6.4.2), the order of clusters in a combination does not affect the similarity score of its materializations. This optimization exploits this property to avoid generating and evaluating redundant combinations (e.g., (B,A) if (A,B) has been considered, or (B,A,C) if (A,B,C) has). This is achieved by enforcing a canonical ordering during candidate generation, tied to cluster IDs assigned during hierarchical clustering. Enforcing canonical ordering ensures that only one representative combination is generated for each unique set of clusters, preventing the evaluation of redundant combinations that would yield the same results. Namely, by (1) assigning IDs to clusters in a depth-first manner during the hierarchical clustering step, and (2) only generating combinations with clusters in a specific order (e.g., decreasing order of IDs, always splitting the leftmost largest cluster first), we can ensure that only one representative combination is generated for each unique set of clusters.

Example: We show how new combinations are generated from the root combination (C_0, C_0) in Figure 6.6.

O11: Duplicate Objects Check (Candidate Generation): If the query specifies that a object cannot appear multiple times in a result combination (e.g., finding sets of *distinct* time series),

this check is performed when generating candidates from an indecisive cluster combination. Namely, cluster combinations including the same singleton cluster (i.e., a cluster containing only one object) are immediately pruned.

Example: In our example, say we are splitting the combination $(\{C_5, C_1\}, C_1)$, where $C_5 = \{\boldsymbol{c}\}$, the candidate $(\{C_5, C_5\}, C_1)$ is ignored as all its materializations will contain the same vector $\boldsymbol{c}$ twice.

O12: Branch Prioritization for Top-k queries (Branch Prioritization step): Recall from Chapter 5 that the base algorithm for answering top-k queries involves initializing a running top-k threshold τ at the lowest possible value (e.g., 0) and iteratively exploring cluster combinations, updating τ with the highest score found so far, and maintaining a running top-k result set. In other words, the execution strategy is effectively identical to that of threshold queries, with the additional step of maintaining the result set at exactly k results.

Also recall that the pruning power of the algorithm is directly tied to the running top-k threshold τ; the higher it is, the more combinations can be pruned. Therefore, the pruning power (and thus performance) at top-k queries can be optimized by (1) initializing τ at a higher value, and (2) finding high-scoring combinations quickly to raise the threshold early in the execution.

This optimization focuses on the second aspect by prioritizing the exploration of branches (cluster combinations) that are more likely to yield high-scoring combinations, which in turn allows for a higher running top-k threshold τ earlier in the execution. This is achieved by restructuring the default top-k algorithm into two phases:

1. *Initial Bounded Traversal*: For a given complexity level, the algorithm starts by traversing the tree of cluster combinations in a *breadth-first manner*. As usual, during the bounding process, it computes the standard lower bound (LB) and upper bound (UB) for each combination. However, for prioritization, an "optimistically tightened" or "shrunk" upper bound (UB_{shrunk}) is also calculated using the formula: $UB_{\text{shrunk}} = (1-\gamma)\frac{UB+LB}{2} + \gamma UB$. Here, γ is a configurable shrink factor between -1 and 1, with a default of 0. If UB_{shrunk} is below the current top-k threshold (τ) but the original UB is still above τ (i.e., $UB_{\text{shrunk}} \leq \tau < UB$), the cluster combination is not immediately split further. Instead, it is marked as "postponed" for prioritized exploration.

2. *Prioritized Depth-First Exploration*: Postponed cluster combinations are placed into a priority queue, sorted by a their "critical shrink factor" (γ^*), which is the minimum value of γ for which UB_{shrunk} would exceed the current top-k threshold τ. Intuitively, a smaller γ^* indicates that a larger portion of the combination's bound range ($UB - LB$) is above τ, making it a more promising candidate as it is more likely to yield many high-scoring combination. The framework then explores these postponed branches in a *depth-first manner*, starting with those having the smallest γ^*.

By processing more promising branches first, this strategy aims to rapidly increase the running top-k threshold. A higher threshold, in turn, allows for more effective pruning in subse-

quent explorations. The choice of the γ parameter can influence the balance between the initial breadth-first-like pass and the subsequent depth-first exploration of promising branches, but it does not affect the completeness or correctness of the final results. Empirically, a value of $\gamma = 0$ has shown to be most effective in practice, balancing the two phases.

Example: In our example, say the running threshold is $\tau = 0.5$ and the cluster combination $(\{C_1, C_2\}, C_3)$ has similarity bounds $LB = 0.4$ and $UB = 0.55$. Then, given $\gamma = 0$, the optimistically tightened upper bound would be $UB_{\text{shrunk}} = (1-0)\frac{0.55+0.4}{2} + 0 \cdot 0.55 = 0.475$. As $UB_{\text{shrunk}} \leq \tau < UB$, the combination is marked as postponed for prioritized exploration, with a critical shrink factor of $\gamma^* = \frac{\tau-\mu}{UB-\mu} = \frac{0.5-0.475}{0.55-0.475} = \frac{0.025}{0.075} = \frac{1}{3}$ as $\mu = \frac{UB+LB}{2} = \frac{0.55+0.4}{2} = 0.475$. Then, in the prioritized depth-first exploration phase, this combination would be explored before any other postponed combinations with a smaller critical shrink factor (e.g., a combination with bounds $LB = 0.3$ and $UB = 0.51$), as it is more likely to yield high-scoring results.

Summary

The described optimization suite seamlessly integrates with and builds upon the foundations of the query execution workflow of Section 6.4.3, significantly improving its operational efficiency. The inherent flexibility of the base workflow – efficient traversal of the full combinatorial search space – makes it highly amenable to many optimizations from literature (cf. Section 6.2). Particularly, the simplicity of abstracting object combinations into cluster combinations allows us to reason about multiple candidates at once while also allowing the application of powerful set-based pruning techniques such as Apriori filtering.

Collectively, these optimizations transform SD into a more powerful and versatile framework for multivariate similarity search. By dynamically tailoring its execution strategy to the similarity measure and query configuration at hand, the optimization suite expands the range of problems that can be efficiently addressed, including those that would be computationally infeasible otherwise. This ultimately solidifies the framework's ability to handle a wide variety of multivariate similarity search tasks, from simple top-k queries on correlations to complex queries using non-distributive distance/similarity measures with multiple constraints.

6.4.6 Measure Interface

To enable the integration of user-defined problems into the framework, we establish a standardized *measure interface*. This interface functions as a *contract* for implementing multivariate similarity measures, ensuring they conform to a consistent structure and operational behavior. Furthermore, through the use of general or *abstract* methods on the interface, we can implement default behavior across all measures (e.g., implementation of the general theoretical bounds in Theorem 6.4.6), allowing for a simpler and more efficient implementation process.

In Appendix G, we provide a full interface implementation in Python including the logic for all methods. Note that the interface can be implemented in any programming language, and the choice of Python is arbitrary.

The measure interface is designed to be comprehensive yet flexible, streamlining the process of integrating new measures into the framework. It consists of the following main parts:

- **Properties**: Defined in the initialization method, these are boolean flags on the measure properties discussed in Section 6.4.2, and can be overridden by the implementer. These properties describe the fundamental characteristics of the measure and are used by the framework to determine appropriate behaviors, such as which bounding strategies or default methods are applicable. The properties are:
  - *is_similarity_function*: Indicates whether the measure is a similarity function (as opposed to a distance function).
  - *is_monotonically_increasing*: Indicates whether the measure is monotonically increasing with respect to the number of objects in the input sets.
  - *is_monotonically_decreasing*: Indicates whether the measure is monotonically decreasing with respect to the number of objects in the input sets.
  - *is_metric*: Indicates whether the measure satisfies the properties of a metric (non-negativity, identity of indiscernibles, symmetry, and triangle inequality).
  - *is_on_vector_space*: Indicates whether the measure operates over a vector space.
  - *is_permutation_invariant*: Indicates whether the measure is permutation invariant with respect to the order of objects in the input sets.
  - *is_continuous*: Indicates whether the measure operates on continuous data.
- **Abstract Methods**: These methods form the core contract that any concrete measure must fulfill by providing specific implementations.
  - *compute(X,Y)*: This is the core method that computes the multivariate similarity or distance between two sets of objects X and Y. For one-sided measures, Y should be ignored or combined with X within the implementation.
  - *get_empirical_bounds(CX, CY)*: This method calculates the measure-specific empirical bounds for a cluster combination (C_X, C_Y). Users can derive these bounds through the General Empirical Bounds in Theorem 6.4.1.
- **Optional Methods**: These methods come with default implementations (the details of which can be found in Appendix G) but can be overridden by the implementer if the specific measure requires a different behavior. Key optional methods include:
  - *preprocess(data)*: Optional data preprocessing step applied before initializing the search. Corresponds to step 1 of the workflow in Section 6.4.3.

  - *aggregate(objects)*: Aggregation function for two-sided measures. Defaults to the element-wise mean in case of vectors.
  - *cdist(x, y)*: Distance function used in hierarchical clustering. Defaults to Euclidean distance for continuous data and Hamming distance for categorical data.
  - *dist2sim(dist)*: Converts a distance value to a similarity score. Used as part of the general theoretical bounds in Theorem 6.4.6 for measures that are non-metric but can be expressed as functions of metric distances (cf. Section 6.4.4).
  - *sim2dist(sim)*: Inverse of *dist2sim*. Used as part of the general theoretical bounds.
  - *check_additional_constraints(CX, CY)*: Allows for checking problem-specific conditions or additional constraints that objects in the final result set must satisfy. This is where constraints like minimum jump or irreducibility are enforced.

- **Generic Methods**: These methods provide the standardized, generic logic that come as part of the framework. Key generic methods include:
  - *bound(CX, CY)*: Main method for determining bounds between cluster combinations. Selects the tightest bounds by comparing theoretical and empirical bounds, if available.
  - *get_pairwise_bounds(c1, c2, d)*: Computes bounds between two clusters using their centroids and radii, based on the equations in Lemma 6.4.2.
  - *get_theoretical_distance_bounds(CX, CY)*: Implements the general theoretical bounds for metric distances from Theorem 6.4.6.
  - *get_theoretical_bounds(CX, CY)*: Converts distance bounds to similarity bounds using the dist2sim/sim2dist transformations when applicable.

When a user wishes to integrate a new multivariate similarity or distance measure into the framework, their main tasks are:

1. Define a new Python class that inherits from the "Measure" abstract base class.
2. In the initialization method of their class, override any properties that are not applicable.
3. Provide implementations for *compute* and (optionally) *get_empirical_bounds*.
4. Review the default implementations of the optional methods. If the default behavior is not suitable for the new measure (e.g., if a different aggregation strategy or clustering distance measure is required), these methods should be overridden with custom logic.

This structured approach ensures that new measures are compatible with the framework's existing functionalities, such as pruning through recursive bounding, while providing the necessary flexibility to accommodate the unique aspects of each specific measure.

Table 6.3: Overview of example measures, their properties, and optimizations applied by the framework.

Property/Feature	***mPC***	***MP***	***mED***	***TC***	***mMD***
Type	Two-sided	One-sided	Two-sided	One-sided	Two-sided
Aggregation	Average	-	Average	-	Sum
Metric	✓	✓	✓	✗	✓
Mono. Increasing	✗	✓	✗	✓	✗
Mono. Decreasing	✗	✗	✗	✗	✗
Perm. invariant	✓	✓	✓	✓	✓
Continuous	✓	✓	✓	✗	✓
Preprocessing	Z-norm	Z-norm	None	Quant.	Rand. Proj.
O1: Dim. reduction	✗	✗	✗	✗	✓
O2: Pairwise matrix	✓	✓	✓	✓	✗
O3: Distance measure	✗	✗	✗	Info. Dist.	Manhattan
O4: Skip first level	✓	✓	✓	✓	✗
O5: Result expanding	✓	✓	✓	✓	✓
O6: Apriori filtering	✗	✗	✗	✗	✗
O7: Centroid & radius caching	✗	✗	✗	✗	✓
O8: Cluster pair caching	✓	✓	✓	✓	✗
O9: Emp. bound discounting	✓	✓	✓	✓	✗
O10: Canonical gen.	✓	✓	✓	✓	✓
O11: Duplication check	✓	✓	✓	✓	✓
O12: Branch priority	✓	✓	✓	✓	✓

6.4.7 Examples

To illustrate the integration of different measures into the framework, this section presents three examples of multivariate similarity measures. For each example, we demonstrate: (1) the derivation of bounds based on the theorems in Section 6.4.4; (2) the implementation of the measure using the defined interface; and (3) the optimizations automatically applied by SD. The complete Python implementations for all three measures are provided in Appendix H.

Multivariate Euclidean Distance (*mED*)

The Euclidean distance d_{eucl} is a well-known metric that measures the straight-line distance between two points in Euclidean space. We define its multivariate version *mED* as a two-sided measure with averaging as the aggregation function, formally defined as:

$$mED(X,Y) = d_{eucl}\left(\frac{\sum_{\boldsymbol{x}\in X}\boldsymbol{x}}{|X|}, \frac{\sum_{\boldsymbol{y}\in Y}\boldsymbol{y}}{|Y|}\right) = \left\|\frac{\sum_{\boldsymbol{x}\in X}\boldsymbol{x}}{|X|} - \frac{\sum_{\boldsymbol{y}\in Y}\boldsymbol{y}}{|Y|}\right\|_2 \tag{6.17}$$

Note that while we use averaging as the aggregation function here, the bounds derived for *mED* can be trivially extended to any linear aggregation function, not just the average or sum. This distance measure is a metric, operates on vector spaces, is permutation invariant (with averaging or sum aggregation), and operates on continuous data. It also does not require preprocessing. Euclidean distance is ubiquitous in the literature, serving as the de facto

standard distance measure across numerous fields including data mining, machine learning, pattern recognition, and signal processing [24, 26, 52, 69, 73, 81, 82, 158, 191, 230, 237, 239].

The initial step in implementing a measure involves bound derivation. A key aspect is determining if empirical bounds can be established for the measure, which enables several optimizations. As discussed in Section 6.4.4, we observe that *mED* can be expressed as a function of pairwise dot products $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ as follows:

$$mED(X,Y)^2 = \frac{\sum_{\boldsymbol{x}\in X}\sum_{\boldsymbol{x'}\in X}\langle \boldsymbol{x}, \boldsymbol{x'}\rangle}{|X|^2} + \frac{\sum_{\boldsymbol{y}\in Y}\sum_{\boldsymbol{y'}\in Y}\langle \boldsymbol{y}, \boldsymbol{y'}\rangle}{|Y|^2} - \frac{2 * \sum_{\boldsymbol{x}\in X}\sum_{\boldsymbol{y}\in Y}\langle \boldsymbol{x}, \boldsymbol{y}\rangle}{|X| * |Y|} \tag{6.18}$$

Then, following Theorem 6.4.1, empirical bounds on $mED(S_X, S_Y)$ for a cluster combination (S_X, S_Y) can be derived from the minimum and maximum dot products between vectors in the clusters:

$$LB^2_{eucl} = \frac{\sum_{C_X\in S_X}\sum_{C_{X'}\in S_X}\langle C_X, C_{X'}\rangle_{\min}}{|S_X|^2} + \frac{\sum_{C_Y\in S_Y}\sum_{C_{Y'}\in S_Y}\langle C_Y, C_{Y'}\rangle_{\min}}{|S_Y|^2} - \frac{2 * \sum_{C_X\in S_X}\sum_{C_Y\in S_Y}\langle C_X, C_Y\rangle_{\max}}{|S_X| * |S_Y|}$$

and

$$UB^2_{eucl} = \frac{\sum_{C_X\in S_X}\sum_{C_{X'}\in S_X}\langle C_X, C_{X'}\rangle_{\max}}{|S_X|^2} + \frac{\sum_{C_Y\in S_Y}\sum_{C_{Y'}\in S_Y}\langle C_Y, C_{Y'}\rangle_{\max}}{|S_Y|^2} - \frac{2 * \sum_{C_X\in S_X}\sum_{C_Y\in S_Y}\langle C_X, C_Y\rangle_{\min}}{|S_X| * |S_Y|}$$

where $LB_{eucl} \leq d_{eucl}(S_X, S_Y) \leq UB_{eucl}$, and $\langle C_X, C_Y\rangle_{\min}$ and $\langle C_X, C_Y\rangle_{\max}$ are the minimum and maximum dot products between vectors in clusters.

Considering the measure's properties and the derived bounds, its implementation via the measure interface is straightforward, as illustrated in Appendix H.1. Although the general measure interface itself may be complex, the implementation effort required from the user is minimal. The user only needs to implement two methods and optionally has to override certain properties within the constructor. Once the measure is implemented, the framework automatically applies the optimizations detailed in Table 6.3, based on the measure's characteristics and the availability of empirical bounds. Specifically for *mED*, the applicable optimizations include: precomputing the pairwise matrix (O2), skip first level (O4), result expanding (O5), cluster pair caching (O8), empirical bound discounting (O9), canonical candidate generation (O10), duplicate objects checks (O11), and branch prioritization (O12). Recall that the order of applying the optimizations is not important, as they are independent of each other, with the only exception being O4, which requires O2 to be applied first.

Total Correlation (*TC*)

Total Correlation (TC) is an information-theoretic measure from multivariate statistics that quantifies the total amount of information redundancy or dependency among a set of random variables. For a set of p random variables $X = \{X_1, \ldots, X_p\}$ (e.g., time series), *TC* is defined as the difference between the sum of the marginal entropies and their joint entropy:

$$TC(X) = \sum_{i=1}^{p} H(X_i) - H(X_1, \ldots, X_p) \tag{6.19}$$

where $H(X_i)$ is the marginal entropy of X_i, and $H(X_1, \ldots, X_p)$ is the joint entropy of all variables in X.

It is a one-sided measure, typically considered a similarity function (higher TC implies more shared information or structure). It is also monotonically increasing, and permutation invariant. It usually operates on discrete data; if the data is continuous, quantization is applied as a preprocessing step (cf. Table 6.3).

Empirical bounds can be derived for Total Correlation. Based on existing results in information theory, TC can be bounded using sums and differences of marginal, joint entropies as follows [109, 234, 251]:

$$\frac{1}{n-1} \sum_{i=1}^{p} \sum_{j=i+1}^{p} H(X_i, X_j) \leq TC(X) \leq \sum_{i=1}^{p} H(X_i) - \max_{1 \leq j < k \leq p} H(X_j, X_k) \tag{6.20}$$

Following Theorem 6.4.1, empirical bounds on $TC(CX)$ for a list of clusters $S = \{C_1, \ldots, C_p\}$ can be derived by substituting the entropy terms with their respective minimum or maximum values computed over the time series within the involved clusters.

The implementation of Total Correlation via the measure interface would define these calculations, as shown in Appendix H.2. As with other measures, the user primarily implements the compute and empirical bounds methods, and the framework automatically applies relevant optimizations from Table 6.3. For Total Correlation, these optimizations include: precomputing the pairwise matrix (O2), distance measure optimization (O3), skipping level 1 (O4), result expanding (O5), cluster pair caching (O8), empirical bound discounting (O9), canonical candidate generation (O10), duplicate objects checks (O11), and branch prioritization (O12). Furthermore, the 'preprocess' method is overridden to apply quantization to the data, which is necessary for TC to operate on continuous data.

Multivariate Manhattan Distance (*mMD*)

The Manhattan distance d_{manh}, also known as L_1 distance or taxicab geometry, is a metric that measures the distance between two vectors as the sum of the absolute differences of their values. We define its multivariate version *mMD* as a two-sided measure with summation as

the aggregation function:

$$mMD(X,Y) = \left\| \sum_{\boldsymbol{x}\in X} \boldsymbol{x} - \sum_{\boldsymbol{y}\in Y} \boldsymbol{y} \right\|_1 \tag{6.21}$$

where X and Y are sets of vectors.

It is a two-sided measure, uses summation as its aggregation function, is a metric, operates on vector spaces, is permutation invariant (with summation), and operates on continuous data. However, unlike *mED*, *mMD* is not soft-distributive, meaning we cannot derive empirical bounds for it. However, due to its metric properties, the framework can apply the general theoretical bounds in Theorem 6.4.5 to it.

As theoretical bounds are looser than empirical bounds, *mMD* is a good candidate for the use of dimensionality reduction techniques, in order to reduce the cost of distance and bound calculations. An appropriate dimensionality reduction technique for L_1 is random projection with the projection matrix generated from Cauchy distributions [93, 101, 122], and estimating the L_1 distance using the bias-corrected geometric mean estimator proposed by [146]. Random projections are a well-known technique for dimensionality reduction, where the original data is projected into a lower-dimensional space using a random matrix generated from a specific distribution [93, 122]. A key benefit of using random projections is that they often come with probabilistic guarantees that the pairwise distances in the lower-dimensional space will approximate the pairwise distances in the original space, with high probability [93, 101]. Furthermore, by increasing the size of the projection matrix, one can control the trade-off between the accuracy of the distance approximation, the computational cost of the projection and the dimensionality of the transformed data. In the case of L_1 distance, while the exact distances are not preserved in the projected space, we can accurately estimate them using the bias-corrected geometric mean estimator proposed by Li [146]. For two vectors projected to dimension m', the estimator $\hat{d}_{li}$ is defined as:

$$\hat{d}_{li}(\boldsymbol{x}) = \cos^{m'}\left(\frac{\pi}{2m'}\right) \prod_{j=1}^{m'} |\boldsymbol{x}_j|^{1/m'}$$

where $\boldsymbol{x}_j$ represents the j-th component of the projected vector $\boldsymbol{x}$, and the $\cos^{m'}$ term acts as a bias correction factor. This estimator provides an unbiased estimate of the original L_1 distance with provable error bounds that improve as m' increases.

The implementation via the measure interface is shown in Appendix H.3. Note that the 'cdist' method is overridden to use the standard Manhattan distance as the clustering distance, and the 'preprocess' method is overridden to apply random projections to the data. Furthermore, as the measure is metric itself rather than a function of a metric, implementation of 'dist2sim' or 'sim2dist' is not necessary. Through the general implementation of the adaptive 'bound' method (cf. Appendix G), the framework will recognize that no empirical bounds are available for this measure and will apply the theoretical bounds of Theorem 6.4.5 instead, using the general implementation. The optimizations applicable to the Multivariate

Manhattan Distance include: dimensionality reduction (O1), distance measure optimization (O3), result expanding (O5), centroid and radius caching (O7), canonical candidate generation (O10), duplicate objects checks (O11), and branch prioritization (O12).

6.4.8 Extending the Framework to Streaming Data

In Chapter 5, we introduced CDStream, an algorithm for efficiently detecting high *mPC* combinations in streaming data. Its core strategy is to maintain an index of Decisive Cluster Combinations (DCCs) and monitor only the underlying pairwise correlations that determine their bounds. A change in these correlations, caused by new data, signals a potential change in the DCC's status (e.g., from decisive to indecisive), which may alter the final result set. This trigger-based approach avoids a full re-computation over the sliding window, enabling efficient updates.

The effectiveness of this indexing strategy hinges on the soft-distributive property of the *mPC* measure. Because the multivariate correlation can be expressed as a function of the underlying pairwise correlations (cf. Theorem 5.3.2), we can predict how the *mPC* value will change based on a change in any of these pairwise statistics. This direct link allows the index to efficiently monitor for violations. To extend this concept to any similarity measure within the Similarity Detective framework, a similar approach could be adopted for all other empirical measures. If the similarity values of a measure are directly and monotonically proportional to the values of its underlying pairwise statistics, an index tracking these statistics would be effective. This is not the case for all measures; for instance, with the Multipole (*MP*) measure, the similarity depends on the smallest eigenvalue of the correlation matrix. This eigenvalue is not a monotonic function of each individual pairwise correlation in the matrix, meaning a change in a single pairwise value does not predictably alter the final *MP* score. Therefore, an index based on monitoring individual pairs would be ineffective for *MP*.

For theoretical measures that are not soft-distributive, the structure of the index would need to be rethought. Since their bounds are derived from the geometric properties of aggregate clusters – namely their centroids and radii (cf. Theorem 6.4.6) – the index could not rely on pairwise statistics. However, a similar "triggering mechanism" could still be designed. Instead of detecting changes in pairwise correlations, the index would need to detect significant changes in the composition of the aggregate clusters themselves. For instance, as new data points arrive and old ones expire within a sliding window, the contained area of a cluster may grow, shrink, or shift its position in the vector space. Such a change would alter the cluster's centroid and radius, thereby affecting the theoretical bounds of any DCC it is part of. An index designed to monitor these cluster-level properties could thus efficiently identify which DCCs require re-evaluation, providing a viable path for extending the streaming approach to non-empirical measures.

Notice that, as CDHybrid inherits the support of CDStream in terms of measures, it will automatically fall under the SD framework once we successfully complete this integration.

While these approaches provide a path forward, we leave the detailed exploration and development of this extension for future work.

Parameter	Stocks	fMRI	SLP	TMP	Crypto	Deep
Nr. of objects n	1440	1440	1440	1440	708	96
Nr. of time points m	1309	5470	1827	1827	2660	10,000
Top-k **value** k	100	100	100	100	100	100
Critical shrink factor γ	0	0	0	0	0	0
K-means κ	10	10	10	10	10	10
Dimensionality reduction ϵ	0.6	0.6	0.6	0.6	0.6	0.6
Dimensionality reduction δ	0.001	0.001	0.001	0.001	0.001	0.001

Table 6.4: Default parameters for the experiments

6.5 Evaluation

The purpose of this evaluation is threefold: (a) to assess the effectiveness of each of the twelve optimizations in the suite, (b) to compare the performance of our framework with other general-purpose baselines that can answer multivariate similarity queries with any measure, and (c) to analyze the sensitivity of the framework to different datasets, similarity measures, and parameters.

Baselines. The baselines include UNOPT and OPT (cf. Chapter 5), which are two variants of an exhaustive search algorithm that iterate over all possible combinations, and four contemporary database management systems (DBMS). To the best of our knowledge, there are currently no other general-purpose methods proposed for answering multivariate similarity queries with arbitrary measures. In this chapter, we do not compare with specialized algorithms built for specific measures, such as CONTRa [9] for the tripole measure and CoMEtExtended [10] for multipoles, as they were already extensively compared to CD in Chapter 5. Comparing to these methods here would be redundant, as our framework will yield the same results as CD for these measures.

Hardware and implementations. All experiments, except for the comparison with the DBMS systems, were executed on a server equipped with a 32-core 2.6 GHz AMD Rome 7H12 processor and 128 GB of RAM. Due to permission constraints on the server, the DBMS experiments were executed on another machine, with an Intel i7-10750H 12-Core 2.60GHz processor, 32GB RAM, running Ubuntu 22.04.1 LTS. All implementations, except the ones of two DBMS, were multi-threaded. Algorithm performance comparisons are exclusively made under matching execution styles (e.g., comparing single-threaded SD only to DBMS). The reported execution time for SD corresponds to the total execution cost, including normalization, clustering, pairwise matrix precomputation, etc. All reported results are medians from 10 repetitions.

Datasets. We conduct our evaluation on the same six datasets used in Chapter 5, with default parameters as shown in Table 6.4. Namely, the datasets include *Stocks*, *fMRI*, *SLP*, *TMP*, *Crypto*, and *Deep*.

Similarity and distance measures. We evaluate the framework with *Multivariate Pearson Correlation (*mPC*)*, *Multipole (*MP*)*, *Multivariate Euclidean Distance (*mED*)*, *Total correla-*

Table 6.5: Impact of each optimization in the framework on the top-k query time for different similarity measures, run on the *Stocks* dataset. The speedup is calculated as the ratio of the time taken by the algorithm without the optimization to the time taken with the optimization (i.e., leave-one-out comparison). The last three rows show the speedup of all optimizations combined compared to the baseline algorithms and the algorithm without any optimizations. Empty cells indicate that the optimization is not applicable to the corresponding similarity measure.

Optimization	**Speedup w.r.t. not using the optimization**		
	***mPC* (1,3) (empirical)**	***mMD* (1,2) (theoretical)**	***iMP* (4) (apriori)**
O1 Dimensionality Reduction		5.00	
O2 Precompute Pairwise Matrix	1.59		1.20
O3 Distance Measure Optimization	1.13	1.69	1.05
O4 Skip First Level	1.19		
O5 Result Expanding	1.13	1.04	
O6 Apriori Filtering			1274
O7 Centroid & Radius Caching		1.02	
O8 Cluster Pair Caching	84.2		1.70
O9 Empirical Bound Discounting	2.14		1.01
O10 Canonical Candidate Generation	4.50	1.83	1.67
O11 Duplicate Objects Check	1.27	1.02	1.07
O12 Branch Prioritization	2.42	1.36	1.03
All optimizations (vs. no optimizations)	508	22.6	4,998
All optimizations (vs. OPT)	952,259*	93.9	35,955
All optimizations (vs. UNOPT)	1,221,165*	88.9	40,200

* Estimated based on baseline's progress over 72 hours (i.e., extrapolated).

*tion (*TC*)*, *Multivariate Manhattan Distance (*mMD*)*, and *Inverse Multipole (*iMP*)*. Inverse Multipole is defined as $iMP(X) = 1 - MP(X)$, making it a monotonically *decreasing* function measuring the level of linear independence between the time series in X. The other measures are previously defined in Section 6.4.7 and as part of Chapter 5. With empirical bounds applicable to *mPC*, *MP*, *mED*, and *TC*, theoretical bounds applicable to *mMD*, and apriori filtering applicable to *iMP*, these measures were selected to evaluate the framework across the full spectrum of optimizations and bounding strategies. Again, we refer to the combination of a measure and maximum cardinality as a *query pattern* (e.g., $mPC(1, 2)$, $MP(3)$, etc.) for the sake of brevity.

Notably, all results presented in this evaluation were obtained without any manual tuning or optimization of the similarity and distance measures beyond implementing their definitions as described in this chapter.

The remaining of this section is organized as follows. We start with an ablation study of the optimizations in Section 6.5.1, followed by a sensitivity analysis of the framework in Section 6.5.2. Then, we compare the framework with the baselines in Section 6.5.3. As the framework currently only covers queries over static data, we do not include any experiments with streaming data in this evaluation.

6.5.1 Effect of Optimizations

To quantify the benefit of each optimization within the SD framework, we conducted an ablation study. The results, presented in Table 6.5, showcase the speedup achieved by each of the twelve optimizations for three query patterns: *mPC* (1,3) using empirical bounds, *mMD* (1,2) using theoretical bounds, and *iMP* (4) using empirical bounds and enabling apriori filtering. The experiments were performed on the *Stocks* dataset using 3000, 3000, and 1440 time series for the *mPC*, *mMD*, and *iMP* experiments, respectively. These dataset sizes and query patterns were selected as they ensure that the impact of each optimization could be clearly observed, while allowing both the unoptimized SD and the baseline algorithms to generally finish within our 72-hour time limit.

A key observation from Table 6.5 is that every optimization contributes positively to query performance, with all individual speedups being greater than 1.0. The magnitude of this impact, however, varies considerably across different optimizations and similarity measures, highlighting the importance of having a diverse set of optimizations that target different scenarios. For instance, Apriori Filtering (O6) yields a remarkable 1274x speedup for *iMP* (4), as this measure's monotonically decreasing nature allows for aggressive pruning. Furthermore, while relatively ineffective for *iMP* (4), Cluster Pair Caching (O8) provides a staggering 84.2x speedup for *mPC* (1,3), demonstrating the effectiveness in reusing intermediate statistics between empirical bound calculations. Similarly, Dimensionality Reduction (O1) is particularly beneficial for *mMD* (1,2), offering a 5.00x speedup by reducing the cost of distance computations. It should be noted that this optimization introduces a controlled accuracy trade-off: using parameters $\epsilon = 0.6$ and $\delta = 0.001$ for the random projection, we achieve this significant speedup while maintaining a practical recall of approximately 0.74 for the top-k results. Other optimizations, such as Distance Measure Optimization (O3) or Canonical Candidate Generation (O10), provide more modest yet consistent gains across applicable measures, typically ranging from 1.02x to 4.50x.

When all optimizations are enabled, the framework achieves substantial performance improvements over the algorithm without optimizations, with speedups of 508x for *mPC* (1,3), 22.6x for *mMD* (1,2), and an impressive 4,998x for *iMP* (4). This demonstrates the powerful cumulative effect of the optimization suite. In fact, for measures like Multivariate Manhattan Distance and Inverse Multipole, the total speedup achieved by enabling all optimizations is remarkably close to the product of the individual speedups listed in the table. For *mMD* (1,2), the product of individual speedups (e.g., $5.00 \times 1.69 \times \cdots \times 1.36$) is approximately 22.8, closely aligning with the observed combined speedup of 22.6. Similarly, for *iMP* (4), the product of its applicable individual speedups is approximately 5073, compared to the actual combined speedup of 4998. This suggests that, for these measures, the optimizations largely act independently and their benefits multiply, indicating minimal negative interference and often a synergistic relationship where they increase each other's effectiveness. While this direct multiplicative effect is less apparent for *mPC* (product of ~ 6021 vs. actual 508x), likely due to the dominant effect and potential overlaps of optimizations like Cluster Pair Caching (O8) and Canonical Candidate Generation (O10), the overall theme of compounded gains remains clear.

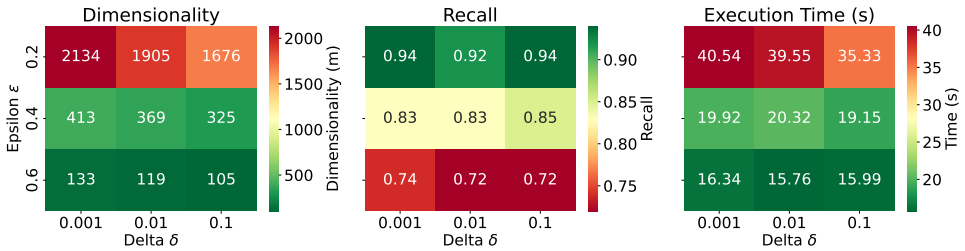


Figure 6.7: Effect of dimensionality reduction parameters on the accuracy and efficiency of SD on a top-k *mMD* (1,2) query on the *fMRI* dataset with $n = 1440$ time series. Epsilon ϵ and delta δ refer to the parameters of the probabilistic error bounds of Johnson Lindenstrauss random projection.

The most striking results emerge when comparing the fully optimized framework against the exhaustive search baselines, UNOPT and OPT. For the *mPC* (1,3) query, SD is over 950,000 times faster than OPT and over 1.2 million times faster than UNOPT. These baselines were estimated to require 135-173 days to complete the query, whereas SD delivered the results in just 5 seconds. This dramatic acceleration, being roughly six orders of magnitude, has profound implications. It implies that complex multivariate similarity queries, which were previously computationally infeasible due to prohibitive runtimes, now become practical and accessible. This capability to tackle previously intractable problems highlights the significant advancement offered by the framework.

Summary. The ablation study demonstrates that all twelve optimizations in the SD framework contribute positively to performance, with varying impacts depending on the similarity measure and its properties. Collectively, these optimizations yield speedups of one to three orders of magnitude over an unoptimized version of SD. For certain measures, the combined speedup approaches the product of individual speedups, indicating a strong synergistic effect. Most notably, when compared to exhaustive search baselines, the fully optimized SD achieves speedups of up to six orders of magnitude, making previously infeasible multivariate similarity search queries tractable.

6.5.2 Sensitivity Analysis

This section investigates the sensitivity of the SD framework to various query configurations and parameters, focusing primarily on top-k queries with $k = 100$ (see Table 6.4 for other default parameters). We analyze how its performance and accuracy are influenced by the parameters of the dimensionality reduction optimization, its robustness when applied to diverse datasets, and its scalability with respect to different query patterns. Understanding these aspects is crucial for evaluating the framework's practical applicability and performance across different scenarios.

While our analysis centers on top-k queries, the findings generally extend to threshold queries as well, provided the threshold is sufficiently selective. top-k queries are particularly suitable for this evaluation as they guarantee the use of an effective pruning threshold that naturally

emerges from the k-th best result found so far. This ensures meaningful performance comparisons across different datasets and similarity measures. In contrast, threshold queries with poorly chosen (i.e., insufficiently selective) thresholds could yield result sets with thousands of combinations, a scenario that is neither interesting nor realistic in most real-world applications.

Dimensionality Reduction

For certain distance measures, such as *mMD*, the Dimensionality Reduction optimization employs techniques like random projection to reduce the dimensionality of the data while approximately preserving the pairwise distances between time series. In the case of Cauchy random projections for Manhattan Distance, the number of dimensions in the projected space is determined by a variant of the Johnson-Lindenstrauss lemma corresponding to the unbiased geometric mean estimator of [146]. This lemma depends on two key parameters, ϵ and δ, which define the desired error bounds $P(|d - \hat{d}| > \epsilon \cdot d) < \delta$, where d is the true distance and $\hat{d}$ is the estimated distance in the reduced space. The number of required dimensions m' is given by $m' \geq G_{gm} \frac{(2 \log m - \log \delta)}{\epsilon^2}$, where m is the number of original dimensions and $G_{gm} = \max\{G_{L,gm}, G_{R,gm}\}$ with $G_{L,gm} = \frac{\pi^2}{2}\left(1 - \epsilon + \left(\frac{1}{12} + \frac{2}{3\pi^2}\right)\epsilon^2 + \ldots\right)$ and $G_{R,gm} = \frac{\pi^2}{2}\left(1 + \epsilon + \left(\frac{1}{12} + \frac{2}{3\pi^2}\right)\epsilon^2 + \ldots\right)$, where both expressions are Taylor series expansions around $\epsilon = 0$ [146]. Adjusting ϵ and δ allows for a trade-off: higher values reduce the number of dimensions, thereby decreasing the computational cost of distance calculations and improving algorithm performance, but at the expense of accuracy.

To analyze this trade-off, we evaluated the effect of varying $\epsilon \in \{0.2, 0.4, 0.6\}$ and $\delta \in \{0.001, 0.01, 0.1\}$ on both the accuracy (i.e., recall of the top-k set) and efficiency (execution time) of SD. The experiment was conducted using a top-k *mMD* (1,2) query on the *fMRI* dataset with $n = 1440$ time series. The results are presented in Figure 6.7. The heatmaps clearly show that ϵ has a substantially larger impact on the dimensionality of the random projection, and consequently on both recall and execution time, compared to δ. For instance, increasing ϵ from 0.2 to 0.6 (while keeping $\delta = 0.001$) reduces the number of projected dimensions from 2134 to 133. This significant reduction in dimensionality leads to a considerable performance gain, speeding up the algorithm by approximately 60%. However, this speedup comes with a decrease in recall from 0.94 to 0.74, for $\epsilon = 0.2$ and $\epsilon = 0.6$, respectively.

Importantly, while the dimensionality reduction's error bound of $\epsilon = 0.6$ may seem large, the actual impact on SD's accuracy is much more modest, with the recall only dropping from 0.94 to 0.74. This demonstrates that distance estimation errors do not fully propagate to the final results, as explained in Section 6.4.5. The framework's bounding mechanisms help maintain reasonable accuracy even with aggressive dimensionality reduction.

The question of whether this trade-off is acceptable depends on the specific application requirements. In many scenarios, a 60% improvement in execution time might be highly valuable, especially if a recall of 0.74 is still sufficient for the task at hand. Given that the drop in recall is not excessively steep relative to the performance gain, this trade-off can be con-

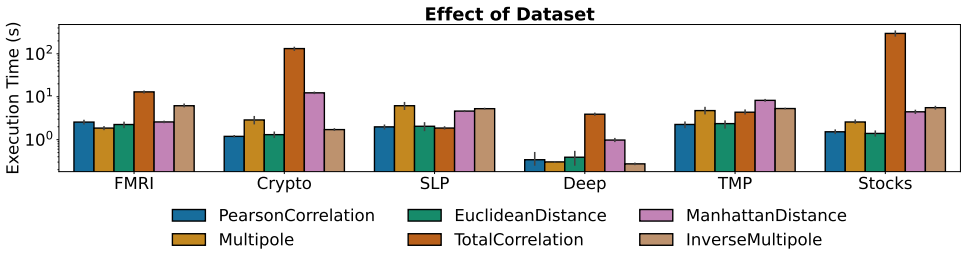


Figure 6.8: Performance of SD on top-k queries across different similarity measures and datasets, using a query pattern (1,2) for two-sided measures, and (3) for one-sided measures.

sidered reasonable for applications where speed is critical. For the subsequent experiments in this chapter, including the ablation study presented in Table 6.5, we adopted $\epsilon = 0.6$ and $\delta = 0.001$ when utilizing the dimensionality reduction optimization, balancing computational efficiency with a practical level of accuracy.

Different Datasets

To assess the robustness of the SD framework, we evaluate its performance across the six datasets detailed previously, using a range of similarity measures. Figure 6.8 presents the execution times for top-k queries, employing a (1,2) query pattern for two-sided measures (*mPC*, *mED*, *mMD*) and a (3) pattern for one-sided measures (*MP*, *TC*, *iMP*).

The results generally indicate stable performance of SD across different datasets and similarity measures. A noticeable exception is the *Deep* dataset, where queries are considerably faster. This is primarily due to its smaller size, containing only 96 time series compared to the 1440 time series in most other datasets, which naturally leads to a reduced search space. For Total Correlation (*TC*), however, performance outliers are observed, particularly for the *Crypto* and *Stocks* datasets. Analysis of the *TC* value distributions for these datasets revealed gamma-like distributions with a mean very close to the correlation of k'th result in the top-k. This characteristic implies that *TC* is not sufficiently discriminating on these specific datasets, leading to less effective pruning by the bounding mechanisms, similar to what we observed for Multipole on certain datasets in Chapter 5. Such situations could potentially be mitigated by performing an initial exploratory analysis on the correlation distribution of a small data sample. If this analysis does not indicate sufficiently high or discriminative correlations, the data analyst should consider switching to an alternative similarity measure that better captures the relevant and meaningful relationships in their data.

Interestingly, *mMD* demonstrates competitive performance, often comparable to measures that benefit from empirical bounds (e.g., *mPC*, *mED*, *MP*), despite relying on theoretical bounds itself. This efficiency can be largely attributed to the effectiveness of the Dimensionality Reduction (O1) optimization ($\epsilon = 0.6, \delta = 0.001$), which significantly reduces the cost of distance computations for *mMD*, as seen in earlier experiments. Overall, while specific dataset characteristics can influence performance for certain measures, the framework exhibits a good degree of robustness, delivering efficient query execution across a variety of data domains and similarity measures.

Table 6.6: Execution times of SD with different query patterns on top-k queries (seconds)

	mED					*MP*		*mPC*				
	(1,2)	**(1,3)**	**(2,2)**	**(1,4)**	**(2,3)**	**(3)**	**(4)**	**(1,2)**	**(1,3)**	**(2,2)**	**(1,4)**	**(2,3)**
fMRI	2	5	11	89	12	1	2	2	8	11	1690	7100
Stocks	1	2	4	28	94	2	24	2	4	3	311	613

	TC		*mMD*			*iMP*		
	(3)	**(4)**	**(1,2)**	**(1,3)**	**(2,2)**	**(3)**	**(4)**	**(5)**
fMRI	11	111	2	3	3	6	6	7
Stocks	12	1570	4	59	40	5	5	6

Query Patterns

Table 6.6 presents execution time of SD for different query patterns. As expected, increasing the complexity of the query pattern leads to an increase of the computational time. However, even though the size of the search space follows $O\left(\binom{n}{p_l+p_r}\right)$, execution time of SD grows at a much slower rate. Indicatively, for the fMRI dataset, the search space size grows 5 orders of magnitude between $mED(1,2)$ and $mED(2,3)$, whereas SD's execution time increases by only one order of magnitude, indicating efficient pruning of the search space and the high effectiveness of the optimizations in the suite.

Particularly, the relatively modest increases in execution time for measures like *mED*, *MP*, *mMD*, and *iMP*, even for more complex patterns, can be attributed to the rapid increase of the running top-k threshold over the complexity levels. For instance, with Multipole (*MP*), the threshold often approaches 0.999 after evaluating just the first two complexity levels (i.e. pairs and triplets). This high threshold enables SD to prune the search space very aggressively at higher complexity levels, often eliminating over 99.9% of candidate combinations high up in the search tree. The relatively lower execution times for *TC* and *mMD* on the fMRI dataset compared to the Stocks dataset can be explained by the fMRI dataset having a larger amount of highly correlated pairs. This results in a higher initial top-k threshold when moving to more complex patterns (e.g., triplets and quadruplets), thereby enhancing pruning efficiency.

In summary, the analysis of different query patterns reveals that while execution time naturally increases with pattern complexity, SD's growth rate is substantially slower than the combinatorial explosion of the search space. This efficiency stems from effective pruning strategies and optimizations like Result Expanding (O5), Bound Discounting (O9), and Branch Prioritization (O12), which involve prioritizing the most promising combinations within the search space first in order to bring the top-k threshold as close to its final value in the early stages of the search, maximing pruning opportunities.

Summary. The sensitivity analysis reveals the key characteristics of the SD framework. Dimensionality reduction provides a configurable trade-off between speed and accuracy, controlled primarily by the ϵ parameter. Performance can vary across datasets, influenced by factors such as dataset size and the distribution of similarity values for specific measures. Despite these variations, SD demonstrates efficient scaling with increasing query pattern complexity, with runtime growing significantly slower than the

```
WITH corrs(vid1, vid2, corr) AS (
    SELECT v1.vid, v2.vid, pearson(v1.vec, v2.vec)
    FROM fMRI v1, fMRI v2
    WHERE v1.vid < v2.vid),
pc12(vid1, vid2, vid3, mcorr) AS (
    SELECT c12.vid1, c12.vid2, c13.vid2,
    (c12.corr + c13.corr) / SQRT(2 + 2*c23.corr)
    FROM corrs c12, corrs c13, corrs c23
    INNER JOIN c12 c13 ON c12.vid1 = c13.vid1
    INNER JOIN c12 c23 ON c12.vid2 = c23.vid1
    INNER JOIN c13 c23 ON c13.vid2 = c23.vid2
    WHERE c12.vid1 != c23.vid1 AND c12.vid1 != c23.vid2)
SELECT * FROM pc12 ORDER BY mcorr DESC LIMIT 10
```

Figure 6.9: $mPC(1, 2)$ top-10 query, implemented with SQL. The pearson correlation is implemented as a stored function.

```
SELECT v1.vid, v2.vid, v3.vid,
manhattan(eavg(v1.vec, v2.vec), v3.vec) AS mmd
FROM fMRI v1, fMRI v2, fMRI v3
WHERE v1.vid < v2.vid AND v1.vid != v3.vid AND v2.vid != v3.vid
ORDER BY mmd DESC LIMIT 10
```

Figure 6.10: $mMD(1, 2)$ top-10 query, implemented with SQL. The manhattan distance manhattan and the element-wise average eavg methods are implemented as stored functions.

search space, highlighting the effectiveness of its pruning strategies and optimizations. These findings illustrate the framework's ability to handle diverse queries while maintaining practical execution times, though careful consideration should be given to the choice of similarity measure for each specific dataset.

6.5.3 Comparison to Baselines

To contextualize the performance of the SD framework, it is essential to compare it against other systems capable of executing multivariate similarity queries with arbitrary similarity measures. Currently, the only alternatives that offer such flexibility are exhaustive search algorithms, either implemented explicitly (such as the UNOPT and OPT baselines) or implicitly through the execution of SQL queries within contemporary Database Management Systems (DBMS). This section evaluates SD against these baselines to demonstrate its relative efficiency and scalability.

Multivariate similarity search can be expressed as an SQL query, as shown in Figures 6.9 and 6.10, which demonstrates a $mPC(1, 2)$ and $mMD(1, 2)$ top-k query on a (z-normalized) table named "fMRI" in SQL. This allows us to compare SD's performance with general-

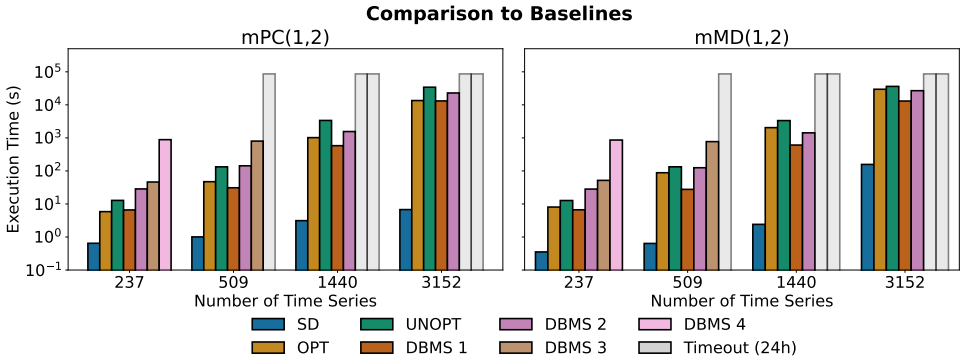


Figure 6.11: Comparison of SD with general-purpose baselines on the *fMRI* dataset with different similarity measures.

purpose state-of-the-art RDBMS. Our experiment used four off-the-shelf databases, all configured with RAM-stored tables for fair evaluation given SD's RAM usage. DBMS1 and DBMS3 supported array attributes, allowing us to develop array functions for Pearson correlation calculation. The other DBMSs stored data in long format (with columns corresponding to a primary key, time series id, time, and value), using a GROUP BY clause for Pearson correlation and Manhattan distance calculations. To maximize the performance of the SQL queries, we created indexes on the time series id columns for databases with long-format data. Indexes were not created for any other columns, as they did not significantly impact query performance. Due to limited multi-threading support, all approaches including SD ran single-threaded with a 24-hour query limit.

Figure 6.11 shows the execution times for each system to detect $mPC(1, 2)$ and $mMD(1, 2)$ on different resolutions of the fMRI dataset. The OPT and UNOPT baselines are also included in this analysis. The reported DBMS times exclude the one-off costs dataset loading and index creation. SD outperforms all baselines by several orders of magnitude. For *mPC*, the performance gap widens as dataset size increases, while for *mMD*, it remains relatively stable though still substantial. The time complexity for all baselines appears to follow $O(n^3)$ (where n is the number of time series), indicative of their triple nested loop execution strategy for this type of query. In contrast, SD's execution time grows at a much slower rate. The results suggest that all DBMS effectively perform an exhaustive search, exploring the entire search space, which accounts for their comparable performance characteristics, with OPT and UNOPT often performing similarly to or slightly better than some DBMS configurations.

Summary. Compared to general-purpose baselines, SD demonstrates a performance advantage of several orders of magnitude. While these baselines exhibit $O(n^3)$ complexity for the evaluated queries, SD's advanced optimizations result in significantly better scaling. This highlights SD's ability to make complex multivariate similarity queries practical for large datasets, a task that remains computationally intractable for existing general-purpose systems due to their exhaustive nature.

6.6 Vision

As discussed in Section 6.1, the current landscape of multivariate similarity search is fragmented, with domain experts implementing their own specialized solutions. While the SD framework provides a unified theoretical foundation and implementation to address this fragmentation, its current form as a software library primarily caters to users with programming expertise. To truly serve as a universal replacement for domain-specific implementations and foster adoption across disciplines, the framework needs to become more accessible to domain experts who may not possess extensive programming knowledge.

The SD framework, as presented, is implemented as a software library. Users interact with it by defining their custom similarity or distance measures through the MEASURE interface (detailed in Section 6.4.6) and by programmatically defining their query though the library's methods for top-k, threshold, or progressive queries. The library then executes these queries, automatically applying optimizations derived from the measure's declared properties to ensure efficient execution.

While this programmatic approach offers flexibility and power, particularly for developers integrating the framework into larger systems, we envision several ways to enhance its accessibility and promote broader adoption across different domains. A key direction for future work is the development of a graphical user interface (GUI) built on top of the existing library. Such a UI could empower users to define their queries and specify measure properties through intuitive elements like drop-down menus for query types, sliders for parameters (e.g., k for top-k, τ for threshold), and radio buttons for boolean measure properties. For defining the core measure logic or empirical bounds, a dedicated text box could allow users to input code in a high-level, accessible language like Python or LaTeX. This approach would significantly lower the barrier to entry for users unfamiliar with programming or those who prefer a more visual interaction with the system. Figure 6.12 illustrates a conceptual mockup of such an interface, showcasing how users might interact with these components to construct and execute a query.

The development of such a user-friendly layer is primarily an engineering task. The current SD framework already incorporates the necessary theoretical foundations and a reference implementation of the core logic, as demonstrated throughout this chapter. The proposed interface would serve as a user-friendly abstraction over this robust core, facilitating easier integration with existing data analysis pipelines and promoting the use of principled multivariate similarity search in a wider range of applications.

Beside query construction, visualization of query results also plays a crucial role in making multivariate similarity search more accessible. Interactive visualizations can help users understand the complex relationships discovered by their queries. Figure 6.13 shows a working implementation of an interactive graph visualization built using the Python PYVIS library, displaying results from a *mPC* (1,2) top-k query over financial time series data. In this interactive visualization, nodes represent individual time series, while edges indicate their contribution to high-order correlations. The color coding (green for left-side patterns, red for right-side patterns) helps users quickly identify the role of each time series in the discovered patterns. Users can interact with the visualization by hovering over nodes to see details,

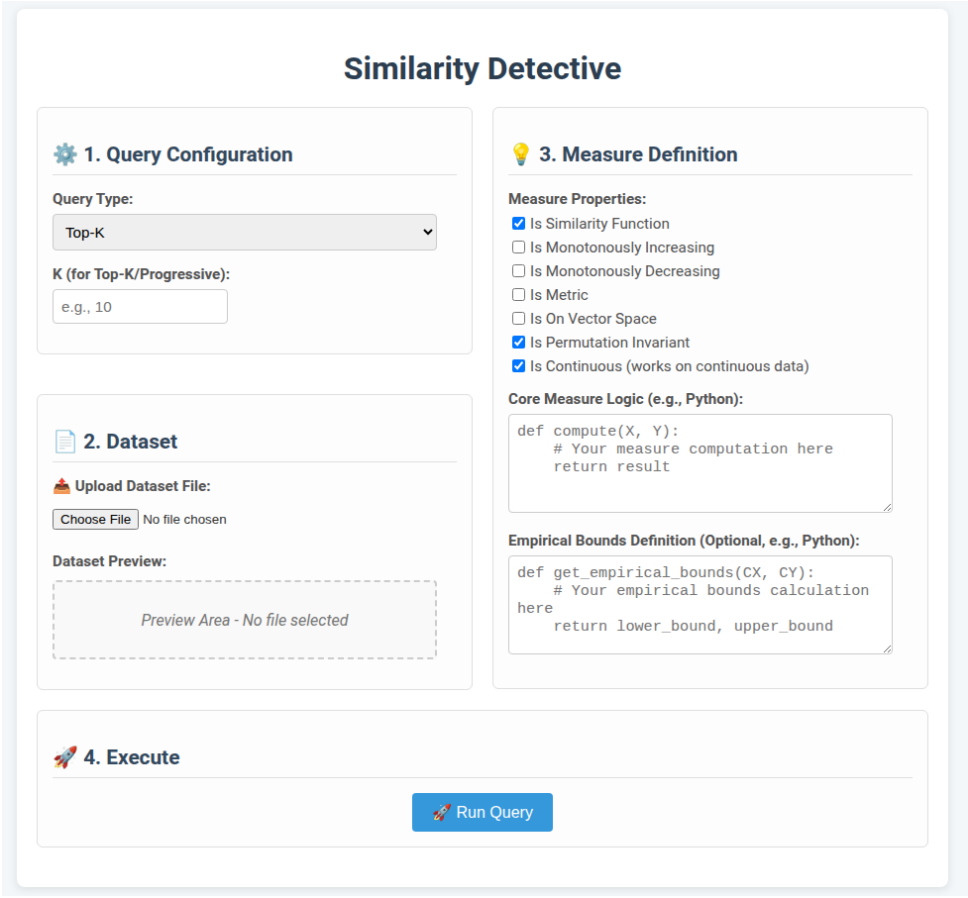


Figure 6.12: Conceptual mockup of a graphical user interface for the SD framework, illustrating how users might interact with the system to define queries and measures.

zooming to focus on specific clusters, dragging nodes to rearrange the graph, or filtering results based on various criteria. This visualization in Figure 6.13 sources from a prototype that was integrated in CD in an effort to make the results more interpretable for domain experts in the agricultural sector (as part of an EU-funded project). Future enhancements to this visualization could include (a) integrating it into the SD framework as well, (b) allowing users to customize the visualization parameters (e.g., color schemes, layout algorithms), (c) providing export options for sharing or further analysis, and (d) integrating it into the proposed GUI for a seamless user experience.

Ultimately, these future directions in query interfaces and result visualization aim to transform the SD library into a comprehensive and accessible system for multivariate similarity search, empowering a broader audience to harness its capabilities. Figure 6.12 illustrates a conceptual mockup of such an interface, showcasing how users might interact with these components to construct and execute a query.

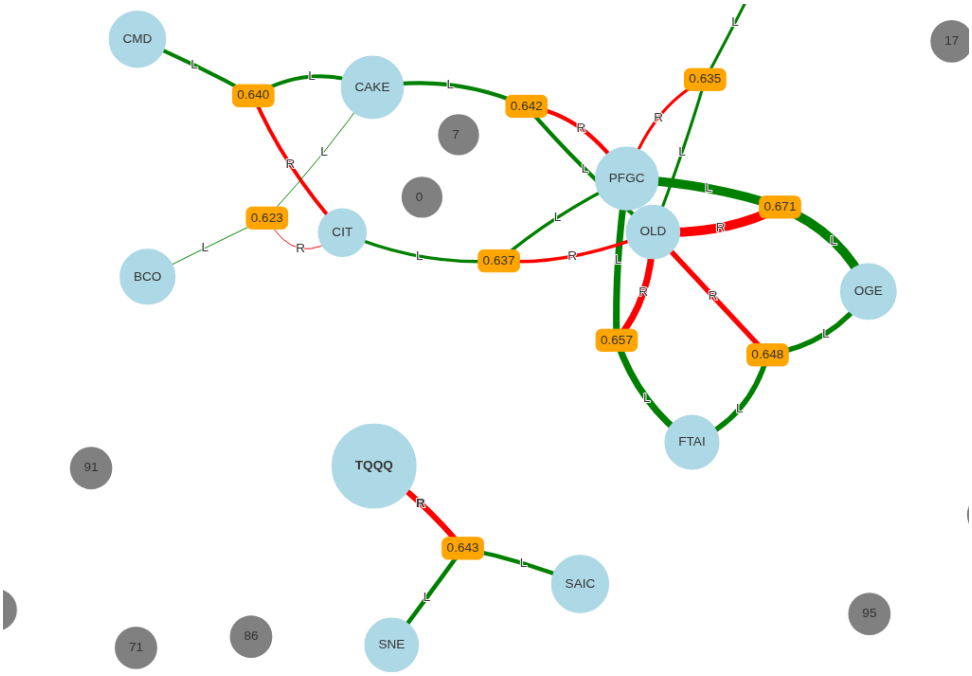


Figure 6.13: Example interactive graph interface of the results of a *mPC* (1,2) top-k query over a dataset of stock prices, implemented in Python using the pyvis library. The graph displays the top-k pairs of time series with the highest Pearson correlation, where nodes represent time series and edges represent contribution to a high-order correlation. Green and red edges indicate that the time series is located on the left-side (resp. right-side) of the query pattern.

6

6.7 Reflection and Conclusions

In this chapter, we addressed the need for a unified approach to multivariate similarity search by presenting the SD framework. Building on the specialized algorithms from the previous chapter, this work generalizes their underlying principles to create a versatile and efficient system applicable to any multivariate similarity (or distance) measure.

Chapter summary. We introduced the SD framework, a generic system designed to address the fragmented nature of existing domain-specific methodologies. A key aspect of the framework is its standardized measure interface, which formalizes the minimally required properties and methods for any measure to be integrated into the system. This interface facilitates easy integration of new user-defined measures, which is key for the framework's flexibility and extensibility. Beyond its practical implementation, the framework also serves as a standardization of the problem itself, aiming to unify the terminology and concepts used across the different applications of multivariate similarity search.

The framework's core query execution strategy employs a level-wise exploration of the combinatorial search space. This process is significantly accelerated by hierarchical clustering of the dataset and the application of recursive bounding techniques to combinations of these

clusters, enabling aggressive pruning of irrelevant candidates. The modularity of this workflow allows for the integration of various effective optimization strategies from existing literature. These optimizations can be automatically applied based on measure properties and query parameters. This leads to substantial performance gains, rendering previously intractable queries feasible.

Transferable insights. The SD framework offers a robust, adaptable, and efficient solution to the general problem of multivariate similarity search. Beyond its practical implementation, the framework serves as a standardization of the problem itself, unifying terminology and concepts across different applications. It empowers researchers and practitioners to apply multivariate similarity search to their datasets in a domain-appropriate manner, without needing to develop custom solutions from scratch. This can catalyze new discoveries in fields where data complexity has been a computational barrier.

Limitations and path forward. Currently implemented as a software library, the SD framework requires users to define custom measures and queries programmatically. While powerful, this approach has a steep learning curve. The vision for the framework involves improving its usability and accessibility, potentially through a higher-level declarative query language or a graphical user interface. Furthermore, while the framework is generic, the performance of specific optimizations can still be measure-dependent, requiring expert knowledge for fine-tuning in novel applications (e.g., selecting appropriate parameters for dimensionality reduction).

Reflection on research question. This chapter represents a crucial milestone in answering our central research question by unifying and generalizing the advances made in the previous chapter. Where Chapter 5 demonstrated the feasibility of a relationship-level extension through a specialized solution, the SD framework proves that this extension can be systematically handled within a single, coherent paradigm. By successfully bridging both the data-level and relation-level extensions while maintaining efficiency, we have effectively shown that the traditional similarity search paradigm can indeed be extended to fully embrace the multivariate nature of real-world data.

CHAPTER 7

Case Study: Sparse Representation

A general-purpose framework is only as valuable as its ability to solve real-world problems. Recall that the multivariate similarity search problem of CD already showed its practical relevance in neuroscience [10] and meterology [9, 150]. In this chapter, we aim to extend this demonstration of practical relevance by applying the more general SD framework to address a different, well-established and computationally hard problem in signal processing and machine learning: sparse representation. We demonstrate how this classic optimization problem can be elegantly reformulated as a multivariate similarity search query. Through a comprehensive empirical evaluation, we show that our general-purpose framework, without any domain-specific tuning, achieves performance competitive with – and in some cases superior to – specialized, state-of-the-art algorithms. This case study not only validates the power and versatility of the SD framework but also illustrates its potential to offer new, efficient solutions to long-standing problems in other domains.

7.1 Introduction

This chapter illustrates the application of SD to the problem of sparse representation, an important task in signal processing and machine learning with numerous practical applications. The goal of this case study is to demonstrate how our general-purpose multivariate similarity search framework can be effectively applied to solve a specific combinatorial problem that has traditionally been addressed with specialized algorithms.

The chapter is organized as follows: we first present the background and problem definition of sparse representation, followed by an overview of current approaches and their limitations. We then introduce our SD-based solution, detailing how the framework is configured to address this problem. Finally, we present an empirical evaluation comparing SD against

baseline methods across multiple dimensions and demonstrate its effectiveness through adversarial examples specifically designed to challenge greedy algorithms.

It is worth emphasizing that our evaluation is not intended to establish superiority over all specialized algorithms in this field. The sparse representation community has developed numerous domain-specific methods with various optimizations beyond those we compare against. Rather, our evaluation demonstrates how our framework enables the creation of a fast and effective algorithm for this combinatorial problem while achieving competitive performance compared to common baseline approaches that serve as reference points.

7.2 Background and Problem Definition

Sparse representation, also known as sparse coding, sparse approximation, or atomic decomposition [67, 223], is a core task in signal processing and machine learning [252]. The problem involves representing a signal or data vector as a linear combination of a minimal number of basis elements (atoms) selected from a large, often overcomplete, dictionary or reference dataset [52, 236, 237, 252]. Formally, given a signal vector $\boldsymbol{q} \in \mathbb{R}^m$ and a dictionary matrix $\boldsymbol{D} = [\boldsymbol{d}_1, \boldsymbol{d}_2, ..., \boldsymbol{d}_n] \in \mathbb{R}^{n \times m}$, where typically $n > m$ (making the dictionary *overcomplete*, meaning it contains more basis vectors than dimensions), the goal is to find a coefficient vector $\boldsymbol{x} \in \mathbb{R}^n$ such that $\boldsymbol{q} \approx \boldsymbol{x^T D}$. Crucially, $\boldsymbol{x}$ must be sparse, meaning it contains very few non-zero entries [52, 237].

Alternatively, the problem can be formulated as an optimization problem where one aims to minimize the count of non-zero coefficients, subject to a constraint on reconstruction error. Specifically, we minimize the L_0 pseudo-norm of $\boldsymbol{x}$, denoted $||\boldsymbol{x}||_0$, which counts the non-zero elements (positive or negative) in $\boldsymbol{x}$:

$$\min_{\boldsymbol{x}} ||\boldsymbol{x}||_0 \quad \text{subject to} \quad ||\boldsymbol{q} - \boldsymbol{x}^T \boldsymbol{D}||_2 \leq \epsilon \tag{7.1}$$

Here, ϵ is a small tolerance parameter defining the permissible reconstruction error. An alternative formulation minimizes the reconstruction error subject to an explicit sparsity constraint:

$$\min_{\boldsymbol{x}} ||\boldsymbol{q} - \boldsymbol{x}^T \boldsymbol{D}||_2 \quad \text{subject to} \quad ||\boldsymbol{x}||_0 \leq p \tag{7.2}$$

Where p denotes the maximum allowable number of non-zero coefficients.

Sparse representation plays a key role in various applications, including image compression and denoising [106, 236, 237, 242], audio processing [113, 162, 200, 252], medical imaging (particularly MRI reconstruction) [161, 255], feature extraction and selection in machine learning [113, 154], mass-spectroscopy in analytical chemistry [117], and portfolio optimization in computational finance [201]. The foundational principle that explains this wide range of applications across domains is that many natural signals can be efficiently represented or closely approximated using only a few components from an appropriate basis (i.e., dictionary), enabling more efficient storage and transmission of the data, and explaining the signal as part of analysis [252].

7.3 Current Approaches

Methods for constructing sparse representations typically fall into two main categories: *exact* techniques, which guarantee finding optimal solutions (within their problem formulation), and *approximate* techniques that trade off solution quality for efficiency. An important distinction between these approaches lies in their solution completeness: exact methods aim to find the globally optimal (sparsest) solution within their problem formulation (e.g., L_1 relaxation), while approximate methods focus on finding a feasible solution that may or may not be optimal. We now briefly discuss these two approaches and their most prominent algorithms, highlighting their relative strengths and weaknesses.

7.3.1 Exact Methods

The main approach for finding exact solutions involves convex optimization techniques. This class of methods is built on the observation that directly minimizing the L_0 norm is computationally intractable. Particularly, prominent works in the field [52, 237] prove that in the case of overcomplete dictionaries, the problem boils down to finding the sparsest solution of an underdetermined system of linear equations, which is known to be NP-hard [41, 70, 223], and difficult to even approximate [12].

To overcome this challenge, a common approach relaxes the L_0 norm constraint to its nearest convex counterpart, the L_1 norm, defined as $||\boldsymbol{x}||_1 = \sum_i |x_i|$. This transforms the problem into a convex optimization task that can be efficiently solved using standard methods like linear programming solvers, coordinate descent, proximal gradient descent, or interior point methods [41, 52]. Importantly, theoretical guarantees show that under specific conditions on the dictionary $\boldsymbol{D}$, particularly the Restricted Isometry Property (RIP) [42], the L_1-minimization solution coincides with the original L_0-minimization solution [41, 42, 76].

Several prominent algorithms leverage this L_1 relaxation approach:

1. **Basis Pursuit** (BP) [52] frames the problem as finding the minimum L_1 norm solution that achieves perfect signal reconstruction:
$$\min_{\boldsymbol{x}} ||\boldsymbol{x}||_1 \quad \text{subject to} \quad \boldsymbol{q} = \boldsymbol{x}^T \boldsymbol{D} \tag{7.3}$$

2. **Basis Pursuit Denoising** (BPDN) [52] extends BP by relaxing the exact reconstruction requirement to accommodate noisy signals:
$$\min_{\boldsymbol{x}} ||\boldsymbol{x}||_1 \quad \text{subject to} \quad ||\boldsymbol{q} - \boldsymbol{x}^T \boldsymbol{D}||_2 \leq \epsilon \tag{7.4}$$

3. The **Least Absolute Shrinkage and Selection Operator** (LASSO) [225], while closely related to BPDN, is typically formulated as a regression problem with L_1 regularization:
$$\min_{\boldsymbol{x}} \frac{1}{2}||\boldsymbol{q} - \boldsymbol{x}^T \boldsymbol{D}||_2^2 + \lambda||\boldsymbol{x}||_1 \tag{7.5}$$
Here, λ is a regularization parameter controlling the balance between reconstruction error and solution sparsity.

Despite their similarities, these methods have important differences. BP and BPDN, under appropriate conditions, can guarantee finding the sparsest possible (L_1) solution. LASSO's solution may slightly deviate due to the influence of the regularization parameter λ, which requires careful tuning. Computationally, BP and BPDN often rely on general-purpose linear programming solvers, which may be less efficient than the specialized algorithms commonly used for LASSO.

7.3.2 Approximate Methods

The main strategy for finding approximate solutions involves greedy algorithms. These iterative methods build the solution by progressively selecting dictionary atoms that best account for the signal or its residual at each stage [52, 70, 195]. Their heuristic nature means they generally cannot guarantee finding the globally optimal sparse solution, but they offer significant computational advantages [52].

The most well-known greedy algorithm is **Orthogonal Matching Pursuit** (OMP) [70, 195]. OMP iteratively identifies the dictionary atom with the highest correlation to the current residual signal (initially, the signal itself). It then projects the residual onto the subspace defined by all currently selected atoms and updates the residual accordingly. This process continues until a predetermined sparsity level or reconstruction error threshold is reached [195].

Another popular greedy technique is **Least Angle Regression** (LARS) [83]. LARS shares conceptual similarities with OMP and forward stepwise regression but can select multiple basis elements simultaneously based on their correlation with the residual, following a path where correlations with the residual decrease equiangularly [83].

7.3.3 Comparison and Limitations

Convex optimization methods provide theoretical guarantees about finding the optimal L_1 solution (and potentially the L_0 solution under RIP conditions), but often at the cost of higher computational demands. Greedy algorithms like OMP and LARS are typically faster and less computationally intensive but lack such optimality guarantees [52].

One well-documented failure mode occurs when dictionary atoms are highly correlated, indicating poor orthogonality within the dictionary. In such cases, the myopic nature of greedy approximate methods may lead to suboptimal selections in early iterations that subsequent steps cannot adequately correct. The classic "twinsine" example illustrates this limitation: a signal composed of two highly correlated sinusoids may be only partially recovered by OMP and LARS, which typically selects just one of the constituent components. This example is illustrated in Figure 7.1, where the signal y is a linear combination of highly correlated sinusoids, particularly $y(t) = 0.5 * \cos(2\pi * 0.121 * \frac{n}{2} * t) + 0.5 * \cos(2\pi * 0.123 * \frac{n}{2} * t)$ with $n = 1000$. The OMP algorithm, when applied to this signal, selects a single component between the two constituent components, leading to suboptimal recovery, while exact algorithms like BPDN recovers both components exactly.

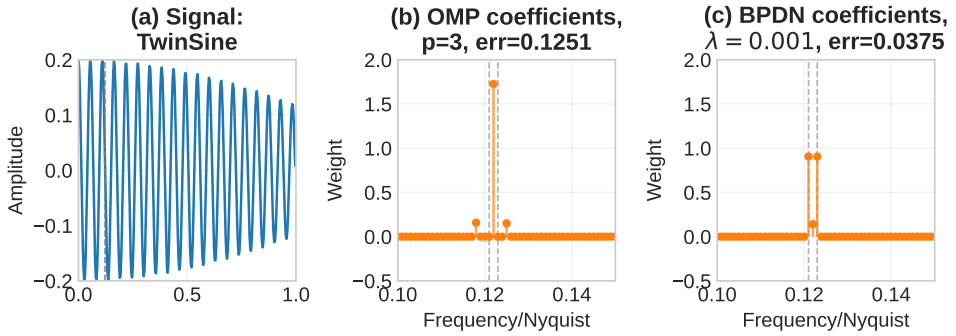


Figure 7.1: The twinsine example: a signal composed of two highly correlated sinusoids (frequencies 0.121 and 0.123 of the Nyquist frequency, which correspond 12.1% and 12.3% of the sampling rate). OMP typically selects only one of the constituent components, leading to suboptimal recovery.

Another case of failure arises when the optimal representation requires multiple atoms, but none individually show strong correlation with the signal [226]. By prioritizing the single most correlated atom in each iteration, greedy approximate algorithms can be misled by atoms with higher individual correlations that don't contribute to the optimal sparse combination. This case is further illustrated in Section 7.5.4, where we present an adversarial example designed to highlight this limitation of greedy approximate algorithms.

7.4 Solving Sparse Representation with SD

We now present how the SD framework can be applied to effectively solve the sparse representation problem, as an alternative approach to the existing optimization-based and greedy approaches. We first discuss the intuition behind using SD for sparse representation, followed by a detailed explanation of the query definition and the process of deriving the coefficients once the atoms are found. Then, we evaluate the performance of SD against existing methods in Section 7.5, focusing on both the quality of the solutions and the computational efficiency of the approach.

7.4.1 Methodology

The sparse representation problem can be reformulated as a multivariate similarity search task by exploiting a fundamental equivalence with multipole maximization. This equivalence is formally established through the following theorem:

Theorem 7.4.1 (Equivalence of Sparse Representation and Multipole Maximization). Given a dataset of vectors $\boldsymbol{D} = [\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_n] \in \mathbb{R}^{m \times n}$, a query vector $\boldsymbol{q} \in \mathbb{R}^m$, and a maximum set cardinality p, the constrained sparse representation problem:

$$\min_{\boldsymbol{x}} \|\boldsymbol{q} - \boldsymbol{D}\boldsymbol{x}\|_2^2 \text{ subject to } \|\boldsymbol{x}\|_0 \leq p$$

is equivalent to finding the top-1 multipole combination S' from a transformed dataset

$\boldsymbol{D}'$, where each vector $\boldsymbol{d}'_i \in \boldsymbol{D}'$ is defined as $\boldsymbol{d}'_i = \boldsymbol{d}_i - \boldsymbol{q}$. The solution to the sparse representation problem is the subset $S \subseteq \boldsymbol{D}$ corresponding to the vectors in S'.

Proof Sketch. The key insight behind this theorem is that by transforming the dictionary vectors as $\boldsymbol{d}'_i = \boldsymbol{d}_i - \boldsymbol{q}$, the multipole query objective of finding maximally linearly dependent vectors becomes equivalent to the sparse representation objective of minimizing reconstruction error. Intuitively, if there exists a sparse representation that closely approximates the query vector $\boldsymbol{q}$ using dictionary atoms, i.e., if $\boldsymbol{q} \approx \sum_{j \in S} v_j \boldsymbol{d}_j$ for some small set S, then rearranging this equation shows that $\sum_{j \in S} v_j (\boldsymbol{d}_j - \boldsymbol{q}) \approx 0$. By definition of our transformation, this means $\sum_{j \in S} v_j \boldsymbol{d}'_j \approx 0$, indicating that the transformed vectors are highly linearly dependent and will thus have a high multipole correlation value. Put differently, when vectors in the transformed dataset $\boldsymbol{D}'$ are maximally linearly dependent (i.e., exhibit high multipole correlation), there exists a linear combination $\sum_{j \in S} v_j \boldsymbol{d}'_j \approx 0$, which translates to $\sum_{j \in S} v_j \boldsymbol{d}_j \approx \boldsymbol{q}$ in the original space. This demonstrates that the subset S' with the highest multipole score corresponds exactly to the subset S that best reconstructs the query vector $\boldsymbol{q}$ with minimal error. The complete proof of this equivalence is provided in Appendix D.1.

This theoretical foundation enables us to solve the sparse representation problem using the SD framework with multipole correlation as the similarity measure. By performing a top-1 multipole query on the transformed dataset $\boldsymbol{D}'$ with maximum cardinality p, we can find the combination of at most p atoms that most accurately represents the query $\boldsymbol{q}$.

Notice, however, that other SD query types are also relevant for this problem. In fact, they offer valuable alternatives that can prioritize different aspects of the solution. For example, when one prioritizes sparsity of the solution over reconstruction accuracy, a progressive query that terminates after finding the first combination above the given threshold is a suitable choice as SD traverses the search space in increasing order of combination cardinality. Conversely, using a threshold query with a sufficiently high threshold (e.g., $\tau = 0.95$) allows us to find *all combinations* that closely approximate the query $\boldsymbol{q}$, which allows the user to select the combination that best fits their needs. This freedom to tune the algorithm to the specific needs of the application presents an additional advantage of the SD approach. The empirical evaluation presented in Section 7.5 will compare the performance characteristics when using SD for sparse representation with the three query types: Threshold, Progressive, and Top-k.

Query Definition To solve the sparse representation problem using SD, we first transform the dictionary by computing $\boldsymbol{d}'_i = \boldsymbol{d}_i - \boldsymbol{q}$ for each dictionary atom $\boldsymbol{d}_i$, creating the transformed dataset $\boldsymbol{D}'$. We then configure the multivariate similarity search query on $\boldsymbol{D}'$ with the following parameters:

- **Similarity Measure:** Multipole correlation.
- **Query Type:** Threshold, Progressive, or Top-k.
- **Threshold τ:** A high value (e.g., 0.95) to ensure representation accuracy — only applicable to Threshold and Progressive queries.

- **Top-k limit:** $k = 1$, to find the best or first acceptable representation in case of Top-k or Progressive queries, respectively.
- **Maximum Set Cardinality (p):** The maximum desired sparsity level ($L_0 \leq p$).

Deriving the Coefficients After SD identifies a suitable subset $S' \subseteq \boldsymbol{D}'$ of transformed vectors that exhibit high multipole correlation, we need to compute the corresponding coefficients for the linear combination that reconstructs $\boldsymbol{q}$ using the original dictionary atoms. Let $S \subseteq \boldsymbol{D}$ be the corresponding subset of original dictionary atoms. The high multipole value corresponds to a small minimum eigenvalue, λ_{min}, of the *Gram matrix* ${S'}^T S'$ constructed from the transformed vectors in S'. The eigenvector $\boldsymbol{v}$ associated with this λ_{min} confirms that S is a relevant subset, as it satisfies the relationship $\sum_{j \in S} v_j(\boldsymbol{d}_j - \boldsymbol{q}) \approx 0$. However, to find the coefficients $\boldsymbol{x_S}$ that formally minimize the reconstruction error $\|\boldsymbol{q} - \boldsymbol{D_S x_S}\|_2^2$, we solve the standard linear least-squares problem for the now-known subset S. This is achieved by solving the normal equations $(\boldsymbol{D_S}^T \boldsymbol{D_S})\boldsymbol{v_S} = \boldsymbol{D_S}^T \boldsymbol{q}$. This computation is efficient, especially given the small number of selected atoms in S.

7.4.2 Reflection

Using SD for sparse representation offers several advantages. First, through the proven equivalence with multipole maximization, we directly address the original L_0 formulation of the problem — the problem that was proven to be NP-hard [41, 70, 223] — without the need for relaxation to an L_1 problem. The dataset transformation approach provides a theoretically grounded method for reformulating sparse representation as a multivariate similarity search task. Second, as SD is an exact algorithm, we can guarantee that the solution is either the sparsest possible under the reconstruction error constraint (in the case of a threshold query with post-processing to select the sparsest solution), or the best possible solution under the sparsity constraint when $L_0 \leq p$ (in the case of the top-k query).

In that, SD combines the strengths of the current approaches: it pursues true sparsity (L_0) like greedy algorithms while offering guarantees for discovering optimal or near-optimal combinations within the search space defined by p, similar to exact methods. The top-k query variant provides a unique capability: identifying the sparse representation (up to size p) with the provably minimal representation error among all combinations of that size, a feature not directly offered by standard greedy or L_1-based methods. Specifically, greedy methods like OMP and LARS may fail to find the optimal sparse representation, as illustrated in the twinsine example (Figure 7.1). The exact methods like BPDN face a different challenge. Due to their L_1 relaxation of the problem, they are not guaranteed to find the sparsest solution under the L_0 constraint. Instead, they may instead yield solutions that are minimal in terms of L_1 norm but not necessarily the sparsest in terms of L_0 norm.

The following evaluation section empirically investigates these claims through a comparative analysis of SD against established sparse representation algorithms.

7.5 Evaluation

The goal of our evaluation is to demonstrate that a general-purpose algorithm like SD can not only solve the sparse representation problem effectively without domain-specific optimizations, but can also achieve performance comparable to specialized algorithms. We aim to show this through three key aspects: (a) SD achieves competitive performance across multiple evaluation dimensions (reconstruction error, sparsity level, computational efficiency, and downstream classification accuracy) compared to traditional sparse representation methods, (b) SD's exact search approach allows it to find optimal solutions in scenarios where popular greedy algorithms fail, as demonstrated through adversarial examples, and (c) SD accomplishes this while maintaining its general-purpose nature, requiring only configuration changes rather than algorithmic modifications to adapt to this domain.

High-Level Setup Our evaluation focuses on using sparse representation for face recognition, following the influential work of Wright et al. [236]. In this setup, we construct a dictionary from a large set of face images of known individuals, where each image represents an *atom* in the dictionary and each individual represents a *class*. There are multiple images per individual in the dataset, capturing variations in pose and lighting conditions. From this dataset, we then take out one image per individual to serve as the *query* images, and the remaining images form the *dictionary* used for sparse representation. Then, given a query image of an unknown person, we attempt to find its sparse representation using this dictionary. The key insight is that if the query image belongs to a specific person, it should be well-approximated using primarily the dictionary images corresponding to that person, with minimal contributions from other images. Therefore, by examining which dictionary images contribute most significantly to the sparse representation, we can classify the query image as belonging to the corresponding person. This approach elegantly combines the power of sparse representation with practical face recognition, providing a concrete application to evaluate our methods.

Datasets Following the experimental setup of Wright et al. [236], we apply sparse representation for image classification on the Extended Yale B dataset [99]. This dataset contains 16,380 grayscale images of 28 unique persons, with 585 images per subject across 9 poses (e.g., frontal, left, right) and 65 illumination conditions (e.g., frontal, left, right, ambient). The dataset was preprocessed by cropping the original images with a resolution of 640×480 pixels to include only the face region, resulting in a resolution of 192×168 pixels. We accomplished this using the Python OpenCV library and the Haar Cascade classifier for face detection [34], as the preprocessed dataset from Wright's original work is no longer available online.[1]

From this preprocessed dataset, we created several subsets to evaluate the performance of sparse representation methods under different conditions, following the design of Wright et al. [236]. The purpose of these subsets is to evaluate the robustness of the algorithms to datasets with different characteristics, such as varying resolution and the presence of specific facial features. We create these subsets by either downsampling the full images or cropping them to isolate specific facial features (nose, mouth, left eye). The goal here is to test if the

[1] The code repository and cropped dataset are available at https://github.com/JdHondt/FaceCrop and https://www.kaggle.com/datasets/jensdhondt/extendedyaleb-cropped-full, respectively.

algorithms can not only recognize a person from their full face but also from individual facial features (e.g., recognizing someone just from their nose or mouth). The following subsets were created:

- **Faces**: Complete face images at full resolution (192×168 pixels). This serves as the baseline test for face recognition.
- **Faces downsampled**: Complete face images downsampled to 12×10 pixels to test robustness to lower resolution.
- **Faces downsampled pose 1**: Only frontal pose images at 12×10 pixels to evaluate performance without pose variations.
- **Noses**: Images cropped to show only the nose region, testing if this feature alone is sufficient for identification.
- **Mouths**: Images cropped to show only the mouth region, similarly testing identification from this single feature.
- **Left-Eye**: Images cropped to show only the left eye region, completing the set of individual facial feature tests.

It is important to note that **the datasets are never mixed** (i.e., there is no dictionary that has both nose and mouth images); each subset is treated independently, with both the query and dictionary constructed from the same subset. For each subset, the goal remains the same: to identify a person by finding a sparse representation of their query image using the corresponding images or features from the dictionary (e.g., recognize a person from their nose image using a dictionary of nose images). The different subsets help evaluate how well each algorithm performs when given either complete faces or isolated facial features as input.

Query and Dictionary Construction For query selection, we extracted the first image of each subject (frontal pose with frontal illumination), resulting in 28 queries. The dictionary consisted of all other images of all subjects, with the dictionary size varying depending on the data subset used (e.g., 16,352 images in the case of the full face dataset).

Classification Task We adopt the classification strategy proposed by Wright et al. [236]. Given a query image $\boldsymbol{y}$ from an unknown subject, we first find its sparse representation $\boldsymbol{x}$ using the dictionary $\boldsymbol{D}$ containing images from all subjects. Then, for each possible subject c, we create a class-specific coefficient vector $\boldsymbol{x}_c$ by keeping only the coefficients in $\boldsymbol{x}$ corresponding to dictionary atoms belonging to class c, setting all other coefficients to zero.[2] We reconstruct the signal using only these coefficients: $\hat{\boldsymbol{y}}_c = \boldsymbol{D}\boldsymbol{x}_c$. The query $\boldsymbol{y}$ is assigned to the class c that yields the minimum reconstruction error:

$$\hat{\text{class}}(\boldsymbol{q}) = \arg\min_c ||\boldsymbol{q} - \hat{\boldsymbol{q}}_c||_2 \tag{7.6}$$

[2]Note that clipping a coefficient vector to only include coefficients for a specific class c does not involve any ground-truth information about the query's class $c_{\boldsymbol{q}}$, as the classes of the atoms are simply annotated in the dictionary.

Compared Methods To compare SD-based sparse representation with an accurate representation of the current state of the art, we select a set of algorithms that are widely used in the literature and have been shown to perform well on the Extended Yale B dataset [236]. Particularly, we include (a) Basis Pursuit Denoising (**BPDN**), and (b) **LASSO** as convex optimization methods, and (c) Orthogonal Matching Pursuit (**OMP**) and (d) Least Angle Regression (**LARS**) as greedy algorithms. Note that we do not include Basis Pursuit as it requires exact reconstruction, which was not always possible in our experiments due to noise, leading to no solutions in most cases. These methods are then compared to three variants of SD with Multipole correlation: (a) a progressive query with $k = 1$ (**SD-Progressive**), (b) a threshold query (**SD-Threshold**), and (c) a Top-k query with $k = 1$ (**SD-Topk**).

Parameterization In terms of parameters, we adopt the settings as Wright et al. [236] where applicable. We employ a tolerance parameter of $\epsilon = 0.05$ and an iteration limit of 10,000 for BPDN and LASSO, and include results under the regularization parameter values $\lambda \in \{10^{-5}, 10^{-4}, 10^{-3}\}$ in the case of LASSO. These values are consistent with the original work [236] and showed to yield solutions with varying sparcity levels across the datasets in our experiments. For the greedy algorithms OMP and LARS, which require specifying the desired number of non-zero coefficients, we test with target sparsities $k \in \{1, 2, 3\}$, as these small values often yielded good results in face recognition tasks. Lastly, to align SD as much as possible with the other methods, we employ a maximum cardinality of $p = 3$ for all SD variants with a threshold of $\tau = 0.95$ for progressive and threshold queries. The choice of $\tau = 0.95$ is motivated by Theorem 7.4.1, which establishes that high multipole correlation directly implies low reconstruction error. Since multipole correlation is bounded by 1, a threshold of 0.95 empirically showed to result in sufficiently low reconstruction errors while allowing for some tolerance to noise in the data, similar to the tolerance parameter ϵ used in BPDN and LASSO.

Hardware and Implementations All experiments were executed on a server equipped with a 48-core 2.4 GHz AMD Genoa 9654 processor and 128 GB of RAM. For LASSO, OMP, and LARS, we utilize the implementations available in the SCIKIT-LEARN library [197], a widely used Python machine learning library. For BPDN, we use the CVXOPT library [13], a Python package for convex optimization. Both libraries employ highly optimized C-compiled code for their core computations to achieve high performance. Our SD implementation (detailed in Chapter 6) is used for the SD variants. All implementations utilized multithreading across all available CPU cores with 48 threads for query answering, and sequential processing for each query.

Evaluation Metrics We evaluate the algorithms based on the following metrics, averaged over the 28 queries for each dataset subset:

- *Representation Error:* The average L_2 norm of the residual, $||\boldsymbol{y} - \boldsymbol{x}^T \boldsymbol{D}||_2$ over all queries.
- *Sparsity Level:* The average L_0 norm of the solution vector $\boldsymbol{x}$, $||\boldsymbol{x}||_0$ over all queries.
- *Computational Efficiency:* The total wall-clock time taken to process all 28 queries for a given dataset subset.
- *Classification Accuracy:* The percentage of query images correctly classified using the procedure described above.

To account for variability, each experiment (algorithm on a specific dataset subset) was run 10 times, and we report the average results. For SD-Threshold, which can potentially return multiple solutions per query, we report aggregate metrics over all found solutions unless otherwise specified. We set a timeout of 12 hours for each algorithm to complete all queries. Any algorithm that fails to complete within this time limit is excluded from the respective plot.

7.5.1 Comparing Sparsity and Reconstruction Error

Figure 7.2 presents the trade-off between reconstruction error and sparsity level (L_0 norm) for all algorithms across the different Extended Yale B subsets. Each point represents the average performance of an algorithm configuration on a specific dataset. The ideal region is the bottom-left corner, representing low error and high sparsity (low L_0).

We observe that the Pareto frontier, representing the best achievable trade-offs, is often occupied by the SD variants and OMP. Specifically, SD variants frequently achieve slightly lower reconstruction errors than OMP for the same sparsity level (typically $L_0 = 1$ or $L_0 = 2$, not counting the query itself). Among the SD variants, the progressive query consistently yields the sparsest solutions (often $L_0 = 1$), as it stops upon finding the first satisfactory combination, which occurs at the lowest cardinality p. In contrast, the threshold and Top-k query variants explore up to $p = 3$ ($L_0 \leq 2$) and thus can produce slightly less sparse solutions, although the differences in L_0 norm are generally small (ranging from 1 to 2 on average).

Comparing the convex optimization methods, BPDN generally achieves lower reconstruction errors than LASSO. It is important to note that while methods like BPDN are considered 'exact' within their convex optimization framework, they do not necessarily yield zero reconstruction error. This is because their formulation, as described in Section 7.3, is designed to handle noisy, real-world data by minimizing the L_1 norm subject to an error tolerance ($\epsilon = 0.05$ in our experiments). Perfect reconstruction (error of 0) is often impossible with noisy data, and forcing it would lead to overfitting; the tolerance parameter ensures a solution can always be found. However, BPDN solutions tend to be much denser (higher L_0 norm) compared to the greedy methods and SD. LASSO, by varying the regularization parameter λ, can produce solutions with different sparsity levels, but its solutions are generally Pareto-dominated by either BPDN (in error) or the greedy/SD methods (in sparsity and often error). Notably, BPDN failed to complete all queries within a 12-hour time limit on the larger, higher-dimensional "Faces" and "Mouths" datasets, highlighting its computational cost.

A notable observation is the relatively high reconstruction error of LARS compared to OMP, despite both being greedy algorithms. This can be attributed to their differing coefficient selection strategies. While OMP explicitly minimizes the reconstruction error at each step for the chosen set of atoms, LARS follows an "equi-angular" path that selects atoms based on their correlation with the residual, which may not always lead to the optimal reconstruction. This can lead to a less optimal reconstruction for the same level of sparsity, though LARS still achieves high classification accuracy (as we will show later), indicating that it is effective at selecting images with the same class as the query image, even if the reconstruction error is higher than that of OMP.

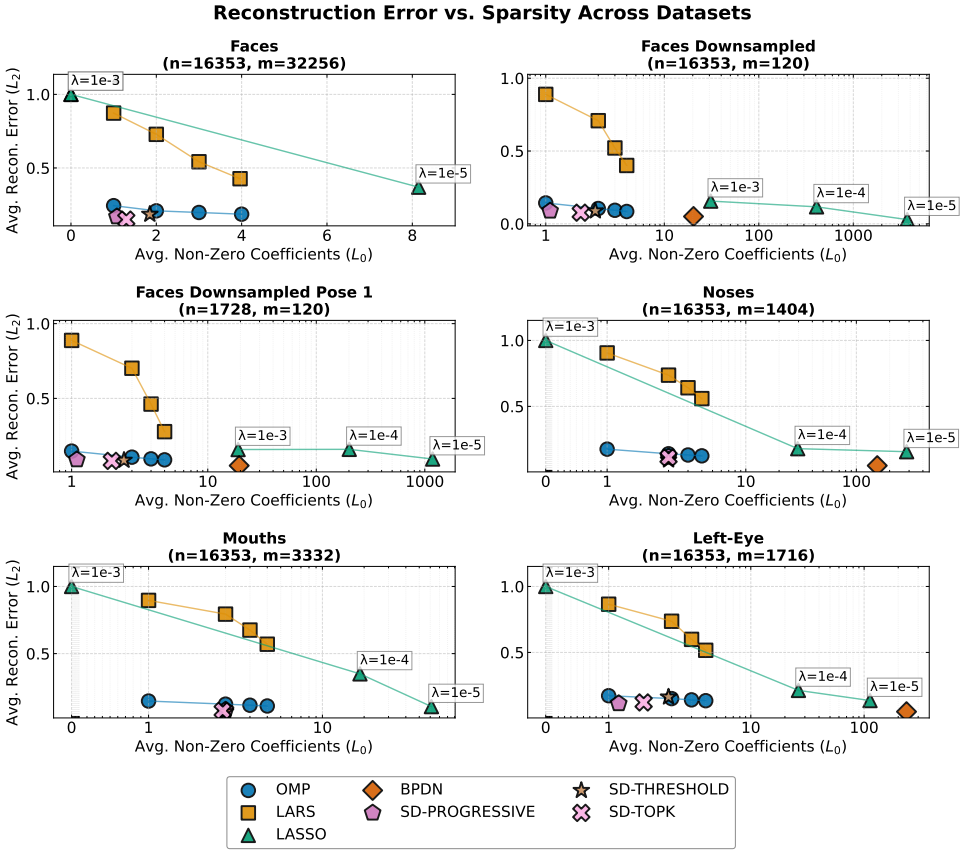


Figure 7.2: Sparsity vs. Reconstruction Error over different datasets and methods.

7

Overall, these results suggest that SD, especially SD-Progressive, offers a highly competitive trade-off between reconstruction error and sparsity. It achieves sparsity levels comparable to OMP but often with better reconstruction fidelity, likely benefiting from its ability to find optimal or near-optimal combinations within its search space. The optimization-based methods, while theoretically powerful, seem less effective in this specific application, potentially due to the L_1 relaxation favoring dense solutions with many small coefficients, effectively overfitting to noise when aiming for very low reconstruction error. The results also imply that for these datasets, the sparsest representations (often $L_0 = 1$) are frequently among the best in terms of reconstruction error; finding the single most similar face often provides the best reconstruction, and adding more faces primarily models noise.

Table 7.1: Average Classification Accuracy across Extended Yale B subsets.

Method	Avg. Accuracy (%)
BPDN	98.15
SD-Progressive	97.15
LARS	96.30
OMP	96.30
SD-Topk	92.59
SD-Threshold	88.02
LASSO (λ^*)	80.25

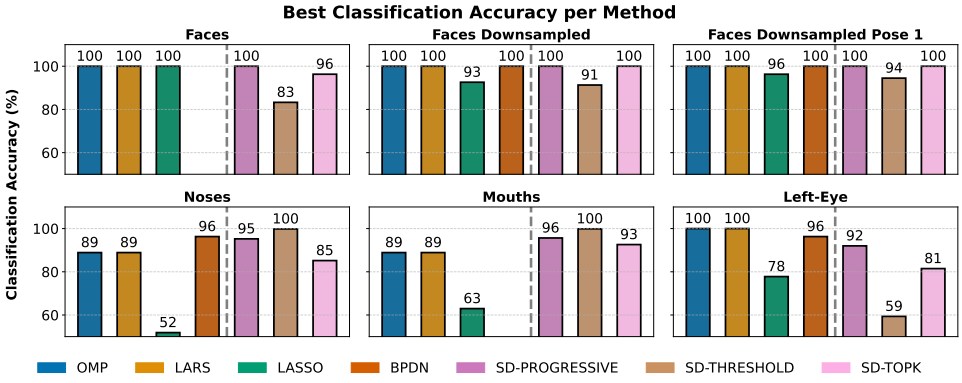


Figure 7.3: Classification accuracy across different datasets and methods.

7.5.2 Comparing Classification Accuracies

Figure 7.3 shows the classification accuracy achieved by each algorithm on the different dataset subsets, while Table 7.1 summarizes the average accuracy across all datasets.[3] Note that for each method and dataset, the parameter configuration yielding the best accuracy is used in both the figure and the table. In real world scenarios, such optimal configurations can be determined through cross-validation on a small validation set, composed of a stratified sample from the dictionary.

BPDN achieves the highest average accuracy (98.15%) among the algorithms that completed the task, although its inability to run on all datasets limits this comparison. It is worth noting that none of the methods achieve 100% accuracy. This is not a failure of the sparse representation algorithms themselves, but rather a characteristic of the classification strategy adopted from Wright et al. [236]. For a small number of queries in the dataset, the solution with the lowest reconstruction error does not lead to a correct class prediction, even when the sparsity of the solution is ignored (i.e., when the algorithm has the freedom to select any

[3] In practice, optimal parameter configurations can be determined through cross-validation on a small validation set, composed of a stratified sample from the dictionary. This way, we can get an idea of what parameter configuration yields the best accuracy for each method and dataset before running classification on the actual queries (i.e., the test set).

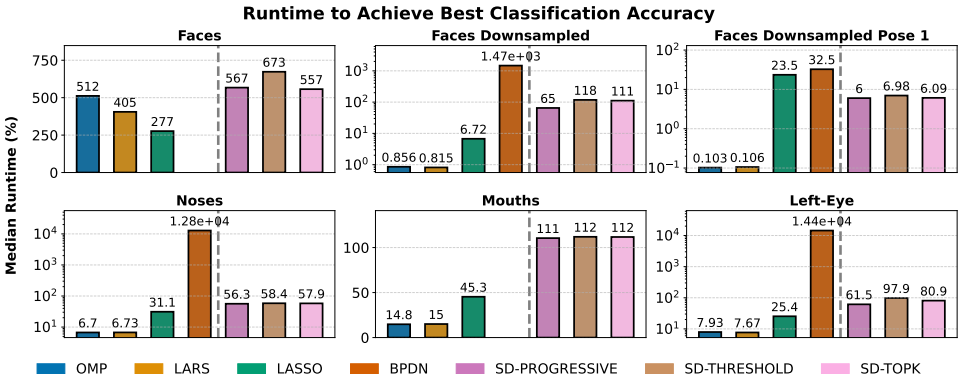


Figure 7.4: Computational efficiency (in seconds) of each algorithm across different datasets. BPDN failed to complete on Faces and Mouths within the 12-hour time limit.

number of atoms). This is not uncommon in face recognition or any classification task for that matter; even state-of-the-art methods rarely achieve perfect accuracy on this dataset, as demonstrated in the study by Wright et al. [236].

SD-Progressive demonstrates the second-best performance, achieving an average accuracy of 97.15%, only marginally lower than BPDN's average on the datasets it completed (but as we will see in the next section, it is significantly faster). OMP and LARS show identical average accuracies (96.30%). This result is somewhat surprising, considering the differences in their selection strategies and LARS's higher reconstruction error, as discussed in the previous section. This suggests that while LARS's equi-angular selection path may not always yield the best reconstruction, it is equally effective at identifying atoms from the correct class, which is sufficient for accurate classification in this context. Also notice though that their performance of both algorithms degrades noticeably on the Noses and Mouths subsets, suggesting that for these more ambiguous facial parts, simply finding the atom with the highest initial correlation (as OMP/LARS tend to do) is not always sufficient for correct identification.

The other SD variants, SD-Topk (92.59%) and SD-Threshold (88.02%), perform less well in terms of classification accuracy compared to SD-Progressive. This further supports the observation that for this task, the sparsest representation found by SD-Progressive is most effective for classification. Allowing slightly denser solutions (as Topk and Threshold might) does not improve, and sometimes hinders, classification. LASSO consistently yields the lowest classification accuracy (around 80.25% for the best λ value), likely due to its tendency to produce denser solutions that may overfit or incorporate features from incorrect classes.

These classification results reinforce the findings from the error-sparsity analysis. Algorithms achieving good, sparse representations (low L_0, low error) tend to perform well in classification. SD-Progressive emerges as a top performer among them, achieving accuracy very close to the best (BPDN) while being applicable across all datasets.

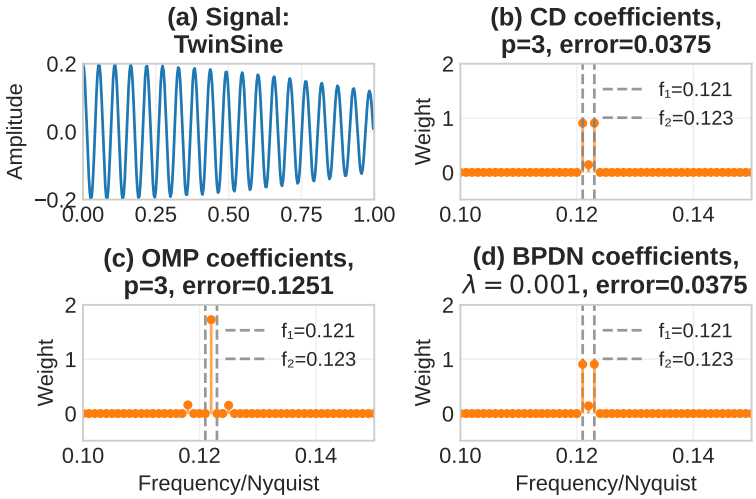


Figure 7.5: Results of the twinsine example including SD-Topk.

7.5.3 Comparing Computational Efficiency

Figure 7.4 illustrates the computational time required by each algorithm to process all 28 queries for each dataset subset. As expected, the greedy algorithms OMP and LARS are generally the fastest, benefiting from their simple iterative nature and early stopping criteria (fixed number of atoms). LASSO, likely due to its highly optimized SCIKIT-LEARN implementation using coordinate descent and Cython, is also computationally efficient [197]. The SD variants (SD-Progressive, SD-Threshold, SD-Topk) exhibit moderate computational costs. They are generally slower than the fastest greedy methods but significantly faster than BPDN. Their runtime increases with the size and dimensionality of the dataset, reflecting the combinatorial nature of the search performed by SD. However, the pruning strategies inherent in the SD framework keep the runtime practical for these dataset sizes. BPDN is by far the slowest algorithm, requiring orders of magnitude more time than the others and failing to complete within the 12-hour limit on the largest datasets (Faces and Mouths). This is relatively unsurprising given its requirement to solve a linear program for each query. While efficient LP solvers exist, they still cannot match the speed of greedy algorithms or SD.

These experiments demonstrate that SD offers a viable solution for sparse representation, providing a slightly better trade-off between reconstruction error and sparsity level than OMP, albeit at a higher computational cost. However, it remains substantially faster than BPDN while additionally offering guarantees on finding the sparsest low-error solution under the L_0 norm, rather than the L_1 norm. To further illustrate the value of these guarantees, we next examine two adversarial examples where SD reliably finds optimal solutions while OMP fails.

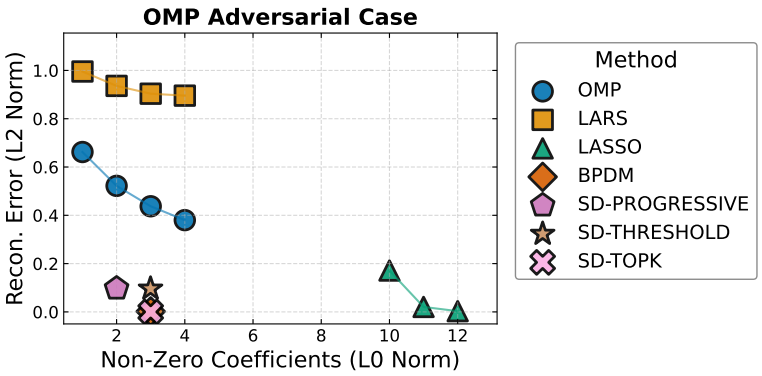


Figure 7.6: Reconstruction Error vs. Sparsity Level of different algorithms on the Adversarial OMP dataset.

7.5.4 Adversarial Examples

To further highlight the difference between SD and greedy algorithms like OMP, we examine scenarios designed to be challenging for greedy approaches.

Twinsine Example Revisited First, recall the classic "twinsine" example of [52], where the target signal is a sum of two highly correlated sinusoids present in the dictionary. In this case, OMP typically identifies only one of the two component sine waves, while exact algorithms correctly find both [52]. When we apply SD-Topk to this example, we obtain results identical to those from BPDN. Particularly, as shown in Figure 7.5, SD successfully identifies both constituent components, achieving a perfect reconstruction of the original signal. This demonstrates SD's ability to overcome the myopia of OMP by considering combinations of atoms.

Adversarial OMP

Second, we construct a new adversarial example specifically targeting a known weakness of OMP: its potential failure when the optimal representation involves multiple atoms, none of which individually exhibit a dominant correlation with the query signal [52, 226]. OMP's reliance on high individual correlations can cause it to be misled by "decoy" atoms that are highly correlated with the query but do not belong to the optimal solution.

To demonstrate this second failure mode, we created a synthetic dataset that exhibits this behavior. This dataset was constructed as follows. First, we create a query signal $\boldsymbol{q}$ in a 100-dimensional space as the average of three random vectors, $\mathbf{y} = (\mathbf{a}_1 + \mathbf{a}_2 + \mathbf{a}_3)/3$. These vectors $\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$ form the true sparse basis (i.e., the optimal solution) for the query signal, and are added to the dictionary $\mathbf{D}$. Then, we create a set of 20 "decoy atoms" $\{\mathbf{d}_1, \mathbf{d}_2, ..., \mathbf{d}_{20}\}$. These vectors are designed to have a high individual correlation (dot product) with $\mathbf{y}$ to act as "decoys" that mislead greedy algorithms like OMP into selecting them over the true basis vectors, thus leading to a suboptimal solution. We accomplished this by projecting the query onto random vectors, scaling them to align their dot product with the query, and

adding small orthogonal perturbations to ensure they don't lie on the subspace spanned by $\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$ nor that they form a subspace that includes the query. These decoy vectors are then also added to the dictionary $\mathbf{D}$. Lastly, we include a set of 177 random vectors uncorrelated with the query, resulting in a dictionary of 200 vectors and a sufficiently large search space of possible combinations.

We then run the different sparse representation algorithm on this "Adversarial OMP" dataset, using the same parameters as in the previous experiments. The results are shown in Figure 7.6, where we plot the reconstruction error against the sparsity level (L_0 norm) for each algorithm. We observe that OMP and LARS fail to recover the true sparse representation. They are misled by the high individual correlations of the decoy atoms and select them instead of the true basis vectors $\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$. LASSO manages to reconstruct the query well but selects a superset of the true basis, including some decoy atoms, resulting in a non-sparse solution. In contrast, both BPDN and SD (specifically SD-Topk with $p = 4$ to allow for 3 atoms) successfully identify the correct three basis atoms $\{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$ and achieve perfect reconstruction with the optimal sparsity. This example clearly demonstrates SD's robustness in scenarios that are challenging for greedy algorithms. By evaluating combinations, SD avoids being trapped by misleading individual correlations and finds the globally optimal sparse solution within its search constraints, a crucial capability in certain real-world applications.

Real-World Implications

The scenario illustrated by the Adversarial OMP example is not merely a contrived academic exercise; it reflects practical situations where sparse representation is applied. Research across various domains has highlighted cases where optimal solutions involve combinations of signals that, individually, show low correlation with the query. Such challenges appear in fields like hyperspectral unmixing [67], financial and biomedical data analysis [250], image reconstruction [85], and seismic data processing [145]. In these situations, methods that rely on myopic, greedy selection criteria may not find the optimal representation.

The theoretical foundations for these challenging scenarios have also been examined. For instance, the performance of greedy algorithms can be linked to dictionary properties like the maximum correlation between any two dictionary atoms [85]. As the maximum correlation increases, the likelihood of a greedy algorithm being misled by individually correlated but suboptimal atoms also increases. This underscores the value of methods that can systematically evaluate combinations, like SD, as they are inherently robust to such scenarios.

7.6 Conclusions

This case study has demonstrated the successful application of the SD framework to the problem of sparse representation. Our results show that SD, leveraging the Multipole correlation measure, is not only applicable to this domain but also highly competitive with established algorithms. In terms of the trade-off between reconstruction error and sparsity, SD variants often reside on or near the Pareto frontier, performing comparably or slightly better than the widely used OMP algorithm.

SD offers a compelling alternative by bridging the gap between fast but heuristic greedy algorithms and slower but theoretically grounded convex optimization methods. As an algorithm that explores combinations of atoms, SD demonstrated superior empirical performance compared to greedy methods in challenging scenarios, as shown by the adversarial examples where OMP failed but SD successfully identified the optimal solution.

Furthermore, SD directly addresses the original L_0 problem, avoiding the need for L_1 relaxation which can lead to denser solutions than desired. In this way, SD offers the best of both worlds: an algorithm that combines the speed of greedy approaches with the exactness of optimization methods.

This case study demonstrates the versatility of the SD framework, showing how its general approach to multivariate similarity search can be adapted to solve domain-specific problems with performance comparable to specialized algorithms. The successful application to sparse representation suggests that SD may prove valuable across a wide range of domains where efficient identification of multivariate relationships is essential. The successful application to sparse representation suggests that SD may prove valuable across a wide range of domains where efficient identification of multivariate relationships is essential.

CHAPTER 8

Conclusions and Future Directions

This thesis has advanced similarity search beyond its traditional univariate, pairwise focus by developing methods to analyze multivariate data and discover high-order relationships. This concluding chapter summarizes these key contributions, then identifies open challenges and outlines future research directions that build upon this new foundation. The aim is to inspire a new generation of multivariate-aware similarity search techniques.

8.1 Summary of Contributions

This thesis aimed to advance the field of similarity search by developing effective and efficient methods. Our approach directly addresses the multivariate nature of both data and the relationships within them, moving beyond the traditional univariate and pairwise focus that has dominated the field for decades.

In Chapter 1, we highlighted the limitations of traditional similarity search in their focus on univariate, pairwise relationships. We subsequently outlined the problem statement, which called for a systematic extension of similarity search to multivariate data and high-order relationships.

In Chapter 2, we laid the theoretical groundwork by establishing formal definitions of similarity search queries. We then briefly introduced known techniques for efficient similarity search, such as approximation methods and index structures. Finally, we positioned the work within the broader research landscape by discussing related work in both MTS similarity search and multivariate similarity search on univariate data.

In Part I, we focused on **pairwise similarity search on multivariate time series**. In Chapter 3, we conducted the first comprehensive study of distance measures for MTS. We introduced a novel taxonomy based on normalization, temporal models, and channel-dependency

models that brought clarity to a fragmented area. Through this work, we provided practical guidelines for measure selection and revealed the surprising effectiveness of *Nonorm* for current MTS datasets, the strong performance of *SBD-D*, and important insights into channel-dependent versus channel-independent approaches. We then introduced *MS-Index* in Chapter 4, a novel algorithm for exact k-NN subsequence search in MTS – a problem previously unaddressed due to its inherent computational complexity and the limitations of existing methods. We demonstrated that *MS-Index* achieves orders of magnitude speedup over existing approaches and offers practical features like ad-hoc channel selection. Crucially, we showed that its design principles, such as unified representation and channel-adaptive partitioning, are transferable to other multivariate search contexts.

In Part II, we extended the scope of similarity search to **high-order relationships** in univariate data. In this domain, we introduced Correlation Detective (CD) in Chapter 5, a highly efficient exact algorithm for detecting multivariate correlations in large static datasets, using the Multivariate Pearson Correlation and Multipole measures. We designed *CD* to leverage theoretical results, bounding, and novel optimizations to dramatically reduce the search space, while supporting different query types and constraints. We then extended this algorithm to streaming data with *CDStream*, which efficiently maintains results over sliding windows using a custom index and a novel stream processing model. Additionally, we developed *CDHybrid* as an adaptive solution that automatically switches between *CD* and *CDStream* based on stream characteristics. Finally, we generalized the concepts from *CD* into the *Similarity Detective Framework* in Chapter 6. This framework provides a principled methodology to detecting multivariate relationships in univariate data. It not only extends the capabilities of *CD* to *any* multivariate similarity measure, and constraint, but also unifies a wide range of field-specific solutions under a common framework. The framework's broad applicability was then demonstrated in Chapter 7, through its successful application to the sparse representation problem, where it showed competitive performance against specialized methods.

Collectively, these contributions lay the theoretical and practical groundwork for a new perspective on similarity search – one that embraces the multivariate nature of both data and the relationships within them.

8.2 Reflection on Central Research Question

Returning to our central research question – whether it is possible to extend the traditional similarity search paradigm to the multivariate setting – this thesis has provided a comprehensive affirmative answer through multiple complementary contributions.

Starting with the extension to multivariate data (Part I), Chapter 3 established that meaningful comparison of multivariate time series is not only possible but can be done effectively through measures like SBD-D. Chapter 4 then proved that such comparisons can be performed efficiently at scale through MS-Index, which achieves orders of magnitude speedup over existing approaches. Together, these chapters demonstrated that the first fundamental extension of similarity search to handle multivariate data objects is both theoretically sound and practically feasible.

For the extension to multivariate relationships (Part II), Chapter 5 showed that detecting meaningful high-order relationships in univariate data is computationally tractable through careful algorithm design and effective pruning strategies. Chapter 6 then generalized these insights into a comprehensive framework that can handle any multivariate similarity measure, proving that this extension is not limited to specific measures but represents a broader development in the field. The practical value of this extension was validated in Chapter 7, demonstrating its effectiveness in real-world applications.

Collectively, these contributions prove that similarity search can indeed be meaningfully extended to embrace the multivariate nature of real-world data, both at the data level and the relation level. Moreover, our findings suggest that such extensions often lead to better and more meaningful patterns than traditional univariate approaches, validating the premise that motivated this research.

8.3 Open Challenges and Future Directions

While this thesis has made significant progress in multivariate similarity search, it also reveals numerous open challenges and exciting future research directions. We now outline a vision to inspire and guide the community in this new domain.

8.3.1 The Grand Challenge: Multivariate Squared

The ultimate goal, for which this thesis builds a foundation, is the search for *multivariate relations in multivariate data*. This represents the combination of the two extensions explored in this thesis, seeking to understand complex, many-to-many relationships between multi-attribute data objects. This challenge involves overcoming multiple conceptual and algorithmic hurdles behind simply combining existing approaches. It requires a fundamental reconsideration of how similarity is defined, measured, and efficiently computed in such complex contexts.

From a conceptual perspective, defining similarity in this context presents novel theoretical challenges, similar to those encountered when extending distance measures for UTS to the MTS domain in Chapter 3. Defining such measures requires answering questions like:

1. How can we meaningfully extend the MTS distance measures explored in Chapter 3 to cover relations over multiple time series?
2. When such extensions are developed, will the empirical patterns and insights from our comprehensive study still hold, or will entirely new patterns emerge?
3. How do we evaluate these measures? Traditional approaches like 1-NN classification become less straightforward when either side of the neighbor relationship can contain multiple objects rather than single entities.

From an algorithmic perspective, the challenges are equally profound. A fundamental question concerns whether the SD framework can be extended to support measures over MTS data. While this framework successfully addresses the combinatorial explosion of the search space for univariate data, it remains unclear whether the same bottlenecks persist in the multivariate context. For instance, the dimensionality of the data may now start playing a more

significant role for empirically bounded measures. Additionally, the added channel axis in the data introduces an additional term in the theoretical complexity of the problem, which might need to be addressed through novel optimizations.

Our vision for addressing this grand challenge follows a systematic, multi-phase approach. The first phase involves applying the SD framework to a diverse range of domains where multivariate similarity search is already applied, but often limited by the performance of existing tools. By enabling large-scale analysis, the SD framework can help uncover new insights in these fields. Furthermore, applying the framework across various domains will allow us to draw connections between different applications and deepen our understanding of the multivariate similarity measures used in practice. Through these case studies, we aim to develop a comprehensive view of the measures employed across domains, identify their meta-properties, and understand the downstream tasks that drive their selection. This, in turn, can help researchers and practitioners select appropriate measures or develop new ones that fill gaps in the current landscape.

Building upon these insights, the second phase involves establishing a structured evaluation framework for multivariate similarity measures, similar to the study conducted for MTS distance measures in Chapter 3. This will help structure the current landscape of multivariate similarity measures, and identify which measures are most effective for specific tasks and data characteristics.

After understanding the current landscape of measures and their performance, the third phase focuses on extending the SD framework to support the most promising multivariate measures identified in our evaluation. This extension requires careful analysis of how the framework's core components (i.e., the bounds, execution workflow, and optimizations) can accommodate the computational and theoretical demands of multivariate measures. We must identify the primary performance bottlenecks that emerge in these scenarios and develop novel optimization techniques specifically designed to address them.

8.3.2 Directions in MTS Search

Building upon Part I, several directions promise to improve our ability to efficiently find pairwise similarities in MTS datasets.

MTS-Specific Normalization Techniques. Our finding in Chapter 3 that existing normalization methods offer little benefit – with *Nonorm* often being optimal – suggests current methods may fail to capture essential multivariate characteristics. This calls for dedicated research into novel normalization techniques specifically designed for MTS, which might consider inter-channel dependencies, temporal dynamics, or be adaptive and task-aware.

Efficient Algorithms for Advanced Distance Measures. There is a need to bridge the gap between the most effective MTS distance measures identified in Chapter 3 and their practical use in large-scale search scenarios. This involves designing novel indexing structures and search algorithms specifically optimized for computationally intensive yet effective MTS distance measures like *SBD-D* or supervised *MSM-I*. A natural first step here would be to de-

rive tight lower bounds for these measures, enabling the development of pruning strategies and index structures that exploit these bounds for efficient search.

Variable-Length and Multi-Scale MTS Subsequence Search. *MS-Index* requires the length of the queries to be known at index construction time, which limits its applicability in scenarios where query lengths vary or are not known in advance. Future work should aim to either extend the index structure of MS-Index to support variable-length queries, or develop new algorithms that natively support variable-length subsequence search. During the writing of this thesis, we have already started exploring this direction, where we extended the *ULISSE* algorithm [152] to the multivariate case in [80]. Preliminary results show that this approach can achieve competitive performance on synthetic datasets, but further work is needed to evaluate its performance on real-world datasets and to develop optimizations that can improve its efficiency.

8.3.3 Directions in High-Order Relationship Search

The work in Part II on discovering high-order relationships can be expanded by extending the SD framework to the streaming domain, a direction already explored conceptually in Section 6.4.8. The central intellectual challenge lies in generalizing the principles of efficient, incremental updates from *CDStream* to the broad class of measures supported by the framework. This requires moving beyond the specific properties of *mPC* and developing a more abstract theory of incremental updates for multivariate similarity.

For empirically bounded measures, the key challenge is to identify which measures possess properties analogous to the soft-distributivity of *mPC*, where changes in aggregate similarity can be predicted from changes in underlying pairwise statistics. For measures lacking this property, a fundamental question is whether alternative, efficiently monitorable statistics exist. For theoretically bounded measures, the challenge shifts to the geometric domain: how can we efficiently track the evolution of cluster properties (e.g., centroids and radii) in response to streaming data, and how do these changes propagate to the similarity bounds? Addressing these questions will require new theoretical insights into the relationship between streaming data, summary statistics, and the bounds of diverse multivariate similarity measures, thereby creating a principled foundation for high-order similarity search in streaming environments.

8.3.4 Overarching Directions

Besides the specific challenges on each of the variants of multivariate similarity search outlined above, we also see several overarching directions that can help advance the field as a whole.

Extreme-Scale Experiments This thesis demonstrated the effectiveness of our algorithms on datasets containing up to 10,000 time series with roughly 10,000 time points each, consistently showing superior scaling behavior compared to existing approaches. While our theoretical analysis and current experimental results strongly suggest this superior scalability will continue at larger scales, these experiments fall short of true ”big data” by today’s

standards (e.g., applications like power grid monitoring can generate billions of data points per day [40]). To validate this expected scalability, future work should evaluate these algorithms on datasets with millions of time series and tens of millions of time points, which will require transitioning from current in-memory implementations to disk-based and potentially distributed architectures. This shift will require rethinking parts of the algorithms, potentially uncovering new performance bottlenecks in I/O operations, data partitioning strategies, and memory management that are not apparent in smaller datasets. While we expect the core algorithmic advantages to persist, addressing these challenges will be crucial for conclusively demonstrating the practical applicability of these algorithms in large-scale scenarios.

Hardware Acceleration and Distributed Systems The computational demands of multivariate similarity search at extreme scales will also require a shift towards hardware-aware algorithm design to leverage modern parallel computing architectures. This involves designing algorithms that can exploit specialized hardware such as GPUs, TPUs, and FPGAs, while simultaneously developing distributed computing strategies that can effectively partition both data and computation across multiple nodes. Core algorithmic components like the indexing structures in *MS-Index* and the hierarchical clustering in *CD* may need to be redesigned to exploit massively parallel processing capabilities, potentially requiring novel data structures and communication patterns optimized for these hardware platforms. Furthermore, subroutines like distance computation with MASS can be adapted to run on GPUs, which can significantly speed up the search process for large datasets.

Explainable and Actionable Insights. The goal is to evolve multivariate similarity search algorithms from pattern-finding tools into analytical engines where discovered patterns come with understandable explanations and actionable recommendations. This involves developing post-analysis techniques that (1) identify the contributing factors to discovered similarities, (2) filter out spurious correlations based on systematic, theoretically grounded rules, and (3) visualize these relationships in ways that are interpretable by domain experts. For example, we can extend the SD framework to not only find correlations but also to explain them by identifying the underlying patterns in the data that contribute to the similarity and incorporate them in knowledge graphs.

Learning-Driven Similarity Search. We envision deep integration of machine learning into similarity search, enabling systems to learn optimal search strategies, data representations, and even the notion of "similarity" from data and user feedback. This includes developing methods for learning task-specific MTS distance measures or multivariate similarity functions directly from data, and using techniques like reinforcement learning to enable algorithms to dynamically adapt their parameters based on observed data distributions and query performance.

Broad Empirical Validation Finally, the potential of this new frontier in similarity search should be demonstrated by systematically applying the developed frameworks to real-world domains to uncover novel insights. We envision applications in diverse fields such as understanding ecosystem dynamics, identifying risks in financial systems, discovering patient subgroup characteristics for personalized medicine, or detecting bias in AI systems. This

involves collaboration with domain experts and development of benchmark suites focused on multivariate and high-order similarity discovery.

8.3.5 Towards an Overarching Framework

The work in Chapter 6 demonstrates that it is possible to structure the space of queries within a specific problem domain, construct a unified theoretical framework for exact query answering under any measure, identify performance bottlenecks across different query types, and develop targeted optimizations that are automatically applied based on measure and query parameters. While this approach was applied specifically to finding multivariate relations in univariate data, the same principles can be applied to any variant of similarity search, whether dealing with univariate or multivariate data, or addressing pairwise or high-order relationships.

This would require conducting surveys and large-scale comparative studies across every variant of the similarity search problem, then crystallizing these insights into decision-tree-like guidelines that specify optimal measures (if not provided), algorithms (i.e., sequential scan, index, or hybrid), and optimizations (e.g., dimensionality reduction strategies like DFT/SAX/SFA, index structures like R*-tree, B-tree, graph, etc.) for given scenarios. Key to these studies is unveiling the theoretical properties of measures and data characteristics (e.g., distributions) that make them suitable for specific query strategies. For example, for exact whole-sequence search on univariate data, the resulting system may recommend Euclidean distance with an iSAX index, while for approximate search, it could suggest graph-based indexes like HNSW [20, 233]. For subsequence search, the framework might prescribe DFT with an R-tree structure unless variable-length queries are required, in which case sequential scanning through MASS becomes the optimal approach [198]. When decision rules are established for every segment of the similarity search space, these rules can be unified into a single system capable of designing and executing efficient query plans for any similarity search query.

This vision involves substantial work, likely requiring multiple dissertations to fully realize such a system. Additionally, the constantly evolving landscape of problems, measures, and algorithms implies that the decision rules will need to be continuously updated. However, we believe that researchers should strive not only to develop domain-specific solutions but also to build a common understanding of the entire similarity search space, providing more general, accessible, and efficient solutions that also reduce redundancy of effort across the field. This approach relates closely to how relational databases revolutionized structured data querying by providing SQL as a common language and relational algebra as a unifying framework, enabling efficient querying of any structured data regardless of domain or specific application.

8.4 Closing Remarks

The complex, interconnected nature of modern data presents a significant challenge to traditional similarity search approaches, which have long relied on pairwise comparisons of univariate entities. This thesis has addressed this gap by systematically extending similarity

search, developing novel theories, algorithms, and frameworks to handle both multivariate data objects and the high-order relationships that connect them.

The presented contributions mark significant advancements. They lay a critical foundation for a more powerful and realistic approach to understanding similarity in complex systems. Yet, this journey into the multivariate realm is just beginning. The outlined challenges, especially the grand challenge of searching multivariate relations within multivariate data and the development of an overarching similarity search system, highlight that there is still plenty of work to be done.

The future of similarity search is *multivariate*. This thesis has argued that similarity search must embrace this multivariate complexity. We hope that the foundations laid in this thesis will catalyze further research within this new field, inspiring new ideas and approaches that improve our ability to discover meaningful patterns in complex, multivariate datasets. By moving beyond one-to-one comparisons and embracing the rich, multivariate nature of real-world data, the next generation of similarity search will unlock unprecedented insights, transforming our ability to understand and interact with complex systems. This thesis serves not as a final destination, but as a starting point and an enthusiastic call to explore these new territories in the quest for meaningful pattern discovery.

CHAPTER A

Proofs to Chapter 4

A.1 Proof of correctness and completeness of MS-Index

Here, prove that the query process described in Section 4.3.3 of Chapter 4 is correct, i.e., that the algorithm described in Algorithm 1 is guaranteed to return the correct and complete k-NN to the query $\boldsymbol{q}$. Specifically, we aim to prove the following lemma:

Lemma A.1.1. *Any indexed subsequence $\boldsymbol{t}$ of length $|\boldsymbol{q}|$ that is part of the k-NN of $\boldsymbol{q}$ is guaranteed to be in the set of subsequences returned by MS-Index, and therefore, after exact distance computations on this set, MS-Index is guaranteed to return the correct k-NN of $\boldsymbol{q}$.*

Proof. The first probe of the index returns k subsequences of length $s \geq |\boldsymbol{q}|$, for which the exact distances with the query are computed. As these k subsequences collectively contain at least k subsequences of length $|\boldsymbol{q}|$, 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 $\tau_k * \sqrt{|\boldsymbol{q}|}$ on the R-tree index, which contains the feature vectors. Based on Eq. 2.1, 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 $|\boldsymbol{q}|$-length subsequence $\boldsymbol{t}$ and the query $\boldsymbol{q}$ is lower-bounded by the distance between their feature vectors $\tilde{\boldsymbol{t}}'$ and $\tilde{\boldsymbol{q}}'$ as:

$$d(\boldsymbol{t}, \boldsymbol{q}) \geq \frac{1}{\sqrt{|\boldsymbol{q}|}} \sqrt{\sum_{i \in c_{\boldsymbol{q}}} d(\tilde{\boldsymbol{t}}_i, \tilde{\boldsymbol{q}}_i)^2} = \frac{d(\tilde{\boldsymbol{t}}'_{c_{\boldsymbol{q}}}, \tilde{\boldsymbol{q}}'_{c_{\boldsymbol{q}}})}{\sqrt{|\boldsymbol{q}|}} \tag{A.1}$$

where $\tilde{\boldsymbol{t}}'_{c_{\boldsymbol{q}}}$ and $\tilde{\boldsymbol{q}}'_{c_{\boldsymbol{q}}}$ correspond to the feature vectors of $\boldsymbol{t}$ and $\boldsymbol{q}$ limited to the channels of $\boldsymbol{q}$. Now, assume that an indexed subsequence $\boldsymbol{t}$ has distance with $\boldsymbol{q}$ less than τ_k. Then, by Eq. A.1 we know that $\frac{d(\tilde{\boldsymbol{t}}_{c_{\boldsymbol{q}}}, \tilde{\boldsymbol{q}}'_{c_{\boldsymbol{q}}})}{\sqrt{|\boldsymbol{q}|}} \leq d(\boldsymbol{t}, \boldsymbol{q}) \leq \tau_k$. This means that when we query the R-tree with a threshold $\tau_k * \sqrt{|\boldsymbol{q}|}$, by correctness of the R-tree algorithm, $\boldsymbol{t}$ is guaranteed to

be included in the returned set.[1] This proves that MS-Index is complete by guaranteeing that all subsequences in the k-NN are found. As we perform exact distance computations for all subsequences returned by this range query, the algorithm is guaranteed to identify all subsequences in the k-NN of $\boldsymbol{q}$, and therefore return the correct query result. □

[1] Note that the $\sqrt{|\boldsymbol{q}|}$ 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.

CHAPTER B

Proofs to Chapter 5

B.1 Proof of Lemma 5.3.1

Lemma B.1.1. Let $\rho(\boldsymbol{x}, \boldsymbol{y})$ denote the Pearson correlation between two z-normalized vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, and $\theta_{\boldsymbol{x},\boldsymbol{y}}$ the angle formed by these vectors. Consider four z-normalized vectors $\boldsymbol{u_1}$, $\boldsymbol{u_2}$, $\boldsymbol{v_1}$, and $\boldsymbol{v_2}$, such that $\theta_{\boldsymbol{v_1},\boldsymbol{u_1}} \leq \theta_1$ and $\theta_{\boldsymbol{v_2},\boldsymbol{u_2}} \leq \theta_2$. Then, correlation $\rho(\boldsymbol{u_1}, \boldsymbol{u_2})$ can be bounded as follows:

$$\cos(\theta^{\max}_{\boldsymbol{v_1},\boldsymbol{v_2}}) \leq \rho(\boldsymbol{u_1}, \boldsymbol{u_2}) \leq \cos(\theta^{\min}_{\boldsymbol{v_1},\boldsymbol{v_2}})$$

where $\theta^{\min}_{\boldsymbol{v_1},\boldsymbol{v_2}} = \max\left(0, \theta_{\boldsymbol{v_1},\boldsymbol{v_2}} - \theta_1 - \theta_2\right), \theta^{\max}_{\boldsymbol{v_1},\boldsymbol{v_2}} = \min\left(\pi, \theta_{\boldsymbol{v_1},\boldsymbol{v_2}} + \theta_1 + \theta_2\right)$.

Proof. Let m be the length of $\boldsymbol{x}$ and $\boldsymbol{y}$. Note that $\rho(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}^T \cdot \boldsymbol{y}}{m \cdot \sigma(\boldsymbol{x}) \cdot \sigma(\boldsymbol{y})}$. As we assume that $\boldsymbol{x}$ and $\boldsymbol{y}$ are z-normalized, $\sigma(\boldsymbol{x}) = \sqrt{\frac{1}{m}} \|\boldsymbol{x}\|_2$ and $\sigma(\boldsymbol{y}) = \sqrt{\frac{1}{m}} \|\boldsymbol{y}\|_2$. As such, $\rho(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\|_2 \cdot \|\boldsymbol{y}\|_2} = \cos(\theta_{\boldsymbol{x},\boldsymbol{y}})$. Now, let us consider $\theta_{\boldsymbol{u_1},\boldsymbol{u_2}}$ with $\boldsymbol{u_1}$ and $\boldsymbol{u_2}$ as defined in the lemma. Since $\theta_{\boldsymbol{u_1},\boldsymbol{v_1}} \leq \theta_1$ and $\theta_{\boldsymbol{u_2},\boldsymbol{v_2}} \leq \theta_2$, we have that

$$\theta_{\boldsymbol{v_1},\boldsymbol{v_2}} - \theta_1 - \theta_2 \leq \theta_{\boldsymbol{u_1},\boldsymbol{u_2}} \leq \theta_{\boldsymbol{v_1},\boldsymbol{v_2}} + \theta_1 + \theta_2$$

by the triangle inequality for angles. Under the convention that all angles between two time series are in $[0, \pi]$, we can rewrite this as

$$\max(0, \theta_{\boldsymbol{v_1},\boldsymbol{v_2}} - \theta_1 - \theta_2) \leq \theta_{\boldsymbol{u_1},\boldsymbol{u_2}} \leq \min(\pi, \theta_{\boldsymbol{v_1},\boldsymbol{v_2}} + \theta_1 + \theta_2)$$

Since $\cos(x)$ is a monotonically decreasing function on $[0, \pi]$, and $\cos(\theta_{\boldsymbol{u_1},\boldsymbol{u_2}}) = \rho(\boldsymbol{u_1}, \boldsymbol{u_2})$, we get the final result:

$$\cos(\min(\pi, \theta_{\boldsymbol{v_1},\boldsymbol{v_2}} + \theta_1 + \theta_2)) \leq \rho(\boldsymbol{u_1}, \boldsymbol{u_2}) \leq \cos(\max(0, \theta_{\boldsymbol{v_1},\boldsymbol{v_2}} - \theta_1 - \theta_2))$$

□

B.2 Proof of Theorem 5.3.2

Theorem B.2.1 (Bounds for *mPC*). For any pair of clusters C_i, C_j, let $l(C_i, C_j)$ and $u(C_i, C_j)$ denote lower/upper bounds on the pairwise correlations ρ between the cluster pair's materializations, i.e., $l(C_i, C_j) \leq \min_{\boldsymbol{x} \in C_i, \boldsymbol{y} \in C_j} \rho(\boldsymbol{x}, \boldsymbol{y})$ and $u(C_i, C_j) \geq \max_{\boldsymbol{x} \in C_i, \boldsymbol{y} \in C_j} \rho(\boldsymbol{x}, \boldsymbol{y})$. Consider sets of clusters $S_l = \{C_i^l\}_{i=1}^{p_l}$ and $S_r = \{C_j^r\}_{j=1}^{p_r}$. Let $L(S_1, S_2) = \sum_{C_i \in S_1, C_j \in S_2} l(C_i, C_j)$, and $U(S_1, S_2) = \sum_{C_i \in S_1, C_j \in S_2} u(C_i, C_j)$. Then, for any two sets of z-normalized time series $X = \{\hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_{p_l}\}$, $Y = \{\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_{p_r}\}$ such that $\hat{\boldsymbol{x}}_i \in C_i^l$, $\hat{\boldsymbol{y}}_i \in C_i^r$, multivariate correlation *mPC*(X, Y), can be bounded as follows:

1. if $L(S_l, S_r) \geq 0$:

$$\frac{L(S_l, S_r)}{\sqrt{U(S_l, S_l)}\sqrt{U(S_r, S_r)}} \leq \textit{mPC}(X, Y) \leq \frac{U(S_l, S_r)}{\sqrt{L(S_l, S_l)}\sqrt{L(S_r, S_r)}}$$

2. if $U(S_l, S_r) \leq 0$:

$$\frac{L(S_l, S_r)}{\sqrt{L(S_l, S_l)}\sqrt{L(S_r, S_r)}} \leq \textit{mPC}(X, Y) \leq \frac{U(S_l, S_r)}{\sqrt{U(S_l, S_l)}\sqrt{U(S_r, S_r)}}$$

3. else:

$$\frac{L(S_l, S_r)}{\sqrt{L(S_l, S_l)}\sqrt{L(S_r, S_r)}} \leq \textit{mPC}(X, Y) \leq \frac{U(S_l, S_r)}{\sqrt{L(S_l, S_l)}\sqrt{L(S_r, S_r)}}$$

Proof. The *mPC* correlation function can be rewritten as follows:

$$\begin{aligned}
\textit{mPC}(X, Y) = \rho(Avg(X), Avg(Y)) &= \frac{\sum_{i=1}^{m} Avg(X)_i \cdot Avg(Y)_i}{m \cdot \sigma(Avg(X)) \cdot \sigma(Avg(Y))} \\
&= \frac{\sum_{i=1}^{m} \left(\sum_{x \in X} x_i\right) \cdot \left(\sum_{y \in Y} y_i\right)}{m \cdot \sigma\left(\sum_{x \in X} x\right) \cdot \sigma\left(\sum_{y \in Y} y\right)} \\
&= \frac{\sum_{x \in X} \sum_{y \in Y} \frac{1}{m} \sum_{i=1}^{m} x_i \cdot y_i}{\sqrt{\sum_{i,j \in x} cov(i, j)} \cdot \sqrt{\sum_{i,j \in Y} cov(i, j)}} \\
&= \frac{\sum_{x \in X} \sum_{y \in Y} \rho(x, y)}{\sqrt{\sum_{i,j \in x} \rho(i, j)} \cdot \sqrt{\sum_{i,j \in Y} \rho(i, j)}}
\end{aligned}$$

Note that, under the mild assumption that the time series in X and the time series in Y are linearly independent, it holds that $\sigma\left(\sum_{x \in X} x\right) > 0$ and $\sigma\left(\sum_{y \in Y} y\right) > 0$. As such, it is safe to say that the denominator of the fraction will be positive for any 2 sets of time

series X, Y. Now, as S is a set of clusters rather than time series, the correlations $\rho(\cdot,\cdot)$ are not fixed, but we do have lower and upper bounds on the pairwise correlations between two clusters (i.e., derived via Lemma 5.3.1). We now proceed to bound the fraction by bounding the numerator and denominator separately:

1. $L(S_\text{l}, S_\text{r}) \leq \sum_{x\in X, y\in Y} \rho(x,y) \leq U(S_\text{l}, S_\text{r})$
2. $\sqrt{L(S_\text{l}, S_\text{l})}\sqrt{L(S_\text{r}, S_\text{r})} \leq \sqrt{\sum_{i,j\in \mathbf{X}} \rho(i,j)}\sqrt{\sum_{i,j\in \mathbf{Y}} \rho(i,j)} \leq \sqrt{U(S_\text{l}, S_\text{l})}\sqrt{U(S_\text{r}, S_\text{r})}$

In (2), we replace $L(\cdot,\cdot)$ with a small $\epsilon > 0$ if its value is negative, as we know that a standard deviation is positive (assuming linear independence). Subsequently distinguishing the three cases as in Theorem 5.3.2 leads to the result. □

B.3 Proof of Theorem 5.3.3

Theorem B.3.1 (Bounds for *MP*). For any pair of clusters C_i, C_j, let $l(C_i, C_j)$ and $u(C_i, C_j)$ denote lower / upper bounds on the pairwise correlations between the cluster's materializations, i.e., $l(C_i, C_j) \leq \min_{\boldsymbol{x}\in C_i, \boldsymbol{y}\in C_j} \rho(\boldsymbol{x}, \boldsymbol{y})$ and $u(C_i, C_j) \geq \max_{\boldsymbol{x}\in C_i, \boldsymbol{y}\in C_j} \rho(\boldsymbol{x}, \boldsymbol{y})$. Consider the set of clusters $S = \{C_i\}_{i=1}^p$. Furthermore, let $\mathbf{L}$ and $\mathbf{U}$ be symmetric matrices such that $\boldsymbol{L}_{ij} = l(C_i, C_j)$ and $\boldsymbol{U}_{ij} = u(C_i, C_j)\ \forall 1 \leq i, j \leq p$. For any set of **z-normalized** time series $X = \{\hat{\boldsymbol{x}}_1, \hat{\boldsymbol{x}}_2, \ldots, \hat{\boldsymbol{x}}_p\}$ such that $\hat{\boldsymbol{x}}_i \in C_i$, multipole correlation $MP(X)$ can be bounded as follows:

$$MP(X) \in 1 - \lambda_{min}\left(\frac{\boldsymbol{L}+\boldsymbol{U}}{2}\right) \pm \frac{1}{2}||\boldsymbol{U}-\boldsymbol{L}||_2 \tag{B.1}$$

where $\lambda_{min}\left(\frac{\boldsymbol{L}+\boldsymbol{U}}{2}\right)$ is the smallest eigenvalue of matrix $\left(\frac{\boldsymbol{L}+\boldsymbol{U}}{2}\right)$.

Proof. The multipole correlation $MP(X)$ can be rewritten as [10]:

$$MP(X) = 1 - \lambda_{\text{min}}(R)$$

where $\lambda_{\text{min}}(A)$ denotes the smallest eigenvalue of a matrix A, and R is the correlation matrix of the time series in X. As $S = \{C_1, \ldots, C_n\}$ is a set of clusters, the elements of R, r_{ij}, are not fixed, but there are two matrices L and U s.t. $\forall_{i,j\in\{0,\ldots,n\}} : l_{ij} \leq r_{ij} \leq u_{ij}$. In what follows, let R denote any correlation matrix corresponding to a set of time series X in which each of its elements $x_i \in C_i$. Furthermore, for conciseness, let $M = \frac{U+L}{2}$ and $E = R - M$. Then:

$$\lambda_{\text{min}}(R) = \lambda_{\text{min}}(M + E)$$

Using Weyl's inequality [138], we can derive:

$$|\lambda_{\min}(M+E) - \lambda_{\min}(M)| \leq ||E||_2 \Leftrightarrow$$
$$|\lambda_{\min}(R) - \lambda_{\min}(M)| \leq ||E||_2$$

B

To complete the proof, it remains to be shown that $||E||_2 \leq \frac{1}{2}||U-L||_2$. To this end, let $x^* = \underset{||x||_2=1}{argmax}||Ex||_2$, and let y denote the time series containing the absolute values of the elements in x^* : $y_i = |x_i^*|$. Note that $||y||_2 = 1$. Then:

$$\begin{aligned}
||E||_2 &= \max_{||x||_2=1} ||Ex||_2 \\
&= ||Ex^*||_2 \\
&= \sqrt{\sum_{i=1}^{p}\left(\sum_{j=1}^{p}\left(r_{ij} - \frac{u_{ij}+l_{ij}}{2}\right)x_j^*\right)^2} \\
&\leq \sqrt{\sum_{i=1}^{p}\left(\sum_{j=1}^{p}\left|r_{ij} - \frac{u_{ij}+l_{ij}}{2}\right|\left|x_j^*\right|\right)^2} \\
&\leq \sqrt{\sum_{i=1}^{p}\left(\sum_{j=1}^{p}\left|u_{ij} - \frac{u_{ij}+l_{ij}}{2}\right| y_j\right)^2} \\
&= \sqrt{\sum_{i=1}^{p}\left(\sum_{j=1}^{p}\left(\frac{u_{ij}-l_{ij}}{2}\right) y_j\right)^2} \\
&= ||\left(\frac{U-L}{2}\right) y||_2 \\
&\leq \max_{||v||_2=1} ||\left(\frac{U-L}{2}\right) v||_2 \\
&= \frac{1}{2}||U-L||_2
\end{aligned}$$

□

CHAPTER C

Proofs to Chapter 6

C.1 Proof of Lemma 6.4.2

Lemma C.1.1 (Bivariate distance bounds). Let d be a bivariate distance function that satisfies the triangle inequality. Then, given two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^m$, each included in a cluster C_X and C_Y, respectively, one can bound $d(\boldsymbol{x}, \boldsymbol{y})$ as follows using the triangle inequality:

$$d(\boldsymbol{c}_X, \boldsymbol{c}_Y) - r_X - r_Y \leq d(\boldsymbol{x}, \boldsymbol{y}) \leq d(\boldsymbol{c}_X, \boldsymbol{c}_Y) + r_X + r_Y \tag{C.1}$$

with $\boldsymbol{c}_X$ and $\boldsymbol{c}_Y$ being the centroids of C_X and C_Y, respectively, and r_X and r_Y being the radii of C_X and C_Y. The radius of a cluster is the maximum distance between the centroid and any point in the cluster, according to d, i.e., $r_X = \max_{\boldsymbol{v} \in C_X} d(\boldsymbol{c}_X, \boldsymbol{v})$.

Proof.

$$\begin{aligned}
d(\boldsymbol{x}, \boldsymbol{y}) &\leq d(\boldsymbol{x}, \boldsymbol{c}_X) + d(\boldsymbol{y}, \boldsymbol{c}_X) && \text{(Triangle inequality)} \\
&\leq d(\boldsymbol{x}, \boldsymbol{c}_X) + d(\boldsymbol{c}_X, \boldsymbol{c}_Y) + d(\boldsymbol{y}, \boldsymbol{c}_Y) && \text{(Triangle inequality)} \\
&\leq d(\boldsymbol{c}_X, \boldsymbol{c}_Y) + \max_{\boldsymbol{v} \in C_X} d(\boldsymbol{v}, \boldsymbol{c}_X) + \max_{\boldsymbol{w} \in C_Y} d(\boldsymbol{w}, \boldsymbol{c}_Y) && \text{(Radius definition)} \\
&= d(\boldsymbol{c}_X, \boldsymbol{c}_Y) + r_X + r_Y
\end{aligned}$$

and

$$\begin{aligned}
d(\boldsymbol{c}_X, \boldsymbol{c}_Y) &\leq d(\boldsymbol{x}, \boldsymbol{c}_X) + d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, \boldsymbol{c}_Y) && \text{(2x Triangle inequality)} \\
\leftrightarrow d(\boldsymbol{x}, \boldsymbol{y}) &\geq d(\boldsymbol{c}_X, \boldsymbol{c}_Y) - d(\boldsymbol{x}, \boldsymbol{c}_X) - d(\boldsymbol{y}, \boldsymbol{c}_Y) && \text{(Rearranging)} \\
&\geq d(\boldsymbol{c}_X, \boldsymbol{c}_Y) - \max_{\boldsymbol{v} \in C_X} d(\boldsymbol{v}, \boldsymbol{c}_X) - \max_{\boldsymbol{w} \in C_Y} d(\boldsymbol{w}, \boldsymbol{c}_Y) && \text{(Radius definition)} \\
&= d(\boldsymbol{c}_X, \boldsymbol{c}_Y) - r_X - r_Y
\end{aligned}$$

□

C.2 Proof of Lemma 6.4.3

Lemma C.2.1 (Centroid of an Aggregated Hypersphere Cluster). Given a set of spherical clusters $S = \{C_i\}_{i=1}^{p}$ with each cluster covering a set of vectors $C_i \subseteq \mathcal{V}$ and a linear map $\text{agg}(X) = \omega^T * X$, the centroid of the hypersphere C_S covering all points in the Minkowski aggregation of S is as follows;

$$\boldsymbol{c}_S = \omega^T [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_p] \tag{C.2}$$

with $\boldsymbol{c}_i$ being the centroid of the cluster C_i.

C

Proof.

$$\begin{aligned}
C_S &= \omega_1 C_1 \oplus \ldots \oplus \omega_p C_p && \text{(Minkowski sum)} \\
\boldsymbol{c}_S &= \frac{\sum_{\boldsymbol{v}_1 \in C_1} \cdots \sum_{\boldsymbol{v}_p \in C_p} \omega^T [\boldsymbol{v}_i, \ldots, \boldsymbol{v}_p]}{\prod_{i=1}^{p} |C_i|} && \text{(Centroid definition)} \\
&= \omega_1 * \frac{\sum_{\boldsymbol{v}_1 \in C_1} \boldsymbol{v}_1}{|C_1|} + \ldots + \omega_p * \frac{\sum_{\boldsymbol{v}_p \in C_p} \boldsymbol{v}_p}{|C_p|} && \text{(Rearranging)} \\
&= \omega_1 * \boldsymbol{c}_1 + \ldots + \omega_p * \boldsymbol{c}_p && \text{(Centroid definition)} \\
&= \omega^T * [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_p] && \text{(Vectorization)}
\end{aligned}$$

□

C.3 Proof of Lemma 6.4.4

Lemma C.3.1 (Radius of an Aggregated Hypersphere Cluster). Given a set of spherical clusters $S = \{C_i\}_{i=1}^{p}$ with each cluster covering a set of vectors $C_i \subseteq \mathcal{V}$ and radius defined using a distance metric d on a *vector space* S with norm $\|\boldsymbol{x}\|_d = d(\boldsymbol{x}, \boldsymbol{0})$, and a linear map $\text{agg}(X) = \omega^T * X$, the radius of the hypersphere C_S covering all points in the Minkowski aggregation of S is bounded as follows;

$$r_S = \omega^T * [r_1, \ldots, r_p] \leq \sum_{i=1}^{p} |\omega_i| r_i \tag{C.3}$$

with $\boldsymbol{c}_i$ being the centroid of the cluster C_i and r_i being the radius of the cluster C_i.

Proof.

$$
\begin{aligned}
r_S &= \max_{\boldsymbol{v}_1 \in C_1, \ldots, \boldsymbol{v}_p \in C_p} d(\boldsymbol{c}_S, \omega^T * [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_p]) && \text{(Radius definition)} \\
&= \max_{\boldsymbol{v}_1 \in C_1, \ldots, \boldsymbol{v}_p \in C_p} d(\omega^T * [\boldsymbol{c}_1, \ldots, \boldsymbol{c}_p], \omega^T * [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_p]) && \text{(Lemma 6.4.3)} \\
&= \max_{\boldsymbol{v}_1 \in C_1, \ldots, \boldsymbol{v}_p \in C_p} d(\sum_{i=1}^{p} \omega_i \boldsymbol{c}_i, \sum_{i=1}^{p} \omega_i \boldsymbol{v}_i) && \text{(De-vectorization)} \\
&= \max_{\boldsymbol{v}_1 \in C_1, \ldots, \boldsymbol{v}_p \in C_p} \left\| \sum_{i=1}^{p} \omega_i (\boldsymbol{c}_i - \boldsymbol{v}_i) \right\|_d && \text{(Norm definition)} \\
&\leq \max_{\boldsymbol{v}_1 \in C_1, \ldots, \boldsymbol{v}_p \in C_p} \sum_{i=1}^{p} \| |\omega_i| (\boldsymbol{c}_i - \boldsymbol{v}_i) \|_d && \text{(Triangle inequality)} \\
&= \max_{\boldsymbol{v}_1 \in C_1, \ldots, \boldsymbol{v}_p \in C_p} \sum_{i=1}^{p} |\omega_i| \, \| (\boldsymbol{c}_i - \boldsymbol{v}_i) \|_d && \text{(Absolute homogeneity)} \\
&\leq \sum_{i=1}^{p} |\omega_i| r_i && \text{(Radius definition)} \\
&= \omega^T * [r_1, \ldots, r_p] && \text{(Vectorization)}
\end{aligned}
$$

□

CHAPTER D

Proofs to Chapter 7

D.1 Proof of Equivalence of Sparse Representation and Multipole Maximization

We show that the constrained sparse representation problem is equivalent to a standard top-1 multipole query performed on a translated version of the original dataset. This approach embeds the query vector into the dataset, aligning the objectives of the two problems.

Theorem D.1.1. Given a dataset of vectors $\boldsymbol{D} = [\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_n] \in \mathbb{R}^{m \times n}$, a query vector $\boldsymbol{q} \in \mathbb{R}^m$, and a maximum set cardinality p, the constrained sparse representation problem (cf. Equation D.1):

$$\min_{\boldsymbol{x}} \|\boldsymbol{q} - \boldsymbol{D}\boldsymbol{x}\|_2^2 \text{ subject to } \|\boldsymbol{x}\|_0 \leq p$$

is equivalent to finding the top-1 multipole combination S' from a transformed dataset $\boldsymbol{D}'$, where each vector $\boldsymbol{d}'_i \in \boldsymbol{D}'$ is defined as $\boldsymbol{d}'_i = \boldsymbol{d}_i - \boldsymbol{q}$. The solution to Equation D.1 is the subset $S \subseteq \boldsymbol{D}$ corresponding to the vectors in S'.

Proof. The proof shows that the sparse representation problem's objective is equivalent to the multipole query objective on the transformed dataset.

The Sparse Representation Objective: The constrained sparse representation problem seeks a subset of vectors $S \subseteq \boldsymbol{D}$ (the support of $\boldsymbol{x}$) such that their linear combination best reconstructs the query vector $\boldsymbol{q}$. Geometrically, this means finding the subset S whose spanned subspace, $span(S)$, is closest to the vector $\boldsymbol{q}$. This is achieved when the reconstruction error, or residual, is minimized:

$$\min_{\boldsymbol{x}} \|\boldsymbol{q} - \boldsymbol{D}\boldsymbol{x}\|_2^2 \text{ subject to } \|\boldsymbol{x}\|_0 \leq p \tag{D.1}$$

where $\boldsymbol{D}_S$ is the matrix of vectors in S and $\boldsymbol{x}_S$ is the corresponding sparse vector of coefficients. The constraint $\|\boldsymbol{x}\|_0 \leq p$ limits the number of vectors in S to at most p. The optimal solution is the subset S for which this error is the smallest.

The Transformed Dataset and Multipole Objective: Let us define a new dataset $\boldsymbol{D}' = \{\boldsymbol{D}'_i | \boldsymbol{D}'_i = \boldsymbol{d}_i - \boldsymbol{q}, \forall \boldsymbol{d}_i \in \boldsymbol{D}\}$. Now, consider a standard top-1 multipole query on this dataset $\boldsymbol{D}'$. The goal of this query is to find the subset $S' \subseteq \boldsymbol{D}'$ of size at most p that is maximally linearly dependent. According to the definition of the multipole value, this is the subset S' for which the term $\min_{\|v\|_2=1} \mathrm{Var}(\boldsymbol{D}'_{S'} \cdot \boldsymbol{v})$ is minimized, where $\boldsymbol{D}'_{S'}$ is the matrix of vectors in S' (see Definition 12).

Establishing Equivalence: The multipole query on $\boldsymbol{D}'$ finds the subset S' whose vectors are "closest to being linearly dependent." This occurs when there exists some non-zero linear combination of the vectors in S' that is approximately equal to the zero vector [103, 104]: $\boldsymbol{D}'_{S'}\boldsymbol{v} \approx 0$. Substituting the definition of our transformed vectors, $\boldsymbol{d}'_i = \boldsymbol{d}_i - \boldsymbol{q}$, we can rewrite this condition in terms of the original vectors in the corresponding subset $S \subseteq \boldsymbol{D}$:

$$\sum_{j \in S} v_j (\boldsymbol{d}_j - \boldsymbol{q}) \approx 0 \tag{D.2}$$

$$\sum_{j \in S} v_j \boldsymbol{d}_j - \sum_{j \in S} v_j \boldsymbol{q} \approx 0 \tag{D.3}$$

$$\sum_{j \in S} v_j \boldsymbol{d}_j \approx \left(\sum_{j \in S} v_j \right) \boldsymbol{q} \tag{D.4}$$

This shows that a linear combination of the vectors in S can approximate a *scaled* version of the query vector $\boldsymbol{q}$, where the scaling factor is the sum of the coefficients v_j for $j \in S$. To show that this relationship is equivalent in form to the sparse representation problem where $\boldsymbol{q} \approx \sum_{j \in S} v_j \boldsymbol{d}_j$, we perform simple algebraic rescaling. Namely, assuming the sum of the coefficients is non-zero, we divide Equation D.4 by this scalar constant and define a new set of coefficients $\boldsymbol{x}_S$ with $x_{S,j} = \frac{v_j}{\sum_{k \in S} v_k}$ for $j \in S$. This gives us:

$$\sum_{j \in S} v_j \boldsymbol{d}_j \approx \left(\sum_{j \in S} v_j \right) \boldsymbol{q} \Leftrightarrow \frac{\sum_{j \in S} v_j \boldsymbol{d}_j}{\sum_{k \in S} v_k} \approx \boldsymbol{q} \Leftrightarrow \boldsymbol{D}_S \boldsymbol{x} \approx \boldsymbol{q}$$

where the latter is equivalent to the sparse representation problem in Equation D.1. This confirms that a subset S' that reconstructs the query vector $\boldsymbol{q}$ with minimal error will also maximize the multipole score on the transformed dataset $\boldsymbol{D}'$, making it the solution to the top-1 multipole query.

Conclusion The subset $S' \subseteq \boldsymbol{D}'$ that is most linearly dependent corresponds exactly to the subset $S \subseteq \boldsymbol{D}$ that best reconstructs the query vector $\boldsymbol{q}$. The dataset transformation aligns the objectives of both problems, proving their equivalence. □

CHAPTER E

Distance Measure Definitions

This appendix provides the mathematical definitions for the distance measures discussed in Chapter 3.

Table E.1: Mathematical Definitions of Distance Measures

Measure	Definition	Notes
	Lock-step Measures	
L_p	$L_p(\boldsymbol{x}, \boldsymbol{y}) = \sqrt[p]{\sum_{i=1}^{c} \sum_{j=1}^{m} \lvert \boldsymbol{x}_j^{(i)} - \boldsymbol{y}_j^{(i)} \rvert^p}$	
Euclidean (L_2)	$L_2(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{c} \sum_{j=1}^{m} (\boldsymbol{x}_j^{(i)} - \boldsymbol{y}_j^{(i)})^2}$	
Manhattan (L_1)	$L_1(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{c} \sum_{j=1}^{m} \lvert \boldsymbol{x}_j^{(i)} - \boldsymbol{y}_j^{(i)} \rvert$	
Lorentzian	$D(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{c} \sum_{j=1}^{m} \ln(1 + \lvert \boldsymbol{x}_j^{(i)} - \boldsymbol{y}_j^{(i)} \rvert)$	
$Avg(L_1, L_\infty)$	$D(\boldsymbol{x}, \boldsymbol{y}) = \frac{L_1(\boldsymbol{x}, \boldsymbol{y}) + \sum_{i=1}^{c} \max_{j \in \{1, \ldots, m\}} \left\lvert \boldsymbol{x}_j^{(i)} - \boldsymbol{y}_j^{(i)} \right\rvert}{2}$	
Canberra	$D(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{c} \sum_{j=1}^{m} \frac{\lvert \boldsymbol{x}_j^{(i)} - \boldsymbol{y}_j^{(i)} \rvert}{\lvert \boldsymbol{x}_j^{(i)} \rvert + \lvert \boldsymbol{y}_j^{(i)} \rvert}$	For $\boldsymbol{x}_j^{(i)} + \boldsymbol{y}_j^{(i)} \neq 0$.
Chord	$D(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{c} \sum_{j=1}^{m} \left(\sqrt{\lvert \boldsymbol{x}_j^{(i)} \rvert} - \sqrt{\lvert \boldsymbol{y}_j^{(i)} \rvert} \right)^2$	
Clark	$D(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{c} \sum_{j=1}^{m} \left(\frac{\lvert \boldsymbol{x}_j^{(i)} - \boldsymbol{y}_j^{(i)} \rvert}{\lvert \boldsymbol{x}_j^{(i)} \rvert + \lvert \boldsymbol{y}_j^{(i)} \rvert} \right)^2}$	For $\boldsymbol{x}_j^{(i)} + \boldsymbol{y}_j^{(i)} \neq 0$.
Jaccard	$D(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i=1}^{c} \sum_{j=1}^{m} (\boldsymbol{x}_j^{(i)} - \boldsymbol{y}_j^{(i)})^2}{\sum_{i=1}^{c} \sum_{j=1}^{m} \left((\boldsymbol{x}_j^{(i)})^2 + (\boldsymbol{y}_j^{(i)})^2 \right) - \sum_{i=1}^{c} \sum_{j=1}^{m} (\boldsymbol{x}_j^{(i)} \boldsymbol{y}_j^{(i)})}$	

Continued on next page

Table E.1 – continued from previous page

Measure	Definition	Notes
Soergel	$D(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i=1}^{c}\sum_{j=1}^{m} \lvert \boldsymbol{x}_j^{(i)} - \boldsymbol{y}_j^{(i)} \rvert}{\sum_{i=1}^{c}\sum_{j=1}^{m} \max(\boldsymbol{x}_j^{(i)}, \boldsymbol{y}_j^{(i)})}$	
Emanon4	$D(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i=1}^{c}\sum_{j=1}^{m} (\boldsymbol{x}_j^{(i)} - \boldsymbol{y}_j^{(i)})^2}{\sum_{i=1}^{c}\sum_{j=1}^{m} \max(\boldsymbol{x}_j^{(i)}, \boldsymbol{y}_j^{(i)})}$	
Topsoe	$D(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{c}\sum_{j=1}^{m} \left(\boldsymbol{x}_j^{(i)} \ln \frac{2\boldsymbol{x}_j^{(i)}}{\boldsymbol{x}_j^{(i)} + \boldsymbol{y}_j^{(i)}} + \boldsymbol{y}_j^{(i)} \ln \frac{2\boldsymbol{y}_j^{(i)}}{\boldsymbol{x}_j^{(i)} + \boldsymbol{y}_j^{(i)}} \right)$	
Sliding Measures		
SBD-D	$SBD\text{-}D(\boldsymbol{x}, \boldsymbol{y}) = 1 - \max_w(NCC2_w(\boldsymbol{x}, \boldsymbol{y}))$ $NCC2_w(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{c} \frac{\sum_{j=1}^{m-w} \boldsymbol{x}_j^{(i)} \cdot \boldsymbol{y}_{j+w}^{(i)}}{\lVert \boldsymbol{x}^{(i)} \rVert_2 \cdot \lVert \boldsymbol{y}^{(i)} \rVert_2}$	w is the temporal shift. $NCC2$ is the normalized cross-correlation.
SBD-I	$SBD\text{-}I = \sum_{i=1}^{c} SBD\text{-}D(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$	Passed as MTS with 1 channel.
Elastic Measures		
DTW-D	$DTW\text{-}D(\boldsymbol{x}, \boldsymbol{y}) = Cost(\lvert \boldsymbol{x} \rvert, \lvert \boldsymbol{y} \rvert)$ $c^D(i,j) = c^V(i,j) = c^H(i,j) = L_2(\boldsymbol{x}_i, \boldsymbol{y}_j)$	See Section 3.4 for definition of $Cost(i,j)$ c^D, c^V, and c^H are the cost functions for diagonal, vertical, and horizontal moves.
DTW-I	$\sum_{i=1}^{c} DTW\text{-}D(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$	Passed as MTS with 1 channel.
LCSS-D	$LCSS\text{-}D(\boldsymbol{x}, \boldsymbol{y}) = 1 - \frac{Cost(\lvert \boldsymbol{x} \rvert, \lvert \boldsymbol{y} \rvert)}{\min(\lvert \boldsymbol{x} \rvert, \lvert \boldsymbol{y} \rvert)}$ $c^D(i,j) = \begin{cases} 1 & \text{if } \lvert \boldsymbol{x}_i - \boldsymbol{y}_j \rvert \geq \epsilon \\ 0 & \text{if } \lvert \boldsymbol{x}_i - \boldsymbol{y}_j \rvert < \epsilon \end{cases}$ $c^V(i,j) = c^H(i,j) = 0$	See Section 3.4 for definition of $Cost(i,j)$
ERP-D	$ERP\text{-}D(\boldsymbol{x}, \boldsymbol{y}) = Cost(\lvert \boldsymbol{x} \rvert, \lvert \boldsymbol{y} \rvert)$ $c^D(i,j) = L_2(\boldsymbol{x}_i, \boldsymbol{y}_j)$ $c^V(i,j) = L_2(\boldsymbol{x}_i, g)$ $c^H(i,j) = L_2(\boldsymbol{y}_i, g)$	See Section 3.4 for definition of $Cost(i,j)$ g is the gap value parameter.
ERP-I	$\sum_{i=1}^{c} ERP\text{-}D(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$	Passed as MTS with 1 channel.
MSM-D	$MSM\text{-}D(\boldsymbol{x}, \boldsymbol{y}) = Cost(\lvert \boldsymbol{x} \rvert, \lvert \boldsymbol{y} \rvert)$ $c^D(i,j) = L_1(\boldsymbol{x}_i, \boldsymbol{y}_j)$ $c^V(i,j) = \begin{cases} c & \text{if } L_2(\boldsymbol{x}_i, \frac{\boldsymbol{x}_{i-1}+\boldsymbol{y}_i}{2}) \leq L_2(\boldsymbol{x}_{i-1}, \boldsymbol{y}_i) \\ c + \min(L_1(\boldsymbol{x}_{i-1}, \boldsymbol{x}_i), L_1(\boldsymbol{x}_i, \boldsymbol{y}_i)) & \text{otherwise} \end{cases}$	See Section 3.4 for definition of $Cost(i,j)$

Continued on next page

Table E.1 – continued from previous page

Measure	Definition	Notes
	$c^H(i,j) = \begin{cases} c & \text{if } L_2(\boldsymbol{y}_i, \frac{\boldsymbol{y}_{i-1}+\boldsymbol{x}_i}{2}) \leq L_2(\boldsymbol{y}_{i-1}, \boldsymbol{x}_i) \\ c + \min(L_1(\boldsymbol{y}_{i-1}, \boldsymbol{y}_i), L_1(\boldsymbol{x}_i, \boldsymbol{y}_i)) & \text{otherwise} \end{cases}$	
MSM-I	$\sum_{i=1}^{c} MSM\text{-}D(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$	Passed as MTS with 1 channel.
TWE-D	$TWE\text{-}D(\boldsymbol{x}, \boldsymbol{y}) = Cost(\lvert\boldsymbol{x}\rvert, \lvert\boldsymbol{y}\rvert)$ $c^D(i,j) = L_2(\boldsymbol{x}_i, \boldsymbol{y}_j) + L_2(\boldsymbol{x}_{i-1}, \boldsymbol{y}_{j-1}) + 2\nu\lvert i-j\rvert$ $c^V(i,j) = L_2(\boldsymbol{x}_i, \boldsymbol{x}_{i-1}) + \nu + \lambda$ $c^H(i,j) = L_2(\boldsymbol{y}_i, \boldsymbol{y}_{j-1}) + \nu + \lambda$	See Section 3.4 for definition of $Cost(i,j)$ ν and λ are parameters.
TWE-I	$\sum_{i=1}^{c} TWE\text{-}D(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$	Passed as MTS with 1 channel.
	Kernel Measures	
RBF	$K_{RBF}(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\gamma * L_2(\boldsymbol{x}, \boldsymbol{y})^2\right)$	γ is the kernel parameter.
GAK-D	$D_{GAK}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\pi \in A(m,n)} \prod_{(i,j)\in\pi} K_{RBF}(\boldsymbol{x}_i, \boldsymbol{y}_j)$	Sum is over all alignment paths π.
GAK-I	$\sum_{i=1}^{c} D_{GAK}(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$	Passed as MTS with 1 channel.
KDTW-D	Uses a $K_{DTW}(\boldsymbol{x}_i, \boldsymbol{y}_j)$ instead of distance in the DTW recursion. $K_{DTW}(\boldsymbol{x}_i, \boldsymbol{y}_j) = \frac{K_{RBF}(\boldsymbol{x}_i, \boldsymbol{y}_j) + \epsilon}{3*(1+\epsilon)}$	 ϵ is the kernel parameter.
KDTW-I	$\sum_{i=1}^{c} KDTW\text{-}D(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$	Passed as MTS with 1 channel.
SINK-D	$D_{SINK}(\boldsymbol{x}, \boldsymbol{y}) = \frac{K_{SINK}(\boldsymbol{x}, \boldsymbol{y})}{\sqrt{K_{SINK}(\boldsymbol{x}, \boldsymbol{x}) K_{SINK}(\boldsymbol{y}, \boldsymbol{y})}}$ $K_{SINK}(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(\gamma * NCC2(\boldsymbol{x}, \boldsymbol{y})\right)$	 γ is the kernel parameter.
SINK-I	$\sum_{i=1}^{c} D_{SINK}(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$	Passed as MTS with 1 channel.
	Feature-based Measures	
$D_{Catch22}$-I	$D(\boldsymbol{x}, \boldsymbol{y}) = L_2(f_k(\boldsymbol{x}) - f_k(\boldsymbol{y}))^2$	f_k are the 22 features from the catch22 set. Applied channel-wise, forming a $22 * c$ size feature vector.
$D_{TSFresh}$-I	$D(\boldsymbol{x}, \boldsymbol{y}) = L_2(f_k(\boldsymbol{x}) - f_k(\boldsymbol{y}))^2$	f_k are the features from the TSFresh library. Applied channel-wise, forming a feature vector of size $\lvert f\rvert * c$.
	Model-based Measures	

Continued on next page

Table E.1 – continued from previous page

Measure	Definition		Notes
KL_{Gauss}-D	$D_{KL}(\boldsymbol{x}, \boldsymbol{y})$ $D_{KL}(\mathcal{N}(\mu_{\boldsymbol{x}}, \Sigma_{\boldsymbol{x}})\|\|\mathcal{N}(\mu_{\boldsymbol{y}}, \Sigma_{\boldsymbol{y}}))$	=	D_{KL} is the Kullback-Leibler divergence. $\mu_{\boldsymbol{x}}$ are the channel-wise means of $\boldsymbol{x}$, $\Sigma_{\boldsymbol{x}}$ is the covariance matrix of $\boldsymbol{x}$.
KL_{Gauss}-I	$\sum_{i=1}^{c} D_{KL}(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$		Passed as MTS with 1 channel.
KL_{HMM}-D	$D_{HMM}(\boldsymbol{x}, \boldsymbol{y})$ $\sum_{i=1}^{n} \tilde{P}(\boldsymbol{t}_i \mid \lambda_{\boldsymbol{x}}) \log \frac{\tilde{P}(\boldsymbol{t}_i \mid \lambda_{\boldsymbol{x}})}{\tilde{P}(\boldsymbol{t}_i \mid \lambda_{\boldsymbol{y}})}$	=	Following the definition of [100]. $\lambda_{\boldsymbol{x}}$ and $\lambda_{\boldsymbol{y}}$ are the HMMs fitted to $\boldsymbol{x}$ and $\boldsymbol{y}$, $\boldsymbol{t}_i$ is the i-th time series in the dataset, $\tilde{P}(\boldsymbol{t}_i \mid \lambda)$ is the likelihood of $\boldsymbol{t}_i$ given the model λ.
KL_{HMM}-I	$\sum_{i=1}^{c} D_{HMM}(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$		Passed as MTS with 1 channel.
	Embedding Measures		
TS2Vec-D/TLoss-D	$D(\boldsymbol{x}, \boldsymbol{y}) = L_2(\text{Enc}(\boldsymbol{x}), \text{Enc}(\boldsymbol{y}))$		Enc$(\cdot)$ is the learned encoder model.
TS2Vec-I/TLoss-I	$D(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{c} L_2(\text{Enc}(\boldsymbol{x}^{(i)}), \text{Enc}(\boldsymbol{y}^{(i)}))$		Passed as MTS with 1 channel.
GRAIL-D	$D_{GRAIL}(\boldsymbol{T}_i, \boldsymbol{T}_j) = L_2(\boldsymbol{Z}_i, \boldsymbol{Z}_j)$ $Z = EQ_W\Lambda_W^{-0.5}$		$\boldsymbol{T}_i$ is the i-th time series in dataset $T \in \mathbb{R}^{n \times c \times m}$, $\boldsymbol{Z}_i$ is its representation, which is the i-th row of the matrix $Z \in \mathbb{R}^{n \times c \times d}$, where d is the embedding dimension. $E \in \mathbb{R}^{n \times d}$ is the distance matrix between T and a dictionary of time series $D \in \mathbb{R}^{d \times c \times m}$ using D_{SINK}, Q_W is the orthogonal matrix of eigenvectors of W, Λ_W is the diagonal matrix of eigenvalues of W. $W \in \mathbb{R}^{d \times d}$ is the distance matrix between D and D using D_{SINK}.
GRAIL-I	$D(\boldsymbol{T}_i, \boldsymbol{T}_j)$ $\sum_{k=1}^{c} L_2(Emb(\boldsymbol{T}^{(k)})_i, Emb(\boldsymbol{T}^{(k)})_j)$	=	$Emb(\boldsymbol{T}^{(k)})$ is the embedding of dataset $\boldsymbol{T}$ using the GRAIL method, only using the k-th channel.
D_{PCA}	$D_{PCA}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{1+S_{PCA}(\boldsymbol{x}, \boldsymbol{y})}$		

Continued on next page

Table E.1 – continued from previous page

Measure	Definition	Notes
	$S_{PCA}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{k} \sum_{j=1}^{k} \cos^2(\theta_i, \theta_j)$	θ_i and θ_j are the i-th and j-th principal components of $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. k is the number of principal components used.
D_{Eros}	$D_{Eros}(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{2 - 2S_{Eros}(\boldsymbol{x}, \boldsymbol{y})}$ $S_{Eros}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{d} \omega_i \lvert\cos(\theta_i, \theta_i)\rvert$	θ_i and θ_j are the i-th and j-th eigenvectors of the covariance matrix of $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. d is the number of components. ω is a weight vector based on the (normalized) eigenvalues of MTS in a training set (detailed in [243]).
	Ensemble Measures	
Ensemble	$D_{Ensemble}(\boldsymbol{x}, \boldsymbol{y}, \mathcal{D}) = \sum_{D \in \mathcal{D}} \bar{D}(\boldsymbol{x}, \boldsymbol{y})$	$\boldsymbol{x}$, $\boldsymbol{y}$ are time series in a dataset $T \in \mathbb{R}^{n \times c \times m}$, $\mathcal{D}$ is a set of distance measures.
	$\bar{D}(\boldsymbol{x}, \boldsymbol{y}) = \frac{D(\boldsymbol{x}, \boldsymbol{y}) - \min_{\boldsymbol{a}, \boldsymbol{b} \in T} D(\boldsymbol{a}, \boldsymbol{b})}{\max_{\boldsymbol{a}, \boldsymbol{b} \in T} D(\boldsymbol{a}, \boldsymbol{b}) - \min_{\boldsymbol{a}, \boldsymbol{b} \in T} D(\boldsymbol{a}, \boldsymbol{b})}$	Normalized distance.

CHAPTER F

Distribution of Contributions in Chapter 5

This appendix provides an overview of the contributions made in Chapter 5 and their distribution across the two degrees involved in this work (Master's thesis and PhD thesis). The contributions are categorized based on the degree they were part of, with each contribution explicitly mapped to the relevant section of the chapter. An important contribution made during the PhD phase is the development of additional optimizations, and a total re-implementation of the CD algorithm, which significantly improved its performance. To quantify the impact of these changes, we compare the performance of the original CD algorithm and the version discussed in Chapter 5 in Section F.2.

F.1 Distribution of Contributions

Results from the Master's thesis [72] or from work of other people [170, 171]:

- **Core CD algorithm** [170, 171]. This includes the theoretical results on *mPC* and *MP* measures (i.e., Theorem 5.3.2 and Theorem 5.3.3), and the bound-based search strategy for threshold and top-k queries (i.e., Algorithm 3), discussed in Section 5.3.

- **Core CDStream algorithm** [72, 171]. This involves the use of an index to monitor changes in pairwise correlations in DCCs, aimed to to detect potential changes in the result set, discussed in Section 5.4.

- **Core CDHybrid algorithm** [72, 171]. This algorithm combines the strengths of both CD and CDStream on streaming data, adaptively switching between the two methods based on the rate and nature of arriving data, discussed in Section 5.4.5.

Results from the work performed during the PhD phase:

- **New top-k query algorithm**. This algorithm is a significant improvement over the original top-k algorithm, which was based on a batch processing approach. The new algorithm breaks up the search into two phases and introduces a new routing prin-

ciple for parsing the hierarchy of cluster combinations. Specifically, it switches between Depth-First (DFS) and Breadth-First (BFS) traversal of the tree of combinations, which ensures that the most promising candidates are processed even earlier in the process, which allows for more aggressive pruning. Additionally, the DFS-BFS routing strategy improved the parallelization of the algorithm. These changes significantly improved the performance of the top-k algorithm, as shown in Section F.2 of this appendix.

- **Novel optimization: Exploiting (soft) monotonicity**. This optimization is applied to the top-k algorithm, allowing it to 'pre-investigate' candidates with a higher chance of being in the top-k result set.

- **New streaming model for CDStream**. Compared to the streaming model in the original CDStream algorithm, the new BW$^+$ model (a) no longer required the streams to be time-aligned and synchronous, and (b) enabled the algorithm to identify uncommon events in the streams sooner. This redesign also involved adding a new knob (the epoch), which allowed the user to instruct the algorithm to be more responsive to large updates. The model is discussed in detail in Section 5.4.

- **More extensive evaluation**. We ran experiments with more datasets (added Crypto and Deep), more queries (progressive queries for all measures), and more baselines, including a comparison with DBMSs. Furthermore, the evaluation of CDStream now included experiments on (1) the development of the index size over time, (2) the sensitivity to the epoch size and number of epochs per basic window, and (3) the sensitivity of CDStream and CDHybrid to different datasets.

- **Total re-implementation of CD**. This included optimizations in the bound computation, the use of a more efficient index structures, and improved parallelization strategies.

F

F.2 Quantitative Comparison between old and new CD algorithms

To evaluate the effect of the changes made to the CD algorithm, we conduct a quantitative comparison between the original CD algorithm presented in [170, 171]. In this comparison, we run the code of the original CD algorithm published in [171] on a subset of the experiments presented in Section 5.5.2, and compare the results with those using the new CD algorithm (i.e., the ones presented in Section 5.5.2). Unfortunately, we cannot compare the original CDStream algorithm with the new one, as the original CDStream algorithm utilized the old streaming model, which assumed synchronous and time-aligned streams. The new streaming model, introduced in Section 5.4, does not have this limitation, making a direct (fair) comparison infeasible.

Comparison over different correlation patterns. To assess the impact of the algorithmic improvements, we performed a top-k query ($k = 100$) on the fMRI and Stocks datasets using a variety of similarity measures and correlation patterns, similar to the experiment underlying the results in Table 5.6 in Section 5.5.2. For each experiment, we imposed a timeout of 8 hours on the total runtime. The results in Table F.1 demonstrate that the new CD algorithm

Table F.1: Comparison of the original and new CD algorithm for top-k queries ($k = 100$) on the fMRI and Stocks datasets with different correlation patterns.

		MP (3)	*MP* (4)	*mPC* (1,2)	*mPC* (1,3)	*mPC* (1,4)	*mPC* (2,2)	*mPC* (2,3)
fMRI	**CD-OLD**	6.67	8.06	4.69	48.17	>28800	32.48	2101.57
	CD-NEW	1.64	2.01	1.82	12.31	3008.03	10.56	683.25
	Speedup	3.6x	4.0x	2.6x	3.9x	>9.6x	3.1x	3.1x
Stocks	**CD-OLD**	13.90	4729.32	5.34	128.72	>28800	121.49	>28800
	CD-NEW	2.50	61.22	1.35	4.50	564.63	3.98	1071.04
	Speedup	5.6x	77.2x	4.0x	28.6x	>50.9x	30.5x	>26.9x

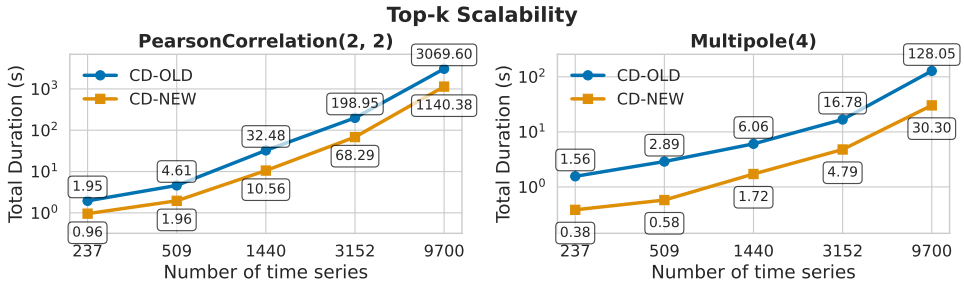


Figure F.1: Comparison of the original and new CD algorithm for top-k queries ($k = 100$) queries with increasing number of time series on the fMRI dataset. Note that both y-axes are in **log scale**.

is significantly faster than the original one, with speedups ranging from 2.6x to 77.2x across different correlation patterns and datasets. Notably, the speedup does not seem to follow a particular pattern; in some cases, it increases dramatically with the complexity of the correlation pattern (for example, the speedup for *MP* (3) to *MP* (4) changes from 5.6x to 77.2x), while in other cases it remains relatively stable. This improvement can be attributed to the new top-k algorithm and the soft-monotonicity optimization introduced in the new algorithm. These improvements allow the algorithm to increase the running threshold between consecutive traversals of two patterns (i.e., between finishing the search on pairs and starting the search on triples) and to raise the threshold earlier within a specific pattern search. Since the search space grows combinatorially with pattern complexity, even a modest increase in the threshold early in the search can result in substantially more pruning. Overall, these results indicate that the improvements made to the CD algorithm have a substantial impact on its performance, especially for more complex correlation patterns.

Comparison of algorithm scalability on top-k and threshold queries. For the scalability experiment, we ran both versions of the CD algorithm on different resolutions of the fMRI dataset, focusing on the correlation patterns *mPC* (2,2) and *MP* (4). The results in Figure F.1 and Figure F.2 show that the algorithms scale at similar rates with the number of time series, but the new version consistently outperforms the original: it is approximately three times faster on top-k queries and about twice as fast on threshold queries. This implies that the speedups reported in Table F.1 are stable across different dataset sizes.

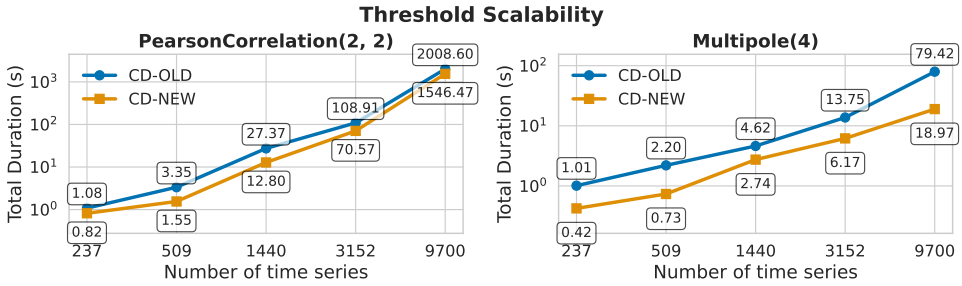


Figure F.2: Comparison of the original and new CD algorithm for threshold queries ($\tau = 0.95$) with increasing number of time series on the fMRI dataset. Note that both y-axes are in **log scale**.

Altogether we can conclude that the changes to the CD algorithm not only improved its scalability for increasingly complex correlation patterns, but also resulted in a constant-factor improvement in performance regardless of dataset size. The new CD algorithm thus represents a significant advancement and implies that the algorithm can be used effectively on larger datasets and more complex correlation patterns.

CHAPTER G

Framework Interface

The following code snippet provides a simplified version of the Measure interface of Chapter 6 in Python. Note that it can be implemented in any programming language, the choice of Python is arbitrary. The interface is designed to be extensible, allowing for the addition of new measures without modifying existing code.

Listing G.1: Python Implementation of the Measure Interface

```python
from abc import ABC, abstractmethod
from typing import List, Optional, Tuple, Dict, Callable
import numpy as np
from scipy.spatial.distance import euclidean, hamming
from clustering import Cluster

class Measure(ABC):
    """Abstract base class for multivariate similarity measures."""

    def __init__(self):
        # Properties of the measure
        self.is_similarity_function = True
        self.is_monotonously_increasing = False
        self.is_monotonously_decreasing = False
        self.is_metric = False
        self.is_on_vector_space = False
        self.is_permutation_invariant = True
        self.is_continuous = True # Works on continuous data?

        # Caches
        self.centroid_cache: Dict[Cluster, np.ndarray] = {}
        self.radius_cache: Dict[Cluster, float] = {}
        self.bounds_cache: Dict[Tuple[Cluster, Cluster], Tuple[float,
        float]] = {} # Cache for bounds between cluster pairs

    # --- Abstract methods ---

    @abstractmethod
    def compute(self, X: List[np.ndarray], Y: List[np.ndarray]) -> float:
        """Compute the similarity on a vector combination."""
        pass
```

```
    @abstractmethod
    def get_empirical_bounds(self, CX: List[Cluster], CY: List[Cluster])
    -> Tuple[float, float]:
        """Calculate empirical bounds for a cluster combination."""
        return None, None

    # --- Optional methods (with default behavior) ---

    def preprocess(self, data: np.ndarray) -> np.ndarray:
        """Preprocess the data if needed."""
        return data

    def aggregate(self, vectors: List[np.ndarray]) -> np.ndarray:
        """Aggregate the vectors into a single vector."""
        return np.mean(vectors, axis=0)  # Default: mean aggregation

    def cdist(self, x: np.ndarray, y: np.ndarray) -> float:
        """Calculate the clustering distance between two vectors."""
        if self.is_continuous:
            return euclidean(x, y)  # Euclidean distance
        else:
            return hamming(x, y)  # Hamming distance

    def dist2sim(self, sim: float) -> float:
        """Conversion from distance to similarity for general theoretical
         bounds."""
        return sim

    def sim2dist(self, dist: float) -> float:
        """Conversion from similarity to distance for general theoretical
         bounds."""
        return dist

    def check_additional_constraints(self, CX: List[Cluster], CY: List[
    Cluster]) -> bool:
        """Check if additional constraints are satisfied."""
        return True

    # --- Generic methods ---

    def bound(self, CX: List[Cluster], CY: List[Cluster]) -> Tuple[float,
     float]:
        """Generic adaptive bounding method."""
        lb, ub = get_empirical_bounds(CX, CY) # Try empirical bounds
        first
        if lb is not None and ub is not None:
            return lb, ub

        if self.is_metric and self.is_on_vector_space: # Then check
        theoretical bounds
            return self.get_multivariate_theoretical_bounds(CX, CY)

        # Fallback to exhaustive search
        lb, ub = float('inf'), float('-inf')
        for (X, Y) in split_to_singletons(CX, CY):
```

```
            sim = compute(X, Y)
            lb = min(lb, sim)
            ub = max(ub, sim)
        return lb, ub

    def get_pairwise_bounds(self, c1: Cluster, c2: Cluster, d: Callable[[
    np.ndarray, np.ndarray], float]) -> Tuple[float, float]:
        """Calculate empirical bounds between two clusters."""
        if (c1, c2) in self.bounds_cache:
            return self.bounds_cache[(c1, c2)]

        lb, ub = float('inf'), float('-inf')
        for x in c1.get_vectors():
            for y in c2.get_vectors():
                dist = d(x, y)
                lb = min(lb, dist)
                ub = max(ub, dist)
        self.bounds_cache[(c1, c2)] = (lb, ub)
        return lb, ub

    def get_theoretical_distance_bounds(self, CX: List[Cluster], CY: List
    [Cluster]) -> Tuple[float, float]:
        """Calculate theoretical bounds for a cluster combination."""
        cX = self.aggregate([c.centroid() for c in CX])
        cY = self.aggregate([c.centroid() for c in CY])
        rX = self.aggregate([c.radius() for c in CX])
        rY = self.aggregate([c.radius() for c in CY])
        lb = self.sim2dist(self.compute([cX], [cY])) - rX - rY
        ub = self.sim2dist(self.compute([cX], [cY])) + rX + rY
        return lb, ub

    def get_theoretical_bounds(self, CX: List[Cluster], CY: List[Cluster
    ]) -> Tuple[float, float]:
        d_lb, d_ub = self.get_theoretical_distance_bounds(CX, CY)
        lb, ub = self.dist2sim(d_ub), self.dist2sim(d_lb)
        return min(lb, ub), max(lb, ub)
```

CHAPTER H

Implementations of Example Measures

The following code snippets provide Python implementations of the three example measures discussed in Section 6.4.7. These implementations demonstrate how different types of multivariate similarity measures can be integrated into the framework using the defined interface.

H.1 Multivariate Euclidean Distance (*mED*)

The Euclidean distance implementation in Listing H.1 demonstrates a two-sided metric measure with empirical bounds derived from pairwise dot products. This example showcases how averaging aggregation and vector space operations are handled in the framework.

H.2 Multivariate Total Correlation (*TC*)

The Total Correlation implementation in Listing H.2 illustrates a one-sided information-theoretic measure that requires discrete data preprocessing. This example demonstrates how entropy calculations and monotonically increasing measures are integrated into the framework.

H.3 Multivariate Manhattan Distance (*mMD*)

The Manhattan Distance implementation in in Listing H.3 shows a metric measure that relies on theoretical bounds rather than empirical ones. This example highlights the use of dimensionality reduction through random projections and specialized distance estimators for L1 norms.

Listing H.1: Python Sketch for Multivariate Euclidean Distance

```
from measure import Measure
from typing import List
import numpy as np
from clustering import Cluster

class MultivariateEuclideanDistance(Measure):
    """Multivariate Euclidean Distance measure."""

    def __init__(self):
        # Call parent class initializer first
        super().__init__()

        # Override some properties
        self.is_similarity_function = False
        self.is_metric = True
        self.is_on_vector_space = True

    def compute(self, X: List[np.ndarray], Y: List[np.ndarray]) -> float:
        return np.linalg.norm(aggregate(X) - aggregate(Y))

    def get_empirical_bounds(self, CX: List[Cluster], CY: List[Cluster])
    -> Tuple[float, float]:
        lb = ub = 0
        for Cx in CX:
            for Cx_ in CX:
                _lb, _ub = get_pairwise_bounds(Cx, Cx_, np.dot)
                lb += _lb
                ub += _ub

        for Cy in CY:
            for Cy_ in CY:
                _lb, _ub = get_pairwise_bounds(Cy, Cy_, np.dot)
                lb += _lb
                ub += _ub

        for Cx in CX:
            for Cy in CY:
                _lb, _ub = get_pairwise_bounds(Cx, Cy, np.dot)
                lb -= 2*_ub
                ub -= 2*_lb

        return np.sqrt(min(0, lb)), np.sqrt(min(0, ub))
```

Listing H.2: Python Sketch for Multivariate Total Correlation

```
from measure import Measure
from typing import List, Tuple
import numpy as np
from clustering import Cluster

class TotalCorrelation(Measure):
    def __init__(self):
        super().__init__()
        self.is_monotonically_increasing = True

    def entropy(self, X: List[np.ndarray]) -> np.ndarray:
        """Compute joint entropy of a list of arrays."""
        values, counts = np.unique(X, return_counts=True)
        probabilities = counts / counts.sum()
        return -np.sum(probabilities * np.log(probabilities + 1e-10))

    def compute(self, X: List[np.ndarray], Y: List[np.ndarray]) -> float:
        """Compute the total correlation over X."""
        X = X + Y
        H_X = sum(self.entropy([x]) for x in X)
        H_XY = self.entropy(X)
        return H_X - H_XY

    def get_empirical_bounds(self, CX: List[Cluster], CY: List[Cluster])
    -> Tuple[float, float]:
        """Compute empirical bounds for total correlation."""
        CX = CX + CY
        lb = ub = 0
        max_joint_lb = 0
        for i in range(len(CX)):
            lb_m, ub_m += self.get_pairwise_bounds(CX[i], CX[i], self.
            entropy)
            lb += lb_m
            ub += ub_m
            for j in range(i+1, len(CX)):
                lb_j, _ = self.get_pairwise_bounds(CX[i], CX[j], self.
                entropy)
                lb += lb_j
                max_joint_lb = max(max_joint_lb, lb_j)
        return lb, ub - max_joint_lb

    def preprocess(self, data: np.ndarray) -> np.ndarray:
        """Quantize the data for continuous variables."""
        # Example quantization logic
        return np.digitize(data, bins=np.linspace(np.min(data), np.max(
        data), num=10))
```

Listing H.3: Python Sketch for Multivariate Manhattan Distance

```
from measure import Measure
from typing import List, Tuple
import numpy as np
from clustering import Cluster
from scipy.stats import cauchy

class ManhattanDistance(Measure):
    def __init__(self, num_projections: int = 100): # Example parameter
        super().__init__()
        self.is_similarity_function = False
        self.is_metric = True
        self.is_on_vector_space = True
        self.num_projections = num_projections

    def preprocess(self, data: List[np.ndarray]) -> List[np.ndarray]:
        """Apply Cauchy random projections to the data."""
        original_dim = data[0].shape[0]
        projection_matrix = cauchy.rvs(size=(original_dim, self.
        num_projections))
        return data @ projection_matrix

    def compute(self, X: List[np.ndarray], Y: List[np.ndarray]) -> float:
        """
        Estimate the L1 distance using the bias-corrected geometric mean
        estimator of Li et al.
        """
        mean_X = aggregate(X) # Assumes aggregate function sums
        mean_Y = aggregate(Y)
        corr_term = np.cos(np.pi / (2 * self.num_projections))
        geomean = np.prod(np.abs(mean_X - mean_Y) ** (1 / self.
        num_projections))
        return corr_term * geomean

    def cdist(self, x: np.ndarray, y: np.ndarray) -> float:
        """Use standard Manhattan distance for clustering."""
        return np.sum(np.abs(x - y))
```

Bibliography

[1] 2020 stock market crash - Wikipedia. https://en.wikipedia.org/wiki/2020_stock_market_crash. Accessed: 2024-10-17.

[2] Correlation Detective Repository. https://github.com/CorrelationDetective/public.

[3] Skoltech Computer Vision | Deep billion-scale indexing. https://sites.skoltech.ru/compvision/noimi/.

[4] MS-Index Repository. https://github.com/JdHondt/MS-Index, 2025.

[5] MTS Distances Evaluation Repository. https://github.com/TheDatumOrg/MTSDistEval, 2025.

[6] **Aburakhia, S., Tayeh, T., Myers, R., and Shami, A.** Similarity-Based Predictive Maintenance Framework for Rotating Machinery, 2022.

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

[8] **Agrawal, R., Faloutsos, C., and Swami, A.** Efficient similarity search in sequence databases. In *Foundations of Data Organization and Algorithms* (Berlin, Heidelberg, 1993), D. B. Lomet, Ed., Springer Berlin Heidelberg, pp. 69–84.

[9] **Agrawal, S., Atluri, G., Karpatne, A., Haltom, W., Liess, S., Chatterjee, S., and Kumar, V.** Tripoles: A New Class of Relationships in Time Series Data. In *Proc. SIGKDD'17*.

[10] **Agrawal, S., Steinbach, M., Boley, D., Chatterjee, S., Atluri, G., Dang, A. T., Liess, S., and Kumar, V.** Mining Novel Multivariate Relationships in Time Series Data Using Correlation Networks. *IEEE Transactions on Knowledge and Data Engineering 32*, 9 (2020), 1798–1811.

[11] **Alemi, A. A., Fischer, I., Dillon, J. V., and Murphy, K.** Deep Variational Information Bottleneck. In *ICLR'17*.

[12] **Amaldi, E., and Kann, V.** On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems. *Theory Comput. Sci. 209*, 1–2 (Dec. 1998), 237–260.

[13] **Andersen, M., Dahl, J., and Vandenberghe, L.** CVXOPT: Convex Optimization. Astrophysics Source Code Library, Aug. 2020.

[14] **Andritsos, P., Miller, R. J., and Tsaparas, P.** Information-theoretic tools for mining database structure from large data sets. In *Proc. SIGMOD '04* (2004), p. 731–742.

[15] **Arfken, G.** *Mathematical Methods for Physicists*, third ed. Academic Press, Inc., 1985.

[16] **Arthur, D., and Vassilvitskii, S.** K-Means++: the advantages of careful seeding. In *Proc. SODA'07* (2007).

[17] **Aßfalg, J., Kriegel, H.-P., Kröger, P., Kunath, P., Pryakhin, A., and Renz, M.** Similarity Search on Time Series Based on Threshold Queries. In *Proc. EDBT'06* (2006), pp. 276–294.

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

[19] **Azizi, I., Echihabi, K., and Palpanas, T.** ELPIS: Graph-Based Similarity Search for Scalable Data Science. In *Proc. VLDB'23* (2023), p. 1548–1559.

[20] **Azizi, I., Echihabi, K., and Palpanas, T.** Graph-Based Vector Search: An Experimental Evaluation of the State-of-the-Art. *Proc. ACM Manag. Data 3*, 1 (2025), 43:1–43:31.

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

[22] **Baeza-Yates, R., Cunto, W., Manber, U., and Wu, S.** Proximity matching using fixed-queries trees. In *Annual Symposium on Combinatorial Pattern Matching* (1994), Springer, pp. 198–212.

[23] **Bagnall, A., Lines, J., Bostrom, A., Large, J., and Keogh, E.** The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery 31*, 3 (2017), 606–660.

[24] **Bagnall, A. J., Dau, H. A., Lines, J., Flynn, M., Large, J., Bostrom, A., Southam, P., and Keogh, E. J.** The UEA multivariate time series classification archive, 2018.

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

[26] **Basri, R., Hassner, T., and Zelnik-Manor, L.** Approximate nearest subspace search. *IEEE transactions on pattern analysis and machine intelligence 33*, 2 (2010), 266–278.

[27] **Baum, L. E., and Petrie, T.** Statistical inference for probabilistic functions of finite state Markov chains. *The annals of mathematical statistics 37*, 6 (1966), 1554–1563.

[28] **Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B.** The R*-tree: an efficient and robust access method for points and rectangles. In *Proc. SIGMOD'90* (1990), p. 322–331.

[29] **Begum, N., and Keogh, E.** Rare time series motif discovery from unbounded streams. In *Proc. VLDB'14* (2014), p. 149–160.

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

[31] **Berndt, D. J., and Clifford, J.** Using dynamic time warping to find patterns in time series. In *Proc. AAAIWS'94* (1994), p. 359–370.

[32] **Bhaduri, K., Zhu, Q., Oza, N. C., and Srivastava, A. N.** Fast and Flexible Multivariate Time Series Subsequence Search. In *Proc. ICDM'10* (2010), pp. 48–57.

[33] **Bonifati, A., Buono, F. D., Guerra, F., Lombardi, M., and Tiano, D.** Interpretable Clustering of Multivariate Time Series with Time2Feat. *Proc. VLDB'23* (2023), 3994–3997.

[34] **Bradski, G.** The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[35] **Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J.** LOF: identifying density-based local outliers. *Proc. SIGMOD'00* (2000), 93–104.

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

[37] **Cai, Y., and Ng, R.** Indexing spatio-temporal trajectories with Chebyshev polynomials. In *Proc. SIGMOD '04* (2004), p. 599–610.

[38] **Camerra, A., Palpanas, T., Shieh, J., and Keogh, E.** iSAX 2.0: Indexing and Mining One Billion Time Series. In *Proc. ICDM'10* (2010), pp. 58–67.

[39] **Camerra, A., Palpanas, T., Shieh, J., and Keogh, E.** iSAX 2.0: Indexing and Mining One Billion Time Series. In *Proc. ICDM'10* (2010), pp. 58–67.

[40] **Camerra, A., Shieh, J., Palpanas, T., Rakthanmanon, T., and Keogh, E.** Beyond one billion time series: indexing and mining very large time series collections with iSAX2+. *Knowledge and Information Systems 39*, 1 (2014), 123–151.

[41] **Candes, E. J., and Tao, T.** Decoding by linear programming. *IEEE Transactions on Information Theory 51*, 12 (Dec. 2005), 4203–4215.

[42] **Candès, E. J.** The restricted isometry property and its implications for compressed sensing. *Comptes Rendus Mathematique 346*, 9, 589–592.

[43] **Carlborg, Ö., and Haley, C. S.** Epistasis: too often neglected in complex trait studies? *Nature Reviews Genetics 5*, 8 (2004), 618–625.

[44] **Ceccarello, M., Levchenko, A., Ioana, I., and Palpanas, T.** Evaluating and Generating Query Workloads for High Dimensional Vector Similarity Search. In *Proc. KDD'25* (2025).

[45] **Chakrabarti, K., Keogh, E., Mehrotra, S., and Pazzani, M.** Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Transactions on Database Systems 27*, 2 (2002), 188–228.

[46] **Chatzigeorgakidis, G., Skoutas, D., Patroumpas, K., Palpanas, T., Athanasiou, S., and Skiadopoulos, S.** Efficient Range and kNN Twin Subsequence Search in Time Series. *IEEE Transactions on Knowledge and Data Engineering 35*, 6 (2023), 5794–5807.

[47] **Chávez, E., Navarro, G., Baeza-Yates, R., and Marroquín, J. L.** Searching in metric spaces. *ACM computing surveys (CSUR) 33*, 3 (2001), 273–321.

[48] **Chen, H. J., Chen, S. J., Chen, Z., and Li, F.** Empirical Investigation of an Equity Pairs Trading Strategy. *Manage. Sci. 65*, 1 (Jan. 2019), 370–389.

[49] **Chen, L., and Ng, R.** On The Marriage of Lp-norms and Edit Distance. In *Proc. VLDB'04* (2004), pp. 792–803.

[50] **Chen, L., Özsu, M. T., and Oria, V.** Robust and fast similarity search for moving object trajectories. In *Proc. SIGMOD '05* (2005).

[51] **Chen, Q., Chen, L., Lian, X., Liu, Y., and Yu, J. X.** Indexable PLA for efficient similarity search. In *Proc. VLDB '07* (2007), p. 435–446.

[52] **Chen, S. S., Donoho, D. L., and Saunders, M. A.** Atomic Decomposition by Basis Pursuit. *SIAM Rev. 43*, 1 (Jan. 2001), 129–159.

[53] **Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P.** InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In *NIPS'16*.

[54] **Chen, X., and Güttel, S.** An Efficient Aggregation Method for the Symbolic Representation of Temporal Data. *ACM Transactions on Knowledge Discovery from Data 17*, 1 (2023), 1–22.

[55] **Chen, Y., Nascimento, M. A., Ooi, B. C., and Tung, A. K. H.** SpADe: On Shape-based Pattern Detection in Streaming Time Series. In *Proc. ICDE'07* (2007).

[56] **Cheng, P., Min, M. R., Shen, D., Malon, C., Zhang, Y., Li, Y., and Carin, L.** Improving Disentangled Text Representation Learning with Information-Theoretic Guidance. In *Proc. ACL'20*.

[57] **Chiang, R. H., Huang Cecil, C. E., and Lim, E.-P.** Linear correlation discovery in databases: a data mining approach. *Data & Knowledge Engineering 53*, 3 (2005), 311–337.

[58] **Chiu, B., Keogh, E., and Lonardi, S.** Probabilistic discovery of time series motifs. In *Proc. KDD '03* (2003), p. 493–498.

[59] **Christ, M., Braun, N., Neuffer, J., and Kempa-Liehr, A. W.** Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package). *Neurocomputing 307* (2018), 72–77.

[60] **Chu, K. K. W., and Wong, M. H.** Fast time-series searching with scaling and shifting. In *Proc. PODS '99* (1999), p. 237–248.

[61] **Ciaccia, P., Patella, M., and Zezula, P.** M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proc. VLDB '97* (1997), p. 426–435.

[62] **Cliff, O. M., Novelli, L., Fulcher, B. D., Shine, J. M., and Lizier, J. T.** Assessing the significance of directed and multivariate measures of linear dependence between time series. *Phys. Rev. Res. 3* (2021), 013145.

[63] **Cole, R., Shasha, D., and Zhao, X.** Fast window correlations over uncooperative time series. In *Proc. KDD '05* (2005), p. 743–749.

[64] **Cortes, C., and Vapnik, V.** Support-vector networks. *Machine Learning 20*, 3 (1995), 273–297.

[65] **Cristianini, N., and Shawe-Taylor, J.** *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[66] **Cuturi, M.** Fast global alignment kernels. In *Proc. ICML'11* (2011), pp. 929–936.

[67] **Dache, A., Vandaele, A., Gillis, N., and Nadisic, N.** Exact and Heuristic Methods for Simultaneous Sparse Coding. In *Proc. EUSIPCO'23* (2023), pp. 1753–1757.

[68] **Dallachiesa, M., Palpanas, T., and Ilyas, I. F.** Top-k nearest neighbor search in uncertain data series. In *Proc. VLDB'14* (2014), p. 13–24.

[69] **Das, A., and Kempe, D.** Algorithms for subset selection in linear regression. In *Proc. STOC '08* (2008), p. 45–54.

[70] **Davis, G., Mallat, S., and Avellaneda, M.** Adaptive greedy approximations. *Constructive Approximation 13*, 1 (1997), 57–98.

[71] **Demšar, J.** Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research 7* (2006), 1–30.

[72] **d'Hondt, J. E.** Multivariate Correlation Discovery in Streaming Data, 2021.

[73] **d'Hondt, J. E., Li, H., Yang, F., Papapetrou, O., and Paparrizos, J.** A Structured Study of Multivariate Time-Series Distance Measures. In *Proc. SIGMOD'25* (2025).

[74] **Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., and Keogh, E.** Querying and mining of time series data: experimental comparison of representations and distance measures. 1542–1552.

[75] **Ding, R., Wang, Q., Dang, Y., Fu, Q., Zhang, H., and Zhang, D.** YADING: fast clustering of large-scale time series data. 473–484.

[76] **Donoho, D.** Compressed sensing. *IEEE Transactions on Information Theory 52*, 4 (2006), 1289–1306.

[77] **Dürsch, F., Stebner, A., Windheuser, F., Fischer, M., Friedrich, T., Strelow, N., Bleifuß, T., Harmouch, H., Jiang, L., Papenbrock, T., and Naumann, F.** Inclusion Dependency Discovery: An Experimental Evaluation of Thirteen Algorithms. In *Proc. CIKM '19* (2019), p. 219–228.

[78] **d'Hondt, J. E., Kortekaas, T., Papapetrou, O., and Palpanas, T.** MS-Index: Fast Top-k Subsequence Search for Multivariate Time Series under Euclidean Distance. In *Proc. VLDB'26* (2025).

[79] **d'Hondt, J. E., Minartz, K., and Papapetrou, O.** Efficient detection of multivariate correlations with different correlation measures. *The VLDB Journal 33*, 2 (2024), 481–505.

[80] **d'Hondt, J. E., Papapetrou, O., and Paparrizos, J.** Beyond the Dimensions: A Structured Evaluation of Multivariate Time Series Distance Measures. In *Proc. ICDEW'24* (2024), pp. 107–112.

[81] **Echihabi, K., Zoumpatianos, K., Palpanas, T., and Benbrahim, H.** The Lernaean Hydra of Data Series Similarity Search: An Experimental Evaluation of the State of the Art. In *Proc. VLDB'18*.

[82] **Echihabi, K., Zoumpatianos, K., Palpanas, T., and Benbrahim, H.** Return of the Lernaean Hydra: experimental evaluation of data series approximate similarity search. In *Proc. VLDB'19* (2019), p. 403–420.

[83] **Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R.** Least angle regression. *The Annals of Statistics 32*, 2 (2004), 407 – 499.

[84] **Elsworth, S., and Güttel, S.** ABBA: adaptive Brownian bridge-based symbolic aggregation of time series. *Data Mining and Knowledge Discovery 34*, 4 (2020), 1175–1200.

[85] **Emadi, M., Miandji, E., and Unger, J.** A Performance Guarantee for Orthogonal Matching Pursuit Using Mutual Coherence. *Circuits, Systems, and Signal Processing 37*, 4 (2018), 1562–1574.

[86] **Esling, P., and Agon, C.** Time-series data mining. *ACM Comput. Surv. 45*, 1 (2012).

[87] **Fagin, R., Lotem, A., and Naor, M.** Optimal aggregation algorithms for middleware. In *Proc. PODS '01* (2001), p. 102–113.

[88] **Faloutsos, C., Ranganathan, M., and Manolopoulos, Y.** Fast subsequence matching in time-series databases. In *Proc. SIGMOD'94* (1994), p. 419–429.

[89] **Feng, K., Wang, P., Wu, J., and Wang, W.** L-Match: A Lightweight and Effective Subsequence Matching Approach. *IEEE Access 8* (2020), 71572–71583.

[90] **Ferhatosmanoglu, H., Tuncel, E., Agrawal, D., and El Abbadi, A.** Vector approximation based indexing for non-uniform high dimensional data sets. In *Proc. CIKM '00* (2000), p. 202–209.

[91] **Fortunato, S.** Community detection in graphs. *Physics Reports 486*, 3–5 (Feb. 2010), 75–174.

[92] **Franceschi, J.-Y., Dieuleveut, A., and Jaggi, M.** Unsupervised scalable representation learning for multivariate time series. *Advances in neural information processing systems 32* (2019).

[93] **Freksen, C. B.** An Introduction to Johnson-Lindenstrauss Transforms, 2021.

[94] **Frentzos, E., Gratsias, K., and Theodoridis, Y.** Index-based Most Similar Trajectory Search. In *Proc. ICDE'07* (2007).

[95] **Friedman, M.** The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance. *Journal of the American Statistical Association 32*, 200 (1937), 675–701.

[96] **Fu, A. W.-c., Keogh, E., Lau, L. Y. H., and Ratanamahatana, C. A.** Scaling and time warping in time series querying. In *Proc. VLDB '05* (2005), p. 649–660.

[97] **Gedik, B., Bordawekar, R. R., and Yu, P. S.** CellJoin: a parallel stream join operator for the cell processor. *The VLDB journal 18* (2009), 501–519.

[98] **Gentleman, W. M., and Sande, G.** Fast Fourier Transforms: for fun and profit. In *Proc. AFIPS '66 (Fall)* (1966), p. 563–578.

[99] **Georghiades, A., Belhumeur, P., and Kriegman, D.** From Few to Many: Illumination Cone Models for Face Recognition under Variable Lighting and Pose. *IEEE Transactions on Pattern Analysis and Machine Intelligence 23*, 6 (2001), 643–660.

[100] **Ghassempour, S., Girosi, F., and Maeder, A.** Clustering multivariate time series using Hidden Markov Models. *Int J Environ Res Public Health 11*, 3 (Mar. 2014), 2741–2763.

[101] **Ghojogh, B., Ghodsi, A., Karray, F., and Crowley, M.** Johnson-Lindenstrauss Lemma, Linear and Nonlinear Random Projections, Random Fourier Features, and Random Kitchen Sinks: Tutorial and Survey, 2021.

[102] **Goldin, D. Q., and Kanellakis, P. C.** On similarity queries for time-series data: Constraint specification and implementation. In *Principles and Practice of Constraint Programming — CP '95* (Berlin, Heidelberg, 1995), U. Montanari and F. Rossi, Eds., Springer Berlin Heidelberg, pp. 137–153.

[103] **Golub, G. H., and Van Loan, C. F.** An analysis of the total least squares problem. *SIAM journal on numerical analysis 17*, 6 (1980), 883–893.

[104] **Golub, G. H., and Van Loan, C. F.** *Matrix computations*. JHU press, 2013.

[105] **Grosse, R., Raina, R., Kwong, H., and Ng, A. Y.** Shift-Invariance Sparse Coding for Audio Classification, 2012.

[106] **Guha, T., and Ward, R. K.** Image Similarity Using Sparse Representation and Compression Distance. *Transactions on Multi. 16*, 4 (June 2014), 980–987.

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

[108] **Hadwiger, H.** Minkowskische Addition und Subtraktion beliebiger Punktmengen und die Theoreme von Erhard Schmidt. *Mathematische Zeitschrift 53*, 3 (1950), 210–218.

[109] **Han, T. S.** Nonnegative entropy measures of multivariate symmetric correlations. *Information and Control 36*, 2 (1978), 133–156.

[110] **Harris, F.** Polyphase Interpolators with Reversed Order of Up-Sampling and Down-Sampling. In *Proceedings of the 55th Asilomar Conference on Signals, Systems, and Computers* (2021), pp. 918–924.

[111] **Heikinheimo, H., Hinkkanen, E., Mannila, H., Mielikäinen, T., and Seppänen, J. K.** Finding low-entropy sets and trees from binary data. In *Proc. KDD '07* (2007), p. 350–359.

[112] **Heise, A., Quiané-Ruiz, J.-A., Abedjan, Z., Jentzsch, A., and Naumann, F.** Scalable discovery of unique column combinations. In *Proc. VLDB'12* (2013), p. 301–312.

[113] **Henaff, M., Jarrett, K., Kavukcuoglu, K., and Lecun, Y.** Unsupervised learning of sparse features for scalable audio classification. In *Proc. ISMIR'11* (2011), pp. 681–686.

[114] **Heunis, S., Lamerichs, R., Zinger, S., Caballero-Gaudes, C., Jansen, J. F., Aldenkamp, B., and Breeuwer, M.** Quality and denoising in real-time functional magnetic resonance imaging neurofeedback: A methods review. *Human Brain Mapping 41*, 12 (2020), 3439–3467.

[115] **Hills, J., Lines, J., Baranauskas, E., Mapp, J., and Bagnall, A.** Classification of time series by shapelet transformation. *Data Min. Knowl. Discov. 28*, 4 (2014), 851–881.

[116] **Hjaltason, G. R., and Samet, H.** Distance browsing in spatial databases. *ACM Transactions on Database Systems 24*, 2 (June 1999), 265–318.

[117] **Hu, H., Helminiak, D., Yang, M., Unsihuay, D., Hilger, R. T., Ye, D. H., and Laskin, J.** High-Throughput Mass Spectrometry Imaging with Dynamic Sparse Sampling. *ACS Measurement Science Au 2*, 5 (2022), 466–474.

[118] **Huijse, P., Estevez, P. A., Protopapas, P., Principe, J. C., and Zegers, P.** Computational Intelligence Challenges and Applications on Large-Scale Astronomical Time Series Databases. *IEEE Computational Intelligence Magazine 9*, 3 (2014), 27–39.

[119] **Härdle, W. K.** *Applied Multivariate Statistical Analysis*, 2 ed. Springer, 2007.

[120] **Jakulin, A., and Bratko, I.** Testing the significance of attribute interactions. In *Proc. ICML '04* (2004), p. 52.

[121] **Jiang, L., Kawashima, H., and Tatebe, O.** Incremental window aggregates over array database. In *Proc. IEEE BigData 2014*.

[122] **Johnson, W. B., Lindenstrauss, J., et al.** Extensions of Lipschitz mappings into a Hilbert space. *Contemporary mathematics 26*, 189-206 (1984), 1.

[123] **Jörges, C., d'Hondt, J. E., and Chatzigeorgakidis, G.** Leaf Area Index Time Series Imputation for Early Yield Prediction. In *Proc.BIDS'23* (2023), pp. 373–376.

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

[125] **Karaca, M., Alvarado, M. M., Gahrooei, M. R., Bihorac, A., and Pardalos, P. M.** Frequent pattern mining from multivariate time series data. *Expert Systems Appl. 194* (2022), 116435.

[126] **Kashino, K., Smith, G., and Murase, H.** Time-series active search for quick retrieval of audio and video. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing* (1999), pp. 2993–2996 vol.6.

[127] **Kashyap, S., and Karras, P.** Scalable kNN search on vertically stored time series. In *Proc. KDD '11* (2011), p. 1334–1342.

[128] **Kaufman, L., and Rousseeuw, P. J.** *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009.

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

[130] **Keller, F., Muller, E., and Bohm, K.** HiCS: High Contrast Subspaces for Density-Based Outlier Ranking. In *Proc. ICDE '12* (2012), p. 1037–1048.

[131] **Keogh, E.** A decade of progress in indexing and mining large time series databases. In *Proc. VLDB '06* (2006), p. 1268.

[132] **Keogh, E., Chakrabarti, K., Pazzani, M., and Mehrotra, S.** Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. *Knowledge and Information Systems 3*, 3 (2001), 263–286.

[133] **Keogh, E., and Kasetty, S.** On the need for time series data mining benchmarks: a survey and empirical demonstration. In *Proc. SIGMOD'02* (2002), pp. 102–111.

[134] **Keogh, E., Palpanas, T., Zordan, V. B., Gunopulos, D., and Cardle, M.** Indexing large human-motion databases. In *Proc. VLDB '04* (2004), p. 780–791.

[135] **Kimura, H., Huo, G., Rasin, A., Madden, S., and Zdonik, S. B.** CORADD: correlation aware database designer for materialized views and indexes. In *Proc. VLDB'10* (2010), p. 1103–1113.

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

[137] **Knobbe, A. J., and Ho, E. K. Y.** Maximally informative k-itemsets and their efficient discovery. In *Proc. KDD '06* (2006), p. 237–244.

[138] **Kolotilina, L.** A generalization of Weyl's inequalities with implications. *Journal of Mathematical Sciences 101* (2000), 3255–3260.

[139] **Kondylakis, H., Dayan, N., Zoumpatianos, K., and Palpanas, T.** Coconut: a scalable bottom-up approach for building data series indexes. In *Proc. VLDB'12* (2018), p. 677–690.

[140] **Korn, F., Jagadish, H. V., and Faloutsos, C.** Efficiently supporting ad hoc queries in large datasets of time sequences. In *Proc. SIGMOD '97* (1997), p. 289–300.

[141] **Lebedev, S.** hmmlearn. https://github.com/hmmlearn/hmmlearn, 2010.

[142] **Lee, J.-G., Han, J., Li, X., and Cheng, H.** Mining Discriminative Patterns for Classifying Trajectories on Road Networks. *IEEE Transactions on Knowledge and Data Engineering 23*, 5 (2011), 713–726.

[143] **Leutenegger, S., Lopez, M., and Edgington, J.** STR: a simple and efficient algorithm for R-tree packing. In *Proc. ICDE'97* (1997), pp. 497–506.

[144] **Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D.** Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proc. NIPS '20* (2020).

[145] **Li, J.** Robust Sparse Recovery via Matching Pursuit Algorithms and applications to simultaneous-source seismic data processing, 2021.

[146] **Li, P., Hastie, T., and Church, K. W.** Nonlinear Estimators and Tail Bounds for Dimension Reduction in l1 Using Cauchy Random Projections. *Journal of Machine Learning Research 8*, Oct (2007), 2497–2532.

[147] **Li, T., Dong, F.-Y., and Hirota, K.** Distance Measure for Symbolic Approximation Representation with Subsequence Direction for Time Series Data Mining. *Journal of Advanced Computational Intelligence and Intelligent Informatics 17*, 2 (2013), 263–271.

[148] **Lian, X., Chen, L., Yu, J. X., Wang, G., and Yu, G.** Similarity Match Over High Speed Time-Series Streams. In *Proc. ICDE'07* (2007), pp. 1086–1095.

[149] **Licher, S., Ahmad, S., Karamujić-Čomić, H., Voortman, T., Leening, M. J. G., Ikram, M. A., and Ikram, M. K.** Genetic predisposition, modifiable-risk-factor profile and long-term dementia risk in the general population. *Nature Medicine 25*, 9 (2019), 1364–1369.

[150] **Liess, S., Agrawal, S., Chatterjee, S., and Kumar, V.** A teleconnection between the West Siberian Plain and the ENSO region. *Journal of Climate 30*, 1 (2017), 301–315.

[151] **Lin, J., Keogh, E., Lonardi, S., and Chiu, B.** A symbolic representation of time series, with implications for streaming algorithms. In *Proc. DMKD '03* (2003), p. 2–11.

[152] **Linardi, M., and Palpanas, T.** Scalable, variable-length similarity search in data series: the ULISSE approach. 2236–2248.

[153] **Lines, J., and Bagnall, A.** Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery 29*, 3 (2015), 565–592.

[154] **Liu, B.-D., Wang, Y.-X., Shen, B., Zhang, Y.-J., and Hebert, M.** Self-explanatory Sparse Representation for Image Classification. In *Computer Vision – ECCV 2014* (Cham, 2014), D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Springer International Publishing, pp. 600–616.

[155] **Liu, Q., and Paparrizos, J.** The Elephant in the Room: Towards A Reliable Time-Series Anomaly Detection Benchmark. In *NeurIPS 2024* (2024).

[156] **Lkhagva, B., Yu, S., and Kawagoe, K.** New Time Series Data Representation ESAX for Financial Applications. In *Proc. ICDEW'06* (2006).

[157] **Löning, M., Bagnall, A. J., Ganesh, S., Kazakov, V., Lines, J., and Király, F. J.** sktime: A Unified Interface for Machine Learning with Time Series.

[158] **Lu, K., Ishikawa, Y., and Xiao, C.** MQH: Locality Sensitive Hashing on Multi-level Quantization Errors for Point-to-Hyperplane Distances. In *Proc. VLDB'12* (2022), p. 864–876.

[159] **Lubba, C. H., Sethi, S. S., Knaute, P., Schultz, S. R., Fulcher, B. D., and Jones, N. S.** catch22: CAnonical Time-series CHaracteristics: Selected through highly comparative time-series analysis. *Data Mining and Knowledge Discovery 33*, 6 (2019), 1821–1852.

[160] **Luo, D., Ding, C., and Huang, H.** Linear discriminant analysis: new formulations and overfit analysis. In *Proc. AAAI'11* (2011), p. 417–422.

[161] **Lustig, M., Donoho, D., and Pauly, J. M.** Sparse MRI: The application of compressed sensing for rapid MR imaging. *Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine 58*, 6 (2007), 1182–1195.

[162] **Lyon, R. F., Rehn, M., Bengio, S., Walters, T. C., and Chechik, G.** Sound retrieval and ranking using sparse auditory representations. *Neural Comput. 22*, 9 (Sept. 2010), 2390–2416.

[163] **Mair, P., Groenen, P. J. F., and de Leeuw, J.** More on Multidimensional Scaling and Unfolding in R: smacof Version 2. *Journal of Statistical Software 102*, 10 (2022), 1–47.

[164] **Malinowski, S., Guyet, T., Quiniou, R., and Tavenard, R.** *1d-SAX: A Novel Symbolic Representation for Time Series*. 2013, pp. 273–284.

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

[166] **Marteau, P., and Gibet, S.** Constructing Positive Elastic Kernels with Application to Time Series Classification.

[167] **Marteau, P.-F.** Time Warp Edit Distance with Stiffness Adjustment for Time Series Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence 31*, 2 (2009), 306–318.

[168] **Megumi, F., Yamashita, A., Kawato, M., and Imamizu, H.** Functional MRI neurofeedback training on connectivity between two regions induces long-lasting changes in intrinsic functional network. *Frontiers in Human Neuroscience 9* (2015).

[169] **Meng, Q., Qian, H., Liu, Y., Cui, L., Xu, Y., and Shen, Z.** MHCCL: Masked Hierarchical Cluster-Wise Contrastive Learning for Multivariate Time Series. In *AAAI* (2023), AAAI Press, pp. 9153–9161.

[170] **Minartz, K.** Correlation Detective: Efficient Multivariate Correlation Discovery, 2021.

[171] **Minartz, K., d'Hondt, J. E., and Papapetrou, O.** Multivariate correlations discovery in static and streaming data. In *Proc. VLDB'22* (2022), p. 1266–1278.

[172] **Mitra, I., Lavillaureix, A., Yeh, E., Traglia, M., Tsang, K., Bearden, C. E., Rauen, K. A., and Weiss, L. A.** Reverse pathway genetic approach identifies epistasis in autism spectrum disorders. *PLoS genetics 13*, 1 (2017), e1006516.

[173] **Moon, Y.-S., Whang, K.-Y., and Han, W.-S.** General match: a subsequence matching method in time-series databases based on generalized windows. In *Proc. SIGMOD '02* (2002), p. 382–393.

[174] **Mueen, A.** Enumeration of Time Series Motifs of All Lengths. In *Proc. ICDM'13*.

[175] **Mueen, A., Keogh, E., and Young, N.** Logical-shapelets: an expressive primitive for time series classification. In *Proc. KDD '11* (2011), p. 1154–1162.

[176] **Mueen, A., Keogh, E., Zhu, Q., Cash, S., and Westover, B.** Exact discovery of time series motifs. In *Proc. SIAM'09* (2009), pp. 473–484.

[177] **Mueen, A., Nath, S., and Liu, J.** Fast Approximate Correlation for Massive Time-Series Data. In *Proc. SIGMOD'10* (2010).

[178] **Mueen, A., Zhing, S., Zhu, Y., Yeh, M., Kamgar, K., Viswanathan, K., Gupta, C., and Keogh, E.** The Fastest Similarity Search Algorithm for Time Series Subsequences under Euclidean Distance, 2022. http://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html.

[179] **Muhammad Ali, P. J.** Investigating the Impact of Min-Max Data Normalization on the Regression Performance of K-Nearest Neighbor with Different Similarity Measurements. *ARO-THE SCIENTIFIC JOURNAL OF KOYA UNIVERSITY 10*, 1 (2022), 85–91.

[180] **Muniz-Cuza, C., Jensen, S., Brusokas, J., Ho, N., and Pedersen, T.** Evaluating the Impact of Error-Bounded Lossy Compression on Time Series Forecasting. In *EDBT'24* (2024).

[181] **Münnix, M. C., Shimada, T., Schäfer, R., Leyvraz, F., Seligman, T. H., Guhr, T., and Stanley, H. E.** Identifying States of a Financial Market. *Scientific Reports 2*, 1 (2012), 644.

[182] **Nemenyi, P. B.** *Distribution-free Multiple Comparisons*. PhD thesis, Princeton University, 1963.

[183] **Nguyen, H. V., Müller, E., Andritsos, P., and Böhm, K.** Detecting Correlated Columns in Relational Databases with Mixed Data Types. In *Proc. SSDBM'14* (2014).

[184] **Nguyen, H. V., Müller, E., Vreeken, J., Efros, P., and Böhm, K.** Multivariate Maximal Correlation Analysis. In *Proc. ICML'14* (2014).

[185] **Oceanic, N., and Administration, A.** NOAA Integrated Surface Dataset. https://www.ncei.noaa.gov/access/search/dataset-search.

[186] **O'sullivan, A., and Sheffrin, S. M.** *Economics: Principles in action*. Pearson Prentice Hall, 2003.

[187] **Papapetrou, O., and d'Hondt, J. E.** Multivariate Similarity Search - A Call for a New Breed of Similarity Search Algorithms. In *Proc. ICDE'24* (2024), pp. 5662–5662.

[188] **Papapetrou, P., Athitsos, V., Potamias, M., Kollios, G., and Gunopulos, D.** Embedding-based subsequence matching in time-series databases. *ACM Transactions on Database Systems 36*, 3 (2011).

[189] **Paparrizos, J., Boniol, P., Palpanas, T., Tsay, R. S., Elmore, A., and Franklin, M. J.** Volume under the surface: a new accuracy evaluation measure for time-series anomaly detection. In *Proc. VLDB'22* (2022), p. 2774–2787.

[190] **Paparrizos, J., and Franklin, M. J.** GRAIL: efficient time-series representation learning. In *Proc. VLDB'12* (2019), p. 1762–1777.

[191] **Paparrizos, J., and Gravano, L.** k-Shape: Efficient and Accurate Clustering of Time Series. In *Proc. SIGMOD '15* (2015), p. 1855–1870.

[192] **Paparrizos, J., Li, H., Yang, F., Wu, K., d'Hondt, J. E., and Papapetrou, O.** *Data Series Management and Analytics: From Time Series to High-dimensional Vectors*. 2025, ch. A Survey on Time-Series Distance Measures. In press.

[193] **Paparrizos, J., Liu, C., Elmore, A. J., and Franklin, M. J.** Debunking Four Long-Standing Misconceptions of Time-Series Distance Measures. In *Proc. SIGMOD '20* (2020), pp. 1887–1905.

[194] **Paraskevopoulos, P., Dinh, T.-C., Dashdorj, Z., Palpanas, T., Serafini, L., et al.** Identification and characterization of human behavior patterns from mobile phone data. *D4D Challenge session, NetMob* (2013).

[195] **Pati, Y., Rezaiifar, R., and Krishnaprasad, P.** Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers* (1993), pp. 40–44 vol.1.

[196] **Pearson, K.** LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 2*, 11 (1901), 559–572.

[197] **Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E.** Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[198] **Pelok, B., and d'Hondt, J. E.** MULISSE: Variable-Length Similarity Search for Multivariate Time Series. In *Proc. ICDEW'25* (2025).

[199] **Perlin, M.** M of a kind: A Multivariate Approach at Pairs Trading. *Available at SSRN 952782* (2007).

[200] **Plumbley, M. D., Blumensath, T., Daudet, L., Gribonval, R., and Davies, M. E.** Sparse Representations in Audio and Music: From Coding to Source Separation. *Proceedings of the IEEE 98*, 6 (2010), 995–1005.

[201] **Procacci, P. F., and Aste, T.** Portfolio optimization with sparse multivariate modeling. *Journal of Asset Management 23*, 6 (2022), 445–465.

[202] **Rafiei, D., and Mendelzon, A.** Similarity-based queries for time series data. In *Proc. SIGMOD '97* (1997), p. 13–25.

[203] **Rafiei, D., and Mendelzon, A. O.** Efficient Retrieval of Similar Time Sequences Using DFT.

[204] **Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., and Keogh, E.** Searching and mining trillions of time series subsequences under dynamic time warping. In *Proc. KDD '12* (2012), p. 262–270.

[205] **Ramaswamy, S., Rastogi, R., and Shim, K.** Efficient algorithms for mining outliers from large data sets. In *Proc. SIGMOD '00* (2000), p. 427–438.

[206] **Rand, W. M.** Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association 66*, 336 (1971), 846–850.

[207] **Rao, M., Seth, S., Xu, J., Chen, Y., Tagare, H., and Príncipe, J. C.** A test of independence based on a generalized correlation function. *Signal Process. 91*, 1 (Jan. 2011), 15–27.

[208] **RE, K., Kalnay, E., Collins, W., Saha, S., White, G., Woollen, J., Chelliah, M., Ebisuzaki, W., Kanamitsu, M., Kousky, V., Dool, H., RL, J., and Fiorino, M.** The NCEP/NCAR 50-year reanalysis: monthly means CD-ROM and documentation. *Bulletin of the American Meteorological Society 82* (2001), 247–268.

[209] **Roddick, J. F., and Hornsby, K. S.** Temporal, Spatial, and Spatio-Temporal Data Mining. In *Lecture Notes in Computer Science* (2001).

[210] **Rostoker, C., Wagner, A., and Hoos, H.** A Parallel Workflow for Real-time Correlation and Clustering of High-Frequency Stock Market Data. In *Proc. IPDPS'07* (2007).

[211] **Ruiz, A. P., Flynn, M., Large, J., Middlehurst, M., and Bagnall, A.** The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery 35*, 2 (2021), 401–449.

[212] **Sakoe, H., and Chiba, S.** Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing 26*, 1 (1978), 43–49.

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

[214] **Samet, H.** *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., 2005.

[215] **Santoro, A., Battiston, F., Petri, G., and Amico, E.** Higher-order organization of multivariate time series. *Nature Physics 19*, 2 (2023), 221–229.

[216] **Schäfer, P., and Högqvist, M.** SFA: a symbolic fourier approximation and index for similarity search in high dimensional datasets. In *Proc. EDBT '12* (2012), p. 516–527.

[217] **Shieh, J., and Keogh, E.** iSAX: indexing and mining terabyte sized time series. In *Proc. KDD '08* (2008), p. 623–631.

[218] **Shifaz, A., Pelletier, C., Petitjean, F., and Webb, G. I.** Elastic similarity and distance measures for multivariate time series. *Knowledge and Information Systems 65*, 6 (2023), 2665–2698.

[219] **Shokoohi-Yekta, M., Hu, B., Jin, H., Wang, J., and Keogh, E. J.** Generalizing DTW to the multi-dimensional case requires an adaptive approach. *Data Min. Knowl. Discov. 31*, 1 (2017), 1–31.

[220] **Sismanis, Y., Brown, P., Haas, P. J., and Reinwald, B.** GORDIAN: efficient and scalable discovery of composite keys. In *Proc. VLDB '06* (2006), p. 691–702.

[221] **Stefan, A., Athitsos, V., and Das, G.** The Move-Split-Merge Metric for Time Series. *IEEE Transactions on Knowledge and Data Engineering 25*, 6 (2013), 1425–1438.

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

[223] **Tan, M., Tsang, I. W., and Wang, L.** Matching Pursuit LASSO Part II: Applications and Sparse Recovery Over Batch Signals. *Transactions on Signal Processing 63*, 3 (Feb. 2015), 742–753.

[224] **Tan, Z., Jamdagni, A., He, X., Nanda, P., and Liu, R. P.** A System for Denial-of-Service Attack Detection Based on Multivariate Correlation Analysis. *IEEE Transactions on Parallel Distributed Systems 25*, 2 (2014), 447–456.

[225] **Tibshirani, R.** Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological) 58*, 1 (1996), 267–288.

[226] **Tropp, J.** Greed is good: algorithmic results for sparse approximation. *IEEE Transactions on Information Theory 50*, 10 (2004), 2231–2242.

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

[228] **Villarreal, D. J., Poonawala, H. A., and Gregg, R. D.** A robust parameterization of human gait patterns across phase-shifting perturbations. *IEEE Transactions on Neural Systems and Rehabilitation Engineering 25*, 3 (2016), 265–278.

[229] **Virtanen et al, P.** SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods 17* (2020), 261–272.

[230] **Vlachos, M., Hadjieleftheriou, M., Gunopulos, D., and Keogh, E.** Indexing Multi-Dimensional Time-Series with Support for Multiple Distance Measures. In *Proc. KDD '03* (2003), p. 216–225.

[231] **Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., and Keogh, E.** Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery 26*, 2 (2013), 275–309.

[232] **Wang, Y., Wang, P., Pei, J., Wang, W., and Huang, S.** A data-adaptive and dynamic segmentation index for whole matching on time series. In *Proc. VLDB'12* (2013), p. 793–804.

[233] **Wang, Z., Wang, P., Palpanas, T., and Wang, W.** Graph-and Tree-based Indexes for High-dimensional Vector Similarity Search: Analyses, Comparisons, and Future Directions. *IEEE Data Eng. Bull. 46*, 3 (2023), 3–21.

[234] **Watanabe, S.** Information Theoretical Analysis of Multivariate Correlation. *IBM Journal of Research and Development 4*, 1 (1960), 66–82.

[235] **Wilcoxon, F.** Individual Comparisons by Ranking Methods. *Biometrics Bulletin 1*, 6 (1945), 80–83.

[236] **Wright, J., Ma, Y., Mairal, J., Sapiro, G., Huang, T. S., and Yan, S.** Sparse Representation for Computer Vision and Pattern Recognition. *Proceedings of the IEEE 98*, 6 (2010), 1031–1044.

[237] **Wright, J., Yang, A. Y., Ganesh, A., Sastry, S. S., and Ma, Y.** Robust Face Recognition via Sparse Representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 31*, 2 (2009), 210–227.

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

[239] **Xu, Y., Liu, X., Wang, B., Tao, R., Xia, K., and Cao, X.** Fast Nearest Subspace Search via Random Angular Hashing. *IEEE Transactions on Multimedia 23* (2021), 342–352.

[240] **Yan, L., Wu, X., and Xiao, J.** An Improved Time Series Symbolic Representation Based on Multiple Features and Vector Frequency Difference. *Journal of Computer and Communications 10*, 06 (2022), 44–62.

[241] **Yang, F., and Paparrizos, J.** SPARTAN: Data-Adaptive Symbolic Time-Series Approximation. In *Proc. SIGMOD'25* (2025).

[242] **Yang, J., Wright, J., Huang, T. S., and Ma, Y.** Image Super-Resolution Via Sparse Representation. *IEEE Transactions on Image Processing 19*, 11 (2010), 2861–2873.

[243] **Yang, K., and Shahabi, C.** A PCA-Based Similarity Measure for Multivariate Time Series. In *Proc. MMDB'04* (2004), p. 65–74.

[244] **Yeh, C.-C. M., Kavantzas, N., and Keogh, E.** Matrix Profile VI: Meaningful Multidimensional Motif Discovery. In *Proc. ICDM'17* (2017), pp. 565–574.

[245] **Yeh, C.-C. M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H. A., Silva, D. F., Mueen, A., and Keogh, E.** Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets. In *Procl ICDM'16* (2016), pp. 1317–1322.

[246] **Yeh, C.-C. M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H. A., Zimmerman, Z., Silva, D. F., Mueen, A., and Keogh, E.** Time series joins, motifs, discords and shapelets: a unifying view that exploits the matrix profile. *Data Min. Knowl. Discov. 32*, 1 (2018), 83–123.

[247] **Yu, Y., Zhu, Y., Wan, D., Liu, H., and Zhao, Q.** *A Novel Symbolic Aggregate Approximation for Time Series.* 2019, pp. 805–822.

[248] **Yue, Z., Wang, Y., Duan, J., Yang, T., Huang, C., Tong, Y., and Xu, B.** Ts2vec: Towards universal representation of time series. In *Proc. AAAI'22* (2022), pp. 8980–8987.

[249] **Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., and Stoica, I.** Discretized streams: fault-tolerant streaming computation at scale. In *Proc. SOSP'13* (2013).

[250] **Zhang, M., Gao, Y., Sun, C., and Blumenstein, M.** A Robust Matching Pursuit Algorithm Using Information Theoretic Learning, 2020.

[251] **Zhang, X., Pan, F., Wang, W., and Nobel, A.** Mining non-redundant high order correlations in binary data. In *Proc. VLDB'08* (2008), p. 1178–1188.

[252] **Zhang, Z., Xu, Y., Yang, J., Li, X., and Zhang, D.** A Survey of Sparse Representation: Algorithms and Applications. *IEEE Access 3* (2015), 490–530.

[253] **Zhao, Z., and Liu, H.** Searching for interacting features. In *Proc. IJCAI'07* (2007), Morgan Kaufmann Publishers Inc., p. 1156–1161.

[254] **Zheng, Y., Capra, L., Wolfson, O., and Yang, H.** Urban Computing: Concepts, Methodologies, and Applications. *ACM Transactions on Intell. Systems Technol. 5*, 3 (Sept. 2014).

[255] **Zhou, G., Lu, B., Hu, X., and Ni, T.** Sparse representation-based discriminative metric learning for brain MRI image retrieval. *Frontiers in Neuroscience 15* (2022), 829040.

[256] **Zhu, H., Kollios, G., and Athitsos, V.** A generic framework for efficient and effective subsequence retrieval. In *Proc. VLDB'12* (2012), p. 1579–1590.

[257] **Zhu, Y., and Shasha, D.** StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In *Proc. VLDB'02* (2002).

[258] **Zilverstand, A., Sorger, B., Zimmermann, J., Kaas, A., and Goebel, R.** Windowed Correlation: A Suitable Tool for Providing Dynamic fMRI-Based Functional Connectivity Neurofeedback on Task Difficulty. *PLOS ONE 9*, 1 (2014), 1–13.

[259] **Zoumpatianos, K., Lou, Y., Ileana, I., Palpanas, T., and Gehrke, J.** Generating data series query workloads. *The VLDB Journal* (2018).

SIKS dissertations

2016 01 Syed Saiden Abbas (RUN), Recognition of Shapes by Humans and Machines
02 Michiel Christiaan Meulendijk (UU), Optimizing medication reviews through decision support: prescribing a better pill to swallow
03 Maya Sappelli (RUN), Knowledge Work in Context: User Centered Knowledge Worker Support
04 Laurens Rietveld (VUA), Publishing and Consuming Linked Data
05 Evgeny Sherkhonov (UvA), Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers
06 Michel Wilson (TUD), Robust scheduling in an uncertain environment
07 Jeroen de Man (VUA), Measuring and modeling negative emotions for virtual training
08 Matje van de Camp (TiU), A Link to the Past: Constructing Historical Social Networks from Unstructured Data
09 Archana Nottamkandath (VUA), Trusting Crowdsourced Information on Cultural Artefacts
10 George Karafotias (VUA), Parameter Control for Evolutionary Algorithms
11 Anne Schuth (UvA), Search Engines that Learn from Their Users
12 Max Knobbout (UU), Logics for Modelling and Verifying Normative Multi-Agent Systems
13 Nana Baah Gyan (VUA), The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
14 Ravi Khadka (UU), Revisiting Legacy Software System Modernization
15 Steffen Michels (RUN), Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
16 Guangliang Li (UvA), Socially Intelligent Autonomous Agents that Learn from Human Reward
17 Berend Weel (VUA), Towards Embodied Evolution of Robot Organisms
18 Albert Meroño Peñuela (VUA), Refining Statistical Data on the Web
19 Julia Efremova (TU/e), Mining Social Structures from Genealogical Data
20 Daan Odijk (UvA), Context & Semantics in News & Web Search
21 Alejandro Moreno Célleri (UT), From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground
22 Grace Lewis (VUA), Software Architecture Strategies for Cyber-Foraging Systems
23 Fei Cai (UvA), Query Auto Completion in Information Retrieval
24 Brend Wanders (UT), Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach
25 Julia Kiseleva (TU/e), Using Contextual Information to Understand Searching and Browsing Behavior
26 Dilhan Thilakarathne (VUA), In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains

27 Wen Li (TUD), Understanding Geo-spatial Information on Social Media
28 Mingxin Zhang (TUD), Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
29 Nicolas Höning (TUD), Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
30 Ruud Mattheij (TiU), The Eyes Have It
31 Mohammad Khelghati (UT), Deep web content monitoring
32 Eelco Vriezekolk (UT), Assessing Telecommunication Service Availability Risks for Crisis Organisations
33 Peter Bloem (UvA), Single Sample Statistics, exercises in learning from just one example
34 Dennis Schunselaar (TU/e), Configurable Process Trees: Elicitation, Analysis, and Enactment
35 Zhaochun Ren (UvA), Monitoring Social Media: Summarization, Classification and Recommendation
36 Daphne Karreman (UT), Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
37 Giovanni Sileno (UvA), Aligning Law and Action - a conceptual and computational inquiry
38 Andrea Minuto (UT), Materials that Matter - Smart Materials meet Art & Interaction Design
39 Merijn Bruijnes (UT), Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
40 Christian Detweiler (TUD), Accounting for Values in Design
41 Thomas King (TUD), Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance
42 Spyros Martzoukos (UvA), Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
43 Saskia Koldijk (RUN), Context-Aware Support for Stress Self-Management: From Theory to Practice
44 Thibault Sellam (UvA), Automatic Assistants for Database Exploration
45 Bram van de Laar (UT), Experiencing Brain-Computer Interface Control
46 Jorge Gallego Perez (UT), Robots to Make you Happy
47 Christina Weber (UL), Real-time foresight - Preparedness for dynamic innovation networks
48 Tanja Buttler (TUD), Collecting Lessons Learned
49 Gleb Polevoy (TUD), Participation and Interaction in Projects. A Game-Theoretic Analysis
50 Yan Wang (TiU), The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

---

2017 01 Jan-Jaap Oerlemans (UL), Investigating Cybercrime
02 Sjoerd Timmer (UU), Designing and Understanding Forensic Bayesian Networks using Argumentation
03 Daniël Harold Telgen (UU), Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
04 Mrunal Gawade (CWI), Multi-core Parallelism in a Column-store
05 Mahdieh Shadi (UvA), Collaboration Behavior
06 Damir Vandic (EUR), Intelligent Information Systems for Web Product Search
07 Roel Bertens (UU), Insight in Information: from Abstract to Anomaly
08 Rob Konijn (VUA), Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
09 Dong Nguyen (UT), Text as Social and Cultural Data: A Computational Perspective on Variation in Text

10 Robby van Delden (UT), (Steering) Interactive Play Behavior
11 Florian Kunneman (RUN), Modelling patterns of time and emotion in Twitter #anticipointment
12 Sander Leemans (TU/e), Robust Process Mining with Guarantees
13 Gijs Huisman (UT), Social Touch Technology - Extending the reach of social touch through haptic technology
14 Shoshannah Tekofsky (TiU), You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
15 Peter Berck (RUN), Memory-Based Text Correction
16 Aleksandr Chuklin (UvA), Understanding and Modeling Users of Modern Search Engines
17 Daniel Dimov (UL), Crowdsourced Online Dispute Resolution
18 Ridho Reinanda (UvA), Entity Associations for Search
19 Jeroen Vuurens (UT), Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
20 Mohammadbashir Sedighi (TUD), Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
21 Jeroen Linssen (UT), Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
22 Sara Magliacane (VUA), Logics for causal inference under uncertainty
23 David Graus (UvA), Entities of Interest — Discovery in Digital Traces
24 Chang Wang (TUD), Use of Affordances for Efficient Robot Learning
25 Veruska Zamborlini (VUA), Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
26 Merel Jung (UT), Socially intelligent robots that understand and respond to human touch
27 Michiel Joosse (UT), Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
28 John Klein (VUA), Architecture Practices for Complex Contexts
29 Adel Alhuraibi (TiU), From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT"
30 Wilma Latuny (TiU), The Power of Facial Expressions
31 Ben Ruijl (UL), Advances in computational methods for QFT calculations
32 Thaer Samar (RUN), Access to and Retrievability of Content in Web Archives
33 Brigit van Loggem (OU), Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
34 Maren Scheffel (OU), The Evaluation Framework for Learning Analytics
35 Martine de Vos (VUA), Interpreting natural science spreadsheets
36 Yuanhao Guo (UL), Shape Analysis for Phenotype Characterisation from High-throughput Imaging
37 Alejandro Montes Garcia (TU/e), WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
38 Alex Kayal (TUD), Normative Social Applications
39 Sara Ahmadi (RUN), Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
40 Altaf Hussain Abro (VUA), Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems
41 Adnan Manzoor (VUA), Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle

42 Elena Sokolova (RUN), Causal discovery from mixed and missing data with applications on ADHD datasets
43 Maaike de Boer (RUN), Semantic Mapping in Video Retrieval
44 Garm Lucassen (UU), Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
45 Bas Testerink (UU), Decentralized Runtime Norm Enforcement
46 Jan Schneider (OU), Sensor-based Learning Support
47 Jie Yang (TUD), Crowd Knowledge Creation Acceleration
48 Angel Suarez (OU), Collaborative inquiry-based learning

2018 01 Han van der Aa (VUA), Comparing and Aligning Process Representations
02 Felix Mannhardt (TU/e), Multi-perspective Process Mining
03 Steven Bosems (UT), Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction
04 Jordan Janeiro (TUD), Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
05 Hugo Huurdeman (UvA), Supporting the Complex Dynamics of the Information Seeking Process
06 Dan Ionita (UT), Model-Driven Information Security Risk Assessment of Socio-Technical Systems
07 Jieting Luo (UU), A formal account of opportunism in multi-agent systems
08 Rick Smetsers (RUN), Advances in Model Learning for Software Systems
09 Xu Xie (TUD), Data Assimilation in Discrete Event Simulations
10 Julienka Mollee (VUA), Moving forward: supporting physical activity behavior change through intelligent technology
11 Mahdi Sargolzaei (UvA), Enabling Framework for Service-oriented Collaborative Networks
12 Xixi Lu (TU/e), Using behavioral context in process mining
13 Seyed Amin Tabatabaei (VUA), Computing a Sustainable Future
14 Bart Joosten (TiU), Detecting Social Signals with Spatiotemporal Gabor Filters
15 Naser Davarzani (UM), Biomarker discovery in heart failure
16 Jaebok Kim (UT), Automatic recognition of engagement and emotion in a group of children
17 Jianpeng Zhang (TU/e), On Graph Sample Clustering
18 Henriette Nakad (UL), De Notaris en Private Rechtspraak
19 Minh Duc Pham (VUA), Emergent relational schemas for RDF
20 Manxia Liu (RUN), Time and Bayesian Networks
21 Aad Slootmaker (OU), EMERGO: a generic platform for authoring and playing scenario-based serious games
22 Eric Fernandes de Mello Araújo (VUA), Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
23 Kim Schouten (EUR), Semantics-driven Aspect-Based Sentiment Analysis
24 Jered Vroon (UT), Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots
25 Riste Gligorov (VUA), Serious Games in Audio-Visual Collections
26 Roelof Anne Jelle de Vries (UT),Theory-Based and Tailor-Made: Motivational Messages for Behavior Change Technology
27 Maikel Leemans (TU/e), Hierarchical Process Mining for Scalable Software Analysis
28 Christian Willemse (UT), Social Touch Technologies: How they feel and how they make you feel
29 Yu Gu (TiU), Emotion Recognition from Mandarin Speech

	30	Wouter Beek (VUA), The "K" in "semantic web" stands for "knowledge": scaling semantics to the web
2019	01	Rob van Eijk (UL),Web privacy measurement in real-time bidding systems. A graph-based approach to RTB system classification
	02	Emmanuelle Beauxis Aussalet (CWI, UU), Statistics and Visualizations for Assessing Class Size Uncertainty
	03	Eduardo Gonzalez Lopez de Murillas (TU/e), Process Mining on Databases: Extracting Event Data from Real Life Data Sources
	04	Ridho Rahmadi (RUN), Finding stable causal structures from clinical data
	05	Sebastiaan van Zelst (TU/e), Process Mining with Streaming Data
	06	Chris Dijkshoorn (VUA), Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
	07	Soude Fazeli (TUD), Recommender Systems in Social Learning Platforms
	08	Frits de Nijs (TUD), Resource-constrained Multi-agent Markov Decision Processes
	09	Fahimeh Alizadeh Moghaddam (UvA), Self-adaptation for energy efficiency in software systems
	10	Qing Chuan Ye (EUR), Multi-objective Optimization Methods for Allocation and Prediction
	11	Yue Zhao (TUD), Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
	12	Jacqueline Heinerman (VUA), Better Together
	13	Guanliang Chen (TUD), MOOC Analytics: Learner Modeling and Content Generation
	14	Daniel Davis (TUD), Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses
	15	Erwin Walraven (TUD), Planning under Uncertainty in Constrained and Partially Observable Environments
	16	Guangming Li (TU/e), Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
	17	Ali Hurriyetoglu (RUN),Extracting actionable information from microtexts
	18	Gerard Wagenaar (UU), Artefacts in Agile Team Communication
	19	Vincent Koeman (TUD), Tools for Developing Cognitive Agents
	20	Chide Groenouwe (UU), Fostering technically augmented human collective intelligence
	21	Cong Liu (TU/e), Software Data Analytics: Architectural Model Discovery and Design Pattern Detection
	22	Martin van den Berg (VUA),Improving IT Decisions with Enterprise Architecture
	23	Qin Liu (TUD), Intelligent Control Systems: Learning, Interpreting, Verification
	24	Anca Dumitrache (VUA), Truth in Disagreement - Crowdsourcing Labeled Data for Natural Language Processing
	25	Emiel van Miltenburg (VUA), Pragmatic factors in (automatic) image description
	26	Prince Singh (UT), An Integration Platform for Synchromodal Transport
	27	Alessandra Antonaci (OU), The Gamification Design Process applied to (Massive) Open Online Courses
	28	Esther Kuindersma (UL), Cleared for take-off: Game-based learning to prepare airline pilots for critical situations
	29	Daniel Formolo (VUA), Using virtual agents for simulation and training of social skills in safety-critical circumstances
	30	Vahid Yazdanpanah (UT), Multiagent Industrial Symbiosis Systems
	31	Milan Jelisavcic (VUA), Alive and Kicking: Baby Steps in Robotics
	32	Chiara Sironi (UM), Monte-Carlo Tree Search for Artificial General Intelligence in Games

33 Anil Yaman (TU/e), Evolution of Biologically Inspired Learning in Artificial Neural Networks
34 Negar Ahmadi (TU/e), EEG Microstate and Functional Brain Network Features for Classification of Epilepsy and PNES
35 Lisa Facey-Shaw (OU), Gamification with digital badges in learning programming
36 Kevin Ackermans (OU), Designing Video-Enhanced Rubrics to Master Complex Skills
37 Jian Fang (TUD), Database Acceleration on FPGAs
38 Akos Kadar (OU), Learning visually grounded and multilingual representations

---

2020 01 Armon Toubman (UL), Calculated Moves: Generating Air Combat Behaviour
02 Marcos de Paula Bueno (UL), Unraveling Temporal Processes using Probabilistic Graphical Models
03 Mostafa Deghani (UvA), Learning with Imperfect Supervision for Language Understanding
04 Maarten van Gompel (RUN), Context as Linguistic Bridges
05 Yulong Pei (TU/e), On local and global structure mining
06 Preethu Rose Anish (UT), Stimulation Architectural Thinking during Requirements Elicitation - An Approach and Tool Support
07 Wim van der Vegt (OU), Towards a software architecture for reusable game components
08 Ali Mirsoleimani (UL),Structured Parallel Programming for Monte Carlo Tree Search
09 Myriam Traub (UU), Measuring Tool Bias and Improving Data Quality for Digital Humanities Research
10 Alifah Syamsiyah (TU/e), In-database Preprocessing for Process Mining
11 Sepideh Mesbah (TUD), Semantic-Enhanced Training Data AugmentationMethods for Long-Tail Entity Recognition Models
12 Ward van Breda (VUA), Predictive Modeling in E-Mental Health: Exploring Applicability in Personalised Depression Treatment
13 Marco Virgolin (CWI), Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming
14 Mark Raasveldt (CWI/UL), Integrating Analytics with Relational Databases
15 Konstantinos Georgiadis (OU), Smart CAT: Machine Learning for Configurable Assessments in Serious Games
16 Ilona Wilmont (RUN), Cognitive Aspects of Conceptual Modelling
17 Daniele Di Mitri (OU), The Multimodal Tutor: Adaptive Feedback from Multimodal Experiences
18 Georgios Methenitis (TUD), Agent Interactions & Mechanisms in Markets with Uncertainties: Electricity Markets in Renewable Energy Systems
19 Guido van Capelleveen (UT), Industrial Symbiosis Recommender Systems
20 Albert Hankel (VUA), Embedding Green ICT Maturity in Organisations
21 Karine da Silva Miras de Araujo (VUA), Where is the robot?: Life as it could be
22 Maryam Masoud Khamis (RUN), Understanding complex systems implementation through a modeling approach: the case of e-government in Zanzibar
23 Rianne Conijn (UT), The Keys to Writing: A writing analytics approach to studying writing processes using keystroke logging
24 Lenin da Nóbrega Medeiros (VUA/RUN), How are you feeling, human? Towards emotionally supportive chatbots
25 Xin Du (TU/e), The Uncertainty in Exceptional Model Mining
26 Krzysztof Leszek Sadowski (UU), GAMBIT: Genetic Algorithm for Model-Based mixed-Integer opTimization
27 Ekaterina Muravyeva (TUD), Personal data and informed consent in an educational context

28 Bibeg Limbu (TUD), Multimodal interaction for deliberate practice: Training complex skills with augmented reality
29 Ioan Gabriel Bucur (RUN), Being Bayesian about Causal Inference
30 Bob Zadok Blok (UL), Creatief, Creatiever, Creatiefst
31 Gongjin Lan (VUA), Learning better – From Baby to Better
32 Jason Rhuggenaath (TU/e), Revenue management in online markets: pricing and online advertising
33 Rick Gilsing (TU/e), Supporting service-dominant business model evaluation in the context of business model innovation
34 Anna Bon (UM), Intervention or Collaboration? Redesigning Information and Communication Technologies for Development
35 Siamak Farshidi (UU), Multi-Criteria Decision-Making in Software Production

---

2021 01 Francisco Xavier Dos Santos Fonseca (TUD),Location-based Games for Social Interaction in Public Space
02 Rijk Mercuur (TUD), Simulating Human Routines: Integrating Social Practice Theory in Agent-Based Models
03 Seyyed Hadi Hashemi (UvA), Modeling Users Interacting with Smart Devices
04 Ioana Jivet (OU), The Dashboard That Loved Me: Designing adaptive learning analytics for self-regulated learning
05 Davide Dell'Anna (UU), Data-Driven Supervision of Autonomous Systems
06 Daniel Davison (UT), "Hey robot, what do you think?" How children learn with a social robot
07 Armel Lefebvre (UU), Research data management for open science
08 Nardie Fanchamps (OU), The Influence of Sense-Reason-Act Programming on Computational Thinking
09 Cristina Zaga (UT), The Design of Robothings. Non-Anthropomorphic and Non-Verbal Robots to Promote Children's Collaboration Through Play
10 Quinten Meertens (UvA), Misclassification Bias in Statistical Learning
11 Anne van Rossum (UL), Nonparametric Bayesian Methods in Robotic Vision
12 Lei Pi (UL), External Knowledge Absorption in Chinese SMEs
13 Bob R. Schadenberg (UT), Robots for Autistic Children: Understanding and Facilitating Predictability for Engagement in Learning
14 Negin Samaeemofrad (UL), Business Incubators: The Impact of Their Support
15 Onat Ege Adali (TU/e), Transformation of Value Propositions into Resource Re-Configurations through the Business Services Paradigm
16 Esam A. H. Ghaleb (UM), Bimodal emotion recognition from audio-visual cues
17 Dario Dotti (UM), Human Behavior Understanding from motion and bodily cues using deep neural networks
18 Remi Wieten (UU), Bridging the Gap Between Informal Sense-Making Tools and Formal Systems - Facilitating the Construction of Bayesian Networks and Argumentation Frameworks
19 Roberto Verdecchia (VUA), Architectural Technical Debt: Identification and Management
20 Masoud Mansoury (TU/e), Understanding and Mitigating Multi-Sided Exposure Bias in Recommender Systems
21 Pedro Thiago Timbó Holanda (CWI), Progressive Indexes
22 Sihang Qiu (TUD), Conversational Crowdsourcing
23 Hugo Manuel Proença (UL), Robust rules for prediction and description
24 Kaijie Zhu (TU/e), On Efficient Temporal Subgraph Query Processing

25 Eoin Martino Grua (VUA), The Future of E-Health is Mobile: Combining AI and Self-Adaptation to Create Adaptive E-Health Mobile Applications
26 Benno Kruit (CWI/VUA), Reading the Grid: Extending Knowledge Bases from Human-readable Tables
27 Jelte van Waterschoot (UT), Personalized and Personal Conversations: Designing Agents Who Want to Connect With You
28 Christoph Selig (UL), Understanding the Heterogeneity of Corporate Entrepreneurship Programs

---

2022 01 Judith van Stegeren (UT), Flavor text generation for role-playing video games
02 Paulo da Costa (TU/e), Data-driven Prognostics and Logistics Optimisation: A Deep Learning Journey
03 Ali el Hassouni (VUA), A Model A Day Keeps The Doctor Away: Reinforcement Learning For Personalized Healthcare
04 Ünal Aksu (UU), A Cross-Organizational Process Mining Framework
05 Shiwei Liu (TU/e), Sparse Neural Network Training with In-Time Over-Parameterization
06 Reza Refaei Afshar (TU/e), Machine Learning for Ad Publishers in Real Time Bidding
07 Sambit Praharaj (OU), Measuring the Unmeasurable? Towards Automatic Co-located Collaboration Analytics
08 Maikel L. van Eck (TU/e), Process Mining for Smart Product Design
09 Oana Andreea Inel (VUA), Understanding Events: A Diversity-driven Human-Machine Approach
10 Felipe Moraes Gomes (TUD), Examining the Effectiveness of Collaborative Search Engines
11 Mirjam de Haas (UT), Staying engaged in child-robot interaction, a quantitative approach to studying preschoolers' engagement with robots and tasks during second-language tutoring
12 Guanyi Chen (UU), Computational Generation of Chinese Noun Phrases
13 Xander Wilcke (VUA), Machine Learning on Multimodal Knowledge Graphs: Opportunities, Challenges, and Methods for Learning on Real-World Heterogeneous and Spatially-Oriented Knowledge
14 Michiel Overeem (UU), Evolution of Low-Code Platforms
15 Jelmer Jan Koorn (UU), Work in Process: Unearthing Meaning using Process Mining
16 Pieter Gijsbers (TU/e), Systems for AutoML Research
17 Laura van der Lubbe (VUA), Empowering vulnerable people with serious games and gamification
18 Paris Mavromoustakos Blom (TiU), Player Affect Modelling and Video Game Personalisation
19 Bilge Yigit Ozkan (UU), Cybersecurity Maturity Assessment and Standardisation
20 Fakhra Jabeen (VUA), Dark Side of the Digital Media - Computational Analysis of Negative Human Behaviors on Social Media
21 Seethu Mariyam Christopher (UM), Intelligent Toys for Physical and Cognitive Assessments
22 Alexandra Sierra Rativa (TiU), Virtual Character Design and its potential to foster Empathy, Immersion, and Collaboration Skills in Video Games and Virtual Reality Simulations
23 Ilir Kola (TUD), Enabling Social Situation Awareness in Support Agents
24 Samaneh Heidari (UU), Agents with Social Norms and Values - A framework for agent based social simulations with social norms and personal values
25 Anna L.D. Latour (UL), Optimal decision-making under constraints and uncertainty
26 Anne Dirkson (UL), Knowledge Discovery from Patient Forums: Gaining novel medical insights from patient experiences

27 Christos Athanasiadis (UM), Emotion-aware cross-modal domain adaptation in video sequences
28 Onuralp Ulusoy (UU), Privacy in Collaborative Systems
29 Jan Kolkmeier (UT), From Head Transform to Mind Transplant: Social Interactions in Mixed Reality
30 Dean De Leo (CWI), Analysis of Dynamic Graphs on Sparse Arrays
31 Konstantinos Traganos (TU/e), Tackling Complexity in Smart Manufacturing with Advanced Manufacturing Process Management
32 Cezara Pastrav (UU), Social simulation for socio-ecological systems
33 Brinn Hekkelman (CWI/TUD), Fair Mechanisms for Smart Grid Congestion Management
34 Nimat Ullah (VUA), Mind Your Behaviour: Computational Modelling of Emotion & Desire Regulation for Behaviour Change
35 Mike E.U. Ligthart (VUA), Shaping the Child-Robot Relationship: Interaction Design Patterns for a Sustainable Interaction

---

2023 01 Bojan Simoski (VUA), Untangling the Puzzle of Digital Health Interventions
02 Mariana Rachel Dias da Silva (TiU), Grounded or in flight? What our bodies can tell us about the whereabouts of our thoughts
03 Shabnam Najafian (TUD), User Modeling for Privacy-preserving Explanations in Group Recommendations
04 Gineke Wiggers (UL), The Relevance of Impact: bibliometric-enhanced legal information retrieval
05 Anton Bouter (CWI), Optimal Mixing Evolutionary Algorithms for Large-Scale Real-Valued Optimization, Including Real-World Medical Applications
06 António Pereira Barata (UL), Reliable and Fair Machine Learning for Risk Assessment
07 Tianjin Huang (TU/e), The Roles of Adversarial Examples on Trustworthiness of Deep Learning
08 Lu Yin (TU/e), Knowledge Elicitation using Psychometric Learning
09 Xu Wang (VUA), Scientific Dataset Recommendation with Semantic Techniques
10 Dennis J.N.J. Soemers (UM), Learning State-Action Features for General Game Playing
11 Fawad Taj (VUA), Towards Motivating Machines: Computational Modeling of the Mechanism of Actions for Effective Digital Health Behavior Change Applications
12 Tessel Bogaard (VUA), Using Metadata to Understand Search Behavior in Digital Libraries
13 Injy Sarhan (UU), Open Information Extraction for Knowledge Representation
14 Selma Čaušević (TUD), Energy resilience through self-organization
15 Alvaro Henrique Chaim Correia (TU/e), Insights on Learning Tractable Probabilistic Graphical Models
16 Peter Blomsma (TiU), Building Embodied Conversational Agents: Observations on human nonverbal behaviour as a resource for the development of artificial characters
17 Meike Nauta (UT), Explainable AI and Interpretable Computer Vision – From Oversight to Insight
18 Gustavo Penha (TUD), Designing and Diagnosing Models for Conversational Search and Recommendation
19 George Aalbers (TiU), Digital Traces of the Mind: Using Smartphones to Capture Signals of Well-Being in Individuals
20 Arkadiy Dushatskiy (TUD), Expensive Optimization with Model-Based Evolutionary Algorithms applied to Medical Image Segmentation using Deep Learning
21 Gerrit Jan de Bruin (UL), Network Analysis Methods for Smart Inspection in the Transport Domain
22 Alireza Shojaifar (UU), Volitional Cybersecurity

23 Theo Theunissen (UU), Documentation in Continuous Software Development
24 Agathe Balayn (TUD), Practices Towards Hazardous Failure Diagnosis in Machine Learning
25 Jurian Baas (UU), Entity Resolution on Historical Knowledge Graphs
26 Loek Tonnaer (TU/e), Linearly Symmetry-Based Disentangled Representations and their Out-of-Distribution Behaviour
27 Ghada Sokar (TU/e), Learning Continually Under Changing Data Distributions
28 Floris den Hengst (VUA), Learning to Behave: Reinforcement Learning in Human Contexts
29 Tim Draws (TUD), Understanding Viewpoint Biases in Web Search Results

---

2024 01 Daphne Miedema (TU/e), On Learning SQL: Disentangling concepts in data systems education
02 Emile van Krieken (VUA), Optimisation in Neurosymbolic Learning Systems
03 Feri Wijayanto (RUN), Automated Model Selection for Rasch and Mediation Analysis
04 Mike Huisman (UL), Understanding Deep Meta-Learning
05 Yiyong Gou (UM), Aerial Robotic Operations: Multi-environment Cooperative Inspection & Construction Crack Autonomous Repair
06 Azqa Nadeem (TUD), Understanding Adversary Behavior via XAI: Leveraging Sequence Clustering to Extract Threat Intelligence
07 Parisa Shayan (TiU), Modeling User Behavior in Learning Management Systems
08 Xin Zhou (UvA), From Empowering to Motivating: Enhancing Policy Enforcement through Process Design and Incentive Implementation
09 Giso Dal (UT), Probabilistic Inference Using Partitioned Bayesian Networks
10 Cristina-Iulia Bucur (VUA), Linkflows: Towards Genuine Semantic Publishing in Science
11 withdrawn
12 Peide Zhu (TUD), Towards Robust Automatic Question Generation For Learning
13 Enrico Liscio (TUD), Context-Specific Value Inference via Hybrid Intelligence
14 Larissa Capobianco Shimomura (TU/e), On Graph Generating Dependencies and their Applications in Data Profiling
15 Ting Liu (VUA), A Gut Feeling: Biomedical Knowledge Graphs for Interrelating the Gut Microbiome and Mental Health
16 Arthur Barbosa Câmara (TUD), Designing Search-as-Learning Systems
17 Razieh Alidoosti (VUA), Ethics-aware Software Architecture Design
18 Laurens Stoop (UU), Data Driven Understanding of Energy-Meteorological Variability and its Impact on Energy System Operations
19 Azadeh Mozafari Mehr (TU/e), Multi-perspective Conformance Checking: Identifying and Understanding Patterns of Anomalous Behavior
20 Ritsart Anne Plantenga (UL), Omgang met Regels
21 Federica Vinella (UU), Crowdsourcing User-Centered Teams
22 Zeynep Ozturk Yurt (TU/e), Beyond Routine: Extending BPM for Knowledge-Intensive Processes with Controllable Dynamic Contexts
23 Jie Luo (VUA), Lamarck's Revenge: Inheritance of Learned Traits Improves Robot Evolution
24 Nirmal Roy (TUD), Exploring the effects of interactive interfaces on user search behaviour
25 Alisa Rieger (TUD), Striving for Responsible Opinion Formation in Web Search on Debated Topics
26 Tim Gubner (CWI), Adaptively Generating Heterogeneous Execution Strategies using the VOILA Framework
27 Lincen Yang (UL), Information-theoretic Partition-based Models for Interpretable Machine Learning

28 Leon Helwerda (UL), Grip on Software: Understanding development progress of Scrum sprints and backlogs
29 David Wilson Romero Guzman (VUA), The Good, the Efficient and the Inductive Biases: Exploring Efficiency in Deep Learning Through the Use of Inductive Biases
30 Vijanti Ramautar (UU), Model-Driven Sustainability Accounting
31 Ziyu Li (TUD), On the Utility of Metadata to Optimize Machine Learning Workflows
32 Vinicius Stein Dani (UU), The Alpha and Omega of Process Mining
33 Siddharth Mehrotra (TUD), Designing for Appropriate Trust in Human-AI interaction
34 Robert Deckers (VUA), From Smallest Software Particle to System Specification - MuD-ForM: Multi-Domain Formalization Method
35 Sicui Zhang (TU/e), Methods of Detecting Clinical Deviations with Process Mining: a fuzzy set approach
36 Thomas Mulder (TU/e), Optimization of Recursive Queries on Graphs
37 James Graham Nevin (UvA), The Ramifications of Data Handling for Computational Models
38 Christos Koutras (TUD), Tabular Schema Matching for Modern Settings
39 Paola Lara Machado (TU/e), The Nexus between Business Models and Operating Models: From Conceptual Understanding to Actionable Guidance
40 Montijn van de Ven (TU/e), Guiding the Definition of Key Performance Indicators for Business Models
41 Georgios Siachamis (TUD), Adaptivity for Streaming Dataflow Engines
42 Emmeke Veltmeijer (VUA), Small Groups, Big Insights: Understanding the Crowd through Expressive Subgroup Analysis
43 Cedric Waterschoot (KNAW Meertens Instituut), The Constructive Conundrum: Computational Approaches to Facilitate Constructive Commenting on Online News Platforms
44 Marcel Schmitz (OU), Towards learning analytics-supported learning design
45 Sara Salimzadeh (TUD), Living in the Age of AI: Understanding Contextual Factors that Shape Human-AI Decision-Making
46 Georgios Stathis (Leiden University), Preventing Disputes: Preventive Logic, Law & Technology
47 Daniel Daza (VUA), Exploiting Subgraphs and Attributes for Representation Learning on Knowledge Graphs
48 Ioannis Petros Samiotis (TUD), Crowd-Assisted Annotation of Classical Music Compositions

---

2025 01 Max van Haastrecht (UL), Transdisciplinary Perspectives on Validity: Bridging the Gap Between Design and Implementation for Technology-Enhanced Learning Systems
02 Jurgen van den Hoogen (JADS), Time Series Analysis Using Convolutional Neural Networks
03 Andra-Denis Ionescu (TUD), Feature Discovery for Data-Centric AI
04 Rianne Schouten (TU/e), Exceptional Model Mining for Hierarchical Data
05 Nele Albers (TUD), Psychology-Informed Reinforcement Learning for Situated Virtual Coaching in Smoking Cessation
06 Daniël Vos (TUD), Decision Tree Learning: Algorithms for Robust Prediction and Policy Optimization
07 Ricky Maulana Fajri (TU/e), Towards Safer Active Learning: Dealing with Unwanted Biases, Graph-Structured Data, Adversary, and Data Imbalance
08 Stefan Bloemheuvel (TiU), Spatio-Temporal Analysis Through Graphs: Predictive Modeling and Graph Construction
09 Fadime Kaya (VUA), Decentralized Governance Design - A Model-Based Approach

10 Zhao Yang (UL), Enhancing Autonomy and Efficiency in Goal-Conditioned Reinforcement Learning
11 Shahin Sharifi Noorian (TUD), From Recognition to Understanding: Enriching Visual Models Through Multi-Modal Semantic Integration
12 Lijun Lyu (TUD), Interpretability in Neural Information Retrieval
13 Fuda van Diggelen (VUA), Robots Need Some Education: on the complexity of learning in evolutionary robotics
14 Gennaro Gala (TU/e), Probabilistic Generative Modeling with Latent Variable Hierarchies
15 Michiel van der Meer (UL), Opinion Diversity through Hybrid Intelligence
16 Monika Grewal (TU Delft), Deep Learning for Landmark Detection, Segmentation, and Multi-Objective Deformable Registration in Medical Imaging
17 Matteo De Carlo (VUA), Real Robot Reproduction: Towards Evolving Robotic Ecosystems
18 Anouk Neerincx (UU), Robots That Care: How Social Robots Can Boost Children's Mental Wellbeing
19 Fang Hou (UU), Trust in Software Ecosystems
20 Alexander Melchior (UU), Modelling for Policy is More Than Policy Modelling (The Useful Application of Agent-Based Modelling in Complex Policy Processes)
21 Mandani Ntekouli (UM), Bridging Individual and Group Perspectives in Psychopathology: Computational Modeling Approaches using Ecological Momentary Assessment Data
22 Hilde Weerts (TU/e), Decoding Algorithmic Fairness: Towards Interdisciplinary Understanding of Fairness and Discrimination in Algorithmic Decision-Making
23 Roderick van der Weerdt (VUA), IoT Measurement Knowledge Graphs: Constructing, Working and Learning with IoT Measurement Data as a Knowledge Graph
24 Zhong Li (UL), Trustworthy Anomaly Detection for Smart Manufacturing
25 Kyana van Eijndhoven (TiU), A Breakdown of Breakdowns: Multi-Level Team Coordination Dynamics under Stressful Conditions
26 Tom Pepels (UM), Monte-Carlo Tree Search is Work in Progress
27 Danil Provodin (JADS, TU/e), Sequential Decision Making Under Complex Feedback
28 Jinke He (TU Delft), Exploring Learned Abstract Models for Efficient Planning and Learning
29 Erik van Haeringen (VUA), Mixed Feelings: Simulating Emotion Contagion in Groups
30 Myrthe Reuver (VUA), A Puzzle of Perspectives: Interdisciplinary Language Technology for Responsible News Recommendation
31 Gebrekirstos Gebreselassie Gebremeskel (RUN), Spotlight on Recommender Systems: Contributions to Selected Components in the Recommendation Pipeline
32 Ryan Brate (UU), Words Matter: A Computational Toolkit for Charged Terms
33 Merle Reimann (VUA), Speaking the Same Language: Spoken Capability Communication in Human-Agent and Human-Robot Interaction
34 Eduard C. Groen (UU), Crowd-Based Requirements Engineering
35 Urja Khurana (VUA), From Concept To Impact: Toward More Robust Language Model Deployment
36 Anna Maria Wegmann (UU), Say the Same but Differently: Computational Approaches to Stylistic Variation and Paraphrasing
37 Chris Kamphuis (RUN), Exploring Relations and Graphs for Information Retrieval
38 Valentina Maccatrozzo (VUA), Break the Bubble: Semantic Patterns for Serendipity
39 Dimitrios Alivanistos (VUA), Knowledge Graphs & Transformers for Hypothesis Generation: Accelerating Scientific Discovery in the Era of Artificial Intelligence
40 Stefan Grafberger (UvA), Declarative Machine Learning Pipeline Management via Logical Query Plans

41 Mozhgan Vazifehdoostirani (TU/e), Leveraging Process Flexibility to Improve Process Outcome - From Descriptive Analytics to Actionable Insights
42 Margherita Martorana (VUA), Semantic Interpretation of Dataless Tables: a metadata-driven approach for findable, accessible, interoperable and reusable restricted access data
43 Krist Shingjergji (OU), Sense the Classroom - Using AI to Detect and Respond to Learning-Centered Affective States in Online Education
44 Robbert Reijnen (TU/e), Dynamic Algorithm Configuration for Machine Scheduling Using Deep Reinforcement Learning
45 Anjana Mohandas Sheeladevi (VUA), Occupant-Centric Energy Management: Balancing Privacy, Well-being and Sustainability in Smart Buildings
46 Ya Song (TU/e), Graph Neural Networks for Modeling Temporal and Spatial Dimensions in Industrial Decision-making
47 Tom Kouwenhoven (UL), Collaborative Meaning-Making. The Emergence of Novel Languages in Humans, Machines, and Human-Machine Interactions
48 Evy van Weelden (TiU), Integrating Virtual Reality and Neurophysiology in Flight Training
49 Selene Báez Santamaría (VUA), Knowledge-centered conversational agents with a drive to learn
50 Lea Krause (VUA), Contextualising Conversational AI
51 Jiaxu Zhao (TU/e), Understanding and Mitigating Unwanted Biases in Generative Language Models
52 Qiao Xiao (TU/e), Model, Data and Communication Sparsity for Efficient Training of Neural Networks
53 Gaole He (TUD), Towards Effective Human-AI Collaboration: Promoting Appropriate Reliance on AI Systems
54 Go Sugimoto (VUA), MISSING LINKS Investigating the Quality of Linked Data and its Tools in Cultural Heritage and Digital Humanities
55 Sietze Kai Kuilman (TUD), AI that Glitters is Not Gold: Requirements for Meaningful Control of AI Systems
56 Wijnand van Woerkom (UU), A Fortiori Case-Based Reasoning: Formal Studies with Applications in Artificial Intelligence and Law
57 Syeda Amna Sohail (UT), Privacy-Utility Trade-Off in Healthcare Metadata Sharing and Beyond: A Normative and Empirical Evaluation at Inter and Intra Organizational Levels
58 Junhan Wen (TUD), "From iMage to Market": Machine-Learning-Empowered Fruit Supply
59 Mohsen Abbaspour Onari (TU/e), From Explanation to Trust: Modeling and Measuring Trust in Explainable Decision Support
60 Marcel Jurriaan Robeer (UU), Beyond Trust: A Causal Approach to Explainable AI in Law Enforcement
61 Shuai Wang (VUA), Links in Large Integrated Knowledge Graphs: Analysis, Refinement, and Domain Applications
62 Khaleel Asyraaf Mat Sanusi (OU), Augmenting a learning model within immersive learning environments for psychomotor skills
63 Rashid Zaman (TU/e), Online Conformance Checking on Degraded Data
64 **Jens d'Hondt** (TU/e), Effective and Efficient Multivariate Similarity Search

Author's Publications

The author has made the following publications in scientific journals, workshop proceedings during the course of his PhD research.

The following publications are included in this dissertation (ordered chronologically by publication data):

1. **d'Hondt, J. E., Minartz, K., and Papapetrou, O.** Efficient detection of multivariate correlations with different correlation measures. *The VLDB Journal 33*, 2 (2024), 481–505

2. **d'Hondt, J. E., Papapetrou, O., and Paparrizos, J.** Beyond the Dimensions: A Structured Evaluation of Multivariate Time Series Distance Measures. In *Proc. ICDEW'24* (2024), pp. 107–112

3. **Papapetrou, O., and d'Hondt, J. E.** Multivariate Similarity Search - A Call for a New Breed of Similarity Search Algorithms. In *Proc. ICDE'24* (2024), pp. 5662–5662

4. **Paparrizos, J., Li, H., Yang, F., Wu, K., d'Hondt, J. E., and Papapetrou, O.** *Data Series Management and Analytics: From Time Series to High-dimensional Vectors.* 2025, ch. A Survey on Time-Series Distance Measures. In press

5. **d'Hondt, J. E., Li, H., Yang, F., Papapetrou, O., and Paparrizos, J.** A Structured Study of Multivariate Time-Series Distance Measures. In *Proc. SIGMOD'25* (2025)

6. **Pelok, B., and d'Hondt, J. E.** MULISSE: Variable-Length Similarity Search for Multivariate Time Series. In *Proc. ICDEW'25* (2025)

7. **d'Hondt, J. E., Kortekaas, T., Papapetrou, O., and Palpanas, T.** MS-Index: Fast Top-k Subsequence Search for Multivariate Time Series under Euclidean Distance. In *Proc. VLDB'26* (2025)

The following publication made by the author is not included in this dissertation:

1. **Jörges, C., d'Hondt, J. E., and Chatzigeorgakidis, G.** Leaf Area Index Time Series Imputation for Early Yield Prediction. In *Proc.BIDS'23* (2023), pp. 373–376

Summary

Every day, millions of sensors monitor our power grids, medical devices track vital signs, and satellites observe our planet's climate. These systems generate vast amounts of complex, interconnected data with patterns that span multiple measurements and variables. At the heart of identifying these patterns lies the fundamental task of similarity search; given a specific pattern or variable of interest, what are the most similar variables in the data? As we describe in **Chapter 1**, existing methods for such search have remained stuck in a simpler era where objects or phenomena were measured through single variables, and relationships were only measured between pairs of objects. In this dissertation, we aim to advance the traditional similarity search paradigm to embrace the multivariate nature of modern data and the complex relationships within it.

We pursue this goal through two complementary directions. First, how can we effectively search for patterns in multivariate data? And second, how can we discover relationships that span multiple variables? In **Chapter 2**, we establish the theoretical foundations for both directions and identify why existing methods have struggled with these challenges.

The first part of this dissertation focuses on searching patterns in multivariate data. In **Chapter 3**, we tackle the fundamental question of how to effectively measure similarity between multivariate objects, and how to prepare the data for such comparisons. Through a comprehensive study, we develop practical guidelines for choosing appropriate measures and preprocessing steps for different downstream tasks such as classification, clustering, and anomaly detection. Our findings challenge common assumptions about data normalization and reveal that the cost of complex measures are not always justified in terms of accuracy. Building on these insights, **Chapter 4** introduces new techniques that make such comparisons computationally efficient, enabling millisecond-level responses on large-scale datasets.

The second part turns to discovering relationships between multiple variables. In **Chapter 5**, we develop new algorithms that can efficiently detect multivariate relationships in both static and streaming data. These methods achieve up to 100x speedup over existing approaches while guaranteeing complete results, making previously intractable analyses possible. We then generalize these ideas in **Chapter 6** into a comprehensive framework that can mine multivariate relationships from the data under any (domain-specific) measure of similarity. The framework automatically applies effective optimizations based on the properties of the measure, resulting in orders of magnitude speedup without requiring expert knowledge from users. To demonstrate its practical value, **Chapter 7** shows how this framework can help solve a long-standing problem in signal processing in a manner that provides a unique trade-off between accuracy and efficiency.

Finally, in **Chapter 8**, we reflect on these contributions and outline future research directions. The ultimate goal is to combine both advances - enabling the discovery of multivariate relationships within multivariate data. This represents a crucial step towards better understanding the complex, interconnected systems that shape our world.

About the Author

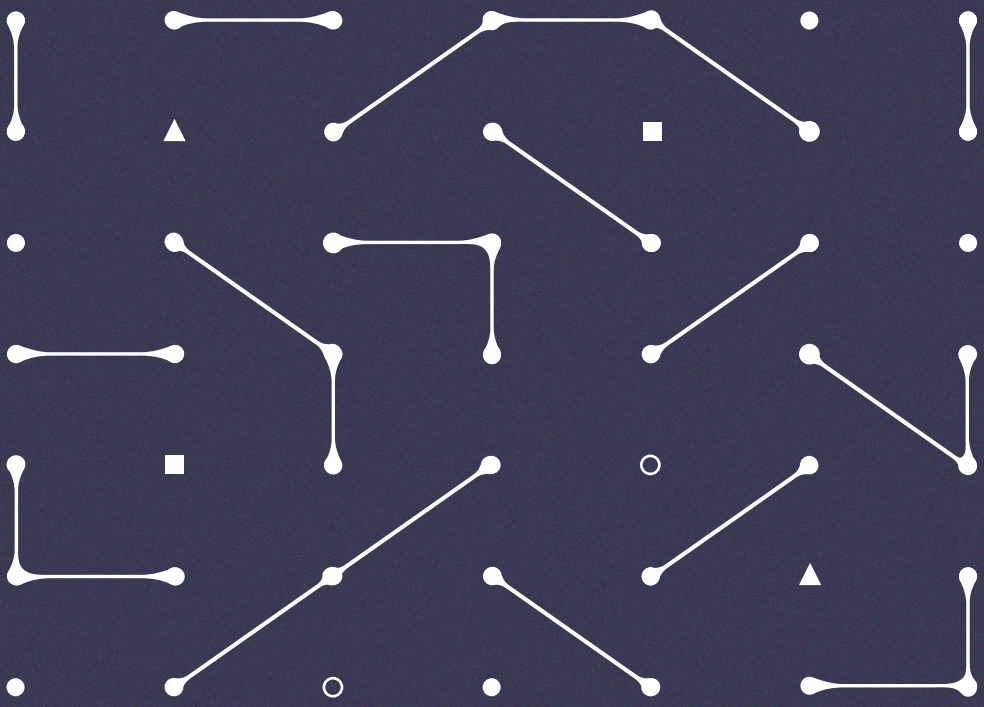


Jens d'Hondt was born on 9 January 1998 in Sint-Niklaas, Belgium, growing up in the nearby town of Hulst (NL). He obtained his BSc degree in Industrial Engineering cum laude at TU/e in 2019, followed by an MSc in Data Science and Engineering, graduating cum laude in 2021. Throughout his studies, he gained research experience through an internship at BMW in Munich, and working as a freelance research engineer at the Information Systems group at TU/e, under guidance of Dr. Pieter van Gorp. Besides his academic career, he has worked as a data scientist at Crossyn Automotive and co-founded a student recruitment agency, called Dpasse.

In 2021, d'Hondt started as a PhD candidate at the Database group at TU/e, under the supervision of Dr. Odysseas Papapetrou and Prof. Dr. George Fletcher. His research focused on the development of efficient algorithms for large-scale time series analysis, with a particular emphasis on similarity search. His work has been published in various top-level scientific conference proceedings and journals, including SIGMOD, VLDB, ICDE, and the VLDB Journal. He has also served as the publication chair for the 2024 and 2025 editions of the International Workshop on Multivariate Time Series Analysis (MulTiSA), and as a reviewer for several conferences and journals, such as ICDEW, the Data Mining and Knowledge Discovery journal, and the Big Data Research journal. In terms of teaching, d'Hondt has (co-)supervised over 8 MSc thesis students, and has been involved in the courses "Big Data Management" and "Data Intensive Systems and Applications" at TU/e.

During his PhD, d'Hondt was an active member of the consortium of the EU-funded STELAR project, where he led research on Correlation Discovery and made significant contributions to the development of remote sensing technologies for precision agriculture and crop monitoring.

After defending his PhD, d'Hondt will continue his research career as a postdoctoral researcher at the Barcelona Supercomputing Center (BSC) in Spain, working under the supervision of Dr. Hervé Petetin.



We live in a world where every event is the result of a complex web of interacting factors. Scientists have mastered the art of finding the simple, pairwise relationships in data, but what about the complex, higher-order interactions that truly shape our reality?

In this dissertation, Jens E. d'Hondt explores how we can expand our analytical toolkit to model and understand the true multi-faceted nature of the world around us, without having to break the bank on computational resources.