Feature Extraction from Multi-variate Time Series and Resource-Aware Indexing

by

Sicong Liu

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved February 2020 by the
Graduate Supervisory Committee:

K. Selçuk Candan, Chair
Hasan Davulcu
Maria Luisa Sapino
Mohamed Sarwat

ARIZONA STATE UNIVERSITY

May 2020

ABSTRACT

In the presence of big data analysis, large volume of data needs to be systematically indexed to support analytical tasks, such as feature engineering, pattern recognition, data mining, and query processing. The volume, variety, and velocity of these data necessitate sophisticated systems to help researchers understand, analyze, and discover insights from heterogeneous, multidimensional data sources. Many analytical frameworks have been proposed in the literature in recent years, but challenges to accuracy, speed, and effectiveness remain hence a systematic approach to perform data signature computation and query processing in multi-dimensional space is in people's interest. In particular, real-time and near real-time queries pose significant challenges when working with large data sets.

To address these challenges, I develop an innovative robust multi-variate feature extraction algorithm over multi-dimensional temporal datasets, which is able to help understand and analyze various real-world applications. Furthermore, to answer queries over these features, I develop a novel resource-aware indexing framework to approximately solve top-k queries by leveraging onion-layer indexing in conjunction with locality sensitive hashing. The proposed indexing scheme allows people to answer top-k queries by only accessing a bounded amount of data, which optimizes big data small for queries.

To my grandparents

ACKNOWLEDGMENTS

I would like to first thank my advisor, Dr. K. Selçuk Candan for his support and guidance on my PhD work through this journey. He has lead a great example of being a research scientist and he has helped me build habbits of being a life long hard working researcher. I am grateful to have him as my PhD advisor.

I would like to thank all my committee members with heart and soul, Dr. Hasan Davulcu, Dr. Maria Luisa Sapino and Dr. Mohamed Sarwat for their insightful guidance and comments on my research. I cannot finish my thesis and complete my work without their help.

I would like to thank all the researchers and engineers that I work with through the past years, including Tony Ma my manager from Apple internship, Yuanyuan Tian and Fatma Ozcan, my mentor and my manager from IBM Almaden, Fangqiu Han, Matthias Boehm, Ruocheng Guo, Jia Yu and many more. It's people like them help me stand on the shoulders of giants. I am thankful for entire Emitlab members that helped and guided me during my study at ASU, especailly Jung Hyun Kim, Silvestro Poccia and Hans Behrens.

Pursuing a PhD degree is a difficult process and I am grateful for Marilyn Zellet, Simon and Critina Bently who has supported me since the very beginning of this journey, and Christafer Suddarth who has supported me at the most difficult times. I want to give special thanks to Caleb Lepisto and Steve Chiapelli who have always been encouraging me through the past years. I'm very grateful for my coach Zachary Moran, who has encouraged and challenged me mentally and physically at the most critial momements.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

It is now the era of big data, a large volumne of which needs to be indexed adequiately and systematically to support information retreival tasks including pattern recognition, data structure mining and top-k query processing etc. Hence sophisticated system/framework is in need to help researchers understand, analyze and discover the relationships of hetergenous data. For instance, one would like to analyze the semantic meaning of human motion and would like to retrive top-k most similar motions based on a given query of interest. Moreover, the data and/or query might come in streaming manner which introduces challenges about how to incrementally maintain indexing schemes to achieve performance guarantee including query accuracy and efficiency. In real world data in multi-dimensional space is often associated with temporal information hence makes it time-series data, for instance, data from weather forecast, stock market, medical ECG etc. Essentially a time series data is a series of data points indexed in timely order, such as a sequence taken at successive equally spaced points in time. Time series can be *a univariate time series* is a series of data with a single time-dependent variable whereas a *multi-variate time serires* has more than one time-dependent variable such that each variable might depend not only on its past values but also has some dependeny on other variables.

1.1 Time Series

A *time series* is a series of data points indexed in time order which embeds observations with temporal information and is ubiquitous in the presence of Internet-of-Things(IoT). Analysis and exploration tasks of time series such as prediction, clustering, classification, sub-sequence matching and similarity search often require proper indexing structure for various types of time series data under different application scenarios. Here I will first address the challenges and common strategies in uni-variate time series indexing then discuss the difficulties and techniques in the presence of multi-variate time series. Then I will introduce the proposed RMT feature based indexing.

1.1.1 Uni-variate Time Series

A time series T is an ordered list of observations: $T = (t_1, t_2, \ldots, t_m)$ where $1, 2, \ldots, m$ represent time steps, which can also be viewed as dimensions. To index T the major challenge lies in the precense of dimensionality with timely order. A common approach is to reduce the dimensionality of a given time series data record either through global or local features or aggregation based representation methods. More specifically,

- patterns and features are able to describe the salient properties of the data and they often contain less volume compared to the original data T hence can be used for indexing Candan and Sapino (2010a). Popular approaches include extraction of *global features of the times series*, for instance, spectral properties quantified using a transformation (Discrete Cosine or Wavelet Transforms). *Global fingerprints* for example, can also be leveraged for indexing puroses, for instance correlations, transfer functions, variate clusters and spectral properties Li *et al.* (2010) that can be computed using SVD or eigen-decompositions. Alternatively, it has been noticed in research community that repeated patterns of interest can also describe the behvior of a given time series data record T hence can be studied for indexing. These repeated patterns are commonly known as motifs which again can be searched and computed for idnexing purpose

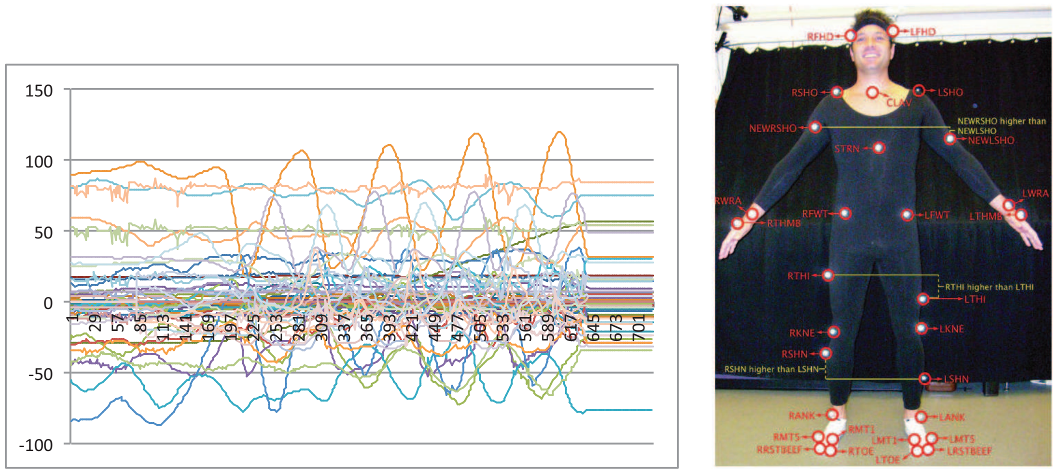
Esling and Agon (2012); Mohammad and Nishida (2009). *Local features or features carry localized temporal information of interest*, can also be leveraged for time series indexing. Popular approaches include landmarks Perng *et al.* (2000), shaplets Rakthanmanon and Keogh (2013); Ye and Keogh (2009), snippets Wang and Candan (2010), longest common subsequences (LCSS) Vlachos *et al.* (2006). Candan *et al.* (2012b) has been proposed in the past for computing salient local features, which are robust against noice. Additionally, RPM Wang *et al.* (2016a) and STS3 Peng *et al.* (2016a) are two recent methods that search informative patterns that serve indexing purposes.

- A popular aggregation based method is Symbolic Aggregate approXimation (SAX) Lin *et al.* (2003a) which is based on Piecewise Aggregate Approximate (PAA) Keogh *et al.* (2001): the design of PAA relies on the fact that time series data can be approximated by segmenting the sequences into equi-length subsections and recording the corresponding mean value, which is simple and quick to obtain. In SAX, a uni-variate time series is first transformed into its PAA representation with a fixed window size. Afterwards, each sub-sequence mean is quantized over a symbolic alphabet through a Gaussian filter that takes into account the minimum and maximum values from the time series of interest.

1.1.2 Multi-variate Time Series

With the notion of uni-variate time series discussed above, the multi-variate counter part can be represented as: $Y(t) =< T_1(t), T_2(t), \ldots, T_n(t) >$ where n is the number of variates. A straightfoward indexing technique for multi-variate time series is that one can treat each and every variate independently, apply the uni-variate indexing scheme then aggregating them together. This method will introduce ambiguity at later phase when performing similarity search, ranking or classification tasks since pair-wise variate comparison has to be enforced and how to combine and aggregate similarity or score across different variates can introduce overhead cost.

Tasks such as *effective search and analysis of multi-variate time series* often requires more complicated tools or systems, with monitoring and understanding the temporal evolution of complex phenomena.



(a) Sample motion data (b) Sensors

Figure 1.1: A multi-variate time series, tracking 62 sensors, created by body motion capture Mocap (2001)

In presence of multiple aspects of recorded sensory data, researchers often face a complex sequence, as seen in Figure 1.1 which is a multi-variate time series of 62 variates from a gesture dataset (MoCap Mocap (2001)). The variates do not show the same trend, but often there are subgroups of variates evolving together. Treating them as uni-variate time series and aggregating them together might not be sufficient enough to completely understand the pattern.

Operations such as clustering and indexing of multi-variate time series are in need, yet these tasks can be difficult due to the nature of the data:

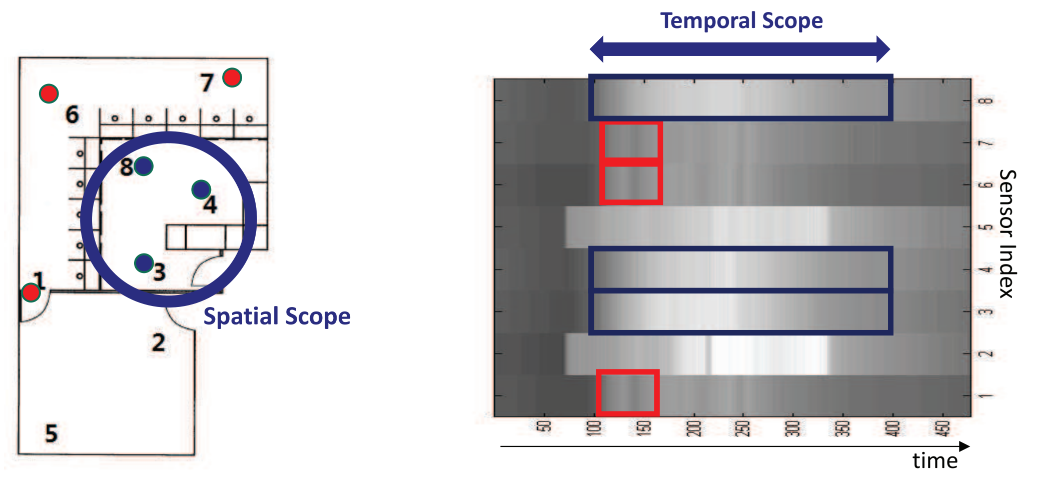
- *time series can be extremely large*: i.e. the recording of a temperature in a room every five minutes for one year.

- *time series can be affected by noise.*

- There can be *missing or anormaly subsequences* within multi-variate time series.

External knowledge such as how the variates are connected can be leveraged to extra local features that invovle multiple variates, to describe temporcal characteristics potentially at multiple scales. These features can thus be indexed to represent mult-variate time series. Given these observations including variate relationships that are nkown a priori, multi-variate time series can be described as $\mathbb{Y} =< Y(t), R >$ where R represents dependency/correlation among individual variate. By leveraging the relationships in between different variates, Wang *et al.* (2014b) proposed a multi-variate feature extraction technqiue: as in Candan *et al.* (2012b), it also (a) smoothes the data to generate different version of the input object corresponding to different scales; (b) compares neighboring points both in time and scale to identify regions where the gradients are sufficiently large. SIFT-like features descriptors Lowe (2004a) are then extracted to support indexing and search.

To understand the pattern and analyze the relationships among data, a well-known approach is to learn models that contain a label indicating the ground truth information. During pattern learning phase, current State-of-Art methods over time series, such as Dynamic Time Warping (DTW) Bemdt and Clifford (1994); Chen and Ng (2004), Symbolic Aggregate Approximation (SAX) Lin *et al.* (2003a); Sakoe and Chiba (1978); Keogh (2002), focus on uni-variate time series data including feature extraction Patel *et al.* (2002), indexing and corresponding retrieval tasks. Many real-world scenarios, however, generate and/or consume multi-variate time series, such as IoT sensor data from heterogeneous sources, human motion sensors etc. A common method is to treat each and every variate independently using the approaches for uni-variate time series and perform aggregations afterwards to achieve the analysis for the multi-variate counter-part. Such methods would ignore the important correlation among variates when performing feature extraction and indexing tasks at the beginning. Moreoever, points of interest or features of time series often spatial and temporal scopes which can also have multiple scales. For instance, Figure 1.2 depicts the spatial and temporal scope of a point of interest that invovles multiple variates for energy building: one event of interest occurs invovles sensor 1, 6, 7 and another invovles sensor 3, 4, and 8, which indicates their spatial group within a building; each of these two events also has temporal scopes respecitvely indicating the duration of the patterns.



(a) Spatial Scope (b) Temporal Scope

Figure 1.2: Multi-variate time series event of interest with scope information

Hence a novel framework to help experts adequately and systematically search for and interpret multi-variate observations is in need.

1.1.3 Robust Multi-variate Temporal Features (RMT)

To solve these research challenges I introduce *Robust Multi-variate Temporal (RMT) Features of Multi-variate Time Series* Liu *et al.* (2018) which aims at developing data models and algorithms to detect features that are robust against noise and can be indexed for efficient and accurate retrieval as well as for supporting data exploration and analysis. More specifically, I first observe that multi-variate time series often carry localized multi-variate temporal features that are robust against noise Lowe (2004a). I then argue that these multi-variate temporal features can be extracted by simultaneously considering, at multiple scales, temporal characteristics of the time-series along with external knowledge, including variate relationships that are known a priori Wang *et al.* (2014b). Generally speaking, features detected by *RMT* share the following properties:

- they are more effective local-feature sets that can be identified if the aspects of time and data can be considered independently, which leads to features with heterogeneous scales along time and variates.
- these local features that involve multiple variates can help researchers better interpret data which considers the variate-relationships for indexing constructions of various tasks, supporting decision making department Poccia *et al.* (2017); Liu *et al.* (2016, 2015) all of which will be introduced in the following sections.

1.2 Analytical Systems Based on RMT Algorithm

With the concept of RMT features, researchers could design and leverage the model and algorithms to help decision making systems detect and analyze the pattern from large volumes of data. One major data exploration scenario is to analyze diffusion process of infectious diseases which requires the insights of demographic data, contact networks, age-specific contact rates, mobility networks and health-care and control intervention data and models. I explore how to leverage RMT features and corresponding techniques to build systems to solve real-world problems. I introduce *NOTES2: Networks-of-Traces for Epidemic Spread Simulations* to help with decision making and intervention against infectious diseases which require analysis of large volumes of data including demographic data, contact networks, age-specific contact rates, mobility networks, and health-care and control intervention data and models. More specially, *NOTES2* supports analysis and indexing of simulation data sets as well as parameter and feature analysis, including identification of unknown dependencies across the input parameters and output variables spanning the different layers of the observation and simulation data. It aims at assisting experts and helping them explore existing simulation trace data sets. Figure 1.3 depicts different layers of NOTES2 framework.

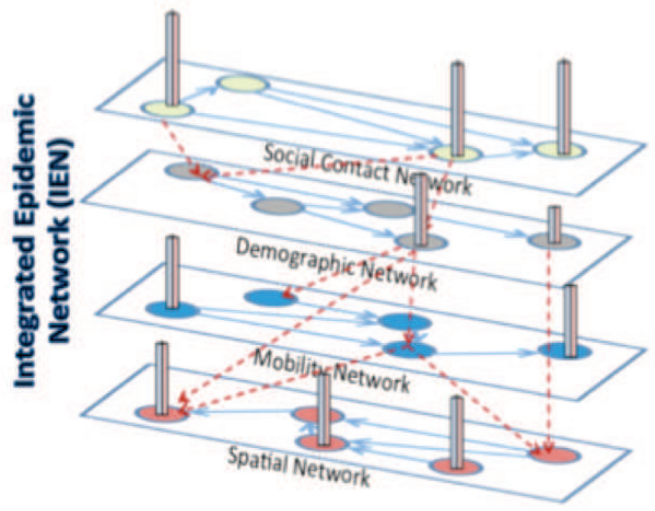


Figure 1.3: Differnt layers from NOTES2 framework

More specifically, the networks-of-traces(NT) data refers to:

- Network Layers: An epidemic simulation requires one or more layers of networks, from local and global mobility patterns to contact networks.

- Disease Models: it describes the epidemiological parameters relevant to a simulation and the parameter dependencies necessary in the computation of the disease spread.

- Simulaiton Traces: For a given disease study, researchers and decision makers often perform multiple simulations, each corresponding to different sets of assumptions (disease parameters or models) or context (e.g. spatiotemporal context, outbreak conditions, interventions).

- Disease Observation Traces: These include real-world observations relating to particular epidemic, including the spread and severity of the disease and observations about other relevant parameters, such as the average length of recovery or percentage of infectious individuals that undergo pharmaceutical treatment.

- External Interventions: In an outbreak, public health and disease control agencies implement various medical or social interventions, quarantines and/or school closures.

Carefully calibrated large-scale computational models of epidemic spread are being increasingly used for generating forecasts of the spatialtemporal progression of epidemics at different spatial scales and assessing the likely impact of different intervention strategies. In such real-life scenarios, experts often lack of a systematic analytical tool to study the pattern and indexing scheme for simulation ensembles, hence I extend *NOTES2* to *EpiDMS: Data Management and Analytics for Decision Making from Epidemic Spread Simulation Ensembles. EpiDMS* is designed to solve the challenges posed by the management and analysis of simulation ensembles stemming from large-scale computational model especially when dealing with multiple inter-dependent parameters, spanning multiple layers and geo-spatial frames, affected by complex dynamic processes operating at different resolutions. Moreover, it helps solve the problems that arise from the need to generate, search, visualize, and analyze in a scalable manner which invovles large volumes of epidemic simulation ensembles and observations during the progression of an epidemic. EpiDMS aims to fill an important gap in decision making during health-care emergencies and enabling critical services with significant economic and health impact.

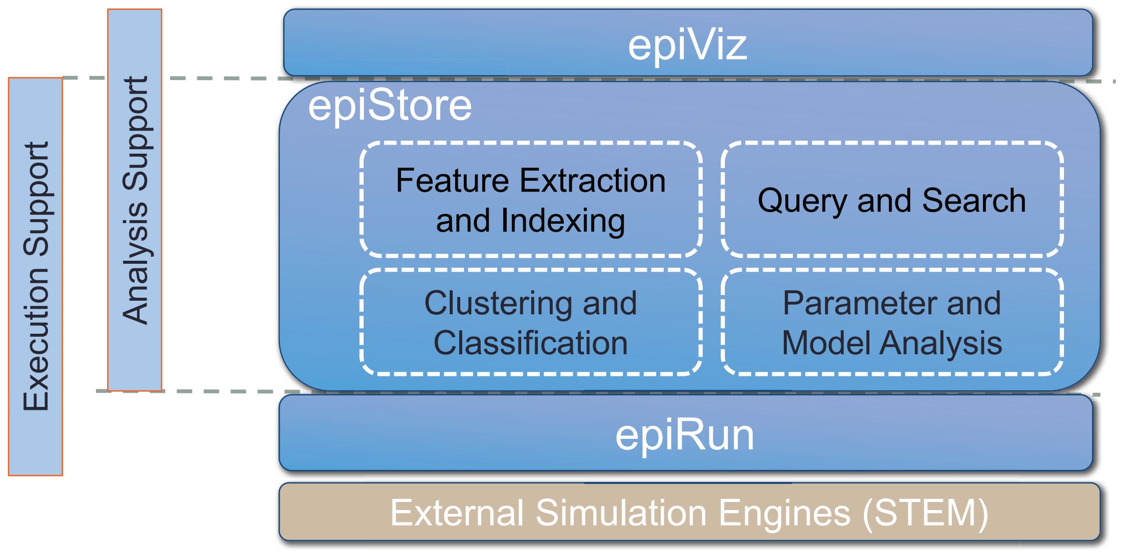

Execution Support
Analysis Support
epiViz
epiStore
Feature Extraction and Indexing
Query and Search
Clustering and Classification
Parameter and Model Analysis
epiRun
External Simulation Engines (STEM)

Figure 1.4: EpiDMS system framework

EpiDMS shown as Figure 1.4 is proposed and designed to solve the challenges posed by the management and analysis of simulation ensembles stemming from large-scale computational model especially when dealing with multiple inter-dependent parameters, spanning multiple layers and geo-spatial frames, affected by complex dynamic processes operating at different resolutions. This problem is compounded by the need to generate near real-time decision-making assessments as the situation in the field changes, which may require the generation of ensembles consisting of 1000s of simulation sets in order to capture a comprehensive range of plausible transmission and control scenarios. EpiDMS also helps solve the problems that arise from the need to generate, search, visualize, and analyze in a scalable manner which invovles large volumes of epidemic simulation ensembles and observations during the progression of an epidemic. This system framework aims to fill an important gap in decision making during health-care emergencies and enabling critical services with significant economic and health impact.

Simulations that are data- and model-driven may track 100s or 1000s of interdependent parameters, spanning multiple layers and spatial-temporal frames, affected by complex dynamic processes operating at different resolutions. Because of the size and complexity of the data and the varying spatial and temporal scales at which the key processes operate, experts often lack the means to analyze results of large simulation ensembles, understand relevant processes, and assess the robustness of conclusions driven from the resulting simulations. Moreover, data and models dynamically evolve over time requiring continuous adaptation of simulation ensembles. To solve these problems, **SIMDMS: Data Management and Analysis to Support Decision Making through Large Simulation Ensembles** is introduced as an extension of EpiDMS: it aims to address the key challenges underlying the creation and use of large simulation ensembles and enables (a) execution, storage, and indexing of large ensemble simulation data sets and the corresponding models; and (b) search, analysis, and exploration of ensemble simulation data sets to enable ensemble-based decision support. *SIMDMS* shown in Figure 1.5, shares similar architecture with *EpiDMS*.

Obtaining and leveraging simulations remains challenging due to parameter track-

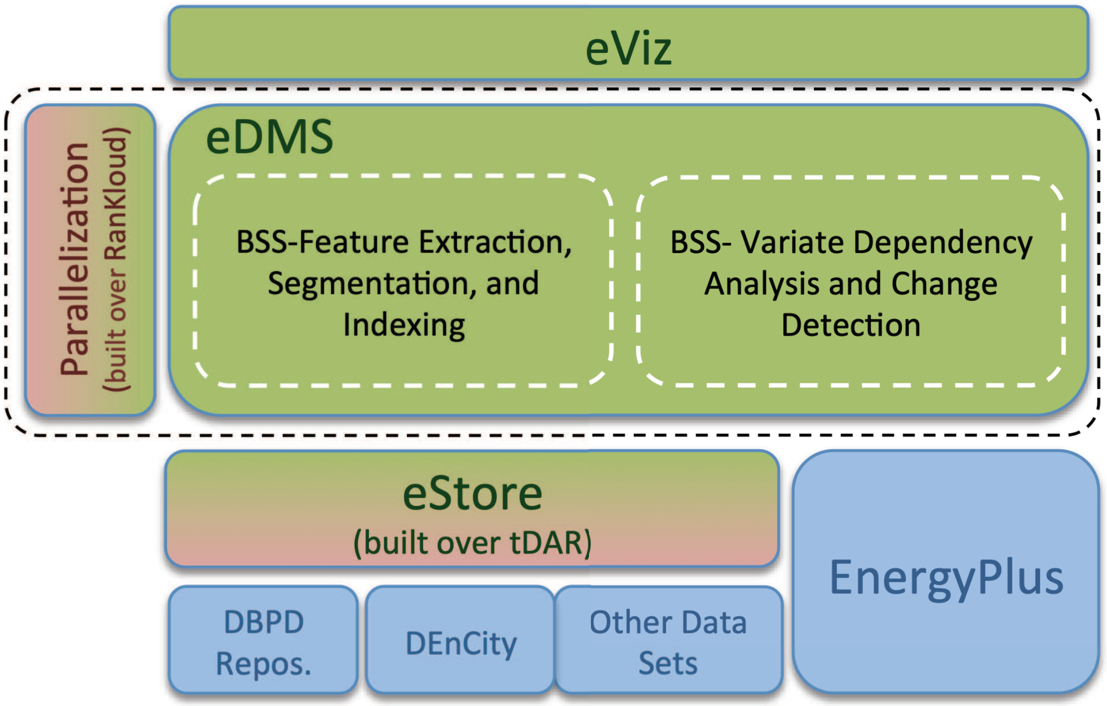


Figure 1.5: SIMDMS system framework

ing and large number of unknowns, hence decision makers usually need to generate ensembles of stochastic realizations, requiring 10s-1000s of individual simulation instances. The situation on the ground evolves unpredictably, requiring continuously adaptive simulation ensembles. To address those challenges from a more general point of view, I and my research team developed *DataStorm-FE: A Data- and Decision-Flow and Coordination Engine for Coupled Simulation Ensembles*, as shown in Figure 1.6, which extends the application scenarios from epidemics to more general cases: natural disasters. *DataStorm-FE* is introduced for creating and maintaining coupled, multi-model simulation ensembles, which also enables end-to-end ensemble planning and optimization, including parameter-space sampling, output aggregation and alignment, state and provenance data management, to improve the overall simulation process. It also aims to work eciently, producing results while working within a limited simulation budget, and incorporates a multivariate, spatiotemporal data browser to empower decision-making based on these improved results.

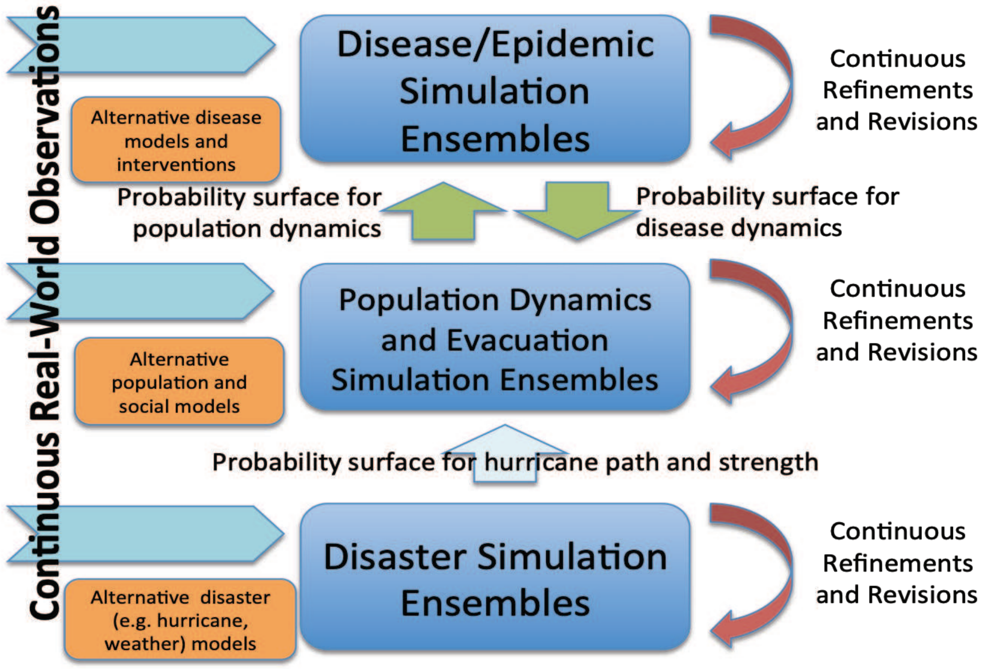

Continuous Real-World Observations
Disease/Epidemic Simulation Ensembles
Alternative disease models and interventions
Continuous Refinements and Revisions
Probability surface for population dynamics
Probability surface for disease dynamics
Population Dynamics and Evacuation Simulation Ensembles
Alternative population and social models
Continuous Refinements and Revisions
Probability surface for hurricane path and strength
Disaster Simulation Ensembles
Alternative disaster (e.g. hurricane, weather) models
Continuous Refinements and Revisions

Figure 1.6: DataStorm system framework and data flow

1.3 Retrieval Tasks in Multi-Dimensional Space

To further explore queries and information retrieval over data in multi-dimensional space, I extend my reserach topic from multi-variate time series data to multi-dimensional data indexing Fagin *et al.* (2001); Heo *et al.* (2010); Gionis *et al.* (1999); Borzsony *et al.* (2001); Papadias *et al.* (2005); Tan *et al.* (2001); Papadias *et al.* (2003) for analysis such as top-k query processing which is critical and remains challenging in many decision support applications under different sceanrios.

1.3.1 Top-K Query Processing in Multi-dimensional Space

My research then extends to the topic of indexing and query processing for data in multi-dimensional space, especially for retrieval tasks such as *top-k* query processing. A naive and straightfoward approach to answer top-k query is to sequantially scan all database objects, compute and combine the scores, sort and output top-k most promising results. An alternative method is to leverage the join operation within relational database management system (RDBMS): map the query into a join query that joins the output from single-attribute queries, and then sorts the joined results based on combined scores. Neither of the two approahces scale well *w.r.t* number of attributes (dimensions) and database size. In this section, I will first discuss the existing optimziations to answer top-k queries then I will briefly introduce my proposed framework.

Existing algorithms, especially within popular RDBMS have made optimziations to process top-k queries from various aspects such as query model design, how to access data, ranking function, etc Ilyas *et al.* (2008). Some popular approahces include:

- Both Sorted and Random Access: In this case, top-k processing strategies assume the availability of both sorted and random access among all data resources. One of the well-adapted algorithms TA Fagin *et al.* (2003). It leverages *bound* information such that it first scans multiple lists representing different rankings of the same set of objects. An upper bound score threshold T is maintained for the overall score of unseen objects, where it is computed by applying the scoring function to the partial scores of the last seen objects in different lists. T is updated together with the scores of seen objects, every time a new object appears in one of the lists. This algorithm terminates when T is smaller than the k-th most promising score from the output. TA algorithm assumes the costs of different access methods are the same and it does not have a restriction on the number of random accesses to be performed, which can potentially be expensive. Alternatively, Fagin *et al.* (2003) discussed a method, Combined Algorithm (CA) that assumes the costs of access methods can be different and comes up with a ratio between the costs of different accesses.

- No Random Access: In this case, top-k processing strategies consider the underlying sources only with sorted access to data objects according to their scores. NRA Fagin *et al.* (2003) discussed above is a typical example, which leverages *bound* information as well to achieve top-k retrieval. which receives a set of sorted lists (each represents sorted attributes) and searches for top-k answers by exploiting only sorted accesses. It may not report the exact object scores since it produces top-k answers using bounds computed over the exact scores.

- Generic (Arbitrary) Ranking Function: In some scenario it is unncessary or sometimes infeasible to define a ranking function. Instead, users can simply establish a relationship which can be arbitrary simply to inform the system which data object is preferrable over the other. For instance, *skyline queries* Borzsony *et al.* (2001) can fall into this category: it returns a set of points that are not dominated by any other point across all dimensions. The *dominance*-relationship can be arbitrary across each and every dimension (data attribute).

- Approximate Query Processing: RDBMS is often used for exact query processing yet with careful design of algorithms and frameworks, the model can be leveraged to answer approximate queries: either as a trade-off in between query accuracy and query processing time, or over incomplete data with a performance guarantee. For instance, Theobald *et al.* (2004) adapts TA algorithm Fagin *et al.* (2003) to fit into the scenario of approximate query processing by associating probabilistic guanratees with top-k answers. Additionally, in the presence of high dimensional data, known as the *curse-of-dimensionality*, retrieval queries such as top-k processing, becomes computationally difficult, also *good enough* top-k result sets are often sufficient instead of exact ones in a timely response manner. Taking these into consideration, Locality Sensitive Hashing (LSH) Indyk and Motwani (1998) is proposed to solve *nearest neighbor*, which can be extended to answer top-k query as well.

Because of the inherent cost of top-k query processing, several indexing, optimization and approximation techniques has been explored. Among them, Locality Sensitive Hashing (LSH) Gionis *et al.* (1999); Gan *et al.* (2012); Datar *et al.* (2004) has been proved to be an efficient and effective tool for indexing and query processing. In real life, however, researchers often face limited computation resources such as RAM, CPU, etc. Furthermore, information such as data distribution and clustering from data itself can provide critical insights of how to leverage limited resources effectively.

To address these challenges discussed above, I introduce a novel indexing structure, **OLSH: Onion-LSH with Layer-Aware Resource Allocation for Approximate Top-K Query Processing** to approximately answer top-k queries in multi-dimensional space. More specifically, the indexing structure relies on the well-known Onion technique Chang *et al.* (2000) to organize the data in layers of convex-hulls to produce top-k results incrementally. Distinctly from the prior work, however, to improve efficiency and reduce data access, data in each layer is stored in Locality Sensitive Hashing (LSH) buckets that enable approximately accurate retrieval of the relevant data elements from each layer in a faster retrieval manner. One key challenge with this approach is that under LSH scheme Neyshabur and Srebro (2015); Shrivastava and Li (2014) there is often a trade-off between available hashing resources and the accuracy. Moreover, the accuracy guarantees that conventional LSH schemes provide are often formulated as a function of the distance from the query in the multi-dimensional space as opposed to the number of top results returned. I complement the OLSH index structure with a layer-aware (LA) resource allocation strategy, which takes into account the distribution of the data and the number of results required to allocate the available hashing resources among the storage layers. More specifically, OLSH is distinct from the prior work such that:

- To improve efficiency and reduce data access, data in each layer is stored in LSH buckets that enable approximately accurate retrieval of the relevant data elements from each layer. One key challenge with this approach is that under LSH there is often a trade-off between available hashing resources and the accuracy.
- The accuracy guarantees that conventional LSH schemes provide are often formulated as a function of the distance from the query in the multi-dimensional space as opposed to the number of top results returned.

As dimensionality goes up, however, convex hull computation of Onion technique becomes computational infeasible. To tackle the case of high dimensionality and potential streaming data and queries with limited hash resources using LSH indexing, I introduce a LSH framework **PLSH**, based on a partition strategy using the l_2 norm of data such that database is pre-processed into different bins. To answer top-k queries, candidates from different bins need to be calcualted and yet data from different bins vary. To quantify this approach, the proposed framework first learn the statistics by leveraging a set of test queries, to see the distribution of ground truth results across different bins; it then learns the distribution to represent the weight or contribution, followed by resource allocation strategy to distribute limited hash resources such that the more contribution a bin has, the more computation resource it should be allocated. In a nutshell, the main contributoin of PLSH are in two fold:

- An efficient partition strategy that helps deal with data in high-dimensional space with performance guarantee
- A data driven statistics learning progress based on queries, to help direct the resource allocation process

1.4 Thesis Outline

The rest of this thesis is organized as follows: Chapter 2 describes related works in the literature. I then discuss the RMT feature extraction framework in Chapter 3, followed by RMT-based frameworks and systesms in Chapter 4. In Chapter 5 and 6 I formulate the tasks of solving top-k retrieval problems in multi-dimensional space using approximate indexing scheme with limited computation resources. Finally I conclude my thesis in Chapter 7.

Chapter 2

BACKGROUND AND RELATED WORK

In this chapter, I will introduce the literature of time series representation and indexing, including the challenges people are facing as well as the background on query processing in high dimensional space, especially for answering top-k queries.

2.1 Time Series Feature Extraction And Similarity Measures

The analysis and exploration of time series often start with extraction of patterns and features that describe salient properties of the data. Popular approaches in the literature include extraction of global features of the time series (such as spectral properties quantified using a transformation; e.g. Discrete Cosine or Wavelet Transforms) and the use of these global features (which describe properties of the time series as a whole) for indexing Candan and Sapino (2010a). In Papadimitriou *et al.* (2005) the authors proposed SPIRIT, this approach discovers patterns from multiple time series streaming. The authors proposed to identify correlations and hidden variables among the multiple time series by applying Principal Component Analysis, in order to summarize the entire set of streams and provide useful means in efficient forecasting. The SPIRIT also satisfies the important requirements of an efficient streaming pattern discovery procedure scaling linearly with the number of time series, adapting to changes and being automatic. In Ji *et al.* (2007) the authors proposed a feature-extraction algorithm that extracts minimal distinguishing sub-sequences that can be used as features. Correlations, transfer functions, variate clusters, and spectral properties De Silva *et al.* (2010).

SVD and similar eigendecompositions can be used for extracting global fingerprints of multi-variate time series data Li *et al.* (2010). The analogous analysis operation on a tensor, which can be used to represent temporal evolution of multi-modal data, is known as tensor decomposition Kolda and Bader (2009). Both matrix and tensor decomposition operations, as well as other techniques, such as probabilistic techniques (such as Dynamic Topic Modeling, DTM Blei and Lafferty (2006)), and AutoRegressive Integrated Moving-Average (ARIMA) based analyses (which separate a time series into autoregressive, moving-average, and integrative components for modeling and forecasting Mills and Mills (1991)) are expensive. Several researchers noticed that significant amount of waste in processing and exploration can be avoided if the attention is directed towards parts of a given time series that are likely to contain interesting patterns Candan *et al.* (2012b); Mohammad and Nishida (2009). One way to achieve this involves searching for frequently repeating patterns; this is commonly known as the *motif search* problem Esling and Agon (2012). Most of the common approaches for motif search involve incrementally moving (or *shift*ing) a fixed-length time window starting from the beginning of the given time series. For each window interval a temporal signature (such as SAX words Lin *et al.* (2003a)) is generated (to speed up the matching of subsequences) and frequent subsequences are discovered using different indexing and hashing algorithms and leveraging pruning techniques for eliminating non-promising subsequences Yankov *et al.* (2007). Other local features of uni-variate time-series include landmarks Perng *et al.* (2000), perceptually important points (PIP) Chung *et al.* (2001), patterns Batal *et al.* (2012), shapelets Rakthanmanon and Keogh (2013); Ye and Keogh (2009), snippets Wang and Candan (2010), longest common subsequences (LCSS Vlachos *et al.* (2006)), and motif-based schemes (which search for frequently repeating temporal patterns) Castro and Azevedo (2010). Morchen in Mörchen (2003) proposed using DFT (Discrete Fourier Transform) and DWT (Discrete Wavelet Transform) for feature extraction because in DWT both temporal properties and frequencies are preserved whereas in DFT only frequency-based aspects are preserved Li *et al.* (2005). In Aggarwal (2002), higher frequency feature generated from DWT illustrates global features whereas lower frequency features denotes local features on temporal axis. PCA-Similarity Factor Krzanowski (1979) and EROS (Extended Frobenius norm) Yang and Shahabi (2004), that use matrix factorization techniques, such as singular vector decomposition (SVD) and principle component analysis to transform the input time series in equi-length time series allowing the usage of cosine similarity over them. Applications of SVD and DTW for various multimedia tasks, such as similarity search, classification, recognition, and watermarking, include Jin and Prabhakaran (2011a); Kim and Prabhakaran (2011b); Li *et al.* (2007a); Mehta *et al.* (2013a). In Ji *et al.* (2007), authors proposed a feature-extraction algorithm that extracts minimal distinguishing subsequences that can be used as features. uni-variate time series often carry localized temporal features which can be used for efficient search and analysis, in Candan *et al.* (2012b), authors developed an sDTW algorithm for extracting salient local features (that are robust against various types of noise) of uni-variate time series and showed that these can help align similar time series more efficiently and effectively. RPM Wang *et al.* (2016a) and STS3 Peng *et al.* (2016a) are two recent approaches that also seek informative patterns from uni-variate time series. In Wang *et al.* (2014c), the authors, starting from the sDTW idea, proposed to extract and use SIFT Lowe (1999a, 2004a)-like robust multi-variate temporal features to determine similarity between multi-variate time series. Assuming that a relation across variates of a multi-variate time series exists, one RMT feature describes one sub-sequence of a multi-variate time series (for both temporal and dependency) allowing fast similarity search.

Various similarity (distance) metrics are proposed and applied afterwards to compute pair-wise similarity (distance) score such that tasks such as pattern matching, top-k retrieval, etc, can be realized. Euclidean distance and, more generally L_p-norm measures, were among the first used to determine the similarity between two time series. They require that the time series being compared are of *same temporal length* and, since they assume strict synchrony among time series, they are not suitable when two time series can have different speeds or are shifted in time Chen (2005); Keogh and Ratanamahatana (2005). Other measures that require equal length and perfect synchrony across time series include *cosine* and (Pearson's) correlation similarity Salton and McGill (1983). In contrast, edit distance Kruskal (1983) measures aim to determine the minimum sequence of *edit* operations that are required to measure similarity. In 70s Sakoe Sakoe and Chiba (1978) and then in mid 90s, Berndt Bemdt and Clifford (1994) proposed dynamic time warping (DTW) technique to find an optimal alignment between two given (time-dependent) sequences under certain restrictions. Intuitively, DTW considers all possible warping paths that can warp (or transform) one series into the other and picks the warping path that has the lowest cost. DTW has found wide acceptance and last two decades have seen several innovations Chen and Ng (2004); Keogh and Ratanamahatana (2005); Ding *et al.* (2008); Keogh (2002); Rakthanmanon *et al.* (2012). For example, while the original DTW is not metric (does not satisfy triangular inequality) Chen and Ng (2004) proposed an extended version of DTW that satisfies triangular inequality. Most of the above algorithms, including DTW, are initially designed for comparing uni-variate time series. More recently, various extensions of DTW have been proposed for *multi*-dimensional time series Sanguansat (2012); Poccia and Garg (2017). The most prevalent of these are the *vectorized* and *independent* extensions. In *vectorized DTW*, a multi-variate time series is considered as a sequence of vectors, where the length of vector is equal to number of variates in the time series. The DTW algorithm is then applied using the distances among these vectors instead of differences in signal amplitude. In *independent DTW*, however, each variate is treated independently from the others and DTW is applied separately to each; finally, these independent DTW distances are added to compute the overall distance between the given pair of multi-variate series.

An alternative approach to the above techniques is to extract *features* from the given time series and use these features to compute similarity/distance instead of the original series. *We provided a detailed discussion of the related work on global and local features in the previous section.* Ji *et al.* (2007), for example, proposed a feature-extraction algorithm that extracts minimal distinguishing subsequences that can be used as features. Morchen in Mörchen (2003) proposed using DFT (Discrete Fourier Transform) and DWT (Discrete Wavelet Transform) for feature extraction. *PCA-Similarity Factor* Krzanowski (1979) and EROS (*Extended Frobenius norm*) Yang and Shahabi (2004), that use matrix factorization techniques, such as singular vector decomposition (SVD) and principle component analysis, have also been proposed to transform the input multi-variate time series into equal length and then apply cosine similarity over them. Applications of SVD and DTW for various multimedia tasks, such as similarity search, classification, recognition, and watermarking, include Jin and Prabhakaran (2011b); Kim and Prabhakaran (2011a); Li *et al.* (2007b); Mehta *et al.* (2013b); Shuai *et al.* (2017). Figure 2.1 shows a classification of all the possible uni-variate representations.

In terms of multi-variate time series, widely adopted approaches often apply feature extraction process first over each and every variate respectively then aggregate points of interest from single variates to arrive at a representation for multi-variate counterparts. This method, however, failed to capture the relationship among different variates to represent more meaningful features which carry (a) scope information

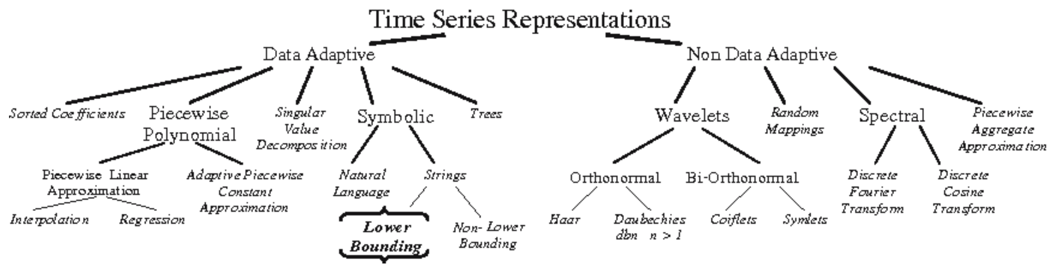


Figure 2.1: A taxonomy of the time series representations Lin *et al.* (2003a)

(b) multiple scales. For instance, as shown in Figure 1.2 users can query events of interest with different scopes and/or scales and to answer this type of queries, novel approaches are in need.

Wang *et al.* (2014b) proposed to extract and use SIFT Lowe (1999b, 2004b)-like robust multi-variate temporal features to determine similarity between time series. I then extend the approach with more general scale-space construction and pruning techniques to obtain better classification performances. The general framework is named RMT for both papers, but the underlying techniques here are significant extensions of Wang *et al.* (2014b): In particular, Wang *et al.* (2014b) utilizes a special case of our proposed generalized scale-space construction and pruning techniques, where only diagonal scale space is considered. In my work, however, I argue (and experimentally show) that, in general, using a more complete scale-space can be more effective. While Wang *et al.* (2014b) considers only one scheme for time series matching using RMT features, my work introduces several alignment measures (including for alignment of features pairs and measuring feature significance) and provides a detailed study of the impact of these measures on the classification accuracies.

2.2 Locality Sensitive Hashing

As we discussed in the introduction, as the data dimensionality gets higher, the performance of multi-dimensional index structures to support nearest neighbor search gets negatively effected. Locality Sensitive Hashing (LSH) presents an alternative that often scales better than traditional index structure, but achieves this performance gain through relaxation of the exact retrieval requirement.

Initially proposed in Indyk and Motwani (1998); Gionis *et al.* (1999), LSH uses hash functions to map high dimensional data to lower dimensional representations such that the similarity relationships among data elements are preserved. Intuitively, data elements that are similar in high dimensional space are likely to be similar to each other in the lower dimensional representations. This is often formulated in the form of an (r, c)-Near Neighbors problem: every point p that lies within a distance of r from query point q should be reported with a probability guarantee of at least $1 - \delta$ (where δ is a user-specified error probability), whereas points lie beyond distance of $c \times r$, for some $c > 1$, from the query point q should have a very low probability of being reported in the result set. The higher the success probability, the higher is the accuracy for the results:

Definition 1 *A hash function family H is said to be (r, c, P_1, P_2)-sensitive if it satisfies all the following conditions for any two points x and y in a data set $D \subset R^d$:*

- *if $|x - y| \leq r$, then $Pr[h(x) = h(y)] \geq P_1$ and*
- *if $|x - y| > cr$, then $Pr[h(x) = h(y)] \leq P_2$.*

Here c is the approximation ratio, P_1 and P_2 are probabilities, typically $c > 1$ and $P_1 > P_2$ to guarantee the hashing scheme to work. In general, the lower dimensional representations are obtained through random projections. For instance, in the original LSH scheme for Euclidean distance, a given vector v is hashed using the function $h_{a,b}(v) = \dfrac{a.v + b}{w}$, where a is a $d-$dimensional random vector with values generated independently from the standard normal distribution and b is a real number chosen uniformly from $[0, w)$, such that w is the width of the hash bucket Datar *et al.*

(2004). Various other LSH hash families Indyk and Motwani (1998); Wang *et al.* (2014a); Neyshabur and Srebro (2015); Shrivastava and Li (2014); Sun *et al.* (2014); Huang *et al.* (2018) have been explored for different distances or similarities, including l_p distance (LSH with $p-$stable distributions).

The LSH scheme indexes all data into hash tables: data points are mapped to buckets based on their hash values and LSH searches for near items via hash table lookups. Such bucketing may, however, lead to misses as well as false positives. Given a distance metric and a corresponding LSH family, LSH data structures control their precision and recall by using multiple independently chosen hash functions organized into several hash layers: intuitively, conjunctively combined hashes within each hash layer reduce false positives where disjunctively combined layers of hash functions help avoid misses. To control false positives, basic LSH concatenates [1] κ hash functions $(h_1(x), \ldots (h_\kappa(x))$ to create a compound hash function $g(x) = (h_1(x), \ldots h_\kappa(x))$ for a single hash layer. The output of this compound hash function identifies a bucket ID in a hash table. As the number of hash functions in layer increases, the false positive rate drops, but (as a side effect) the recall also suffers. In order to improve the recall rate, LSH creates L hash layers (tables) $g_1, g_2, \ldots, g_L$ each consisting of κ hash functions. At each layer, the corresponding hash functions are concatenated to associate a single combined hash value for each data object and L hash tables are constructed to index the buckets corresponding to individual hash layers. During query processing, items lying in (or near) the corresponding L hash buckets are retrieved as potential answers to the given query. Various bucket organization schemes, including query and data distribution aware techniques, have been proposed to help reduce the data redundancy, while providing accurate retrieval Lv *et al.* (2007); Nagarkar and Candan

[1]Note that, traditionally, the number of hash functions in a given layer is denoted using k; however, since in this paper we use k to denote the number of results for the query, we use κ for the number of hashes.

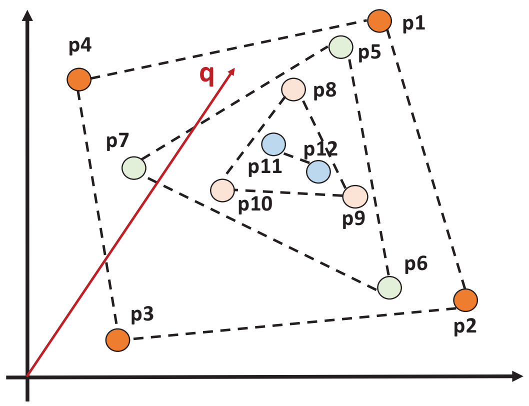


Figure 2.2: Onion layers in 2-dimensions

(2018); Zhang *et al.* (2010); Joly and Buisson (2008); Liu *et al.* (2014); Bawa *et al.* (2005); Gao *et al.* (2015); Gan *et al.* (2012); E2L (1999).

2.3 Layer-based Indexing and Query Processing in Multi-dimensional Space

The skyline Borzsony *et al.* (2001) of a d-dimensional dataset contains the points that are not dominated by any other point on all dimensions. A point dominates another if it is as good or better in all dimensions and better in at least one dimension and the skyline operator is crucial for applications that invovle multi-criteria decision making. There are a few variations of skyline queries Papadias *et al.* (2005), including *skyband query*: similar to K-nearest neighbor queries, a K-skyband query reports the set of points which are dominated by at most K points such that K represents the thickness of a skyline with $K = 0$ corresponding to a conventional skyline.

The Onion technique Chang *et al.* (2000) is a layer-based indexing structure (Figure 2.2) designed to answer top-k quereis. More specifically input data is partitioned into *convex-hull* layers in multi-dimensional space, such that the outermost layer is the convex hull for the entire dataset, the second outermost layer is the convex hull for the dataset *minus* the data in the outermost layer, and so on. Geometrically the partitioned structure looks like an onion, with layers corresponding to peels and it guarantees that the optimal value for any *linear preference* function can always be found at one or more its vertices. The property of outer layers enclosing innter ones geometrically guarantee that given a top-k query, the Onion indexing structure searches for the top result in the outermost layer and searches the 2^{nd} best result in the two outermost layers, up to the k outermost layers combined for the top-k search.

Chapter 3

ROBUST MULTI-VARIATE TEMPORAL (RMT) FEATURES OF MULTI-VARIATE TIME SERIES

3.1 Introduction

In this section, we discuss data models and algorithms to detect local, *robust multi-variate temporal* (RMT) features of multi-variate time series. Recently, in Wang *et al.* (2014b), authors proposed a multi-variate feature extraction technique, which considered the relationships and dependencies between the individual uni-variate time-series that make up the multi-variate series. The local, robust multi-variate temporal (RMT) features are extracted leveraging known correlations and dependencies among the variates. As in Candan *et al.* (2012a), for uni-variate series, Wang *et al.* (2014b) also (a) smoothes the data to generate different version of the input object corresponding to different scales and (b) compares neighboring points both in time (or in x and y dimensions) and scale to identify regions where the gradients are large. As in Lowe (2004b), SIFT-like feature descriptors are extracted to support search.

What makes the problem of extracting local features from multi-variate series difficult, however, is that the concepts of neighborhood, gradient, and smoothing are not well-defined in the presence of multiple variates. In Wang *et al.* (2014b), authors argued that this difficulty can be overcome by leveraging metadata (known correlations and dependencies among the variates in the time series) to define neighborhoods, support data smoothing, and construct scale spaces in which gradients can be measured. Based on this observation, authors proposed topology-sensitive smoothing and topology-sensitive gradient computation techniques to identify local features of multi-variate time series at different time/variate scales. In this section, I show that unlike Wang *et al.* (2014b) (where the time and variate scales are shrank and expanded together), more effective local-feature sets can be located if we allow for the time and variate aspects of the multi-variate time series to be considered independently from each other – leading to multi-variate features with heterogeneous time- and variate-scales.

3.2 Metadata-Enriched Multi-Variate Time Series (MMTS) Model

Before describing the process through which we extract RMT features, we first introduce the metadata-enriched, multi-variate time series model underlying the proposed approach. We present a *metadata-enriched multi-variate time series* (MMTS) model which minimizes the assumptions that need to be made about the data structure:

Definition 2 (Metadata-Enriched Multi-Variate Time Series (MMTS)) *A metadata-enriched multi-variate (MM) time series is a four-tuple* $\mathbf{Y} = (\mathcal{V}, \mathcal{M}, \mathbb{Y}, \mathcal{D})$*,*

- $\mathcal{V}$ *is a set of variates,*
- $\mathcal{M} = \{M_1, \ldots, M_m\}$ *is a set of metadata modalities, where each modality* M_i *describes how the corresponding subset* $V_i \subseteq \mathcal{V}$ *of variates are related to each other,*
- $\mathbb{Y}$ *is a* $(d_1 + d_2 + \ldots + d_m) \times l$ *data matrix, where*
  - l *is the temporal length of the multi-variate time series,*
  - $d_i = |V_i|$*, and*
  - *cells of the matrix* $\mathbb{Y}$ *take values from the data domain* $\mathcal{D}$*.* $\diamond$

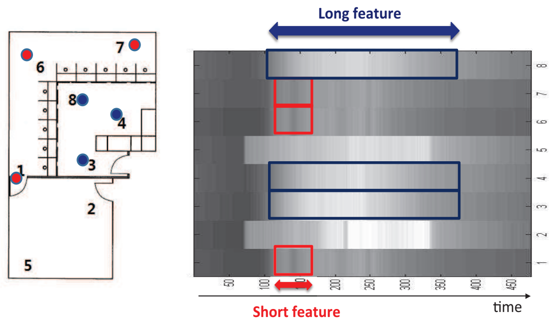
As defined above, each variate is associated with a modality metadata describing how it is related with other variates. In this paper, without loss of generality, we consider *graph-organized* (`G`) representation of variate modalities: Each modality, i, has an associated graph $G_i(V_i, E_i, W_i)$ that relates the variates V_i of the given mode. Depending on the application, the graph maybe directed or undirected and weights may have *distance* or *similarity* semantics. If the underlying graph is unweighted, then for all $e_k \in E_i$, $W_i(e_k) = 1$.

Note that graph-based description of variate relationships is a common way of modeling temporal dynamics of multi-variate time series Eichler (2006); Harvey and Koopman (1997); Silva *et al.* (2010). Note also that, while the meta-data describes the relationship between the variates, this relationship may or may not have causal impact on the observed temporal characteristics of the data:

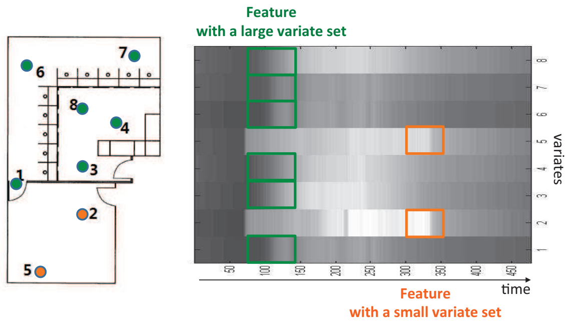
Definition 3 (Metadata-Defined Variate Causality Model) *Let us assume that we have metadata $\mathcal{M}$ that describe the relationship between the variates in the data. Given $\mathcal{M}$, under the variate causality model, we have $\mathbb{Y}[t] = \mathbf{R}_{\mathcal{M}}\mathbb{Y}[t-1] + \vec{E}(t)$, where $\mathbb{Y}[t]$ is a column vector extracted from $\mathbb{Y}$ and corresponds to the observations at time t, $\mathbf{R}_{\mathcal{M}}$ is a (row-normalized) matrix defining how the values of $\mathbb{Y}$ at time $t-1$ impact the values of $\mathbb{Y}$ at time t, and $\vec{E}$ is a multi-variate time series denoting independent, external inputs.* $\diamond$

Intuitively, $\mathbf{R}_{\mathcal{M}}$ is a matrix describing how the values of one variate are impacted by the past values of the variates in the data. Alternatively, $\mathbf{R}_{\mathcal{M}}$ may be a matrix describing the relationships among simultaneous observations:

Definition 4 (Metadata-Defined Variate Correlation Model) *Let us assume that we have metadata $\mathcal{M}$ that describe the relationship between the variates in the data. Given $\mathcal{M}$, under the variate correlation model, we have a matrix $\mathbf{R}_{\mathcal{M}}$ such that*



(a) Different lengths



(b) Different numbers of variates

Figure 3.1: Multi-variate features can be of different sizes (in this example, the multi-variate series represent temperature readings in a floor split into zones)

$\mathbf{R}_{\mathcal{M}}[i,j] = \Phi(\mathbb{Y}[*,i], \mathbb{Y}[*,j]) \in [0,1]$. *Here* $\mathbb{Y}[*,i]$ *and* $\mathbb{Y}[*,j]$ *are rows corresponding to observations for variates* i *and* j*, respectively, and* Φ *is an application specific similarity function.* ◇

Here, Φ may be computed by comparing (recent) historical data of the time series or may reflect available domain knowledge, such as the distance of the sensors recording the variates or known relationships parameters.

It is important to note that the algorithms presented in the paper are applicable under both of the above models [1] and we use the matrix $\mathbf{R}_{\mathcal{M}}$ to denote both relationships.

3.3 Temporal and Variate Smoothing of Multi-Variate Time Series

Let $\mathbf{Y} = (\mathcal{V}, \mathcal{M}, \mathbb{Y}, \mathcal{D})$ be a metadata-enriched multi-variate time series, as defined in previous Section. The first step in identifying multi-variate features of $\mathbf{Y}$ is to generate a scale-space representing versions of the given multi-variate series with different amounts of details. As shown in Figure 3.2, the scale-space, $\mathfrak{Y}$, of $\mathbf{Y}$ is obtained through iterative smoothing across both time and variate relationships,

[1]Thus, without loss of generality, we sometimes focus on the dependency model and, other times, use the correlation model.

starting with an initial smoothing parameter $\Sigma_0 = \langle \sigma_{time,0}, \sigma_{var,0} \rangle$ and iteratively increasing the smoothing degree up to $\Sigma_{max} = \langle \sigma_{time,max}, \sigma_{var,max} \rangle$, obtaining differently smoothed versions of the time series.

The values of Σ_0 and Σ_{max} control the sizes of the smallest and largest features sought in the data. In the rest of this section, we will first describe temporal and variate smoothing techniques. We will then describe optimizations to reduce the cost of the scale-space construction step of the process. For each of the techniques, we will also discuss the relationship among Σ_0, Σ_{max}, and the sizes of the features identified.

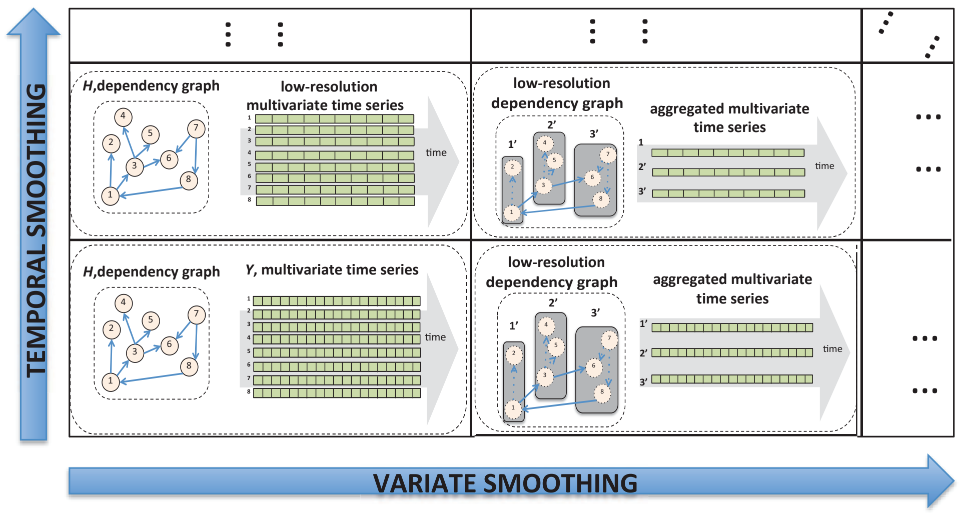

TEMPORAL SMOOTHING
VARIATE SMOOTHING
H,dependency graph
low-resolution multivariate time series
time
low-resolution dependency graph
aggregated multivariate time series
H,dependency graph
Y, multivariate time series
time
low-resolution dependency graph
aggregated multivariate time series
time

Figure 3.2: Scale-space construction through smoothing in time and variates

3.3.1 *Temporal Smoothing*

Let $\mathbf{Y} = (\mathcal{V}, \mathcal{M}, \mathbb{Y}, \mathcal{D})$ be a metadata-enriched multi-variate time series and let $Y_v = \mathbb{Y}[*, v]$ be a uni-variate time series corresponding to one of its variates. Let $Y_v^{(\sigma)}$ indicate a version of the uni-variate time series, Y_v, smoothed through convolution with the Gaussian function, $G(t,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-t^2}{2\sigma^2}}$, with temporal smoothing parameter σ (Figure 3.3). Given this, $\mathbf{Y}^{(time,\sigma)} = (\mathcal{V}, \mathcal{M}, \mathbb{Y}^{(time,\sigma)}, \mathcal{D})$, is a version of the multi-variate time series, $\mathbf{Y}$, where each row of $\mathbb{Y}^{(time,\sigma)}$ is a uni-variate time series smoothed

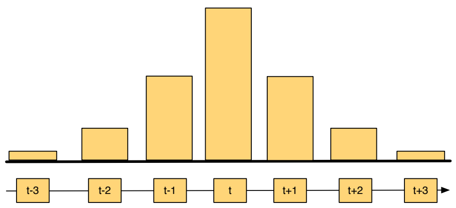


Figure 3.3: Gaussian smoothing of a uni-variate series for time instant, t

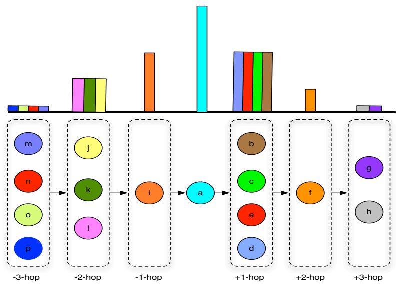


Figure 3.4: Graph smoothing for a node

with temporal smoothing parameter σ, independently of the other uni-variate series.

Temporal Scope: Let us consider a time instant t on which we are applying Gaussian smoothing with parameter σ. Since, under Gaussian smoothing, 3 standard deviations (i.e. 3σ both directions) would cover $\sim 99.73\%$ of the contributions to the smoothed values, we can define the corresponding *temporal scope* as a time interval, centered at t, of length 6σ; in other words, we have $scope_T(t, \sigma) = [t-3\sigma, t+3\sigma)$. Consequently, if the temporal length of a multi-variate time series is L, then we must have $\sigma_{time,max} \leq L/6$. Similarly, since we expect that the smallest feature should involve a time instant and at least its two immediate neighbors, we also have $\sigma_{time,0} \geq 2/6$.

Octaves of Temporal Smoothing: Let σ_1 and σ_2 be two smoothing parameters. The parameter σ_2 is said to be an *octave* larger than σ_1 if $\sigma_2 = 2\sigma_1$. σ_2 defines features twice as large as σ_1 by using a Gaussian smoothing parameter twice as large.

3.3.2 Variate Smoothing

As described above, the temporal smoothing process relies on a convolution operation that leverages the temporal ordering of the time instants in the series. The challenge is that a similar total order does not necessarily exist among the variates – therefore, the definition of variate smoothing is not as straightforward.

Gaussian Smoothing of Graph-Organized Variates: Let $\mathbf{Y} = (\mathcal{V}, \mathcal{M}, \mathbb{Y}, \mathcal{D})$ be a metadata-enriched multi-variate time series, where the metadata $\mathcal{M}$ is graph-structured; i.e., there is a graph $G(V, E, W)$ that relates the variates, V, of the data. Let us further define $frwd_G(v_l, \delta)$ and $bkwd_G(v_l, \delta)$, as the forward and backward neighbors of variate $v_l \in V$ at a distance $\geq (\delta - 0.5)$ and $< (\delta + 0.5)$ on G. Intuitively, $frwd_G()$ and $bkwd_G()$ functions order all the variates into a partial order relative to the variate v_l.

Given the partial order defined by the $frwd_G()$ and $bkwd_G()$ functions and a non-negative smoothing parameter σ, we then obtain the Gaussian smoothed version, $\mathbb{Y}^{(var,\sigma)}$ of the matrix $\mathbb{Y}$ as follows: Let $\mathbb{Y}[t]$ be a column vector extracted from $\mathbb{Y}$ corresponding to the observations at time t. Then, for all v_l, we have $\mathbb{Y}^{(var,\sigma)}[t, l]$ equal to

$$\left(\underset{v_h \in \substack{frwd_G(v_l,0) \\ \cup \\ bkwd_G(v_l,0)}}{AVG} \mathbb{Y}[t,h] \right) + \sum_{\delta=1}^{\infty} G(\delta, \sigma) \left(\underset{v_h \in frwd_G(v_l,\delta)}{AVG} \mathbb{Y}[t,h] \right) + \sum_{\delta=1}^{\infty} G(\delta, \sigma) \left(\underset{v_h \in bkwd_G(v_l,\delta)}{AVG} \mathbb{Y}[t,h] \right).$$

Figure 3.4 shows how we apply Gaussian smoothing over a relationship graph. The lower half of the figure shows a variate a and its forward and backward k-hop neighbors in the relationship graph. As shown in the upper half of the figure, when identifying the contributions of the variate on a, Gaussian smoothing is applied along the hop distance. Since at a given hop distance there may be more than one variate, all the variates at the same distance have the same degree of contribution and the degree of contribution gets progressively smaller as we get away from the variate for which the smoothing is performed.

Variate Scope under Gaussian Smoothing: Similarly to the temporal scope, we define the variate scope corresponding to variate v_l at smoothing level σ as

$$scope_V(v_l, \sigma) = \{v_l\} \cup \left(\bigcup_{\delta \leq 3\sigma} frwd_G(v_l, \delta) \right) \cup \left(\bigcup_{\delta \leq 3\sigma} bkwd_G(v_l, \delta) \right).$$

The variate smoothing parameters, $\sigma_{var,0}$ and $\sigma_{var,max}$, must be selected such that for each variate v_l, $\sigma_{var,0}$ includes its immediate (one hop) graph neighbors, $forward\_neighbors(v_l)$ and $backward\_neighbors(v_l)$ on G, and the value of δ corresponding to $\sigma_{var,max}$ should be compatible with the diameter of the graph G.

Octaves of Gaussian Variate Smoothing: Let σ_1 and σ_2 be two Gaussian graph smoothing parameters. Under Gaussian smoothing, the graph smoothing parameter σ_2 is said to be an *octave* larger than σ_1 if $\sigma_2 = 2\sigma_1$.

3.3.3 Combined Time and Variate Smoothing

Given the definitions of temporal and variate smoothing functions, we now define combined time and variate smoothing of metadata-enriched multi-variate time series:

Definition 5 (TV-Smoothing of a Multi-Variate Time Series) *Let* $\mathbf{Y} = (\mathcal{V}, \mathcal{M}, \mathbb{Y}, \mathcal{D})$ *be a metadata-enriched multi-variate (MM) time series. Recall that* $\mathbb{Y}$ *is a* $(d_1 + d_2 + \ldots + d_m) \times l$ *data matrix, where* l *is the temporal length of the multi-variate time series,* $d_i = |V_i|$*, and* $\mathbb{Y}$ *takes values from the data domain* $\mathcal{D}$*. For a given smoothing parameter,* $\Sigma = \langle \sigma_{time}, \sigma_{var} \rangle$*, the TV-smoothed version,* $\mathbf{Y}\{\Sigma\}$*, of the multi-variate time series,* $\mathbf{Y}$*, is defined as* $\mathbf{Y}\{\Sigma\} = \left(\mathbb{Y}^{(time, \sigma_{time})} \right)^{(var, \sigma_{var})}$,

- $\mathbb{Y}^{(time, \sigma_{time})}$ *is a version of* $\mathbb{Y}$ *where each row (i.e., each uni-variate time series) is temporally smoothed with smoothing parameter* σ_{time}*, independently from the rest; and*

- $\mathbb{X}^{(var, \sigma_{var})}$ *is a version of* $\mathbb{X}$ *where each column (i.e., time instant) is smoothed with smoothing parameter* σ_{var}*, using the variable relationships and modalities described by the metadata* $\mathcal{M}$*.* $\diamond$

3.4 Step 1: Scale-space Construction for Multi-Variate Time Series

As we have seen in Section 3.3, given a metadata-enriched multi-variate time series, $\mathbf{Y} = (\mathcal{V}, \mathcal{M}, \mathbb{Y}, \mathcal{D})$, first step in identifying multi-variate features of $\mathbf{Y}$ is to generate a scale-space representing versions of the multi-variate series with different amounts of details. In this paper, we consider two types of scale-spaces: *diagonal* and *full* scale-spaces, described below.

Diagonal Scale-Spaces: Let $\Sigma_0 = \langle \sigma_{time,0}, \sigma_{var,0} \rangle$ be the user provided smallest temporal and variate smoothing parameters and let l indicate the total number of layers in the scale space. An l-layer *diagonal* state space, $\mathfrak{Y}_{diag}$, is defined as a set of data matrices $\{\mathbb{Y}_0, \ldots, \mathbb{Y}_l\}$, where $\mathbb{Y}_i = \mathbb{Y}\{\langle \sigma_{time,0} \times k^i, \sigma_{var,0} \times k^i \rangle\}$, for some scaling parameter $k > 1$. Note that, in this case, we have $\sigma_{time,max} = \sigma_{time,0} \times k^l$ and $\sigma_{var,max} = \sigma_{var,0} \times k^l$. This will generate only the diagonal entries in the scale-space shown in Figure 3.2.

Full Scale-Spaces: In contrast, the complete scale-space shown in Figure 3.2 is generated as follows: Let

- $\Sigma_0 = \langle \sigma_{time,0}, \sigma_{var,0} \rangle$ be the smallest temporal and variate smoothing parameters,
- $\mathcal{L} = \langle l_{time}, l_{var} \rangle$ indicate the number of temporal and variate smoothing layers, and
- $\mathcal{K} = \langle k_{time,}, k_{var} \rangle$ be scaling parameters for temporal and variate smoothings.

An $\mathcal{L}$-layer *full* state space, $\mathfrak{Y}_{full}$, is defined as a set of data matrices $\langle \mathbb{Y}_{0,0}, \ldots, \mathbb{Y}_{i,j}, \ldots, \mathbb{Y}_{l_{time},l_{var}} \rangle$, where $\mathbb{Y}_{i,j} = \mathbb{Y}\{\langle \sigma_{time,0} \times k^i_{time}, \sigma_{var,0} \times k^j_{var} \rangle\}$. In this case, we have $\sigma_{time,max} = \sigma_{time,0} \times k^{l_{time}}_{time}$ and $\sigma_{var,max} = \sigma_{var,0} \times k^{l_{var}}_{var}$.

Optimization: Time and Variate Subsampling: In the process described above, the multi-variate time series is incrementally smoothed both in time and relationships, halving details at each octave. We note that, once the details have been halved at an octave boundary, performing the feature extraction operation at the same level detail is going to be wasteful. To avoid such waste, we subsample the multi-variate time series at octave boundaries (Figure 3.2). More specifically, at temporal octave boundaries (where temporal details have been halved) we drop one out of every two consecutive temporal observations, reducing the size of the data by half. Similarly, at variate octave boundaries (where variate relationship details have been halved) we reduce the numbers of variates by half by applying a variate clustering algorithm [2] .

3.5 Step 2: Identifying Multi-Variate Temporal Feature Candidates

Building on the observation Lowe (2004a); Candan *et al.* (2012b) that robust localized features are often located where the differences between neighboring regions (possibly in different scales) are large, we seek RMT features of the given multi-variate time series at the *local extrema* of the scale space defined by the difference-of-smoothing (DoS) series. Naturally, the DoS generation and feature identification process will be slightly different depending on whether a diagonal or full scale-space is used.

Local Extrema in Diagonal Scale-Spaces: An l-layer *diagonal* state space of a metadata-enriched multi-variate time series, $\mathbf{Y} = (\mathcal{V}, \mathcal{M}, \mathbb{Y}, \mathcal{D})$, is defined as a set of data matrices $\{\mathbb{Y}_0, \ldots, \mathbb{Y}_l\}$, where $\mathbb{Y}_i = \mathbb{Y}\{\langle \sigma_{time,0} \times k^i, \sigma_{var,0} \times k^i \rangle\}$, for some scaling parameter $k > 1$. Given this, we create the corresponding DoS by considering a sequence of difference matrices $\{\mathbb{D}_0, \ldots, \mathbb{D}_{l-1}\}$, where $\mathbb{D}_i = |\mathbb{Y}_{i+1} - \mathbb{Y}_i|$. We detect RMT feature candidates by seeking the local maxima and minima of the resulting

[2]In the experiments reported in During experimentation, I use a k-means algorithm, where k is equal to the half of the number of variates, based on the distances among sensors on the underlying sensor-distance graph

DoS: each variate-time-scale (VTS) triple, $\langle v, t, s\rangle$, is compared to its neighbors (both in time and variate relationships) in the same scale as well as the scales above and below, and the triple is selected as a candidate only if it is close to being an extremum; i.e., each $\langle v, t, s\rangle$ is compared against its 26 ($= 3^3 - 1$) *neighbors* in time, scale, and variate relationships [3] . More specifically, for each $\langle v, t, s\rangle$, we compare $\mathbb{D}_s[v, t]$ against

$$max \left\{ \begin{array}{ccc} \mathbb{D}_{s-1}[v,t-1] & \mathbb{D}_{s}[v,t-1] & \mathbb{D}_{s+1}[v,t-1] \\ \mathbb{D}_{s-1}[v,t] & & \mathbb{D}_{s+1}[v,t] \\ \mathbb{D}_{s-1}[v,t+1] & \mathbb{D}_{s}[v,t+1] & \mathbb{D}_{s+1}[v,t+1] \\ \mathbb{FD}_{s-1}[v,t-1] & \mathbb{FD}_{s}[v,t-1] & \mathbb{FD}_{s+1}[v,t-1] \\ \mathbb{FD}_{s-1}[v,t] & \mathbb{FD}_{s}[v,t] & \mathbb{FD}_{s+1}[v,t] \\ \mathbb{FD}_{s-1}[v,t+1] & \mathbb{FD}_{s}[v,t+1] & \mathbb{FD}_{s+1}[v,t+1] \\ \mathbb{BD}_{s-1}[v,t-1] & \mathbb{BD}_{s}[v,t-1] & \mathbb{BD}_{s+1}[v,t-1] \\ \mathbb{BD}_{s-1}[v,t] & \mathbb{BD}_{s}[v,t] & \mathbb{BD}_{s+1}[v,t] \\ \mathbb{BD}_{s-1}[v,t+1] & \mathbb{BD}_{s}[v,t+1] & \mathbb{BD}_{s+1}[v,t+1] \end{array} \right\},$$

where $\mathbb{FD}_s[v, t] = \Big(F\mathbb{D}_s\Big)[v, t]$, $\mathbb{BD}_s[v, t] = \Big(B\mathbb{D}_s\Big)[v, t]$, and F and B are two matrices describing forward and backward relationships among variates. Intuitively, $\mathbb{FD}$ accounts for the combined DoS values of the forward neighbors and $\mathbb{BD}$ accounts for the combined DoS values of the backward neighbors of v. We declare the triple, $\langle v, t, s\rangle$, a candidate if the corresponding DoS value, $\mathbb{D}_s[v, t]$, is greater than Θ% of the maximum of its 26 scale-neighbors in DoS, for some user provided $\Theta \sim 100$.

Local Extrema in Full Scale-Spaces: As seen earlier, an $\mathcal{L}$-layer *full* state space, $\mathfrak{Y}_{full}$, is defined as a set of data matrices $\langle \mathbb{Y}_{0,0}, \ldots, \mathbb{Y}_{i,j}, \ldots, \mathbb{Y}_{l_{time},l_{var}}\rangle$, where $\mathbb{Y}_{i,j} = \mathbb{Y}\{\langle \sigma_{time,0} \times k^i_{time}, \sigma_{var,0} \times k^j_{var}\rangle\}$. Given this, for each $s = \langle i, j\rangle$ pair, we can define three differences:

$$\mathbb{D}^t_{i,j} = |\mathbb{Y}_{i+1,j} - \mathbb{Y}_{i,j}|\,, \;\; \mathbb{D}^v_{i,j} = |\mathbb{Y}_{i,j+1} - \mathbb{Y}_{i,j}|\,, \;\; \text{and} \;\; \mathbb{D}^{t,v}_{i,j} = |\mathbb{Y}_{i+1,j+1} - \mathbb{Y}_{i,j}|\,.$$

[3]The number of neighboring triples may be less than 26 if the triple is at the boundary in terms of time, scale, or variate relationship graph.

Local extrema are then identified by considering each variate-time-scale (VTS) triple, $\langle v, t, s\rangle$, and comparing $max(\mathbb{D}^t_{i,j}, \mathbb{D}^v_{i,j}, \mathbb{D}^{t,v}_{i,j})$ to 78 ($= 3 \times 26$) neighboring triples [4] of $\langle v, t, s\rangle$ in time, scale, and relationships for each of the $\mathbb{D}^t$, $\mathbb{D}^v$, and $\mathbb{D}^{t,v}$. We finally declare the triple, $\langle v, t, s\rangle$, a candidate if the corresponding DoS value, $\mathbb{D}_s[v, t]$, is greater than Θ% of the maximum of its 78 neighbors in DoS.

3.6 Step 3: Eliminating Poor RMT Feature Candidates

: Local extrema of DoS can include candidate triples that are poorly localized. In order to identify whether a triple $\langle v, t, s\rangle$ is well or poorly localized in the scale-space, we can consider the principal curvatures at the point $\langle v, t, s\rangle$ of the scale-space generated earlier: a poorly defined peek in the difference-in-smoothing will have a large principal curvature in the scale space in one direction, but a small one in the perpendicular direction. Consequently, as was observed in Harris and Stephens (1988); Lowe (2004a), we can search for well-localized candidates by considering the ratio of the eigenvalues of the 2×2 Hessian matrix, which describes the local curvature of the scale-space in terms of the second-order partial derivatives.

Given the above observation, the major challenge, in this case, is to define and compute the partial derivatives for metadata enhanced multi-variate time series to obtain the Hessian matrix we seek. More specifically, for each VTS triple, $\langle v, t, s\rangle$, we need to construct a 2×2 time/variates Hessian matrix, $\mathfrak{D}^{TV}_{v,t,s} = \begin{bmatrix} D_{T,T} & D_{T,V} \\ D_{V,T} & D_{V,V} \end{bmatrix}$,

- $D_{T,T} = D_T D_T$ is the second derivate along time for the triple $\langle v, t, s\rangle$,
- $D_{V,V} = D_V D_V$ is the second derivative along "variate relationships" for $\langle v, t, s\rangle$,
- $D_{T,V} = D_T D_V$ is the partial derivative along time of the partial derivate along

[4] The number of neighboring triples may be less than 78 if the triple is at the boundary in terms of time, scale, or variate relationship graph.

variate relationships of the triple $\langle v, t, s \rangle$, and

- $D_{V,T} = D_V D_T$ is the partial derivative along variate relationships of the partial derivate along time of the triple $\langle v, t, s \rangle$.

In this paper, we propose to estimate the derivatives along time and variate relationships by taking differences of neighboring sample points:

$$
\begin{aligned}
D_T(v,t,s) &= \mathbb{Y}_s[v,t+1] - \mathbb{Y}_s[v,t-1], \\
D_V(v,t,s) &= \begin{cases} (\mathbb{F}\mathbb{Y}_s[v,t] - \mathbb{B}\mathbb{Y}_s[v,t]) & \text{for directed relationships} \\ (\mathbb{F}\mathbb{Y}_s[v,t] - \mathbb{Y}_s[v,t]) & \text{for undirected relationships} \end{cases}
\end{aligned}
$$

Here, $\mathbb{F}\mathbb{Y}_s$ and $\mathbb{B}\mathbb{Y}_s$, account for the (weighted) averages of the forward and backward variate neighbors at the corresponding scale: i.e., $\mathbb{F}\mathbb{Y}_s[v,t] = \Big(F\mathbb{Y}_s\Big)[v,t]$ and $\mathbb{B}\mathbb{Y}_s[v,t] = \Big(F\mathbb{Y}_s\Big)[v,t]$, where (as was discussed in the previous section) F and B are two matrices describing forward and backward relationships among variates.

Once the Hessian matrix, $\mathfrak{D}^{TV}_{v,t,s}$, is constructed for the triple $\langle v, t, s \rangle$, whether the triple is poorly localized can be checked using eigenvalue-based techniques Harris and Stephens (1988); Lowe (2004a). Note that derivatives (with respect to time) will be high at the boundaries of time (i.e., the beginning and end of the time series). Similarly, in directed variate relationship graphs, source and sink nodes are likely to have large derivatives with respect to the relationship space. Since many of these triples at the boundary of time and relationship do not correspond to real features of the data, but are essentially boundary *noise*s, such candidate triples are removed even if they are well-localized.

3.7 Step 4: RMT Feature Descriptor Creation

For data objects that can be represented as 2D matrices (such as images), Lowe (2004a) proposed that a gradient histogram based descriptor around the given point

$\langle x, y \rangle$ on the matrix could be constructed by computing a gradient for each element in the neighborhood of the point Lowe (2004a). The resulting gradients are then quantized into c orientations. Finally a $2a \times 2b$ grid is superimposed on the neighborhood region centered around the point and the gradients for the elements that fall into each cell are aggregated into a c-bin gradient histogram. This process leads to a feature descriptor vector of length $2a \times 2b \times c$. In Candan *et al.* (2012b), we have shown that gradient histograms (created from data vectors instead of data matrices) are also effective in describing temporal features of uni-variate time series. In the case of multi-variate time series, however, we cannot directly apply the above techniques. Instead, we first need to construct an *extractor matrix* to enable the gradient extraction process.

Extractor Matrix: Let $\mathfrak{Y}$ be a scale space defined over the given metadata-enriched multi-variate time series and the VTS triple, $\langle v, t, s \rangle$, be an RMT feature identified from $\mathfrak{Y}$. The multi-variate feature defined by a variate-time-scale triple, $\langle v, t, s \rangle$, has an associated scope, defined by the scale, s, in which it is identified. The pair, $\langle v, t \rangle$, forms the center of the feature in time and variates. Given this feature center, under scale, s, which corresponds to temporal and variate smoothing parameter pair, $\Sigma = \langle \sigma_{time}, \sigma_{var} \rangle$, the temporal and variate scopes of the feature are computed respectively.

As we have also seen in Section 3.3, observations closer in time and relationships to the triple will have significantly larger contributions to the feature than the points closer to the boundaries of the scope. Therefore, to identify gradients across time and variate relationships, we first construct an N-step aggregation series:

Definition 6 (N-Step Aggregation Series) *For directed variate relationships, we define the N-step aggregation series corresponding to scale s as follows:*

For $-N < a \leq N$,

$$W_s[a] \quad = \quad \begin{cases} \text{if } a > 0 & \Big(F^a \mathbb{Y}_s\Big) \\ \text{if } a = 0 & \mathbb{Y}_s \\ \text{if } a < 0 & \Big(B^a \mathbb{Y}_s\Big), \end{cases}$$

where, as before, F and B are two matrices describing forward and backward relationships among variates. Similarly, in the case of undirected variate relationships, we define the N-step aggregation series, such that for $0 \leq a \leq N$ we have

$$W_s[a] \quad = \quad \begin{cases} \text{if } a > 0 & \Big(F^a \mathbb{Y}_s\Big) \\ \text{if } a = 0 & \mathbb{Y}_s. \end{cases} \qquad \diamond$$

Once the N-step aggregation series are obtained, we can then construct the extractor matrices from which the feature descriptors will be obtained:

Definition 7 (Extractor Matrix) *Let $\langle v, t, s\rangle$ be a VTS triple on the scale space. In the case of directed variate relationships, we define the corresponding extractor matrix as a $2N \times 2M$ matrix, $X_{v,t,s}$, such that for $-N < a \leq N$ and $-M < b \leq M$, we have $X_{v,t,s}[a,b] = \Big(W_s[a]\Big)[v, t+b]$. In the case of undirected variate relationships, we define the extractor matrix as a $(N+1) \times 2M$ matrix, $X_{v,t,s}$, such that for $-N < a \leq N$ and $0 \leq b \leq M$, we have $X_{v,t,s}[a,b] = \Big(W_s[a]\Big)[v, t+b]$.* $\diamond$

The values of N and M should be selected to cover the scope of the feature.

Descriptor Extraction Given this extractor matrix, $X_{v,t,s}$, the feature descriptor is created as a c-directional gradient histogram of this matrix, sampling the gradient magnitudes around the salient point using a $2a \times 2b$ grid (or $2a \times b$ grid for undirected relationship graphs) superimposed on the matrix, $X_{v,t,s}$. To give less emphasis to gradients that are far from the point $\langle v, t\rangle$, a Gaussian weighting function is used to reduce the magnitude of elements further from $\langle v, t\rangle$.

This process leads to a feature descriptor vector of length $2a \times 2b \times c$ (or $2a \times b \times c$ for undirected graphs). The descriptor size must be selected in a way that reflects the temporal characteristics of the time series; if a multi-variate time series contains many similar features, it might be more advantageous to use large descriptors that can better discriminate: these large descriptors would not only include information that describe the corresponding features, but would also describe the temporal contexts in which these features are located.

3.8 RMT Feature Set of a Multi-Variate Time Series

Given the above, the RMT features of a metadata-enriched multi-variate (MM) time series, $\mathbf{Y} = (\mathcal{V}, \mathcal{M}, \mathbb{Y}, \mathcal{D})$, with respect to the parameters

- $\Sigma_0 = \langle \sigma_{time,0}, \sigma_{var,0} \rangle$; i.e., the smallest temporal and variate smoothing parameters,
- $\mathcal{L} = \langle l_{time}, l_{var} \rangle$; i.e., the number of temporal and variate smoothing layers, and
- $\mathcal{K} = \langle k_{time,}, k_{var} \rangle$; i.e., the scaling parameters for temporal and variate smoothings,

is defined as a set, $\mathcal{F}$, where each feature, $f \in \mathcal{F}$, extracted from $\mathbf{Y}$, is a pair of the form, $f = \langle pos, \vec{d} \rangle$:

- $pos = \langle v, t, s \rangle$ is a VTS triple denoting the position of the feature in the scale-space of the multi-variate time series, where v is the index of the variate at which the feature is *centered*, t is the time instant around which the duration of the feature is *centered*, and s is the temporal/variate smoothing scale in which the feature is identified. Note that this triple also defines the *temporal and variates scopes* of the RMT feature.
- $\vec{d}$ is a vector of length $2a \times 2b \times c$ for directed relationship graphs and $2a \times b \times c$ for undirected graphs, as described in the previous section.

Note that this set contains RMT features of potentially different sizes. In particular, we have $\sigma_{time,min} = \sigma_{time,0}$, $\sigma_{var,min} = \sigma_{var,0}$, $\sigma_{time,max} = \sigma_{time,0} \times k_{time}^{l_{time}}$, $\sigma_{var,max} = \sigma_{var,0} \times k_{var}^{l_{var}}$, and these define the minimum and maximum temporal and variate scopes of the features identified from the given multi-variate time series.

3.9 Time Series Matching using RMT Features

This feature set can be used for various applications, including alignment, indexing, and classification of multi-variate series. Let us be given two metadata-enriched multi-variate (MM) time series, $\mathbf{Y}_1$ and $\mathbf{Y}_2$, and their feature sets $\mathcal{F}_1$ and $\mathcal{F}_2$. We rely on the alignments of the feature pairs in $\mathcal{F}_1$ and $\mathcal{F}_2$ to measure how well these two series match each other.

3.9.1 Alignment of Feature Pairs

Let $f_1 = \langle\langle v_1, t_1, s_1\rangle, \vec{d_1}\rangle$ and $f_2 = \langle\langle v_2, t_2, s_2\rangle, \vec{d_2}\rangle$ be two RMT features in $\mathcal{F}_1$ and $\mathcal{F}_2$, respectively. When matching f_1 and f_2, we consider how well aligned as well as how important these two features are.

Temporal Alignment of a Pair of Features

Two features are said to be temporally aligned if their temporal scopes overlap significantly and temporal centers are close.

Definition 8 (Temporal Overlap) *Let $[ts_1, te_1)$ denote the temporal scope of the first feature defined by t_1 and the temporal smoothing parameter corresponding to the feature scale s_1. Similarly, let $[ts_2, te_2)$ denote the temporal scope of the first feature defined by t_2 and the feature scale s_2. We define the temporal overlap score of the two features as $Overlap_T(f_1, f_2) = \frac{min(te_1,te_2)-max(ts_1,ts_2)}{max(te_1,te_2)-min(ts_1,ts_2)}$.* ◇

Definition 9 (Temporal Center Proximity) *We define temporal proximity score as* $Prox_T(f_1, f_2) = 1 - \frac{|t_1 - t_2|}{maxLength}$*, where* $maxLength$ *is the length of time series.* $\diamond$

Given these, we define temporal alignment score as follows:

Definition 10 (Temporal Alignment) *We define the temporal alignment score of the two features as* $Align_T(f_1, f_2) = \frac{Overlap_T(f_1,f_2)+Prox_T(f_1,f_2)}{2}$. $\diamond$

Variate Alignment of a Pair of Features

Two features are said to be variate aligned if their variate scopes overlap significantly:

Definition 11 (Variate Alignment) *Let* $scope(v_1, \sigma_{var,1})$ *denote the variate scope of the first feature defined by parameter* $\sigma_{var,1}$ *corresponding to feature scale* s_1*. Similarly, let* $scope(v_2, \sigma_{var,2})$ *be the variate scope of the second feature. We define the variate alignment score of the two features as* $Align_V(f_1, f_2) = \frac{scope(v_1,\sigma_{var,1}) \cap scope(v_2,\sigma_{var,2})}{scope(v_1,\sigma_{var,1}) \cup scope(v_2,\sigma_{var,2})}$.

$\diamond$

Descriptor Alignment of a Pair of Features

Two features are said to be descriptor aligned if their descriptor vectors are similar to each other:

Definition 12 (Descriptor Alignment) *We define the descriptor alignment score of the two features as* $Align_D(f_1, f_2) = sim(\vec{d_1}, \vec{d_2})$ *or as* $Align_D(f_1, f_2) = \Big(1 + \Delta(\vec{d_1}, \vec{d_2})\Big)^{-1}$ *for a given similarity, sim(), or distance,* $\Delta()$*, function.* $\diamond$

Amplitude Alignment of a Pair of Features

Two features are said to be amplitude aligned if the average amplitudes of the time series within the corresponding feature scopes are similar to each other:

Definition 13 (Amplitude Alignment) *We define the amplitude alignment score as* $Align_A(f_1, f_2) = \Big(1 + |ampl_1 - ampl_2|\Big)^{-1}$, *where* $ampl_1$ *and* $ampl_2$ *are the average amplitudes of the time series,* $\mathbf{Y}_1$ *and* $\mathbf{Y}_2$*, within the scopes of* f_1 *and* f_2*.* $\diamond$

3.9.2 Feature Significance

Scope Significance of a Given Pair of Features

The size of temporal and variate scopes may impact the significance of a feature.

Definition 14 (Temporal Scope Significance) *The combined temporal scope significance of* f_1 *and* f_2 *is defined as* $Sig_T(f_1, f_2) = \frac{\sigma_{time,1} + \sigma_{time,2}}{2}$, *where* $\sigma_{time,1}$ *and* $\sigma_{time,2}$ *are the two temporal smoothing parameters corresponding to temporal scales,* s_1 *and* s_2*.* $\diamond$

Definition 15 (Variate Scope Significance) *The combined variate scope significance of* f_1 *and* f_2 *is defined as* $Sig_V(f_1, f_2) = \frac{\sigma_{var,1} + \sigma_{var,2}}{2}$, *where* $\sigma_{var,1}$ *and* $\sigma_{var,2}$ *are the two variate smoothing parameters corresponding to feature scales,* s_1 *and* s_2*.* $\diamond$

3.9.3 Overall Feature Matching Score

Given the above, the overall matching score of two features is a combination of the individual measures of alignment and importance:

Definition 16 (Overall Feature Matching Score) *We define the overall matching score,* $match(f_1, f_2)$*, of the two features as*

$$\mu\left(\begin{array}{l} Align_D(f_1, f_2), Align_T(f_1, f_2), Align_V(f_1, f_2), Align_A(f_1, f_2), \\ Sig_T(f_1, f_2), Sig_V(f_1, f_2), Sig_C(f_1, f_2) \end{array}\right),$$

where μ *is a merge function that combines the individual scores.* $\diamond$

While there exist different merge functions (such as *min*, *max*, *avg*, *product*), in the experiments reported we use *product*, which approximates the boolean operator *and* when individual scores are zeros and ones Candan and Sapino (2010b).

3.9.4 Identifying Candidate Matching Pairs

Given a query time series, $\mathbf{Y}_q$, and a data series, $\mathbf{Y}_d$, and their feature sets $\mathcal{F}_q$ and $\mathcal{F}_d$, the next step is to identify a set $\mathcal{P} \subseteq \mathcal{F}_q \times \mathcal{F}_d$ of candidate feature pairs

- $\forall f_{q,i} \in \mathcal{F}_q \ \exists f_{d,j} \in \mathcal{F}_d \ \ s.t. \ \langle f_{q,i}, f_{d,j} \rangle \in \mathcal{P}$ (i.e., for each query RMT feature on the query object, at least one matching feature on the data object is located),
- $\forall \langle f_{q,i}, f_{d,j} \rangle, \langle f_{q,h}, f_{d,k} \rangle \in \mathcal{P} \ \ (f_{q,i} = f_{q,h}) \rightarrow (f_{d,j} = f_{d,k})$ (i.e., for each query RMT feature on the query object, at most one matching feature on the data object is located), and
- $\sum_{\langle f_{q,i}, f_{d,j} \rangle \in \mathcal{P}} match(f_{q,i}, f_{d,j})$ is maximized.

It is easy to see that, for each query feature, $\mathcal{P}$ contains one and only one matching data feature and since we aim to maximize the overall matching score, the set $\mathcal{P}$ can be obtained by considering each feature $f_{q,i} \in \mathcal{F}_q$ and selecting the feature $f_{d,j} \in \mathcal{F}_d$ with the maximum $match(f_{q,i}, f_{d,j})$ value. The feature pairs in $\mathcal{P}$ obtained this way may not be mutually consistent and such inconsistencies need to be eliminated.

3.9.5 Inconsistency Pruning of Candidate Pairs

Intuitively, we call a set of feature matchings *temporally consistent* if the corresponding features are similarly ordered in both time series. Figure 3.5 shows several temporal inconsistencies, where temporal scope boundaries of matching features are not similarly ordered in two time series. We define temporal consistency as follows:

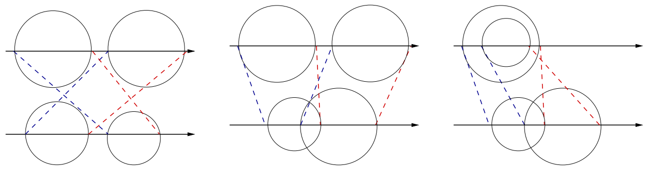


Figure 3.5: Example scope boundary conflicts: blue lines mark corresponding starting points of the matching scopes, red lines mark the corresponding end points

Definition 17 (Temporal Consistency) *Let us be given two metadata-enriched multi-variate (MM) time series,* $\mathbf{Y}_1$ *and* $\mathbf{Y}_2$*, and their feature sets* $\mathcal{F}_1$ *and* $\mathcal{F}_2$*. Let* $p_1 = \langle f_1^1, f_2^1 \rangle, p_2 \in \langle f_1^2, f_2^2 \rangle \in \mathcal{P}$ *be two candidate feature pairs and* $bounds_b^a = \{ts_b^a, te_b^a\}$ *be the start and end points of the temporal scope of feature* f_b^a *for* $a, b \in \{1, 2\}$*. We call* p_1 *and* p_2 temporally consistent *if and only if*

- *for all pairs of end points* $t_i^1, t_j^1 \in bounds_1^1 \cup bounds_2^1$ *in the first pair, we have*

$$((t_i^1 > t_j^1) \quad \rightarrow \quad (t_i^2 \not< t_j^2)) \wedge ((t_i^1 < t_j^1) \rightarrow (t_i^2 \not> t_j^2)),$$

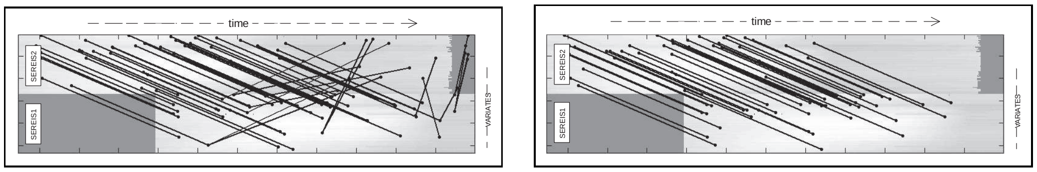
  where $t_i^2, t_j^2 \in bounds_1^2 \cup bounds_2^2$ *are the two end points in the second pair corresponding to* t_i^1 *and* t_j^1*; and*

- *for all pairs of end points* $t_i^2, t_j^2 \in bounds_1^2 \cup bounds_2^2$ *in the second pair, we have*

$$((t_i^2 > t_j^2) \quad \rightarrow \quad (t_i^1 \not< t_j^1)) \wedge ((t_i^2 < t_j^2) \rightarrow (t_i^1 \not> t_j^1)),$$

  where $t_i^1, t_j^1 \in bounds_1^1 \cup bounds_2^1$ *are the two end points in the first pair corresponding to* t_i^2 *and* t_j^2*.* ◇

Figure 3.6 provides an example with inconsistent matches: here we see that the matching process identified some very distant pairs of RMT features as matches. Note also that there are many matching pairs that cross each other in time, implying temporal features that are differently ordered in time in two time series. To improve the accuracy of the matching process, we need to eliminate such inconsistencies. The outline of the process to eliminate inconsistencies is as follows



(a) Pairs of matching RMT features (b) Remaining pairs of matches

Figure 3.6: (a) Candidate RMT feature pairs for two multi-variate time series, and (b) the remaining subset of matching RMT feature pairs after inconsistency pruning

1. For each pair, $\langle f_1, f_2 \rangle \in \mathcal{P}$ of matching features, we compute a dominance score, $dom(f_1, f_2)$, as

$$\rho \left(\begin{array}{l} Align_D(f_1,f_2), Align_T(f_1,f_2), Align_V(f_1,f_2), Align_A(f_1,f_2), \\ Sig_T(f_1,f_2), Sig_V(f_1,f_2), Sig_C(f_1,f_2) \end{array} \right).$$

   Note that this dominance score may, but is not required to, be the same as the overall matching score discussed in Section 3.9.3.

2. We next initialize an empty set ($\mathcal{R}$) to collect the committed consistent feature pairs and two empty lists ($list_1$ and $list_2$) to keep track of their temporal scopes: i.e., we set $\mathcal{R} = \emptyset$, $list_1 = \perp$, and $list_2 = \perp$.

3. Next, we consider all pairs of matching features in $\mathcal{P}$ in descending order of their dominance scores. Let $\langle f_1, f_2 \rangle \in \mathcal{P}$ be the pair we are currently considering.

   (a) *Temporal consistency verification:* Let $\langle ts_1, te_1 \rangle$ and $\langle ts_2, te_2 \rangle$ be the temporal scopes of f_1 and f_2, respectively

      i. We *attempt* to insert the ts_1 and te_1 into $list_1$ ordered in increasing order of time; similarly we *attempt* to insert ts_2 and te_2 into the list, $list_2$, also ordered in increasing order of time.

      ii. Let $rank(ts_1)$, $rank(ts_2)$, $rank(te_1)$, and $rank(te_2)$ be the corresponding ranks of the time points in their respective time ordered lists.

iii. If $rank(ts_1) = rank(ts_2)$ and $rank(te_1) = rank(te_2)$, then we confirm the insertion and we keep the pair [5] .

iv. Else, we drop the pair $\langle f_1, f_2 \rangle$ and eliminate the corresponding scope boundaries from the lists $list_1$ and $list_2$.

(b) If the candidate pair $\langle f_1, f_2 \rangle$ has not been dropped due to temporal inconsistency, then insert the pair in $\mathcal{R}$: i.e., $\mathcal{R} \rightarrow \mathcal{R} \cup \{\langle f_1, f_2 \rangle\}$.

Note that the reason why the feature pairs are considered in descending order of dominance scores is that, when an inconsistency is identified, the most recently considered pair –which is less dominant (relatively less aligned, smaller, and less similar) –can be eliminated without affecting the already committed boundaries.

3.9.6 RMT-Based Multi-variate Time Series Matching Score

Given two metadata-enriched multi-variate (MM) time series, $\mathbf{Y}_1$ and $\mathbf{Y}_2$, and their feature sets $\mathcal{F}_1$ and $\mathcal{F}_2$, the above process results in a set $\mathcal{R}(\mathcal{F}_1, \mathcal{F}_2) = \{\langle f_{1,i}, f_{2,i} \rangle\}$, where $f_{1,i} \in \mathcal{F}_1$ and $f_{2,i} \in \mathcal{F}_2$, respectively. We define the overall matching score, $score(\mathbf{Y}_1, \mathbf{Y}_2)$, of the two multi-variate series, using this set of matching feature pairs:

$$\sum_{\langle f_1, f_2 \rangle \in \mathcal{R}(\mathcal{F}_1, \mathcal{F}_2)} \phi \left(\begin{array}{l} Align_D(f_1, f_2), Align_T(f_1, f_2), Align_V(f_1, f_2), Align_A(f_1, f_2), \\ Sig_T(f_1, f_2), Sig_V(f_1, f_2), Sig_C(f_1, f_2) \end{array} \right),$$

where ϕ is a combined scoring function.

[5]The process is slightly more complex in that there can be exceptions where the ranks are different, but time values are the same. We also confirm the insertion in these special cases.

3.10 Evaluation

In this section, we present experiment results that assess the efficiency and effectiveness of the *robust multi-variate temporal* (RMT [6]) feature extraction algorithms. In our preliminary work Wang *et al.* (2014b), we had shown that the *diagonal* scale-space based RMT features (Section 3.4) are more effective in partial time series search and classification tasks than alternative techniques, including **SVD,** where we created a single fingerprint for each multi-variate time series using the SVD transformation; and **DTW,** where distances were computed directly using dynamic time warping Chen *et al.* (2015). In the appendix, we also consider **SAX**Lin *et al.* (2003b) **DTW**, which provides time savings over DTW, possibly at the expense of accuracy. Therefore, instead of replicating the experiments reported in Wang *et al.* (2014b), we focus on the impact of *full* scale-space based RMT (Section 3.4) features with respect to the use of *diagonal* scale space based RMT (Section 3.4) and also investigate the impacts of the alternative matching and inconsistency removal strategies described in Section 3.9, within the context of a motion recognition task.

3.10.1 Settings

Hardware/Software

In order to ensure results are comparable to those reported in Wang *et al.* (2014b), all experiments were run on the identical set up, with 4-core Intel Core i5-2400 3.10GHz machines with 8GB RAM, running 64-bit Windows 7 Enterprise, using Matlab.

[6]RMT source code is available at EmitLab-ASU (2017).

Table 3.1: (a) Default configuration and (b) alternative matching/pruning strategies

(a) Default configuration

RMT	
# iterations, L	6
# of octaves, o	3
initial smoothing for time, $\sigma_{time,0}$	2.8
initial smoothing for relationships, $\sigma_{var,0}$	0.5
candidate pruning threshold, $\omega_{\top}$	10
descriptor size, $2a \times 2b \times c$	$(4 \times 4 \times 8 =)$ 128
relationship reduction algorithm	k-means
SVD	
degree of energy preservation	95%

(b) Matching/pruning strategies

TO	Temporal overlap (Definition 8)
TP	Temporal proximity (Definition 9)
TA	Temporal alignment (Definition 10)
VA	Variate alignment (Definition 11)
DA	Descriptor alignment (Definition 12)
AA	Amplitude alignment (Definition 13)
TS	Temporal scope significance (Definition 14)
VS	Variate scope significance (Definition 14)

Data Set

For the experiments in this section, we use the Mocap time series data set Mocap (2001): The data set consists of movement records from markers placed on subjects' bodies as they perform 8 types of tasks. We use ASF/AMC format where the original coordinate readings are converted into 62 joint angles data. We treat readings for each joint angle as a different uni-variate time series. The hierarchical spatial distribution (e.g. left foot, right foot, left leg, etc.) of the joint angles on the body is used to create the underlying correlation matrix used as metadata [7] .

We consider additional data sets in the online appendix.

[7]Note that this provides an *intentionally rough* metadata, enabling us to observe accuracy of RMT features under imperfect domain knowledge

Evaluation Metrics

For evaluating accuracy, we use take-one-out methodology with the following criteria: *(a) top-5 precision:* the number of series, among the nearest 5 results, that are of the same class of movement as the query series, and *(b) top-*$\|c\|$ *precision:* the number of series, among the nearest $\|c\|$ results (where $\|c\|$ is the size of the movement class containing the query series) that are of the same class as the query series. The first measure reflects how effective a particular approach is for nearest-neighbor classification, whereas the second measure reflects how well defined the classes.

In addition, we also report pairwise matching times for the alternative approaches. Note that since we are using top-5 and top-$\|c\|$ classification, the classification time is a function of the value of $\|c\|$, the number of labeled data in the training data set, and the pairwise matching time. To ensure that the efficiency different algorithms can be compared independently of the value of $\|c\|$ and the training data set, in the paper, we report the pairwise matching time as an indicator of the classification cost.

Alternative RMT Features

We consider different types of RMT features:

- *diagonal scale-space based RMT (DIA):* This is the version of the RMT features studied in our prior work. As described in Section 3.4, these features are extracted only by considering the diagonal scales of the scale space; in other words, the features' temporal and variate scopes grow in synch to each other.

- *full scale-space based RMT (FULL):* This is the version of the RMT features proposed in this paper. As described in Section 3.4, these are extracted by considering all scales of the scale space; features' temporal and variate scopes grow independently from each other, enabling heterogeneously shaped features.

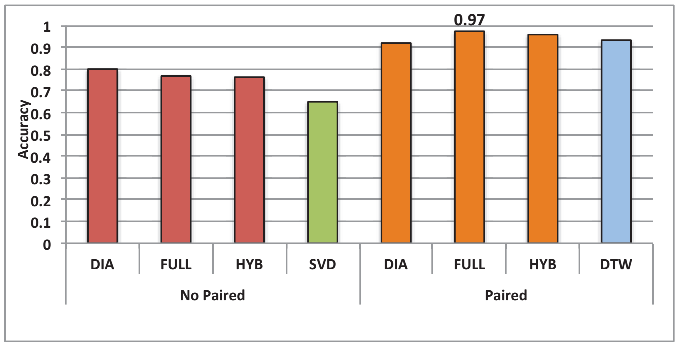
- *hybrid RMT (HYB):* We also consider hybrid feature sets, where diagonal scale-space features and full scale-space features are combined. Note that due to the feature candidate elimination strategy described in Section 3.6, feature set obtained using the full scale-space is not necessarily the superset of the features obtained using the diagonal scale-space. This hybrid strategy re-introduces the diagonal scale-space features which may have been eliminated due to some features in the non-diagonal scales of the space.

- *diagonal scale-space based RMT - alt. 2 (DIA2):* Note that diagonal scale-space based RMT features can be obtained either by using only the diagonal scales of the scale-space as described in Section 3.4, or can be obtained by selecting the subset of the full scale-space based RMT features such that the temporal and variate scales are the same. We refer to this second alternative as DIA2.

Table 3.1(a) provides the outline of the default parameter configuration and describes how these parameters are varied in the experiments.

Alternative Alignment Strategies

In this section, we experiment with the various temporal and variate alignment metrics presented in Section 3.9.1 and listed in Table 3.1(b). When needed, for combining these measures in Table 3.1(b), we use multiplication as the merge function. In addition, we consider two alignment strategies: *(a) all octaves alignment (AoA):* Under this strategy, any two pair of features can be considered for alignment irrespective of their scales. *(b) same octave alignment (SoA):* Under this strategy, only those pairs of features that have the same time and variate octaves are considered for alignment.



(a) Average top-5 precision (%)

Average Top-5 Precision (%)

		Non-paired		Paired	
Class	num	RMT	SVD	RMT	DTW
climb	18	58.9	52.2	85.6	68.9
dribble	14	32.9	28.6	87.1	84.3
jumping	30	100	82.0	100	100
running	19	100	100	100	100
salsa	30	50.0	59.3	100	87.1
soccer	6	43.3	30.0	93.3	96.7
walk	36	100	89.4	100	100
walk (un-even)	76.1	100	58.7	100	98.7
Average Confidence Interval	184	76.9 72.8-80.8%	69.0 65.2-72.8%	97.4 96.5-98.3%	93.3 91.7-94.9%

(b) Per-class top-5 precision

Average Pairwise Matching Time

Non-paired		Paired	
RMT	SVD	RMT	DTW
0.18s	0.003s	0.19s	0.38s

(c) Matching time (in seconds)

Figure 3.7: Top-5 matching accuracy and matching time – default configuration: descriptor alignment (DA) based feature matching and DA based inconsistency pruning and overall score computation

Alternative Inconsistency Elimination (Pruning) Strategies

In this section, we also consider the impact of the measures presented in Table 3.1 on inconsistency elimination process (Section 3.9.5). In addition, we consider two pruning strategies: *(a) all octaves pruning (AoP):* Under this strategy, any two pairs of features can be considered inconsistent irrespective of their scales. *(b) same octave pruning (SoP):* Under this strategy, only those pairs of features that have the same time and variate octaves can be considered inconsistent.

3.10.2 Discussion of the Results

Overview

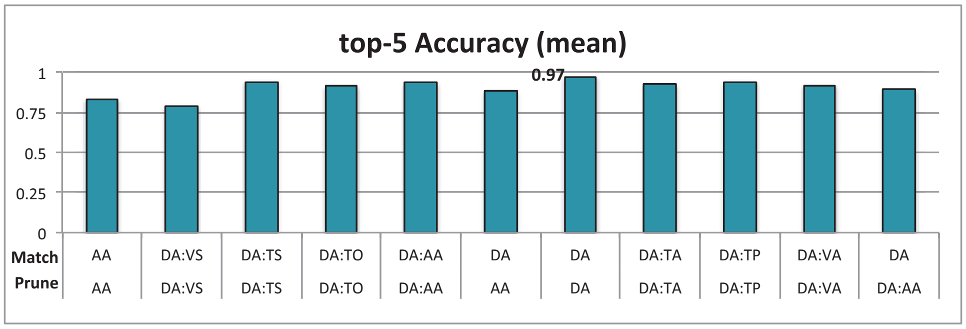
Figure 3.7(a) compares classification accuracy of RMT using 8 classes with 184 motions in the Mocap dataset against alternative approaches:

- *variate-paired* alignment: This is the default configuration where we assume that the pairing of the variates in the query and in the database are known in advance. DTW requires that this pairing is known. In the case of RMT, we leverage the pairing information by ignoring feature matches during the feature alignment phase unless at least 50% of the variates are common. As we see in Figure 3.7, paired RMT provides the best overall accuracy.

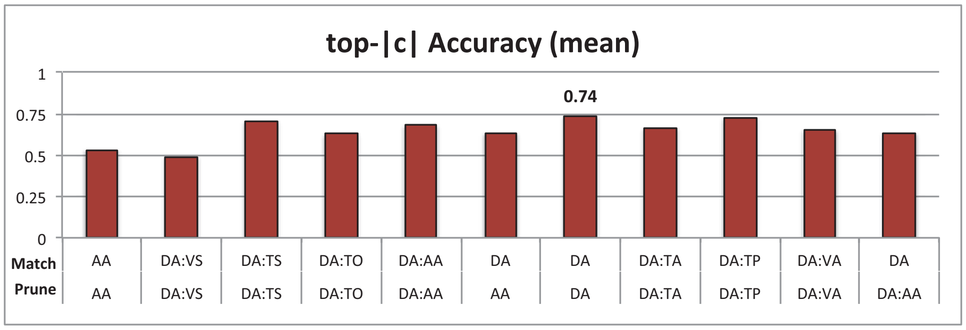
- non-*variate-paired* alignment: Both SVD and RMT can operate without requiring pairing of the variates. Given two multi-variate time series, SVD uses the decomposed series rather than the series themselves, thus it does not require the series to be variate paired. Similarly, RMT can be implemented in such a way that variate alignments are completely ignored during the matching phase. As we see in Figure 3.7, non-paired RMT works better than SVD – and thus is applicable when pairing information is not available. While SVD supports fast matching, the accuracy is significantly lower to render it a feasible approach.

Note that Figure 3.7(b) also includes confidence intervals for the accuracies of various techniques. As we see here, RMT's confidence intervals do not overlap with the other techniques' accuracy confidence intervals, providing additional evidence for the advantage of using RMT features. Moreover, the confidence intervals of RMT are significantly tighter than the confidence intervals of other techniques, again providing evidence that RMT is more robust than the other approaches.

Figure 3.7(a) shows that, as expected, we obtain highest accuracy when we consider the full scale space. It is also important to note that in these experiments we have not leverage RMT feature significance (FS) to boost matching accuracy. Unlike DTW, RMT can further boost accuracy through relevance feeback and other (semi-)supervised learniing techniques.



(a) top-5 precision



(b) top-$\|c\|$ precision

Figure 3.8: The impact of the alternative feature matching and inconsistency pruning strategies – full (FULL) feature set, same octave alignment (SoA), and same octave pruning (SoP)

Impact of Alternative Feature Matching and Inconsistency Pruning

As we have seen in Section 3.9 and Table 3.1, one can use several strategies to match features across multi-variate time series and prune inconsistencies. For the results above, as the default configuration, we considered descriptor alignment (DA) based feature matching, inconsistency, pruning and overall score computation. While the best strategy is application dependent, as we see in Figure 3.8, in this application, RMT is able to achieve high accuracy by using descriptor alignment (DA) for feature matching, inconsistency pruning, and overall score computation. The result also shows that considering additional criteria, such as temporal or variate alignment, is not necessary (and can, in fact, be harmful) in this particular application.

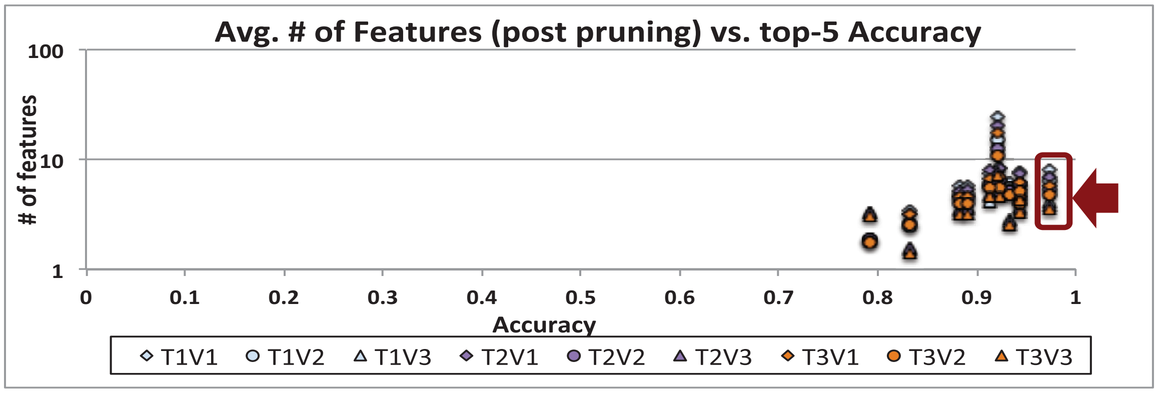


Figure 3.9: Average number of feature pairs for the alternatives considered in Figure 3.8. Here TiVj refers to query features (remaining after inconsistency pruning) that are of time octave i and variate octave j

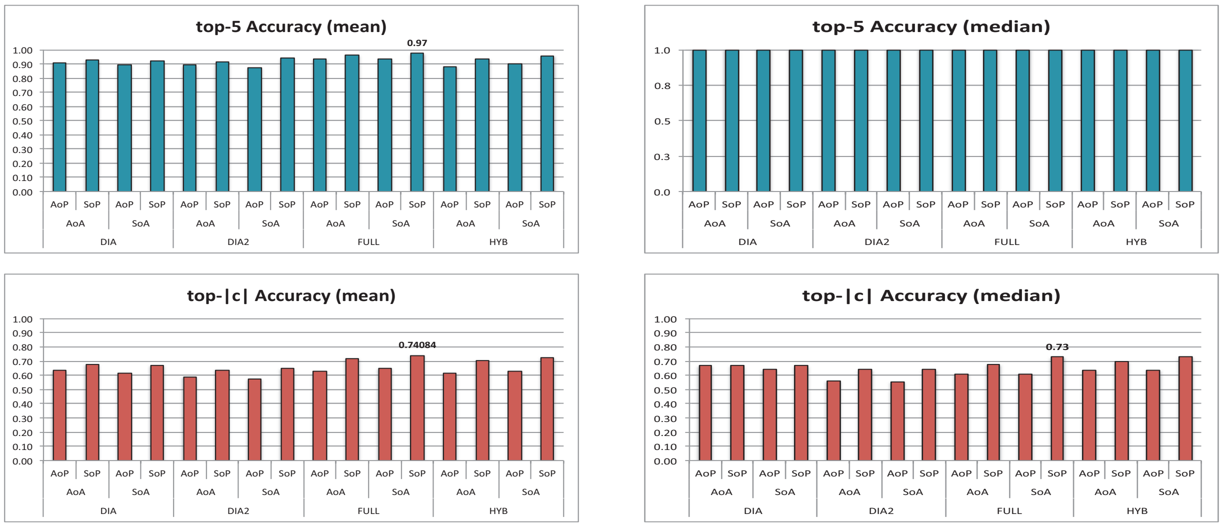


Figure 3.10: Impact of feature discovery and octave management – using descriptor alignment (DA) for feature matching, inconsistency pruning, and overall scoring

This shows that the RMT feature descriptors are highly informative. This is further confirmed by Figure 3.9, where we see the average number of (post-pruning) matching feature pairs for the alternative strategies considered in Figure 3.8. As we see in this Figure, a higher number of matching feature pairs does not translate into more accurate matches. This indicates that the resulting RMT features are highly informative and a small number of feature pairs at different scales and shapes are sufficient to characterize different types of motion.

Table 3.2: # feature pairs before and after inconsistency pruning for the optimal configuration in Figure 3.10

	Before inconsistency pruning	After inconsistency pruning
DIA	952.2	21.5
FULL	1252.1	48.2
HYB	2204.3	56.3

Impact of the Feature Discovery and Octave Management Strategies

For the default results presented above, we leveraged full (FULL) feature set with same octave alignment (AoA) and same octave pruning (SoP) strategies. In Figure 3.10, we study the impact of these strategies in further detail. As we see in this figure, the default configuation indeed leads to highest accuracy: Firstly, as expected, feature matches and inconsistencies need to be considered at each octave scale separately. Secondly, the figure shows that the full scale space provide more information than the diagonal scale space – in fact, extending the FULL feature set with diagonal features (i.e., using the HYB strategy) does not lead to any better results than just using the FULL or DIA feature sets.

This is further studied in Table 3.2, which shows the average matching # feature pairs, before and after inconsistency pruning, for the optimal configuration in Figure 3.10. As we see here, inconsistency pruning eliminates a large number of feature pairs. We also see that the hybrid option (HYB), has more feature pairs than both DIA and FULL, but, as we have seen Figure 3.10, these additional feature pairs do not contribute to the accuracy.

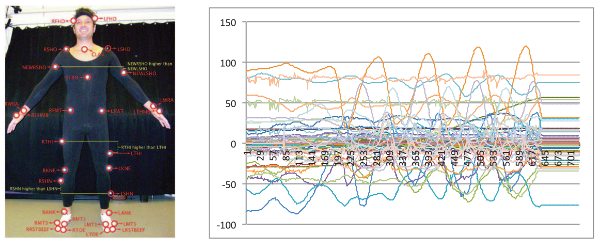


Figure 3.11: A multi-variate time series capturing body movement: the structure of the human body relates the positions of the body sensors during motion capture

3.11 Experiments With Additional DataSets And Algorithms

3.11.1 Experiments with Additional Data Sets

Many time series data sets are (a) multi-variate, (b) interrelated, and (c) multi-resolution: For the experiments reported in this paper, we used several multi-variate data sets that (a) offer the ability to leverage supporting metadata and (b) offer ground truth that can be used for evaluation purposes. The Mocap data sets used in the experiments in Section 3.10 represented human movement in the form of multi-variate time series (Figure 3.11, Mocap (2001)). In this section, we also consider two additional multimedia data sets:

The *Australian sign language* data [8] includes sign gestures captured using a glove-based capture system. The capture data includes 100 per second tracking for all five fingers for both hands: each position tracker provides six degrees of freedom (roll, pitch, yaw, x, y, and z). The data set contains 95 signs, with 27 examples per sign. This data set has 22 variates (11 per hand) and contains a total of 2565 (= 95 × 27) multi-variate time series of average time length, 57. We associated with this data set a metadata file that considers the positions of the fingers within each hand. For this data set we set $\sigma_{time,0}$ to 0.5 (proportional to the average length of the series relative to Mocap - but sufficiently large that the temporal scope of the smallest feature

[8]https://archive.ics.uci.edu/ml/datasets/Australian+Sign+Language+signs+(High+Quality)

covers more than one time instant). Note that ASL data set is selected because it is a relatively synchronized data set where Euclidean based measures perform well.

The *Bird Song* data set [9] contains Mel-frequency cepstral coefficient (MFCC) features for different bird calls. Intuitively, each MFCC coefficient captures short-term power spectrum of a sound for a given frequency band. The MFCC bands are equally spaced on the Mel scale (indicating that they are judged to be of equal distance from each other by listeners). The data set contains 13 MFCC coefficients (i.e., variates) for 154 bird calls of 8 classes, with the average time length of 397 time stamps. We associated with this data set a metadata file that records which MFCC co-efficient is neighbor to which other MFCC coefficients. For this data set we set $\sigma_{time,0}$ to 1.6 (proportional to the average length of the series, relative to Mocap).

Figure 3.12 shows top-5 accuracies and matching times for paired RMT, DTWBemdt and Clifford (1994), and SAXLin *et al.* (2003b) DTW. SAX (Symbolic Aggregate approXimation Lin *et al.* (2003b)) is a symbolic representation for time series, which provides a lower-bound for distance measurements such as dynamic time warping and in general can be computed faster than traditional DTW. Here we also provide SAX [10] as a baseline competitor. We set the parameters for SAX representation: use 10 symbols[11] for representations and 20 segments for each multi-variate time series. Since DTW can be made faster by considering narrower bands Keogh (2002); Sakoe and Chiba (1978) (rather than the whole sequences), we also consider accuracies and execution time for different DTW band sizes. As an approximation method, the performance of SAX shares a similar behavior as DTW method.

We can see in this figure that, while it may help make DTW process faster, placing a significant band length constraint ($\leq 80\%$) on DTW may reduce accuracy

[9]http://www.xeno-canto.org/explore/taxonomy

[10]http://www.cs.ucr.edu/ eamonn/SAX.htm

(for Mocap and bird song data sets, which are less temporally synchronized than the ASL data set). Most importantly, the figure shows that while SAX's accuracy widely fluctuates from one data set to the other, RMT provides consistently better (and overall *the best*) top-5 accuracies, at a matching time cost comparable to DTW. These results indicate that, whenever (even rough) metadata relating the variates is available, RMT can leverage this information to improve classification accuracy.

3.11.2 Experiments with Additional Algorithms

In the previous sections, we compared the proposed RMT algorithm to approaches that are based on SVD, DTW, and SAX-based feature extraction. In this section, we consider two recent systems, namely RPM Wang *et al.* (2016b) and STS3 Peng *et al.* (2016b), that provide parameter selection and hyper-parameter estimation functionalities for uni-variate time series matching and time-series classification tasks; in particular, both use training data to learn feature patterns as well as contextually relevant hyper-parameters.

For both of these techniques, we obtained original code from the authors. However, since both of these approaches were originally designed for uni-variate time series data, we revised their code to account for multi-variate series as follows:

- RPM Wang *et al.* (2016b) creates SAX sequences and grammar rules for uni-variate time series from each class. More specifically, RPM concatenates all uni-variate time series from the same class in the training data and then extracts and selects SAX symbol sequences that are most representative for this given class. RPM then uses Sequitur to learn the context free grammars from the SAX representations as the grammar induction rules to represent this class. Given the output of this process, it uses SVM classifier for classification tasks.

Since in this paper, we consider multi-variate time series data, we modified the original implementation to account for the existence of multiple variates. The training phase stays the same: RPM generates a grammar pattern for each variate. We concatenate all variate pattern vectors from the same class into one vector and use these concatenated pattern vectors from testing data for the SVM classifier. In order to ensure that RPM results and other results presented in our paper are comparable, we set the same SAX parameters as it was described in our manuscript: 20 SAX segments for each multi-variate time series data elements and up to 10 symbols for grammar rule-based representations.

- Instead of concatenating all uni-variate time series from the same class together, STS3 Peng *et al.* (2016b) learns patterns (sets of cell IDs) for every time series of each class. During testing phase, it computes sets of cells for testing data and it uses Jaccard similarity between training and testing data to assign class labels for the testing class.

  Once again, the original STS3 algorithm is designed for uni-variate time series. Therefore, we modified the implementation such that it extracts sets of cells for each variate of class per training data element and aggregates the final Jaccard similarities for each pair of corresponding variates between two multi-variate time series to measure time series similarity.

RPM Wang *et al.* (2016b) provides classification through SVM, whereas the code of STS3 Peng *et al.* (2016b) provided by the authors is designed for 1-NN matching. Therefore, to be fair to STS3, in Table 3.3, we provide 1-NN accuracy for RMT (rather than 5-NN and $\|c\|$-NN accuracies as reported elsewhere in this paper).

As the results in the table shows, the accuracy of RPM is lower than that of RMT, especially for the BirdSong data set, which results in significaanly lower accurcy, Note

Data set	**RPM Acc.** (SVM)	**STS3 Acc.** (1-NN)	**RMT Acc.** (1-NN)
MoCap	0.847	0.078	0.989
BirdSong	0.436	0.145	0.481
ASL	N/A	0.234	0.715

Table 3.3: Accuracies for the multi-dimensional extensions of RPM Wang *et al.* (2016b) and STS3 Peng *et al.* (2016b) algorithms

that, we are not able to report RPM accuracy results for the Australian Sign Language (ASL) dataset because there are multiple variates from various classes with values all zero and these cannot be used to generate grammar rules for classification. The table also shows that STS3 performs significantly worse than both RPM and RMT. While, unlike RPM, STS3 is able to handle the ASL data, it still provides very low accuracy due to the existence of these highly non-discriminating variates.

We note that the reason why STS3 performs rather poorly on multi-variate time series may be due to the way these algorithms learn patterns or the way they compare the series (or both). In order to better understand the underlying reason, we also considered a simple strategy that creates SAX symbols as in RPM, but uses Jaccard similarity of the resulting SAX term vectors for similarity computation as in STS3. More specifically, we counted the frequencies of each SAX symbol within an uni-variate vector and summed up the resulting weighted Jaccard similarities among variate pairs to obtain the similarity between two multi-variate time series: let $\vec{s}$ and $\vec{t}$ represent the symbol frequency vectors for two time series, S and T; the corresponding weighted Jaccard similarity is computed as

$$sim_{Jacc}(S,T) = \frac{\|\vec{s}\|_1 + \|\vec{t}\|_1 - \|\vec{s} - \vec{t}\|_1}{\|\vec{s}\|_1 + \|\vec{t}\|_1 + \|\vec{s} - \vec{t}\|_1},$$

where $\|*\|_1$ represents norm-1 for the corresponding vector. Furthermore, as a control scenario, we also considered the cosine similarity between the two vectors.

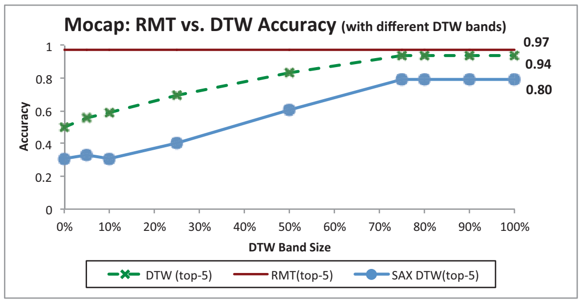
Data set	**Jaccard SAX Acc.** (1-NN)	**Cosine SAX Acc.** (1-NN)	**RMT Acc.** (1-NN)
MoCap	0.826	0.782	0.989
BirdSong	0.357	0.305	0.481
ASL	0.525	0.504	0.715

Table 3.4: Accuracies for the multi-dimensional extensions of Jaccard and cosine similarity based extensions of SAX

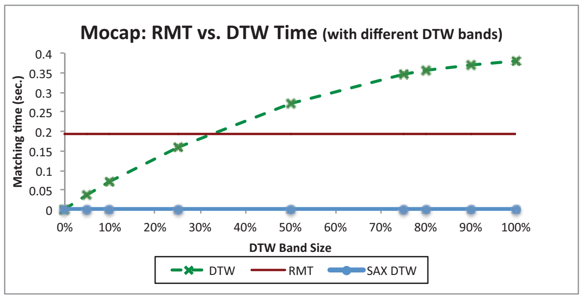
The results under the same evaluation conditions are presented in Table 3.4. This rather simple technique, based on SAX features matched using Jaccard similarity, approaches to that of RPM (on MoCap and BirdSong data sets where RPM results are available) and significantly improves over that of STS3; however results are still not as good as the RMT accuracies. Moreover, when using cosine similarity, matching accuracy slightly drops under that of the Jaccard similarity, indicating that STS3 is using a good measure for matching, but the core problem is that the underlying pattern extraction scheme cannot be directly expanded for multi-variate series.

3.12 Conclusion

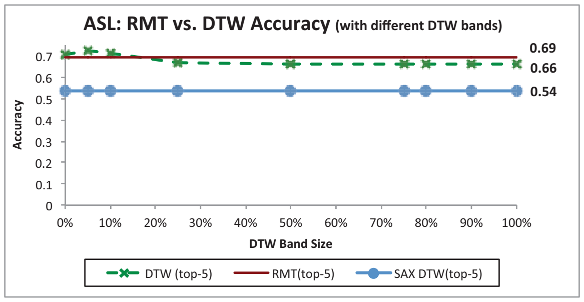
Many time series data sets are (a) multi-variate, (b) interrelated, and (c) multi-resolution. These include motion and gesture data, as described in this paper, as well as data from other domains. We presented a metadata-enriched multi-variate time series model, in which a dependency/correlation model relates the individual variates to each other. Recognizing that multi-variate temporal features can be extracted by simultaneously considering, at multiple scales, differences among individual variates along with the dependency/correlation model that relates them, we developed algorithms to detect robust multi-variate temporal (RMT) features that are multi-resolution and invariant against various types of noise. Experiments confirmed that the RMT features are highly effective in multi-variate series search and classification.



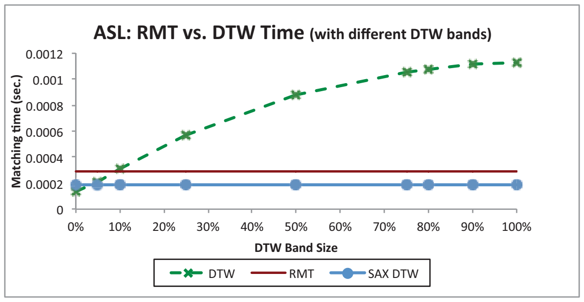
(a) accuracy (Mocap)



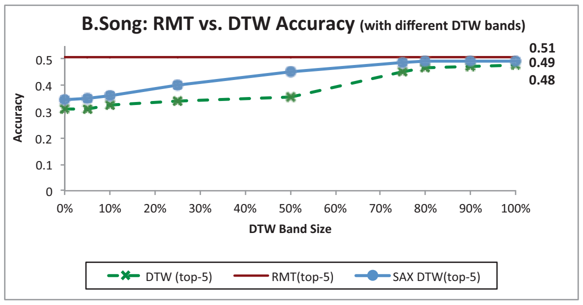
(b) matching time (Mocap)



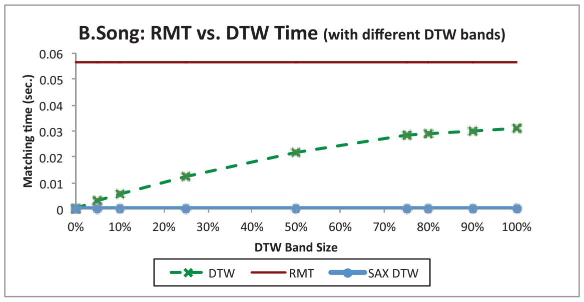
(c) accuracy (ASL)



(d) matching time (ASL)



(e) accuracy (Bird Song)



(f) matching time (Bird Song)

Figure 3.12: (a,c,e) top-5 accuracies for RMT of DTW for different bands (note: 0% band length corresponds to traditional Euclidean distance) and (b,d,f) matching times: for all experiments, the smallest RMT feature is set to be 5% of the average time series in the data set.

Chapter 4

RMT FEATURES BASED SYSTEMS

4.1 Problem Definition

With the concept of RMT features, researchers could design and leverage the model and algorithms to help decision making systems detect and analyze the pattern from large volumes of data. One major data exploration scenario is to analyze diffusion process of infectious diseases which requires the insights of demographic data, contact networks, age-specific conatct rates, mobility networks and health-care and control intervention data and models. We propose **Networks-Of-Traces for Epidemic Spread Simulations (NOTES2)** model and system which aim at assisting experts and helping them explore existing simulation trace datsets. It supports analysis and indexing of simulation data as well as parameter and feature analysis including identification of unknown dependencies across the input parameters and output variable spanning the different layers of observation and simulation data. More specifically, the networks-of-traces(NT) data refers to:

- Network layers: An epidemic simulation requires one or more layers of networks, from local and global mobility patterns to contact networks.
- Disease models: it describes the epidemiological parameters relevant to a simulation and the parameter dependencies necessary in the computation of the disease spread.
- Simulaiton traces: For a given disease study, researchers and decision makers often perform multiple simulations, each corresponding to different sets of

assumptions (disease parameters or models) or context (e.g. spatiotemporal context, outbreak conditions, interventions).

- Disease observation traces: These include real-world observations relating to particular epidemic, including the spread and severity of the disease and observations about other relevant parameters, such as the average length of recovery or percentage of infectious individuals that undergo pharmaceutical treatment.
- External interventions: In an outbreak, public health and disease control agencies implement various medical or social interventions, quarantines and/or school closures.

An extended system tool is proposed as **EpiDMS: Data Management and Analytics for Decision Making from Epidemic Spread Simulation Ensembles**. We argue that the management and analysis of simulation ensembles stemming from large-scale computational models poses challenges particularly when dealing with multiple inter-dependent parameters, spanning multiple layers and geo-spatial frames, affected by complex dynamic processes operating at different resolutions. This problem is compounded by the need to generate near real-time decision-making assessments as the situation in the field changes, which may require the generation of ensembles consisting of 1000s of simulation sets in order to capture a comprehensive range of plausible transmission and control scenarios. In this work we propose the epidemic simulation data management system (epiDMS) which was developed to address the challenges that arise from the need to generate, search, visualize, and analyze in a scalable manner, large volumes of epidemic simulation ensembles and observations during the progression of an epidemic. EpiDMS aims to fill an important gap in decision making during health-care emergencies and enabling critical services with significant economic and health impact.

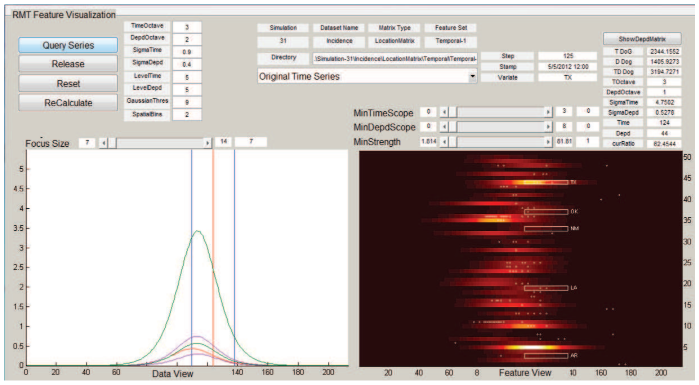
Later on we extend the framework to support data processing to fit in more general cases and we propose **SIMDMS: Data Management and Analysis to Support Decision Making through Large Simulation Ensembles** and **DataStorm-FE: A Data- and Decision-Flow and Coordination Engine for Coupled Simulation Ensembles**, both of which aim to address the key challenges underlying the creation and use of large simulation ensembles and enables

- execution, storage and indexing or lage ensemble simulation datasets and the corresponding models
- search, analysis and exploration of ensemble simulation datasets to enable ensemble-based decision support.

4.2 Notes2

Decision making and intervention against infectious diseases require analysis of large volumes of data, including demographic data, contact networks, age-specific contact rates, mobility networks, and health-care and control intervention data and models. In this paper, we present our *Networks-Of-Traces for Epidemic Spread Simulations (NOTES2)* model and system which aim at assisting experts and helping them explore existing simulation trace data sets. NOTES2 supports analysis and indexing of simulation data sets as well as parameter and feature analysis, including identification of unknown dependencies across the input parameters and output variables spanning the different layers of the observation and simulation data.

Real-time and continuous analysis and decision making for infectious disease understanding and intervention involve multiple aspects, including (i) estimating transmissibility of an epidemic disease, such as influenza Abubakar *et al.* (2012); (ii) forecasting the spatio-temporal spread of pandemic disease at different spatial scales



TimeOctave 3
DepdOctave 2
SigmaTime 0.9
SigmaDepd 0.4
LevelTime 5
LevelDepd 5
GaussianThres 9
SpatialBins 2

Simulation	Dataset Name	Matrix Type	Feature Set
31	Incidence	LocationMatrix	Temporal-1

Directory .\Simulation-31\Incidence\LocationMatrix\Temporal\Temporal-
Original Time Series
Step 125
Stamp 5/5/2012 12:00
Variate TX

MinTimeScope 0 3 0
MinDepdScope 0 8 0
MinStrength 1.814 81.81 1
Focus Size 7 14 7

T DoG 2344.1552
D Dog 1405.9273
TD Dog 3194.7271
TOctave 3
DepdOctave 1
SigmaTime 4.7502
SigmaDepd 0.5278
Time 124
Depd 44
curRatio 62.4544

Figure 4.1: Simulation trace exploration interface of NOTES2

Merler *et al.* (2011); (iii) assessing the effect of travel controls during the early stage of the pandemic Colizza *et al.* (2007); (iv) predicting the effect of implementing school closures Wu *et al.* (2010); and (v) assessing the impact of pharmaceutical interventions on pandemic disease Ferguson *et al.* (2005); Deodhar *et al.* (2014) through simulations. While highly modular and flexible epidemic spread simulation software, such as GLEaMviz gle (2019) and STEM STEM (2016), exist, these suffer from two major challenges that prevent real-time decision making:

- **Data and model complexity**: A sufficiently useful disease spreading simulation tool requires models, including demographic data, social contact networks, age-specific contact rates, local and global mobility patterns of individuals Balcan *et al.* (2009); Merler and Ajelli (2014), epidemiological parameters for the infectious disease (e.g., infectious period), and control intervention data and models. Moreover, these dynamically evolve over time due to preventive actions taken by individuals and public health interventions, requiring continuous adaptation.

- **Complexity of the simulation and observation data**: Epidemic simulations track 10s or 100s of inter-dependent parameters, spanning multiple layers and geo-spatial frames, affected by complex dynamic processes operating at different resolutions. Moreover, generating an appropriate ensemble of stochastic epidemic realizations may require multiple simulations, each with different parameters settings corresponding to slightly different, but plausible, scenarios. Thus, running and interpreting simulation results (along with the real-world observations) to generate timely actionable results are difficult.

I propose *Networks-Of-Traces for Epidemic Spread Simulations (NOTES2)* to assist experts in exploring large simulation ensembles (Figure 4.1). The NOTES2 system supports

- *analysis and indexing of simulation data sets,* including extraction of salient multi-variate temporal features from the inter-dependent parameters, spanning multiple layers and spatial-temporal frames, driven by complex dynamic processes operating at different resolutions.
- *parameter and feature analysis,* including identification of unknown dependencies across the input parameters and output variables spanning the different layers of the observation and simulation data.

4.2.1 Networks-of-Traces for Epidemic Simulations

If effectively leveraged, models reflecting past outbreaks, existing simulation traces obtained from simulation runs, and real-time observations incoming during an outbreak can be collectively used for better understanding the epidemic's characteristics and the underlying diffusion processes, forming and revising models, and performing exploratory, if-then type of hypothetical analyses of epidemic scenarios.

There are five major types of data associated to epidemic spread simulations.

- *Network layers*: An epidemic simulation requires one or more layers of networks, from local and global mobility patterns to contact networks.

- *Disease models*, describing the epidemiological parameters relevant to a simulation and the parameter dependencies necessary in the computation of the disease spread.

- *Simulation traces*: For a given disease study, researchers and decision makers often perform multiple simulations, each corresponding to different sets of assumptions (disease parameters or models) or context (e.g. spatio-temporal context, outbreak conditions, interventions).

- *Disease observation traces*: These include real-world observations relating to particular epidemic, including the spread and severity of the disease and observations about other relevant parameters, such as the average length of recovery or percentage of infectious individuals that undergo pharmaceutical treatment.

- *External interventions*: In an outbreak, public health and disease control agencies implement various medical or social interventions, quarantines and/or school closures.

We collectively refer to these data (network layers, disease models, simulation traces, observation traces, and interventions) as the networks-of-traces (NT) data.

4.2.2 Disease Spread Simulation Understanding and Analysis

Epidemic spread simulations are complex. However, parameter dependencies and the network structures of the layers (e.g. mobility, social contact networks) are implicitly evident in the simulation traces and these carry temporal features (that may correspond to major changes in the underlying networks and/or temporal dynamics) that are robust against noise. The detection of these robust multivariate features constitutes the first step towards leveraging the NT data for understanding

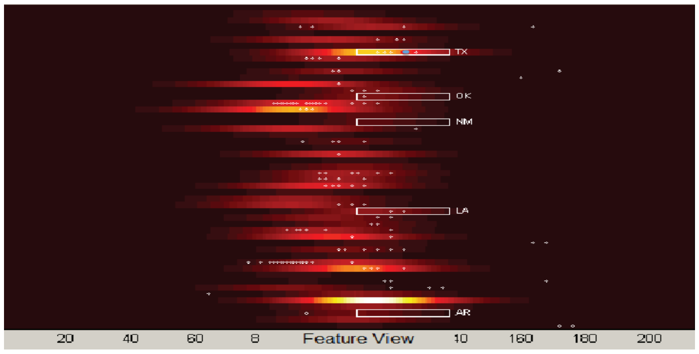


Figure 4.2: Each row corresponds to a time series of incidences for a sample epidemic simulation and each dot corresponds to the center of an identified multivariate feature

epidemics' characteristics and the diffusion processes, revising models, and performing exploratory, if-then type of hypothetical analyses of epidemic scenarios.

4.2.3 Networks-of-Traces (NT) Feature Extraction

An NT data trace is multi-variate and the analysis of the relevant processes requires multi-variate temporal features spanning multiple inter-dependent trace parameters. Intuitively, a robust temporal feature in a multi-variate time series corresponds to a *multi-variate segment* of the series which significantly differs from its neighborhood. The multi-variate segment is represented by a center, $\langle \mu_t, \mu_v \rangle$, and a scope, $\langle \sigma_t, \sigma_v \rangle$. Intuitively, μ_t marks the center of the segment in time and σ_t is the corresponding time interval. On the other hand, μ_v is one or more nodes/variates of the graph on which the segment is centered and σ_v denotes all the graph vertices covered by the segment.

Figure 4.2 shows an epidemic simulation heatmap, where each row corresponds to a different state. In the figure, centers of identified features are highlighted by white dots. The figure also expands one of these robust features (tail end of the epidemic on a set of neighboring states): the center, $\langle \mu_t = 125, \mu_v = \{TX\}$, is marked with a blue dot and its scope, $\langle \sigma_{time} = [111, 139], \sigma_v = \{AR, LA, NM, OK, TX\} \rangle$, is visualized using rectangles.

4.2.4 Robust Feature Detection

Let $Y(t) = \langle Y_1(t), ..., Y_m(t)\rangle$ be a multi-variate trace, from time $t = 1$ to $t = n$. As in Lowe (2004c), we detect stable multi-variate features at the extrema of the scale space. However, unlike Lowe (2004c) (which operates on images with two ordered dimensions; i.e., rows and columns of pixels), extracting multi-variate features of the simulation trace Y (at various temporal and variate scales) requires detecting local maxima and computing gradients relative to not only the (ordered) time dimension, but also to the underlying variate graph.

Time-and-Variate Smoothing. We construct the scale space of $Y(t)$ (corresponding to the versions of the series smoothed at different temporal and variate scales) relying on the following time-and-variate smoothing process:

- Let $Y_i(t, s_t)$ indicate a version of uni-variate series, Y_i, smoothed with parameter s_t: $Y_i(t, s_t) = \mathtt{G}(t, s_t) * Y_i(t)$, where $*$ is convolution in t and $\mathtt{G}(t, s_t)$ is the Gaussian. Let $Y(t, s_t) = \langle Y_1(t, s_t), .., Y_m(t, s_t)\rangle$ be a version of Y, where each uni-variate series is independently smoothed.

- Let us also define the variate smoothing function, $S(\mathbf{R}, s_v, X) = [\mathtt{G}(0, s_v)I + \sum_{(j=1)}^{\infty} 2 \times \mathtt{G}(j, s_v)\mathbf{R}^j]X$, where (a) $\mathbf{R}$ is an $m \times m$ matrix describing the variate dependencies, (b) $X = \langle X_1, ..., X_m\rangle$ is a m-vector, and (c) s_v is a variate smoothing parameter. Since $\mathtt{G}(j, s_v)$ approaches 0 quickly as j increases, the smoothing term in front of X can be approximated by a finite summation.

The time-and-variate smoothed version of $Y(t, s)$ at scale $s = \langle s_t, s_v\rangle$ is defined as $\mathbf{Y}(t, s) = (\mathcal{H}_1(s); ...; \mathcal{H}_t(s))$, where $\mathcal{H}_t(s) = S(H, s_v, Y(t, s_t))$ is the version of $Y(t, s_t) = \langle Y_1(t, s_t), ..., Y_m(t, s_T)\rangle$, variate-smoothed at scale s_v at time instant, t.

Iterative Scale Space Construction. We construct the scale space by incrementally smoothing Y (both in time and variates) starting from an initial scale $s_0 = \langle s_{t,0}, s_{v,0} \rangle$. Let $\mathbf{Y}_i(t,s)$ be a time-and-variate smoothed version of $Y_i(t)$ at scale $s = \langle s_t, s_v \rangle$. Given a pair, $k = \langle k_t, k_v \rangle$, of time and variate scale multipliers, we add the three *scale-space neighbors* (or `ss`-neighbors) of $Y_i(t)$ into the scale space:

$$\begin{aligned}
\mathbf{Y}_i(t, k \circ_t s) &\equiv_{def} \mathbf{Y}_i(t, \langle k_t \times s_t, s_v \rangle), \\
\mathbf{Y}_i(t, k \circ_v s) &\equiv_{def} \mathbf{Y}_i(t, \langle s_t, k_v \times s_v \rangle), \text{ and} \\
\mathbf{Y}_i(t, k \circ_{t,v} s) &\equiv_{def} \mathbf{Y}_i(t, \langle k_t \times s_t, k_v \times s_v \rangle).
\end{aligned}$$

The process continues iteratively until maximum temporal and variate scales (bounded by the length of the simulation trace and the number of variates) are met.

Local Extrema Detection. For detecting extrema, for each $\mathbf{Y}_i(t,s)$ in the constructed scale space, we compute

$$\begin{aligned}
D_i^t(t,s) &= abs(\mathbf{Y}_i(t,s) - \mathbf{Y}_i(t, k \circ_t s)), \\
D_i^v(t,s) &= abs(\mathbf{Y}_i(t,s) - \mathbf{Y}_i(t, k \circ_v s)), \\
D_i^{t,v}(t,s) &= abs(\mathbf{Y}_i(t,s) - \mathbf{Y}_i(t, k \circ_{t,v} s)).
\end{aligned}$$

Local extrema are identified by considering each $\langle i,t,s \rangle$ triple and comparing $max(D_i^t(t,s), D_i^v(t,s), D_i^{t,v}(t,s))$ against the 78 `ss`-neighbors of $\langle i,t,s \rangle$ in terms of time (*before, same time, after*), variates (*impacting, same variate, impacted_by*), and scales (*smaller, same scale, larger*). Poorly defined extrema (i.e., an extremum that has a large principal curvature in one direction but a small one in the perpendicular direction) are eliminated.

Feature Descriptor Creation. Let us be given a triple $\langle i,t,s \rangle$. Let also N and M be two integers such that $N \sim 3\sigma_t$ and $M \sim 3\sigma_v$. We create the local feature descriptor corresponding to this triple using a $2N \times 2N$ matrix W: Let $\mathbf{Y}_{(i,s)}$ be the time series

Y_i at scale s; then, for all $-N < a \leq N$ and $-N < b \leq N$, $W[a,b]$ is defined as follows: (a) if $b > 0$, $W[a,b] = (\mathbf{R}^b \mathbf{Y}_{(i,s)})[t+a]$; (b) if $b = 0$, $W[a,b] = \mathbf{Y}_{(i,s)}(t+a)$, and (c) if $b < 0$, $W[a,b] = (\mathbf{R}^{-1})^b \mathbf{Y}_{(i,s)})[t+a]$.

Finally, we construct a $(2u \times 2v \times c)$-dimensional descriptor for the triple $\langle i, t, s \rangle$ in the form of a gradient histogram based on the matrix, W: we sample c gradient magnitudes on the descriptor using a $2u \times 2v$ grid superimposed on the matrix, W. A Gaussian weighting function is used to reduce the magnitude of elements further from the center.

4.2.5 Feature Search and Alignment

Features extracted from a networks-of-traces data play important roles in the NOTES2 system. Here, we discuss how the similarity between two triples, $\langle i_1, t_1, s_1 \rangle$ and $\langle i_2, t_2, s_2 \rangle$, and the corresponding descriptors, $desc_1$ and $desc_2$, are computed in NOTES2. Depending on the use context, feature similarity has three major components:

- *Descriptor alignment:* Since the feature descriptors are gradient histograms, their similarity is measured through a histogram similarity function (in the experiments, we use inverse of Euclidean distance).

- *Temporal alignment:* For temporal alignment between two features, we consider both the distance between the temporal centers of the features as well as the degree of overlap between the temporal scopes of the features.

- *Variate alignment:* For variate alignment, we consider both the distance between the variates in the underlying relationship graph as well as the degree of overlap between the variates within the scopes of the two features.

Depending on the application, we also consider alignments of (a) the *average amplitudes* and (b) *sizes* of the temporal and variate scopes of the two triplets. These various components of feature similarity are combined using a similarity merge function, such as *max*, *min*, or *product* based on the desired matching semantics.

4.2.6 Evaluation

To assess whether the features extracted from epidemic simulations truly reflect the underlying *variate networks*, we created a set of simulations, using the STEM simulator, based on the US border network, where there is an edge between states if they share a border, and air network, which is a clique. For a given pair of transmission and recovery rates, we created 51 simulations (of length 213 units of time) assuming a different US state as the ground zero and recorded incidence rates [1] . We then extracted three sets of features from each simulation, using parameters in Table 4.1, and assuming different connectivity structures:

- *Border network (BN):* For this case, we used the border network denoting states sharing borders.
- *Air network (AN):* In this case, features are extracted assuming the air network (which is a clique).
- *Random network (RN):* In this case, a random graph (with the same number of edges as the border network) is used for extracting features.

Given these features and their descriptors, we then computed the *confusion* for a simulation with ground zero state, gz_i, as $confusion(gz_i) = AVG_{gz_j \neq gz_i} \frac{sim(gz_i,gz_j)}{sim(gz_i,gz_i)}$. Here the similarity, $sim(gz_i, gz_j)$, between two simulations with ground zero states,

[1]Unless otherwise stated, we use the default STEM parameters

Table 4.1: Target feature parameters

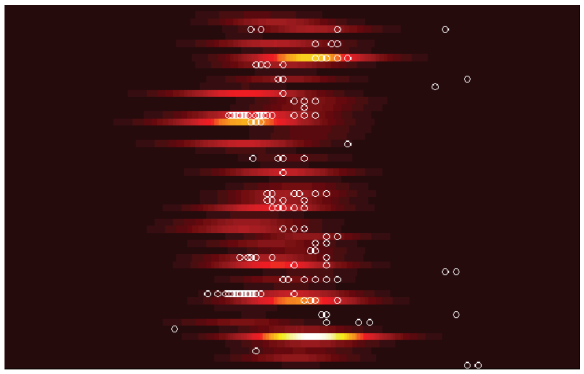
min target feature length	~ 5 time units
max target feature length	~ 40 time units
min target feature size	~ 2 hops
max target feature size	~ 10 hops
descriptor size	$32\ (= 2 \times 2 \times 8)$

Table 4.2: Average confusions for simulations with different transmission and recovery rates (death and birth rates are 0). Features are extracted assuming different network structures

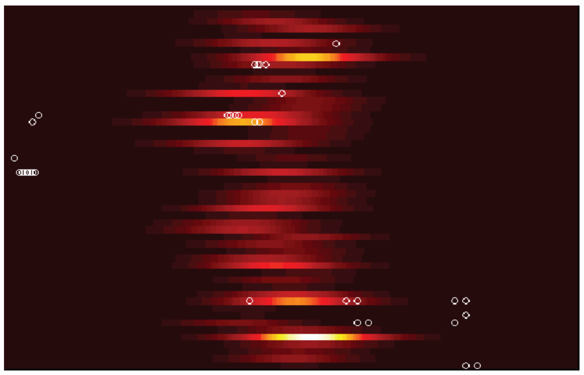
T.Rate	*R.Rate*	*BN*	*AN*	*RN*
1.0	0.5	0.35	0.36 (1.14×)	0.44 (1.28×)
0.75	0.5	0.23	0.25 (1.06×)	0.32(1.37×)
0.25	0.5	0.37	0.63 (1.73×)	0.49 (1.33×)
1.0	0.25	0.44	0.45 (1.02×)	0.54 (1.23×)
0.75	0.25	0.39	0.45 (1.14×)	0.48 (1.23×)
0.5	0.25	0.29	0.34 (1.15×)	0.39 (1.32×)

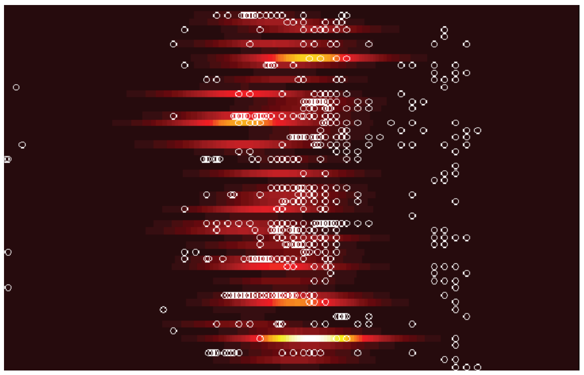
gz_i and gz_j, is defined as $\sum_{f \in features(gz_i)} sim(f, bestmatch_j(f))$. where $bestmatch_j(f)$ is the best matching feature to f in the simulation with ground zero state, gz_j. Temporal alignment parameters were set to be equal; $\alpha_t = \beta_t = 0.5$. Variate alignment parameters were set to $\alpha_v = 0$ and $\beta_v = 0.1$, to avoid penalizing the *wrong network* alternative. We use the product merge function to combine alignment scores.

Intuitively, large confusion implies poor differentiation power and, if the feature extraction process is effective, then we expect that (a) the overall confusion will be the lowest when using features extracted based on a network reflecting the underlying disease propagation, and (b) confusion will be the highest when we use an inappropriate network for feature extraction. Table 4.2 presents results for different transmission and recovery rates. As we see in this table, using the border network for feature



(a) features extracted using the border network





(b) features using the air network (clique) (c) features using a random network

Figure 4.3: Centers of features extracted using different networks (source = "NJ", trans. rate = 0.75 and rec. rate = 0.5)

extraction leads to least amount of confusion. Moreover, these results conform to our expectations listed above and Figure 4.3 helps see why: the (clique structured) air network ignores disease transmissions through land borders (especially when the transmission rate is too small for the flights to have a big impact on the epidemic's diffusion) and, thus, misses useful features. Random networks, on the other hand, result in significant noise.

4.2.7 Conclusion

In this paper, we presented our *networks-of-traces* model, which accounts for layers of disease networks (from local and global mobility patterns to contact networks), disease models, simulation and observation traces, and external interventions. The *Networks-of-Traces for Epidemic Spread Simulations (NOTES2)* system, based on this model, aims to assist experts in exploring large simulation trace data sets, through networks-of-traces feature analysis.

4.3 EpiDMS

The potential for pandemics to rapidly generate morbidity, mortality, and economic impact around the world has highlighted the need to develop quantitative frameworks for supporting public health decision-making in near real-time. For instance, the 2003 SARS coronavirus (Severe Acute Respiratory Syndrome) emergency, which originated in China and spread to 29 countries, generated important nosocomial outbreaks in several regions by August 2003 Chowell *et al.* (2003). More recently, the 2009 A/H1N1 influenza pandemic originating in Mexico rapidly spread around the globe via the airline network and reached 20 countries with highest volume of passengers arriving from Mexico within a few weeks of epidemic onset Khan *et al.* (2009). Importantly, the economic impact associated with a pandemic similar to the 2009 A/H1N1 influenza pandemic has been estimated to cost the global economy between 360 billion and 4 trillion SWI (2019) for the first year of virus circulation. Large-scale computational transmission models of infectious disease spread are increasingly becoming part of the toolkit to carry out inferences on the spread and control of infectious diseases. Examples of real-time analyses of epidemics supported by large-scale transmission models include:

- estimating transmissibility of an epidemic disease, such as influenza,
- forecasting the spatio-temporal evolution of pandemics,
- assessing the effect of travel controls during the early epidemic phase,

- predicting the effect of school closures in mitigating disease spread,
- assessing the impact of reactive vaccination strategies,

These analyses, however, require access to, integration, and analysis of models and large volumes of data, including datasets from diverse sources in order to parameterize demographic characteristics, contact networks, age-specific contact rates, mobility networks, and health-care and control interventions. In this paper, we argue that, if effectively leveraged, existing simulation analyses and real-time observations generated during an outbreak can be collectively used for better understanding the transmission dynamics and refining existing models. At the same time, these model simulations are useful for performing exploratory, if-then type of hypothetical analyses of epidemic scenarios in order to address critical questions including: (a) Can we identify and classify key events (e.g., epidemic peak timing, likely epidemic duration) during an infectious disease outbreak from large simulation ensembles? (b) Can we compare and summarize a large number of epidemic simulations and observations under different epidemiological scenarios? (c) Can we discover latent relationships and dependencies among disease dynamics and social parameters?

4.3.1 Epidemic Simulations

Global epidemic spread can be characterized via simulation through networks of multiple (local and global) scales: individuals within a subpopulation may be infected through local contacts during a localized outbreak. These infected individuals then may seed the infection in other regions, starting a new outbreak. Thus, large-scale epidemic simulation systems (e.g., GLEaM gle (2019)and STEM STEM (2016)) are required to leverage models and data at different spatial scales. These include social contact networks, local and global individual mobility patterns, location-specific control interventions, and epidemiological characteristics of the infectious disease:

- The population model for a global epidemic simulation system can be based, for example, on the Gridded Population of the World project by the Socio-Economic Data and Applications Center (SEDAC) SED (2019), which has a resolution of 15 $\times$ 15 minutes of arc.

- Mobility models can include long-range air travel mobility data, from the International Air Transport Association and the Official Airline Guide and/or short-range commuting patterns between adjacent subpopulations. High-resolution demographic and age-specific contact data has become available for a number of countries including the US Germann *et al.* (2006), and South-East Asia Longini Jr.*et al.* (2005) while age-specific contact rates have been derived from population surveys for a number of European countries Mossong *et al.* (2008). Large-scale computational transmission models, parameterized with high volume air traffic data and country-level seasonality factors, are being increasingly used to assess the global transmission patterns of emerging infectious diseases and the effectiveness of control measures.

- Epidemic models allow the user to specify epidemiological parameters that are specific of the infectious disease (such as transmissibility and seasonality), initial outbreak conditions (e.g. seeding characteristics of the epidemic and the immunity profile of the subpopulation), and the timing, type and intensity of intervention measures. While the disease model can be specific to the type of infection, the parameters of a typical model (the modified Susceptible-Latent-Infectious-Recovered model described in Van den Broeck *et al.* (2011)) include (a) the infection rate of contracting illness when an individual interacts with an infectious person; (b) infection rate scaling factors for asymptomatic infectors and treated infectors; (c) average length of the latency period (in which the individual is infected, but not infecting); (d) probability of symptomatic vs. asymptomatic infections; (e) change in the travelling behavior after the onset of symptoms; (f) average length of recovery; (g) percentage of infectious individuals that undergo pharmaceutical treatment; and (h) impact (e.g. on the length of the infectious period) of the treatment.

The output of a simulation is a multi-variate time series, which tracks for each spatial location (such as the US states) the simulation values of each output parameter, such as the number of infected individuals.

4.3.2 Challenges

While large-scale epidemic simulation systems such as GLEaM gle (2019) or STEM STEM (2016) represent very powerful and highly modular and flexible epidemic spread simulation systems, their power for real-time decision making could be enhanced by addressing the following challenges: (a) Complexity of the simulation and observation data. A sufficiently useful disease spreading simulation system requires models, including social contact networks, local and global mobility patterns of individuals, and epidemiological parameters for the infectious disease (e.g., infectious period). Epidemic simulations track 10s or 100s of inter-dependent parameters, spanning multiple layers and geo-spatial frames, affected by complex dynamic processes operating at different resolutions. Moreover, an ensemble of stochastic epidemic realizations may include 100s or 1000s of simulations, each with different parameters settings corresponding to slightly different, but plausible, scenarios. As a consequence, running and interpreting simulation results (along with the real-world observations) to generate timely actionable results pose challenges. (b) Dynamicity of the real-world observations. A major challenge in using data- and model-driven computer simulations for predicting geo-temporal evolution of epidemics for managing health emergencies, such as the 2014-15 Ebola epidemic in West Africa, is that the data, models, and the underlying model parameters dynamically evolve over time. This necessitates continuous analyses and interpretations of the incoming data and adaptation of the networks and models. Therefore, simulation ensembles may need to be continuously revised and refined as the situation on the ground changes: (a) revisions involve incorporating the real-world observations as well as updated probability surfaces into existing simulations to alter their outcomes; (b) refinements involve identifying new simulations to run based on the changing situation on the ground to provide trustable recommendations. As the situation on the ground and intervention mechanisms evolve, the sampling strategies for the input parameter spaces have to be varied (by eliminating irrelevant scenarios and considering new scenarios or varying the likelihood of old scenarios) in such a way that more accurate simulation results are obtained where it is more relevant.

In order to have a significant impact on disease control and to devise validated epidemic response strategies within a realistic time frame, public health authorities need to adequately and systematically interpret observations, understand the processes driving epidemic outbreaks, and assess the robustness of conclusions driven from simulations. Because of the volume and complexity of the data, the varying spatial and temporal scales at which the key transmission processes operate and relevant observations are made, public health experts could benefit from novel decision support systems. Therefore, tools that help (a) executing large-scale simulation ensembles under a large number of diverse hypotheses/scenarios, and (b) analysis, exploration, interpretation, and visualization of large simulation ensembles (aligned with the real-world observations) to generate timely actionable results are critically needed for understanding the evolution patterns of the outbreaks (including estimating transmissibility, forecasting the spatio-temporal spread at different spatial scales, assessing the cost and impact of interventions, including travel controls, at various stages of the epidemic) and supporting real-time decision making and hypothesis testing through large scale simulations.

The key characteristics of data and models relevant to data-intensive simulations include the following: (a) voluminous, (b) multi-variate, (c) multi-resolution, (d) multi-layer, (e) geo-temporal, (f) inter-connected and inter-dependent, and (g) often incomplete/imprecise. Moreover, data and models dynamically evolve over time, due to control actions taken by individuals and public health interventions, requiring continuous adaptation and re-modeling. The novel epiDMS software framework aims to address the key challenges underlying large epidemic spread simulations, which, today, hinder real-time and continuous analysis and decision making during ongoing outbreaks. The services provided by epiDMS include:

- storage and indexing of large ensemble simulation data sets and the corresponding models; and
- search and analysis of ensemble simulation data sets to enable ensemble-based decision support.

The target user group for epiDMS include a range of public health researchers and decision makers. While creation of models for ensemble simulations and query formulation require moderate infectious disease modeling experience, epiDMS also provides parameterized queries and other interactive user interfaces to enable decision makers with minimal experience to explore large ensemble simulations.

4.3.3 System Overview

The epidemic simulation data management system for managing the data and models for data-driven real-time epidemic simulations consists of three major components (Figure 4.5):

- Epidemic ensemble execution engine (epiRun) takes as input an epidemic model, mobility/connectivity models, interventions, and outbreak conditions (such as ground zero), and creates an epidemic ensemble by sampling the disease parameter space and executing simulations using an external simulation engine. Note that epiRun is not specific to any disease model or simulation engine and can wrap as a black-box software component: any epidemic simulation engine as long as it provides command line invocation. The epidemic model (formulated in the format specific to the simulation engine), the selected input parameter values, and the simulation results (i.e., time series for each output variable) then become inputs for the epidemic data and model store (epiStore), described next.
- Epidemic data and model store (epiStore) stores, and indexes the relevant data and metadata sets. The data and models relevant for modeling large-scale epidemics include the following:
  - Network layers: An epidemic simulation requires one or more layers of networks, from local and global mobility patterns to social contact networks.
  - Disease models, describing the epidemiological parameters relevant to a simulation and the parameter dependencies necessary in the computation of the disease spread.
  - Simulation time series: For a given disease study, researchers and decision makers often perform multiple simulations, each corresponding to different

sets of assumptions (disease parameters or models) or context (e.g. spatio-temporal context, outbreak conditions, interventions).

– Disease observations: These include real-world observations that arise in near real-time relating to a particular epidemic, including the spread and severity of the disease and observations about other relevant parameters, such as the average length of recovery or percentage of infectious individuals that undergo pharmaceutical treatment.

EpiStore captures simulation metadata (simulation model, parameter values, connectivity graphs) and simulation outputs (time series) and provides data analysis (such as clustering, classification, event extraction) to support decision-making. Once again, epiStore is not specific to any disease model or simulation ensembles generated by a specific simulation engine: it can read and store models and simulation results produced by any epidemic simulation engine as long as data wrappers that convert data and metadata into internal epiStore representation are available. Epidemic ensemble query, visualization, and exploration module (epiViz) provides a web-based query and result visualization interface to support user interaction and exploratory decision making through simulation ensembles. Query specification language is also model independent, in the sense that the system does not make any assumptions regarding what the input and output parameters of the simulations are once imported into epiStore, parameters of any model can be queried, visualized, and explored.

4.4 SimDMS

Data- and model-driven computer simulations are increasingly critical in many application domains. These simulations may track 100s or 1000s of inter-dependent parameters, spanning multiple layers and spatial-temporal frames, affected by complex dynamic processes operating at different resolutions. Because of the size and complexity of the data and the varying spatial and temporal scales at which the key processes operate, experts often lack the means to analyze results of large simulation ensembles, understand relevant processes, and assess the robustness of conclusions driven from the resulting simulations. Moreover, data and models dynamically evolve over time requiring continuous adaptation of simulation ensembles. The simDMS platform aims to address the key challenges underlying the creation and use of large simulation ensembles and enables (a) execu- tion, storage, and indexing of large ensemble simulation data sets and the corresponding models; and (b) search, analysis, and exploration of ensemble simulation data sets to enable ensemble-based decision support.

4.4.1 Introduction

Data- and model-driven computer simulations are increasingly critical in many application domains.

- Epidemic Simulation Ensembles: For example, for predicting geo-temporal evolution of epidemics and assessing the impact of interventions, experts often rely on epidemic spread simulation software such as (e.g., GLEaM gle (2019) and STEM STEM (2016)). The GLEaM simulation engine, for example, consists of three layers: (a) a population layer, (b) a mobility layer which includes both long-range air travel and short-range commuting patterns between adjacent subpopulations, and (c) an epidemic layer which allows the user to specify parameters (such as reproductive number and seasonality) for the infectious disease, initial outbreak conditions (e.g. seeding of the epidemic and the immunity profile of the subpopulation), and intervention measures.

- Building Energy Simulation Ensembles: Similarly, effective building energy management, leading to more sustainable building systems and architectural designs with monitoring, prioritization, and adaptation of building components and subsystems, requires large data-driven simulations involving (a) location and climate information for the city in which the building is located, (b) building construc- tion information, such as building geometry and surface constructions (including exterior walls, interior walls, partitions, floors, ceilings, roofs, windows and doors), (c) building use information, including the lighting and other equipment (e.g. electric, gas, etc.) and the number of people in each area of the building, (d) building thermostatic control information, including the temperature control strategy for each area, (e) heating, ventilation, and air conditioning (HVAC) operation and scheduling information, and (f) central plant information for specification and scheduling of boilers, chillers, and other equipment. EnergyPlus software, for example, relies on the description of a building's physical make-up and associated mechanical and other systems including time-step based simulation for many energy-related building parameters.

4.4.2 Challenge

While, in most cases, very powerful simulation software exist, using these simulation software for decision making faces several significant challenges: (a) Creating correct simulation models is a costly operation, and it is often the case that the designed simulation models are incomplete or imprecise. (b) Also, the execution of a simulation can be very costly, given the fact that complex, inter-dependent parameters affected by complex dynamic processes at varying spatial and temporal scales have to be taken into account. (c) A third major source of cost is the simulation ensemble analysis: because of the size and complexity of the data and the varying spatial and temporal

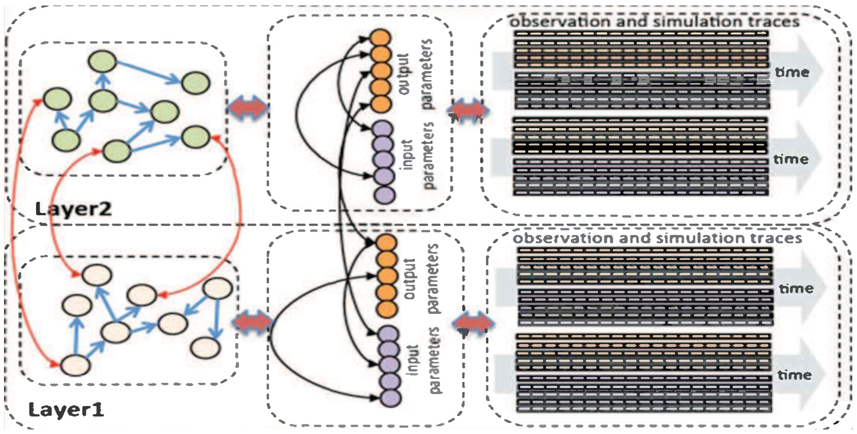


Figure 4.4: Simulation ensembles are (a) multi-variate, (b) multi-modal (temporal, spatial, hierarchical, graphical), (c) multi-layer, (d) multi-resolution, and (e) inter-dependent (i.e., observations of interest depend on and impact each other)

scales at which the key processes operate, experts often lack the means of analyzing results of large simulation ensembles, understanding relevant processes, and assessing the robustness of conclusions driven from the resulting simulations. As visualized in Figure 4.4, the key characteristics of the simulation data sets include the following: (a) multi-variate, (b) multi-modal (temporal, spatial, hierarchical, graphical), (c) multi-layer, (d) multi-resolution, and (e) inter-dependent (i.e., observations of interest depend on and impact each other). In particular, simulations may track 100s or 1000s of inter-dependent parameters, spanning multiple layers and spatial-temporal frames, affected by complex dynamic pro- cesses operating at different resolutions. Moreover, generat- ing an appropriate ensemble of stochastic realizations may require multiple simulations, each with different parameter settings corresponding to slightly different, but plausible, scenarios. As a consequence, running simulations and interpreting simulation results (along with the real-world observations) to generate timely actionable results are difficult. We argue that these challenges can be significantly alleviated using a data-driven approach that addresses the following fundamental questions:

- Given a large parameter space and fixed budget of simulations, can we decide which simulations to execute in the ensemble? Can we revise the ensemble as we receive a stream of real world observations?

- Can we compare a large number of simulation ensembles and observations (under different parameter settings) to identify their similarities and differences? Can we analyze one or more simulation ensembles to discover patterns and relationships between input parameters, key events/interventions, and simulation outcomes? Can we discover key events and summarize a large simulation ensemble to highlight these events? Can we classify these key events?

- Can we search and explore simulation ensembles based on the underlying key events or the overall simulation similarities? Can we keep track of the most relevant and most outlier simulations in an ensemble as we receive a stream of real world observations?

4.4.3 System Overview

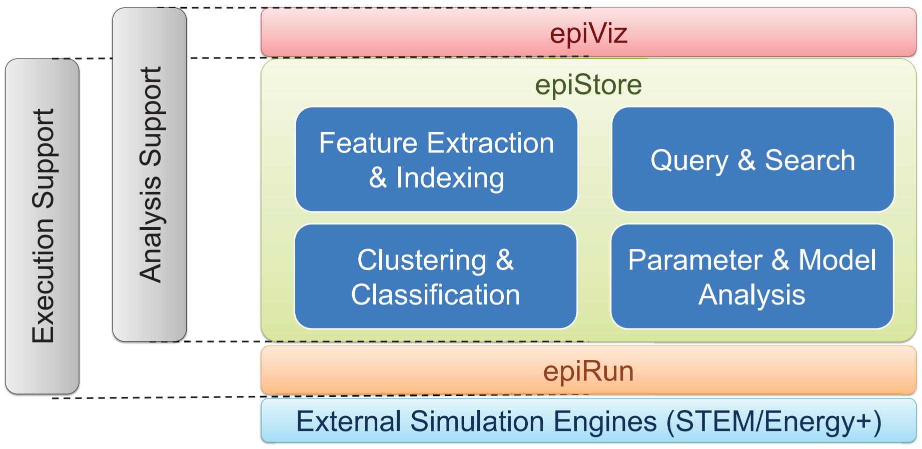

epiViz
epiStore
Feature Extraction & Indexing
Query & Search
Clustering & Classification
Parameter & Model Analysis
epiRun
External Simulation Engines (STEM/Energy+)
Execution Support
Analysis Support
Figure 4.5: simDMS system overview

Figure 4.5 depicts an overview of SIMDMS system which aims at assisting users to explore large simulation ensembles. In particular, it supports:

- analysis and indexing of simulation data sets, including extraction of salient multi-variate temporal features from inter-dependent parameters (spanning multiple layers and spatial-temporal frames, driven by complex dynamic processes operating at different resolutions) and indexing of these features for efficient and accurate search and alignment;

- parameter and feature analysis, including identification of unknown dependencies across the input parameters and output variables spanning the different layers of the observation and simulation data. These, and the processes they imply, can be used for understanding and refining the parameter dependencies and models.

4.5 DataStorm-FE

Data- and model-driven computer simulations are increasingly critical in many application domains. Yet, several critical data challenges remain in obtaining and leveraging simulations in decision making. Simulations may track 100s of parameters, spanning multiple layers and spatial-temporal frames, affected by complex inter-dependent dynamic processes. Moreover, due to the large numbers of unknowns, decision makers usually need to generate ensembles of stochastic realizations, requiring 10s-1000s of individual simulation instances. The situation on the ground evolves unpredictably, requiring continuously adaptive simulation ensembles. We introduce the DataStorm framework for simulation ensemble management, and demonstrate its DataStorm-FE data- and decision-flow and coordination engine for creating and maintaining coupled, multi-model simulation ensembles. DataStorm-FE enables end-to-end ensemble planning and optimization, including parameter-space sampling, output aggregation and alignment, and state and provenance data management, to improve the overall simulation process. It also aims to work efficiently, producing results while working within a limited simulation budget, and incorporates a multivariate, spatiotemporal data browser to empower decision-making based on these improved results.

4.5.1 Introduction

Data- and model-driven computer simulations are increasingly critical in many application domains. For example, when predicting the evolution of epidemics and assessing the impact of interventions, experts often rely on epidemic models and simulation software, such as GLEaM gle (2019) and STEM STEM (2016), and simulation ensemble tools, such as EpiDMS Liu *et al.* (2016). Similarly, data-driven computer simulations for disaster preparedness and response can play a key role in predicting the evolution of disasters and effectively managing emergencies through intervention measures.

4.5.2 Challenge

Yet, several critical data challenges remain in obtaining and leveraging simulations in decision making. Disaster simulations, for example, need to track 100s of inter-dependent parameters, spanning multiple models and geo-spatial frames, affected by complex inter-dependent dynamic processes operating at different resolutions. This is a major challenge due to overlapping and cascading processes, especially when involving multi-hazard scenarios where one hazard (e.g. flooding) is the gateway to the next (e.g. an epidemic). Yet, today's silo-based, de-coupled simulation engines assume that disaster, population dynamics, transportation, and disease/epidemic simulations are not integrated, failing to provide an end-to-end view of the disaster and preventing timely and informed decision making.

Moreover, due to the large number of unknowns, decision makers usually need to generate ensembles of stochastic scenarios, requiring 10s or 1000s of individual simulation instances, each with different parameter settings corresponding to distinct plausible scenarios. Yet, while existing simulation systems provide decision support for well-specified scenarios, when decision making and knowledge discovery in the presence of incomplete information are considered, there is little support for simulation ensemble planning, optimization, and management. In particular, execution of simulation ensembles can be very costly, which leads to simulation budget constraints restricting the number of simulations one can include in an ensemble. To support effective decision making, one must answer the question "Given a parameter space and a fixed simulation budget, which simulation instances we should the ensemble include in order to obtain models with good fit and low complexity?" In addition, since simulation context can dynamically and unpredictably evolve over time (due to for example how a disaster develops and the preventive and reactive actions taken by individuals), continuous adaptation of simulation ensembles is necessary. In particular, as the data arrives in a streaming fashion, simulation ensembles need to be continuously revised and refined as the situation on the ground changes: (a) revisions involve incorporating real-world observations as well as updated probability densities into existing simulations to alter their outcomes; (b) refinements include identifying new simulations to run, incorporating the changing situation on the ground to support improved decision-making.

My research team introduces the DataStorm framework for creation, storage, analysis, and exploration of coupled, multi-model simulation ensembles. At the core of DataStorm is the DataStorm-FE data- and decision-flow engine and coordination engine for creating and maintaining coupled, multi-model simulation ensembles. DataStorm-FE provides a means for coordinating data- and decision- flows among partially-connected simulation engines, and enables end-to-end ensemble planning and optimization (including parameter-space sampling, output aggregation and alignment, continuous data streaming, and state and provenance data management) to improve the predictive accuracy of the overall end-to-end simulation process within a limited simulation budget.

Additionally, DataStorm provides a decision support infrastructure for results obtained by DataStorm-FE, permitting users to query, explore, or visualize relevant data to facilitate the decision-making process.

4.6 Conclusion

Those platforms mentioned above provide metadata and event-driven analysis and visualization of simulation ensembles to assist decision makers to query and explore ensemble simulations and decide strategies such as which additional simulations to execute, which route people can use to evaculate civilians and what actions need to be taken to minimize the impact of natural disasters.

Chapter 5

OLSH: ONION-LSH FOR APPROXIMATE TOP-K QUERY PROCESSING

5.1 Introduction

So far we have discussed information retrieval tasks of multi-variate time series such as pattern matching and classification by leveraging the novel feature extraction process. The motivation is to answer small queries from relatively big data by accessing a small amount of information from the data, the idea of which can be extended to top-k query processing in multi-dimensional space. Therefore, I propose a framework which first learns the structure (importance) of each portion of data, followed by a hashing process for the purpose of fast retreival in later phase, with limited hashing resources. I will first address the problem, then I will go over the proposed OLSH framework and how it can answer top-k queries in high dimensional space.

Essentially the intuition is to deal with a large quantity of big data with limited computation resources such as RAM by accessing a bounded amount of small data, which leads to the motivation of *making big data small for queries.* The methods developed upon this principle is that not all data contributes equally to the final result set hence not all data is necessary for answering queries.

5.2 Problem Definition

Given a database, D, which contains data points that belong to a d-dimensional non-negative space [1] , $S(=\mathbb{R}^d_{\geq 0})$, we define a top-$k$ user preference query, as a vector, q, belonging to S, such that the user is seeking a resulting set, $R \subseteq D$, of objects

[1]In this paper, to ensure the monotonicity of the queries relative to the data, we assume that both data and query vectors have all non-negative components.

where $|R| = k$ and

$$\forall_{p_1 \in R} \nexists_{p_2 \in D \setminus R} \ \ q.p_1 < q.p_2.$$

Note that, when processing queries approximately, the result set R may not satisfy the above constraint. In that case, we define the accuracy, $acc(R, q)$ of the result set R relative to the query q as

$$acc(R, q) = 1 - \frac{|M(R, q) \ominus M(R_{opt}, q)|}{k},$$

where R_{opt} is an accurate solution to the top-k problem, $M(X, q)$ indicates the multi-set of matching scores between the query q and elements of the set X, and $\ominus$ indicates the difference operator for multi-sets. The corresponding error rate is defined as $\delta(R, q) = 1 - acc(R, q)$.

Based on the above, we formulate the *index structure design problem* we are aiming to address in this paper as follows: Given

- a database, D, which contains data points that belong to a multi-dimensional space, S,
- a user preference query, q, defined as a vector in the space S,
- a memory budget, B, and
- a bound, m, on the maximum rank to be supported,

our goal is to create an index structure, such that

- the memory footprint of the index structure is less than B and
- for a top-k query, with $k \leq m$, the corresponding error rate is minimized.

In addition to these, we also wish that the index structure leads to *as little data access* as possible during the execution of the top-k query.

5.3 Onion-LSH (OLSH) for Approximate top-k Processing

As we briefly introduced in Section 5.1, here, we propose a novel *Onion-LSH* (OLSH) index structure to *approximately* solve top-k queries. At the highest level, OLSH relies on the Onion technique Chang *et al.* (2000), to organize the data in layers of convex-hulls, which we refer to as o-layers, to produce top-k results. Unlike prior works, however, to reduce the amount of data access, each o-layer is further organized into (inner product based) LSH layers, referred to as h-layers.

In LSH based retrieval schemes, there is an inherent trade-off between available hashing resources and the accuracy. Since basic LSH does not provide error guarantees for top-k retrieval results (but instead targets accuracy guarantees based on distances), in this section, we discuss how to control this trade-off using an accuracy model that relates accuracy target with the number of h-layers created. We further complement the proposed OLSH index structure with a layer-aware (LA) resource allocation strategy, which takes into account the distribution of the data, the number, k, of results required, and the user's target accuracy, to allocate available hashing resources among the available o-layers.

5.3.1 *Overview of Onion Indexing*

The Onion technique Chang *et al.* (2000) is a layer-based data organization to solve the top-k search problem: the data in the multi-dimensional space are partitioned to convex-hull layers. More specifically, the outermost layer is the convex hull for the entire data set, the second outermost layer is the convex hull for the data set *minus* the data in the outermost layer, and so on.

Onion leverages the geometric properties of the convex hull, which guarantees that the optimal value for any *linear preference* function (which can be represented in the form of an inner product between a query and data vectors) will always be found at one (or more) of its vertices. Given a top-k query, the Onion technique searches for the top result in the outermost layer, searches the 2^{nd} best result in the two outermost layers, and so on, up to the k outermost layers combined for the last (k^{th}) data object in the result. Figure 2.2 provides an example. Here the data is partitioned into four onion layers: the outermost layer consists of points $O_1 = \{p_1, p_2, p_3, p_4\}$; the second outermost layer includes points $O_2 = \{p_5, p_6, p_7\}$; the next layer has $O_3 = \{p_8, p_9, p_{10}\}$, and the very last layer includes two points, $O_4 = \{p_{11}, p_{12}\}$. Given a query, q, to obtain top-2 results, we would first find the top-ranked entry among $\{p_1, p_2, p_3, p_4\}$ in the outermost layer and we would then find the next result among $\{p_j | j = 1, 2, \ldots, 7\}$, combining the two outermost layers.

One major challenge with the basic Onion technique is that, for large data sets, the number of elements in the outermost layers can also be large and this can lead to a large number of candidates to be considered to identify the actual top-k objects. Heo *et al.* (2010) proposed a combination of layer-approach and the list-based ones, where within each layer, the data is organized in a way that leverages a list-based ranked-data enumeration approach, typically Fagin *et al.* (2003). OLSH, instead, relies to an LSH based organization of data in onion layers to help scale to higher dimensional spaces.

5.3.2 OLSH: LSH Indexing of Onion Layers

Let $O_i \subseteq D$ be an onion layer (or o-layer). OLSH organizes the data on this o-layer into LSH data buckets, in a way that provides approximate search for objects matching the query, q. More specifically, for i^{th} o-layer, we pick two parameters, κ_i and L_i, that control the precision and recall, respectively, for that layer. As discussed in Section 2.2, concatenating κ_i independently selected hash functions will help reduce the number of false positives. More specifically, the data in O_i are partitioned into buckets using the combined hash function. We refer to each such grouping of κ_i hash functions as a hash-layer (or h-layer) of the corresponding Onion layer. In order to reduce the misses, we then create L_i many such h-layers and union the results obtained from each of the corresponding buckets, the idea being that if a result is missed by an individual h-layer, it is less likely that it will be missed simultaneously by multiple h-layers.

Definition 18 (OLSH Index) *Let us consider a data set, D, indexed using OLSH for top-k search for $k \leq m$. Let us denote the set of data points in the i^{th} Onion layer (o-layer) as O_i. Let us further denote the number of hash layers (h-layers) and the number of hashes per h-layer for the i^{th} Onion layer as L_i and κ_i as before.*

We represent this OLSH index with a 5-tuple $\langle m, \mathcal{O}, \mathcal{K}, \mathcal{L}, \mathcal{H} \rangle$, where $\mathcal{O} = \{O_1, \ldots, O_m\}$, $\mathcal{K} = \{\kappa_1, \ldots, \kappa_m\}$, $\mathcal{L} = \{L_1, \ldots, L_m\}$, and $\mathcal{H}$ is a suitable hash family.

◇

As we have seen in Section 2.2, the hash family, $\mathcal{H}$, used for mapping the data into hash buckets needs to be compatible with the underlying distance/similarity function. LSH families have been explored for different distances or similarities, including l_p distance (LSH with $p-$stable distributions), Hamming distance, Jaccard coefficient, and so on Indyk and Motwani (1998); Datar *et al.* (2004); Neyshabur and Srebro (2015); Shrivastava and Li (2014); Huang *et al.* (2018). Since, for top-k queries considered in this paper, the degree of match between the query, q, and the data point, p, is defined based on their inner (or dot) product, for OLSH we need a hashing scheme that supports inner product search.

LSHs for Inner Product Search: Given an Onion layer, O_i, the problem we are facing is to organize the data into h-layers in a way to efficiently answer a query, q, described in terms of linear weights on data attributes. In particular, we are looking for a point, $r_i \in (O_1 \cup \ldots \cup O_i)$, that has a large inner product with the query vector, q, but was not returned as a result for any of the outer $i-1$ o-layers:

$$r_i = \underset{x \in ((O_1 \cup \ldots \cup O_i) \setminus \{r_1, \ldots, r_{i-1}\})}{argmax} q.x$$

This indicates that we need a locality sensitive hash function that will provide accuracy guarantees under inner product similarity.

In the original LSH formulation Indyk and Motwani (1998); Motwani *et al.* (2006), the locality sensitive hash function is symmetric in the sense that the same hash function is applied to both data objects p and query object q; moreover, p and q are points in the same vector space. Shrivastava and Li (2014) argued that there is no symmetric locality sensitive hash function which can maximize the inner product for the entire vector space. The authors also showed, however, that when queries are normalized and data vectors are bounded, an asymmetric hash function, which applies two different hash functions to data and query objects, can be developed to answer such inner product based queries. Neyshabur and Srebro (2015) has shown that under certain conditions, one can also develop a symmetric locality sensitive hash function with performance guarantees. In particular, when both query and data vectors are bounded inside unit sphere (i.e., when $||x|| \leq 1$, where x is a query or data vector), there does exists a parameter-free, universal, symmetric LSH.

Note that, in our problem context, given a vector, q, the norm $||q||$ does not affect the outcome. Therefore, we can rescale all vectors in the data set without changing the $argmax$. Thus, in designing OLSH, we rely on a symmetric hash function, which first applies the following transformation, from the original vector space into a 2D-space, to the (data or query) vector, x,

$$P(x) = [\ x; \sqrt{1 - ||x||_2^2}\ ],$$

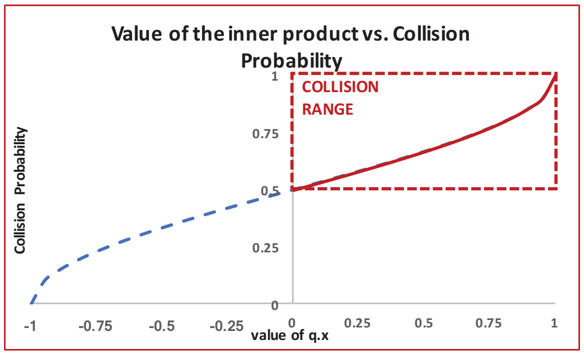


Figure 5.1: Inner product similarity vs collision probability

and then hashes the vector with the following random mapping into the binary alphabet $\Gamma = [\pm 1]$:

$$h_\alpha(x) = sign(\alpha^T P(x)), \tag{5.1}$$

by picking a spherical random vector $\alpha \sim \mathcal{N}(0, I)$ Neyshabur and Srebro (2015).

Accuracy Guarantee of a Single Hash Function: As we discussed earlier, LSH-based index structures utilize multiple hash functions from the same hash family to match the accuracy guarantees that the user or the application requires. This, however, necessitates quantifying the collision probability of a query and a data point for a single hash function. Neyshabur and Srebro (2015) provided the following collision probability for the symmetric hash function introduced above:

$$Pr[h_\alpha(P(q)) = h_\alpha(P(x))] = 1 - \frac{cos^{-1}(q.x)}{\pi}.$$

Since we have normalized q such that $\|q\| = 1$ and also scaled the data vectors such that $\|x\| \leq 1$, we have $-1 \leq q.x \leq 1$, this in general leads to the (blue) curve shown in Figure 5.1.

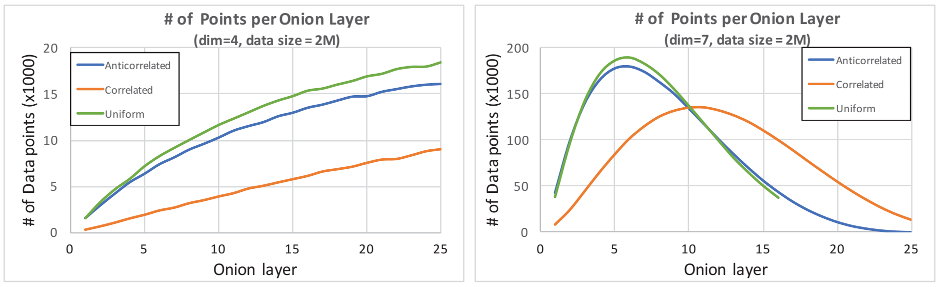
The shape of this collision probability curve poses two challenges for our top-k retrieval problem. First of all, we are given a monotonic query, q, which is constrained to have non-negative values in all data dimensions. As described in Section 5.2, in this paper, we assume that the data set is constrained to have non-negative values for all attributes. Given these, the inner product between a query, q, and a data vector, v, will inherently have a non-negative value and, consequently, the collision probability between query and data points will be greater than 0.5 (see Figure 5.1). Consequently, false positive rates are likely to be high and, in order to keep the number of redundant points that need to be considered as candidates low, we need be especially careful in selecting the number, κ, of hash functions to be concatenated in a given hash layer. Secondly, as we see on the rightmost corner of the curve, the collision probability drops sharply as the value of $q.x$ moves away from a perfect match with 1.0 score, before stabilizing in a more linear pattern. This also indicates that we need to select the number, L, of h-layers carefully to ensure that the top-k retrieval accuracy is maintained sufficiently high.

5.3.3 Accuracy/Resource Trade-offs in OLSH

In traditional LSH, the collision probability for a given hash layer is computed as ψ^{κ}, where ψ is the collision probability (say for a given distance/similarity target) and κ is the number of hash functions concatenated in that layer. This is because hash functions are selected independently and, for any object to appear in the result of a given hash layer, it needs to collide with the query for all κ hash functions in the layer. The collision rate for the entire LSH index (consisting of L hash layer) is, then, $1-(1-\psi^{\kappa})^{L}$, since the only way a data object will fail to appear in the result is if it does not collide with the query for any of the L layers.

In the case of OLSH, not all Onion layers contribute to the result set of a top-k query equally. We therefore need to formulate the accuracy measure in a way that takes into account the contribution of different Onion layers to the final result:

- As we discussed earlier, the top ranked result will certainly reside on the outermost o-layer, the second best result will be in one of the two outermost o-layers



(a) 3D data sets (b) 7D data sets

Figure 5.2: Partitioning of data points among Onion layers, for three sample data sets with different data distributions

and so on. This indicates that outer o-layers are, in general, more critical since (ignoring the data distribution) they are likely to contribute more to top-k result.

- On the other hand, as we see in Figure 5.2, the sizes of the layers often depend on their depths and the dimensionality of the data. This implies that, unless we consider the data distribution across the Onion layers when assigning the resources, deeper layers, which do not produce many top-k results, may nevertheless produce a large number of useless candidates.

We next study the accuracy resource trade-offs in OLSH design.

Resource Consumption: Let us consider a data set, D, indexed using an OLSH index $\mathbb{O} = \langle m, \mathcal{O}, \mathcal{K}, \mathcal{L}, \mathcal{H} \rangle$ as formalized in Definition 18. The data elements corresponding to an Onion layer have to be indexed once for each of the corresponding hash layers. Consequently, the total resource consumption of the OLSH index, $\mathbb{O}$, can be computed as

$$resource(\mathbb{O}) \sim \sum_{i=1}^{m} L_i \times |O_i|.$$

Number of Enumerated Candidates: Let ρ_i denote the average bucket size for a given h-layer in the i^{th} Onion layer. Then, the number, $cand(\mathbb{O}, k)$, of enumerated candidates for a top-k query ($k \leq m$) is such that

$$\sum_{i=1}^{k} \rho_i \leq cand(\mathbb{O}, k) \leq \sum_{i=1}^{k} L_i \times \rho_i.$$

Note that the value of ρ_i depends on the number, $|O_i|$, of data elements in the o-layer, the distribution of these points in the space, the properties of the hash family $\mathcal{H}$, as well as the number κ_i of hash functions per h-layer. In particular, let avg_i be the average inner product among the points in O_i; given this, we can associate an average per-hash function collision probability $\psi_{avg,i}$ to the pairs of points in the Onion layer (see Figure 5.1). Given this collision probability, we can compute the average *non-empty* bucket size, $\rho_{ne,i}$, as follows:

$$\rho_{ne,i} = 1 + \psi_{avg,i}^{\kappa_i} \times (|O_i| - 1).$$

Intuitively, for any non-empty bucket, there is at least one data object *plus* $\left(\psi_{avg,i}^{\kappa_i} \times (|O_i| - 1)\right)$ many other data points that collide with it in the given h-layer.

Note, however, that in general, there is nothing that guarantees that a top-k query, q, will hit a non-empty bucket. Therefore, when computing ρ_i we need to take into account also empty buckets. In particular, since the hash function we utilize for inner-product search is binary (see Equation 5.1 in Section 5.3.2), given κ_i, we can create up to 2^{κ_i} hash buckets, which (assuming non-biased data distribution within o-layers), would give us

$$\rho_i = \frac{|O_i|}{2^{\kappa_i}}.$$

This implies that if $2^{\kappa_i} > |O_i|$, on the average, each h-layer will produce less than 1 result, independently of the underlying collision probability. If, however, $2^{\kappa_i} \ll |O_i|$, this will increase the average bucket size (and the average non-empty bucket size) resulting in a large number, $cand(\mathbb{O}, k)$, of candidates. Since only a small number, k, out of these candidates can be true results, ideally, we should have $2^{\kappa_i} \sim |O_i|$ or, equivalently, $\kappa_i \sim log_2(|O_i|)$.

Error Rate: To compute the error rate for OLSH under a given parameter setting, $\mathbb{O} = \langle m, \mathcal{O}, \mathcal{K}, \mathcal{L}, \mathcal{H} \rangle$, we need to consider that the different onion layers (a) are allocated potentially different amount of resources and (b) have different contributions to the result set of the top-k query. In particular, we have seen that highest ranked result is guaranteed to come from the outermost layer and, in general, i^{th} onion layer have the potential to contribute to any layer at depth i or more. While this intuitively implies that the outer layers are likely to be more important, we also need to consider the fact that inner layers tend to have more points on them (as we have seen in Figure 5.2). This is visualized in Table 5.1: the outermost Onion layer is guaranteed to produce the top ranked result. However, when we consider the lower ranked results, the chance that a deeper Onion layer may contribute becomes non-zero. Thus, when computing the likelihood of the i^{th} Onion layer to produce the h^{th} rank result, we need to consider the size of the i^{th} o-layer relative to the sum of the sizes of all o-layers up to (and including) the h^{th} o-layer. Let us consider a data set partitioned into m outermost Onion layers. Given a top-k query ($k \leq m$), the likelihood of the Onion layer i to produce the h^{th} rank result can be computed as

$$c_{i,h} = \begin{cases} 0 & \text{if } h < i \\ \frac{|O_i|}{\sum_{j=1}^{h} |O_j|} & \text{else} \end{cases}$$

Note that this formulation does not assume any information about the distribution of data across Onion layers, other than their sizes relative to each other. In practice, more precise likelihoods might be obtained through data and query statistics. Having quantified the likelihood of the Onion layer i to produce the h^{th} rank result for a given top-k query, q, we can then compute the expected error rate for a given OLSH index $\mathbb{O} = \langle m, \mathcal{O}, \mathcal{K}, \mathcal{L}, \mathcal{H} \rangle$ for this query as follows:

$$err(\mathbb{O}, k) = \frac{1}{k} \times \sum_{h=1}^{k} \sum_{i=1}^{h} \left(c_{i,h} \times \left((1 - \psi_i^{\kappa_i})^{L_i} \right) \right),$$

Table 5.1: Expected layer contributions of different Onion layers to different ranks in top-5 retrieval

	o-layer 1	o-layer 2	o-layer 3	o-layer 4	o-layer 5
rank1	1	NA	NA	NA	NA
rank2	$\frac{\lvert O_1\rvert}{\sum_{j=1}^{2}\lvert O_j\rvert}$	$\frac{\lvert O_2\rvert}{\sum_{j=1}^{2}\lvert O_j\rvert}$	NA	NA	NA
rank3	$\frac{\lvert O_1\rvert}{\sum_{j=1}^{3}\lvert O_j\rvert}$	$\frac{\lvert O_2\rvert}{\sum_{j=1}^{3}\lvert O_j\rvert}$	$\frac{\lvert O_3\rvert}{\sum_{j=1}^{3}\lvert O_j\rvert}$	NA	NA
rank4	$\frac{\lvert O_1\rvert}{\sum_{j=1}^{4}\lvert O_j\rvert}$	$\frac{\lvert O_2\rvert}{\sum_{j=1}^{4}\lvert O_j\rvert}$	$\frac{\lvert O_3\rvert}{\sum_{j=1}^{4}\lvert O_j\rvert}$	$\frac{\lvert O_4\rvert}{\sum_{j=1}^{4}\lvert O_j\rvert}$	NA
rank5	$\frac{\lvert O_1\rvert}{\sum_{j=1}^{5}\lvert O_j\rvert}$	$\frac{\lvert O_2\rvert}{\sum_{j=1}^{5}\lvert O_j\rvert}$	$\frac{\lvert O_3\rvert}{\sum_{j=1}^{5}\lvert O_j\rvert}$	$\frac{\lvert O_4\rvert}{\sum_{j=1}^{5}\lvert O_j\rvert}$	$\frac{\lvert O_5\rvert}{\sum_{j=1}^{5}\lvert O_j\rvert}$

where ψ_i is the per-hash function success probability for the hash functions used to index the i^{th} onion layer.

5.3.4 OLSH Design Criteria

Given the resource and accuracy trade-offs formalized in the above section, we can then restate the OLSH index design criteria as follows: Let D be a data set. An OLSH index, $\mathbb{O} = \langle m, \mathcal{O}, \mathcal{K}, \mathcal{L}, \mathcal{H} \rangle$, to answer top-$k$ queries for this data set must satisfy the following:

$$\begin{aligned} &\text{minimize} \quad err(\mathbb{O}, m) \\ &\text{subject to} \quad resource(\mathbb{O}) \leq resource_{max}, \end{aligned}$$

where $resource_{max}$ is the maximum resource allocated for the OLSH index. In addition, we would like to keep the number, $cand(\mathbb{O}, k)$, of candidate results produced during the processing of top-k queries ($k \leq m$) as low as possible.

5.3.5 Overview of OLSH

In Algorithms 1 through 3, we present the pseudo-codes for construction and use of the OLSH index structures for top-k query processing.

Constructing OLSH: Algorithms 1 and 2 present the outline of the process through which we create an OLSH index for a given data set D. As we see in Algorithm 1, the

Algorithm 1 Construct_OLSH

```
Input: database $D$, maximum rank $m$, budget $B$, hash family, $\mathcal{H}$
Output: OLSH index structure, $\mathbb{O}$

1: $\mathcal{O}$ = OnionLayerPartition($D$, $m$)
2: $\mathcal{K}$ = AssignPerLayerHashCounts($\mathcal{O}$, $m$, $\mathcal{H}$)
3: $\mathcal{L}_0$ = SolveNonLinear($\mathcal{O}$, $\mathcal{K}$, $m$, $B$, $\mathcal{H}$)
4: $\mathcal{L}$ = ReviseLayers($\mathcal{O}$, $m$, $B$, $\mathcal{K}$, $\mathcal{L}_0$, $\mathcal{H}$)
5: return $\mathbb{O}$ = computeHashTables($m$, $\mathcal{O}$, $\mathcal{K}$, $\mathcal{L}$, $\mathcal{H}$)
```

process takes as input, in addition to a data set, the maximum rank to be supported as well as a suitable hash family. The first step of the algorithm is to extract the m outermost Onion layers from the given data set [2] . In the second step, we compute the per-layer hash counts for each Onion layer, based on the corresponding number of data elements.

In the third step, we solve the optimization problem formulated in the previous subsection. Note that the OLSH design problem is non-linear as the relationships among κ_i, L_i, and the i^{th} layer contribution to the error term are non-linear. The problem is somewhat simplified when, as discussed in Section 5.3.3, we fix $\kappa_i = log_2(|O_i|)$ to minimize the number of candidates – in this case, the numbers, L_i, of hash layers for the different Onion layers are the only unknowns in the optimization problem. Nevertheless, the problem remains non-linear even when after fixing κ_i and, consequently, we need a non-linear solver to obtain an OLSH design for the given problem setting. In our implementation, we use Mathematica's numeric optimizer, `NMinimize`, to solve this problem. However, given the non-linear nature of the problem, obtaining

[2]If the data set does not produce m Onion layers, the top-k search algorithm is suitably modified to deal with this case. For simplicity, we ignore this special case here.

an optimal solution is not feasible. Instead, we seek an approximate solution to the problem by fixing the number of iterations for the optimizer. This, however, potentially leads to sub-optimal designs, where not all available resources are utilized to bring down the error rate. To tackle this problem, after obtaining an initial design $\mathbb{O}_0 = \langle m, \mathcal{O}, \mathcal{K}, \mathcal{L}_0, \mathcal{H} \rangle$, we incrementally improve the design by allocating new hash layers to different onion layers in a way that maximally improves the accuracy. We achieve this by computing, for each Onion layer i, a per-layer resource demand term

$$demand(O_i) = |O_i|$$

and a degree of potential for improvement

$$potential_0(O_i) = err(\mathbb{O}_0) - err(\mathbb{O}_0^{(i\ddagger+1)}),$$

where $L_{i,0}$ is the hash layer assignment for Onion layer i based on the initial solution $\mathbb{O}_0$ and $err(\mathbb{O}_0^{(i\ddagger+1)})$ is the error one would observe if the hash layer assignment for Onion layer i is incremented by one:

$$err(\mathbb{O}_0^{(i\ddagger+1)}) = \frac{1}{k} \times \sum_{h=1}^{k} \sum_{i=1}^{h} \left(c_{i,h} \times \left((1 - \psi_i^{\kappa_i})^{L_{i,0}+1} \right) \right).$$

Given these two terms, for improvement, we select the layer i with the largest potential among those layers with resource demands that are less than the available resources,

$$demand(O_i) \leq resource_{max} - resource(\mathbb{O}_0),$$

and we increase the number of hash layers assigned to this onion layer by one. This gives us a revised resource allocation, $\mathbb{O}_1 = \langle n, \mathcal{O}, \mathcal{K}, \mathcal{L}_1, \mathcal{H} \rangle$. The process, then, is repeated (as shown in Algorithm 2) until the resources have been used up such that no more improvements is possible.

Top-k Search with OLSH: The outline of the top-k search process is given in Algorithm 3. As we see here, the algorithm starts with the first Onion layer and

Table 5.2: Experimental parameters (default values are highlighted in bold)

Parameter	Tested values
Data cardinality	100K, **200K**, 500K, 1M, 1.5M, 2M
Data dimensionality	2, 3, **4**, 5, 6, 7
Data distribution	Independent (Ind), Correlated (Cor), Anticorrelated (Anti)
Max. rank (m)	**25**, 50
Target data redundancy (r)	**2**, 3, 4
Collision probability (ψ)	0.65, **0.75**, 0.85
Per h-layer num hashes (κ)	default = $\mathbf{log_2(\|O_i\|)}$, default + 3, default - 3, uniform

fetches the top-k results, since that layer can potentially contribute up to k elements to the results set. Then, the algorithm visits the deeper Onion layers one-by-one and, from each, it pulls as many results as it can potentially contribute to the result set; in particular, since i^{th} Onion layer can produce at most the i^{th} best result in the result set, from that layer, the algorithm fetches $k - i + 1$ best results relative to the given query, q. The final result is than obtained by combining partial results from the k outermost layers and selecting the best k out of them.

5.4 Experimental Evaluation

In this section, we evaluate the effectiveness of the proposed OLSH hashing scheme for top-k query processing and the underlying layer-aware (LA) resource allocation technique. For these evaluations, we use both synthetic datasets (where we control the

Recall (top-25 query)									
	Anticorrelated			Correlated			Independent		
	OLSH	LSH-MU	LSH-SU	OLSH	LSH-MU	LSH-SU	OLSH	LSH-MU	LSH-SU
w/o Revision	95.1	83.5	81.2	93.3	90.4	89.8	86.3	70.1	72.6
w Revision	97.7	N/A	81.2	93.5	N/A	91.3	89.6	N/A	72.6

Num Candidates (top-25 query)									
	Anticorrelated			Correlated			Independent		
	OLSH	LSH-MU	LSH-SU	OLSH	LSH-MU	LSH-SU	OLSH	LSH-MU	LSH-SU
w/o Revision	6747.3	7482.2	6880.1	4253.2	4990.7	4779.4	3926.8	5203.7	5378.4
w Revision	7067.2	N/A	6880.1	4253.5	N/A	4882.9	4043.8	N/A	5378.4

$\frac{\textbf{Recall}}{\textbf{Num Candidates}}$ (top-25 query)									
	Anticorrelated			Correlated			Independent		
	OLSH	LSH-MU	LSH-SU	OLSH	LSH-MU	LSH-SU	OLSH	LSH-MU	LSH-SU
w/o Revision	0.0141	0.0112	0.0118	0.0219	0.0181	0.0188	0.0220	0.0135	0.0135
w Revision	0.0138	N/A	0.0118	0.0220	N/A	0.0187	0.0222	N/A	0.0135

$\frac{\textbf{Num Candidates}}{\lvert O_1 \cup O_2 \cup \ldots \cup O_{25} \rvert}$ (top-25 query)									
	Anticorrelated			Correlated			Independent		
	OLSH	LSH-MU	LSH-SU	OLSH	LSH-MU	LSH-SU	OLSH	LSH-MU	LSH-SU
w/o Revision	7.1%	7.9%	7.2%	6.7%	7.8%	7.5%	3.7%	4.9%	5.1%
w Revision	7.4%	N/A	7.2%	6.7%	N/A	7.7%	3.8%	N/A	5.1%

Table 5.3: Recall and # of candidates considered for different distributions (200K objects, target redundancy 2, default κ)

data distribution and dimensionality) and real data sets with different characteristics. All experiments were executed on Linux machines with 8 CPUs and 32 GB RAM, running Ubuntu 16.04.

5.4.1 Datasets

Synthetic Data

As synthetic data, we use the standard benchmarks for top-k and preference queries Borzsony *et al.* (2001). Table 5.2 provides an overview of the parameters (cardinality,

Recall (top-25 query)						
	Anticorrelated		**Correlated**		**Independent**	
	OLSH	LSH-SU	OLSH	LSH-SU	OLSH	LSH-SU
2D	92.6	93.3	97.6	99.2	96.2	90.4
3D	96.6	81.0	97.2	92.3	90.2	80.2
4D	97.7	81.2	93.5	91.3	89.6	72.6
5D	90.5	65.5	91.0	88.2	80.3	56.4
6D	64.1	33.7	83.7	76.9	75.6	34.8
7D	31.7	19.4	72.5	65.2	48.4	27.2

Num Candidates (top-25 query)						
	Anticorrelated		**Correlated**		**Independent**	
	OLSH	LSH-SU	OLSH	LSH-SU	OLSH	LSH-SU
2D	577.0	611.2	320.8	366.6	557.2	502.2
3D	2426.0	2139.1	1471.9	1441.6	2089.0	2277.8
4D	7067.2	6880.1	4253.5	4882.9	4043.8	5378.4
5D	10471.6	12141.8	11458.6	11692.9	5582.5	8503.7
6D	8445.6	11981.5	12355.5	12818.0	6364.9	6199.6
7D	7586.6	9946.0	12470.3	12244.2	6476.9	7577.0

$\frac{\textbf{Recall}}{\textbf{Num Candidates}}$ **(top-25 query)**						
	Anticorrelated		**Correlated**		**Independent**	
	OLSH	LSH-SU	OLSH	LSH-SU	OLSH	LSH-SU
2D	0.1604	0.1527	0.3041	0.2706	0.1727	0.1800
3D	0.0398	0.0378	0.0661	0.0640	0.0432	0.0352
4D	0.0138	0.0118	0.0220	0.0187	0.0222	0.0135
5D	0.0087	0.0046	0.0079	0.0075	0.0144	0.0066
6D	0.0076	0.0028	0.0068	0.0060	0.0119	0.0056
7D	0.0042	0.0020	0.0058	0.0053	0.0075	0.0036

Table 5.4: Recall and # of candidates considered for different κ strategies (200K objects, target average redundancy 2)

dimensionality, and distribution) we considered in the experiments.

Real Data

In addition to synthetic data sets described above, we have also considered two real data sets, commonly used as benchmarks in top-k and preference queries. The NBA

Recall						
	Anticorrelated		**Correlated**		**Independent**	
	OLSH	LSH-SU	OLSH	LSH-SU	OLSH	LSH-SU
top-1	99.4	94.6	99.1	97.5	94.6	85.4
top-2	99.4	93.9	99.2	97.7	94.5	83.6
top-5	99.3	90.9	97.8	96.4	93.6	80.0
top-10	99.1	87.2	95.6	93.6	92.6	76.9
top-25	97.7	81.2	93.5	91.3	89.6	72.6

Num Candidates						
	Anticorrelated		**Correlated**		**Independent**	
	OLSH	LSH-SU	OLSH	LSH-SU	OLSH	LSH-SU
top-1	390.2	243.4	129.8	88.9	331.7	108.4
top-2	1122.4	460.8	253.9	201.3	751.0	278.6
top-5	2446.3	1234.0	840.8	763.3	1424.6	760.1
top-10	3796.7	2442.3	1966.6	1519.5	2167.3	1348.5
top-25	7067.2	6880.1	4253.5	4882.9	4043.8	5378.4

Table 5.5: Recall and # of candidates considered for different top-k queries (200K objects, target average redundancy 2, index designed for up to top-25 retrieval)

[3] dataset consists of regular season statistics of NBA players that played at least ten minutes in a given season, from the 1951-1952 season to the 2017-2018 season. This dataset has 23338 unique elements with seven attributes each: `minutes_played`, `total_points`, `field_goals_attempted`, `free_throws_attempted`, `total_rebounds`, `total_assists`, and `total_personal_fouls`. The HOUSE [4] dataset contains 10717764 records, each holding six values that represent an American family's expenditures in `gas`, `electricity`, `water`, `heating`, `insurance`, and `property_tax` between 2013 and 2017.

[3]https://www.basketball-reference.com

[4]https://www.ipums.org

5.4.2 Evaluation Criteria

Our primary performance metric is recall (which we want to keep as close to 100% as possible), but we also report the number of candidates retrieved during the process (which we aim to keep as low as possible). For both synthetic and real-life datasets, we pick 1000 random queries for evaluation. Each experiment configuration was run 5 times (each with a different random hashing) and we report average.

5.4.3 OLSH and Competitors

Our goal is to evaluate the effectiveness of the proposed OLSH hashing scheme for top-k query processing; therefore, we have implemented OLSH and its underlying resource allocation schemes. To obtain the Onion layers through convex hull computation, we use the C++ implementation provided by Barber *et al.* (1996). To solve the optimization problem we formulate in this paper, we use the numerical optimization tool, `NMinimize`, of *Mathematica* (with differential evolution method run for 3000 iterations by default). For the default scenario, with $m = 25$, the optimization process underlying the computation of the layer-aware (LA) resource allocations takes ~ 110 seconds. For $m = 35$, the time cost is ~ 146 and, for $m = 50$, the process takes ~ 216 seconds, indicating that the complexity of the optimization cost is <u>*linear*</u> in the number of Onion layers. Note that this step is applied only once, during the index design time; therefore this has no impact on top-k query processing cost.

For inner product search, we adopt SIMPLE-LSH hash family Neyshabur and Srebro (2015), implemented in C++. When evaluating the `searchLSHOnionLayer()` function in Algorithm 3 for a given query, for each hash layer, we visit up to one nearby bucket (based an Hamming distance) to create a candidate set whose size is as close as possible to the number specified in Line 3 of the algorithm.

The OLSH design parameters considered in our evaluations are shown in Table 5.2. As we see here, we have considered OLSH indexes designed for two different maximum ranks ($m = 25$ and $= 50$). We also considered several alternative target redundancies: when the target redundancy is r, this means that we allocated sufficient memory for the index structure to store each object in the index structure (i.e., in the outermost m onion layers of the data) r many times. We have also varied the number of per hash layer hash functions: as discussed in Section 5.3.3, for the i^{th} Onion layer, we set the default value of the κ_i to $\mathbf{log_2}(|\mathbf{O_i}|)$ – to see the effectiveness of this default setup, we also consider larger $(default + 3)$ and smaller $(default - 3)$ values of κ_i, along with a uniform allocation scheme which allocates the same κ value (maximum of all κ_i) for all Onion layers.

To assess the effectiveness of the proposed Onion layering, we consider two alternative strategies: In **LSH-MU**, we take the maximum available hash resources (based on the target redundancy) and partition it uniformly among the Onion layers. In **LSH-SU**, we also uniformly partition the available resources among the Onion layers, but in this case, instead of considering the maximum available hash resources, we consider the amount of resources used by OLSH under the same conditions. Note that when OLSH uses all available resources under the given target redundancy, **LSH-MU** and **LSH-SU** strategies work similarly.

5.5 Results

We now present and discuss the results. We start with a general overview of the results based on the default setup.

Recall (top-k query)						
	Anticorrelated		**Correlated**		**Independent**	
	OLSH	LSH-SU	OLSH	LSH-SU	OLSH	LSH-SU
k:25,m:25	97.7	87.9	93.7	88.0	83.5	72.1
k:50,m:50	94.4	85.9	90.7	89.3	79.8	73.8

Num Candidates (top-k query)						
	Anticorrelated		**Correlated**		**Independent**	
	OLSH	LSH-SU	OLSH	LSH-SU	OLSH	LSH-SU
k:25,m:25	8081.3	9567.7	5482.3	5914.3	6344.0	7911.7
k:50,m:50	23492.1	26162.9	17083.9	18379.4	15831.9	17041.1

Table 5.6: Recall and # of candidates considered for index structures designed for different maximum target rank (200K objects, target average redundancy 2)

Recall (top-25 query)						
	Anticorrelated		**Correlated**		**Independent**	
κ	OLSH	LSH-SU	OLSH	LSH-SU	OLSH	LSH-SU
default	97.7	81.2	93.5	91.3	89.6	72.6
def.-3	98.5	90.8	97.7	96.8	93.3	83.7
def.+3	91.9	65.5	91.0	88.7	77.8	60.7
uniform	93.6	77.1	89.0	90.0	77.1	68.4

Num Candidates (top-25 query)						
	Anticorrelated		**Correlated**		**Independent**	
κ	OLSH	LSH-SU	OLSH	LSH-SU	OLSH	LSH-SU
default	7067.2	6880.1	4253.5	4882.9	4043.8	5378.4
def.-3	12494.3	12098.8	8098.9	6185.5	7864.5	10102.8
def.+3	4537.1	3325.6	3153.9	3293.5	3401.6	3601.5
uniform	5257.7	6983.4	4696.0	5254.9	4828.7	4777.0

Table 5.7: Recall and # of candidates considered for different κ assignment strategies (200K objects, target average redundancy 2)

5.5.1 Overview

We start the discussion of the results with Table 5.3, which presents recall and number of candidates enumerated for the default scenario, under different data distributions. In the table, we also consider the impact of the layer revision step (which distributes the hash resources remaining) after the optimization among the Onion layers, presented in Section 5.3.5 (Algorithm 2). As we see in the table, under all data distributions, OLSH provides significantly higher accuracies than both LSH-MU and LSH-SU strategies. The table also shows that OLSH is able to benefit from the layer revision process – indicating that we are able to address the loss of performance due to the approximate nature of the numeric non-linear optimizer. It is, however, important to see that, OLSH is able to provide better accuracies even without relying on layer revision.

What is especially impressive is that OLSH is able to achieve this without significantly increasing the number of candidates enumerated – in fact, in many cases, OLSH enumerates less candidates than both competitors. This trade-off is easiest to see in the rows of the table which reports the expected amount of recall gained per candidate enumerated during the query processing (i.e. *recall/num candidates*). As the corresponding two rows show, OLSH provides a significantly higher per-candidate recall gain than the alternative schemes. Moreover, as the last two rows illustrate, the OLSH index considers only a fraction $(3 - 8\%)$ of the data that reside in the first 25 Onion layers of the data set to achieve this high accuracy.

Due to space constraints in the rest of the paper, we report results only for the LSH-SU competitor.

5.5.2 Impact of Different Dimensionalities

Table 5.4 shows the accuracies and the number of candidates for data with different dimensionalities. As we see here, all algorithms suffer in terms of their accuracies. However, as expected, the OLSH index is significantly more robust than the alternative strategy in the presence of higher dimensional data sets. This is especially visible when we consider per-candidate recall gain, which remains significantly more robust for OLSH than the competitor as the number of dimensions increases; moreover, this robustness is especially strong for the two data sets (anticorrelated and uniform) that

Recall (top-25 query)						
	Anticorrelated		**Correlated**		**Independent**	
$\lvert D\rvert$	OLSH	LSH-SU	OLSH	LSH-SU	OLSH	LSH-SU
100K	93.6	75.2	91.6	90.0	89.4	72.2
200K	97.7	81.2	93.5	91.3	89.6	72.6
500K	97.7	87.9	93.7	88.0	83.5	72.1
1M	97.5	93.3	93.6	91.4	85.9	78.0
1.5M	98.7	92.6	94.5	91.5	90.2	79.6
2M	98.1	92.8	93.7	91.6	88.3	77.2

Num Candidates (top-25 query)						
	Anticorrelated		**Correlated**		**Independent**	
$\lvert D\rvert$	OLSH	LSH-SU	OLSH	LSH-SU	OLSH	LSH-SU
100K	6143.1	7109.1	4522.8	3661.9	3876.9	4307.7
200K	7067.2	6880.1	4253.5	4882.9	4043.8	5378.4
500K	8081.3	9567.7	5482.3	5914.3	6344.0	7911.7
1M	11191.8	9862.8	6640.2	5607.2	8139.6	8439.4
1.5M	11851.1	11389.3	6150.9	6771.1	8219.6	9848.1
2M	11180.1	12550.1	7166.7	5913.6	9446.3	9734.2

Table 5.8: Recall and # of candidates considered for different data sizes (target average redundancy 2)

tend to be more difficult in higher dimensions.

5.5.3 Impact of the Data Distribution

We also see in Table 5.4 that, while in lower dimensions the data distribution is not very critical, as the number of dimensions increases, OLSH as well as the two competitors provide their highest accuracies in correlated data, whereas accuracies are lower in anti-correlated and independent data. This is because, when the data is correlated, most points are clustered, and LSH search is able to identify sufficient candidates from each hash-layer, without having to rely on a large data redundancy (reminder: the default target data redundancy in the index structure is only 2).

5.5.4 Impact of Different Values of k

Table 5.5 presents accuracy and number of candidate results for varying values of k. As expected, the index structures are able to provide higher accuracies when k is small and the accuracy suffers as k increases (due to a higher number of opportunities for misses). However, the results presented in the table also indicate that OLSH is significantly more robust as the corresponding drop in accuracy is much lower than the drop in accuracy of the competitor, for the same increase in the value of k.

5.5.5 Impact of Different Values of m

As discussed in Section 5.2, the OLSH index structure is designed for a given maximum target rank value, m, by identifying the corresponding Onion layers and allocating hash resources to each Onion layer. In Table 5.6, we consider top-k queries on index structures designed with different maximum ranks. As we see in the table, OLSH remains the best alternative (both in terms of accuracy and the number of candidates produced) under different values of m.

5.5.6 Impact of Per-Layer Hash Count (κ)

We next consider the impact of the number, κ, of hashes concatenated per hash-layer. As discussed in Section 5.3.3, for the i^{th} Onion layer, we set κ_i to $log_2(|O_i|)$. In Table 5.7, we consider the impacts of larger and smaller values of κ. As we see here, under the proposed κ assignment strategy OLSH is able to provide a very high accuracy ($\sim$ 90% and above). As expected, using a smaller value of κ can further improve the accuracy (by reducing the likelihood of misses in hash layers), however, this comes with a significant jump (almost doubling) in the number of candidates that are enumerated. In other words, the proposed κ assignment strategy provides an effective trade-off between accuracy and the redundant work. This is further confirmed in Figure 5.3, where we visualize the accuracy vs. number of candidates trade-off in the form of XY-charts: the proposed κ assignment strategy (marked using dotted circles) provides a recall that is larger than would be expected purely based on the number of candidates considered (note that the dotted circles, marking the proposed κ strategy, lie above the corresponding trend curves in all the charts).

5.5.7 Impact of Different Data Cardinalities

As we see in Table 5.8, the above observations also hold for different input data sizes: OLSH performs significantly better than the competitor, for all data sizes considered. It is also important to note that as the data size increases, we see an increase in accuracy due to the densification of the space, but this does not reflect on the number of candidates considered: when the input data size increases 20X, from $100K$ to $2M$, the increase in the number of candidates is only <2X for the anticorrelated and correlated data sets and ∼3X for the independent data set. Figure 5.4 presents the information in an alternative format: as we see here, OLSH index structure indeed provides an overall better tradeoff between recall and number of candidates produced for all data sizes considered.

5.5.8 Impact of the Index Budget

In Table 5.9, we study the impact of the index budget, in terms of the average data redundancy that the index allows. As we see here, as the data redundancy increases, the recall also increases, but (as expected) with a corresponding increase in the number of candidates that need to be considered. It is important, however, to note that OLSH provides a better accuracy and a better accuracy/number of candidates trade-off under different target redundancies.

Recall (top-25 query)						
	Anticorrelated		**Correlated**		**Independent**	
r	OLSH	LSH-SU	OLSH	LSH-SU	OLSH	LSH-SU
2	97.7	81.2	93.5	91.3	89.6	72.6
3	99.3	93.7	96.9	93.6	92.0	82.8
4	99.5	95.8	98.8	96.3	95.7	84.8

Num Candidates (top-25 query)						
	Anticorrelated		**Correlated**		**Independent**	
r	OLSH	LSH-SU	OLSH	LSH-SU	OLSH	LSH-SU
2	7067.2	6880.1	4253.5	4882.9	4043.8	5378.4
3	9952.2	9906.3	4291.8	5889.9	6100.0	6196.6
4	11315.1	12793.2	5958.9	7065.2	7735.5	7632.6

$\frac{\textbf{Recall}}{\textbf{Num Candidates}}$ (top-25 query)						
	Anticorrelated		**Correlated**		**Independent**	
r	OLSH	LSH-SU	OLSH	LSH-SU	OLSH	LSH-SU
2	0.0138	0.0118	0.0220	0.0187	0.0222	0.0135
3	0.0100	0.0095	0.0226	0.0159	0.0151	0.0134
4	0.0088	0.0075	0.0166	0.0136	0.0124	0.0111

Table 5.9: Recall and # of candidates considered for different degrees of redundancies in the OLSH index (200K objects)

5.5.9 Impact of the Hash Collision Parameter (ψ)

As we have discussed in Sections 5.3.2 and 5.3.3, the accuracy of OLSH is a function of the per-hash collision probability, ψ and this value is greater than 0.5 for positive data and queries. Consequently, the value of this parameter has an impact on the layer-aware resource allocation. Table 5.10 considers results for three values of ψ between 0.5 and 1.0. As we see in this table, using large values of ψ in the design may lead to reductions in accuracy as, in that case, the layer-aware resource allocation might be done under overly optimistic assumptions. However, we also see that OLSH provides higher accuracy than the competitor under all parameter values considered. The table also shows that, for all data distributions, setting the value of ψ to 0.75 enables OLSH to provide the best accuracy vs. work trade-off.

Recall (top-25 query)						
	Anticorrelated		**Correlated**		**Independent**	
ψ	OLSH	LSH-SU	OLSH	LSH-SU	OLSH	LSH-SU
0.65	96.8	84.3	96.6	92.6	86.0	75.0
0.75	97.7	81.2	93.5	91.3	89.6	72.6
0.85	96.4	81.6	94.2	90.5	75.6	68.2

Num Candidates (top-25 query)						
	Anticorrelated		**Correlated**		**Independent**	
ψ	OLSH	LSH-SU	OLSH	LSH-SU	OLSH	LSH-SU
0.65	6244.9	6953.3	4916.6	4888.6	5393.0	4823.0
0.75	7067.2	6880.1	4253.5	4882.9	4043.8	5378.4
0.85	7585.9	7866.9	5898.6	5012.3	3653.6	4517.1

$\frac{\textbf{Recall}}{\textbf{Num Candidates}}$ **(top-25 query)**						
	Anticorrelated		**Correlated**		**Independent**	
ψ	OLSH	LSH-SU	OLSH	LSH-SU	OLSH	LSH-SU
0.65	0.0155	0.0121	0.0196	0.0190	0.0159	0.0155
0.75	0.0138	0.0118	0.0220	0.0187	0.0222	0.0135
0.85	0.0127	0.0104	0.0160	0.0181	0.0207	0.0151

Table 5.10: Recall and # of candidates considered for different collision parameter values in the OLSH index (200K objects)

5.5.10 Results with Real Data Sets

Finally, Table 5.11 presents results with the two real data sets, HOUSE and NBA. As we see in this table, also on these two real data sets, OLSH provides consistent higher accuracy than LSH-SU. Moreover, especially for large values of k, OLSH is able to provide better accuracy than LSH-SU even though it enumerates significantly smaller number of candidates.

5.6 Experiments with Competitors and Data Setups

In this section I discuss the performance using our approach with alternative methods both exact and approximate ones such as naive linear scan, threshold algorithm and RTree approach built using *Branch and Bound* concept. I show the comparison

Recall (top-k query)				
	HOUSE (6D)		**NBA (7D)**	
	OLSH	LSH-SU	OLSH	LSH-SU
top-1	69.9	34.9	76.1	70.8
top-2	47.6	28.3	69.0	64.4
top-5	26.6	16.2	62.5	68.3
top-10	26.5	17.4	64.2	66.9
top-25	23.4	17.3	70.1	63.3

Num Candidates (top-k query)				
	HOUSE (6D)		**NBA (7D)**	
	OLSH	LSH-SU	OLSH	LSH-SU
top-1	478.9	206.6	166.0	231.2
top-2	613.1	340.8	291.0	342.0
top-5	1424.3	802.3	531.1	953.4
top-10	1756.0	2094.0	1026.3	1602.9
top-25	2783.4	6278.1	2079.0	2827.4

Table 5.11: Recall and # of candidates considered for real data (target average redundancy 4, default κ)

not only the recall accuracy, but also the runtime and indexing memory cost, as well as data accesses for certain algorithms.

5.6.1 Branch-And-Bound Ranked Search

The competitors shown as to the proposed *OLSH* methods are benchmark approahces, yet do not consider the concept of *making big data small for queries*, which treats each portion of data equally. I have implemented in-memory version of RTree approach described in , and more specifically, for RTree approach it leverages the minimal bounding region (MBR) in multi-dimensional space for top-k dot product query retrieval. Let d denote the dimensionality of data, M be the minimal bound region (MBR) then I set the $i-th$ coordinate (dimension) of a point p to the upper or lower bound on this axis with query q to be monotone, l_i, h_i represent the lower and higher bound of a bound region of p. The algorithm shown in 4 describes a possible score of a point p, which will be leveraged for answering top-k queries.

Given the proof and observations, Branch-and-bound Ranked Search (BRS) is described in 5, which traverses the R-Tree nodes in descending order of their *maxscore* values and maintains the set of entries in the nodes accessed so far in a heap H, sorted also in descending order in their *maxscore*. At each step, the algorithm de-heaps the entry having the largest *maxscore*. If it is a leaf entry, the corresponding data point is guaranteed to have the largest score, among all the records that have not been reported yet. On the other hand, for each de-heaped intermediate entry, the algorithm visits its child node and puts all its entries into the heap. Once there are k objects in the result set, the algorithm terminates. BRS algorithm has been proved to be optimal since it visits the smallest number of nodes to correctly answer any top-k query. More intuitively, BRS algorithm accesses only the nodes whose MBRs intersect the search region (the *maxscore* of any other node must be smaller than k-th element in the result set S). Note that his is a approximate algorithm even though it leverages RTree indexing structure, and I have implemented based on the description of the algorithm with RTree indexing structure [5] .

5.6.2 Evaluation Criteria

Considering OLSH, the primary performance metric is recall, which the desired value is 100%. As the performance is from a mix of both exact and inexact solutions, I also report the memory cost which should be as low as possible while keeping the recall at desired values. These run time statistics are gathered based on 1000 random generated queries.

As we can see, BRS is more effective with correlated data due to the creation of minimal bounding region (MBR). Yet OLSH can leverage different hash redundancy

[5]https://github.com/nushoin/RTree

	Recall								
	Anticorrelated			**Correlated**			**Independent**		
	BRS	OLSH(2)	OLSH(4)	BRS	OLSH(2)	OLSH(4)	BRS	OLSH(2)	OLSH(4)
2	99.90	97.35	99.83	100.00	98.93	99.63	100.00	91.68	98.07
3	79.30	92.23	99.18	100.00	96.98	99.66	99.70	89.11	99.04
4	88.80	91.28	99.38	100.00	99.29	99.85	98.40	85.04	98.59
5	18.90	84.75	98.31	100.00	98.80	99.99	99.90	83.81	99.00
6	22.60	83.51	85.54	100.00	92.08	98.46	69.30	77.41	87.13
7	15.20	50.62	84.78	100.00	81.09	92.94	39.30	57.09	84.18

Table 5.12: Recall for different dimensions (200K objects, target average redundancy 2 and 4, index designed for up to top-25 retrieval)

	Memory Cost (KB)					
	Anticorrelated		**Correlated**		**Independent**	
	BRS	OLSH(4)	BRS	OLSH(4)	BRS	OLSH(4)
2	20912	NA	21104	NA	21044	NA
3	23604	8460	23744	5788	23744	11908
4	26420	29400	26360	24320	26360	24320
5	29120	52364	28856	46244	28608	53720
6	31264	55648	31112	57016	31312	54704
7	36716	54280	36392	55400	36776	54644

Table 5.13: Memory Cost for different dimensions (200K objects, target average redundancy 4, index designed for up to top-25 retrieval)

budget to improve recall accuracy and OLHS does not have to index entire dataset, whereas BRS tree-based approach has to index entire dataset which leads to fixed amount of memory and not as flexible as OLSH.

5.7 Experiments with Alternative Layer Indexing Approach

The proposed resource allocation approach is desgiend to allow different layer indexing frameworks to plugin and execute. There can be alternative approaches to build layer indexing structure for layer-aware resource allocation strategies, for instance, Skyline, which is a popular method to answer top-k and is analogous to the layers created using Onion technique. The Skyline Borzsony *et al.* (2001) of a d-

Recall									
	Anticorrelated			Correlated			Independent		
Count(M)	BRS	OLSH(2)	OLSH(4)	BRS	OLSH(2)	OLSH(4)	BRS	OLSH(2)	OLSH(4)
0.2	88.80	91.28	99.38	100.00	99.29	99.85	98.40	85.04	98.59
0.5	79.30	93.88	99.75	100.00	93.55	99.72	98.30	94.30	99.07
1	89.70	93.25	99.70	100.00	96.45	99.74	98.30	90.52	99.50
1.5	84.40	94.26	99.94	100.00	97.70	99.96	98.50	90.28	99.50
2	82.00	89.34	99.83	100.00	93.54	99.55	98.80	90.88	99.51

Table 5.14: Recall for different dimensions (200K objects, target average redundancy 2 and 4, index designed for up to top-25 retrieval)

Memory Cost (KB)						
	Anticorrelated		Correlated		Independent	
Count(M)	BRS	OLSH(4)	BRS	OLSH(4)	BRS	OLSH(4)
0.2	26420	29400	26360	24320	26360	24320
0.5	62728	44248	62528	28568	62528	47784
1	122984	58084	122856	32056	123044	61408
1.5	183560	67492	183824	34208	183440	74412
2	243820	73800	243820	37416	243760	82224

Table 5.15: Memory Cost for different dimensions (200K objects, target average redundancy 4, index designed for up to top-25 retrieval)

dimensional dataset contains the points that are not dominated by any other point on all dimensions. A point dominates another if it is as good or better in all dimensions and better in at least one dimension and the skyline operator is cruicial for applications that invovle multi-criteria decision making. More specifically, data points from outer skyline layers dominate those from inner layers. This layer sturcture can be applied to answer top-k queries and is analogous to the concept of layers created by Onion technique. Hence I decide to test the performance of this alternative approach to see the effectiveness of how it can be used in presence of resource allocation. I name this layer structures created by Skyline with LSH indexing as **SLSH**.

Following the same criteria as mentioned above, the primary performance metric is recall (which we want to keep as close to 100% as possible), as well as the number of candidates retrieved during the process.

Recall (top-25 query 4D)						
	Anticorrelated		**Correlated**		**Independent**	
	OLSH	SLSH	OLSH	SLSH	OLSH	SLSH
0.2	99.69	99.76	97.72	99.97	96.12	99.76
0.5	99.96	99.91	97.31	99.92	97.88	99.85
1	99.96	99.96	98.16	99.7	98.11	99.95
1.5	99.92	99.96	99.55	99.93	97.43	99.98
2	99.94	99.95	97.02	99.78	98.47	99.99

Recall (top-25 query 200K Datasize)						
	Anticorrelated		**Correlated**		**Independent**	
	OLSH	SLSH	OLSH	SLSH	OLSH	SLSH
2	100	99.99	99.58	100	98.87	99.89
3	99.7	100	98.38	99.97	96.3	99.98
4	99.69	99.76	97.72	99.97	96.12	99.76
5	99.85	97.1	100	100	99.98	100
6	97.47	76.51	98.43	100	93.46	99.13
7	85.24	26.62	91.54	100	86.99	99.06

Table 5.16: Recall for different top-k queries (target average redundancy 4, index designed for up to top-25 retrieval)

A sample result output for this setup is as shown in Table 5.16 with default average redundancy 4. By using Skyline as layer for resource allocation, SLSH is able to provide competitive recall accuracies as OLSH, especially for *Correlated* and *Independent* data type due to the *dominance* nature from Skyline creation. For Anti-Correlated data type, Skyline dominance relationship failed to capture the dominanting points that are not along the axises which leads to poor performance in terms of recall accuracy expecially when the dimensionality increases.

5.8 Alternative Mathematical Optimizationand with Model Revision

As discussed in previous sections and Equation 5.3.4, given a fixed amount of hashing resources, the proposed framework is to intelligently re-distribute it among each layer and perform top-k retreival efficiently. Alternatively people can be also interested in the case where, given a desired total error rate $error_{input}$ as threshold, how to minimize the total hashing resources needed. By using the similar notations and mathematical setups as mentioned earlier, the alternative or the reverse optimization can be summarized as:

$$\begin{aligned} \text{minimize} \quad & resource(\mathbb{O}) \\ \text{subject to} \quad & err(\mathbb{O}, m) \leq err_{input}, \end{aligned}$$

An optimization problem as shown in Equation 5.8, however, would require a more precise model. The layer weight model mentioned before, as shown in Table **??** is able to capture the layer importance when hashing resource is limited and eventually generate relatively high recall accuracy, where contributions of each onion layer toward the final top-k result set are considered only based on the number of data elements on a given onion layer. As the convex hull already provides us the knowledge that, top-1 result certainly comes from the outer most layer, top-2 results come from the two outer most layers, etc, the order of onion layer index matters, and should be taken into consideration. Since Equation 5.8 leverages the reverse of the original optimization from Equation 5.3.4, I call the OLSH using reverse optimization concept as *ROLSH*.

For example, as shown in Figure 5.5, the actual *recall* diverges from the layer weight model we introduced previously: the actual recall values across k layers follow a beta distribution whereas the learned model for layer weight follows a linear monotonic decreasing pattern. To model the layer weight in a more precise manner, I first execute a sample query set to generate the recall accruacy at each Onion layer, which can be leveraged to represent the actual layer contribution: the higher recall accuracy from an onion layer, the more important it is for the top-k result set. Following the recall distribution across k layers, the pattern can be fit into a beta distribution to represent layer weight which is then be leveraged for layer aware resource allocation. Figure 5.6 depicts the process of computing layer weight by *slicing* beta distribution to calculate the weight for each onion layer.

Recall						
	Anticorrelated		**Correlated**		**Independent**	
Count(M)	OLSH	ROLSH	OLSH	ROLSH	OLSH	ROLSH
0.2	99.38	71.6	99.85	77.22	98.59	86
0.5	99.75	67.48	99.72	84.14	99.07	78.3
1	99.7	80.02	99.74	88.56	99.5	78.66
1.5	99.94	70.68	99.96	85.7	99.5	69.97
2	99.83	75.44	99.55	84.52	99.51	69.19

Table 5.17: Recall for different top-k queries (4D target average redundancy 4, ROLSH error 0.3, index designed for up to top-25 retrieval)

Experimental studies have shown the efficiency and effectiveness of the updated layer weight model by fixing a desired input error rate to minimize the total resources. I present the results of using various hashing resource redundancy (# of times data gets redplicated) with origin maths (given a limited amount of resources, the goal is to minimize the total error across OLSH, as shown in Equation 5.3.4). Table 5.17, Table 5.18, Table 5.19 and Table 5.20 indicate the results that once the total error rate is fixed, ROLSH can minimize hashing resources accordingly. As shown in the sample experimental study, the total amount of hash used for ROLSH is smaller than the one used by OLSH, hence in general we can see a lower accuracy rate from ROLSH compared to the ones generated using OLSH.

By leveraing the updated model with layer weight calculated from Beta distribution, the overall error rate from OLSH captures the input desired error rate (0.3 in this case) with lower hash cost as shown in Table **??** and Table **??**, which depicts the total of data elements indexed using the LSH framework and the comparison of hash used between OLSH and ROLSH.

5.9 Conclusion

As mentioned above, I discussed a novel Onion-LSH (OLSH) index structure to approximately solve top-k inner product queries. In particular, OLSH indexing frame-

Total Hash Used									
	Anticorrelated			Correlated			Independent		
Count(M)	Elements	OLSH	ROLSH	Elements	OLSH	ROLSH	Elements	OLSH	ROLSH
0.2	95149	380552	132087	63751	254905	76841	105366	422194	161709
0.5	152082	609035	209582	86334	346209	121861	169673	678658	239359
1	206860	829423	278030	103972	415699	132799	232022	930347	319206
1.5	244585	979742	319055	116284	465163	138820	275450	1100803	341285
2	275400	1100991	355907	123335	493860	173600	310299	1243443	394198

Table 5.18: Total amount of hash used for different top-k queries (4D, index designed for up to top-25 retrieval)

Recall						
	Anticorrelated		Correlated		Independent	
	OLSH	ROLSH	OLSH	ROLSH	OLSH	ROLSH
2	99.83	94.13	99.63	98.42	89.70	98.07
3	99.18	89.92	99.66	91.95	79.12	99.04
4	99.38	71.60	99.85	77.22	86.00	98.59
5	98.31	73.64	99.99	98.37	85.92	99.00
6	85.54	87.59	98.46	85.25	80.62	87.13
7	84.78	90.15	92.94	85.80	88.18	84.18

Table 5.19: Recall for different top-k queries (200K objects, target average redundancy 4, ROLSH error 0.3, index designed for up to top-25 retrieval)

work combines the Onion-based data layering with (angular) LSH-based indexing for efficient data access. Since, when using LSH, there is a trade-off between the available hashing resources and the accuracy, I also proposed a top-k based accuracy model and complement the proposed OLSH index structure with a layer-aware (LA) resource allocation strategy, which takes into account the distribution of the data, and the number, k, of results required to allocate available hashing resources among the Onion layers of the index structure. Experimental results with real and synthetic benchmarks showed that that the proposed OLSH technique achieves the target accuracy rates within given resource budget under different scenarios. I also provide sample results of SLSH, where layer index structures are created using Skyline. The results have shown that SLSH follows a similar pattern as OLSH, both of which indi-

Total Hash Used									
	Anticorrelated			**Correlated**			**Independent**		
Count(M)	Elements	OLSH	ROLSH	Elements	OLSH	ROLSH	Elements	OLSH	ROLSH
2	1962	7922	3908	857	3424	1566	2027	8158	3922
3	20869	83525	33434	10931	43700	15996	20869	33434	34934
4	95149	380552	132087	63751	254905	76841	105366	422194	161709
5	188619	752448	317994	166630	668389	214514	192858	767888	334808
6	200000	799991	592958	200000	799993	269745	199998	800559	579663
7	200000	803080	1225236	200000	806067	13780685	199995	799918	1150792

Table 5.20: Total amount of hash used for different top-k queries (200K objects, target average redundancy 4, ROLSH error 0.3, index designed for up to top-25 retrieval)

cate that by learning the structure of input data, one can achieve top-k query retrieval effectively and efficiently. Additionally, I also show the OLSH with reverse mathamtical optimization setup, which is to minimize the total hash resources allocated given a desired error rate input. In a nutshell, results have shown the effectiveness and efficiency of the proposed framework under various setups.

Algorithm 2 ReviseLayers

Input: Onion layers $\mathcal{O}$, maximum rank m, budget B, per-layer hash counts, $\mathcal{K}$, initial layer assignment $\mathcal{L}_0$, hash family, $\mathcal{H}$

Output: Revised layer assignment, $\mathcal{L}$

```
1:  resources = ComputeResources($\mathcal{O}$, $\mathcal{L}_0$)
2:  $\mathcal{L} = \mathcal{L}_0$
3:  BestLayer = $\infty$
4:  Available = $B-$ resources
5:  **while** (Available $> 0$) and (BestLayer $\neq \perp$) **do**
6:      BestPotential = -1
7:      BestLayer = $\perp$
8:      Used = 0
9:      **for** $1 \leq i \leq m$ **do**
10:         $potential_i$ = ComputePotential($\mathcal{O}$, $m$, $\mathcal{K}$, $\mathcal{L}$, $\mathcal{H}$)
11:         $demand_i$ = Size($O_i$)
12:         **if** ($demand_i \leq$ Available) and ($potential_i >$ BestPotential) **then**
13:             BestPotential = $potential_i$
14:             BestLayer = $i$
15:             Used = $demand_i$
16:         **end if**
17:     **end for**
18:     **if** BestLayer $\neq \perp$ **then**
19:         $\mathcal{L}$ = incrementOnionLayer($\mathcal{L}$, BestLayer)
20:         Available = Available - Used
21:     **end if**
22: **end while**
23: return L
```

Algorithm 3 Top-K Search

```
    Input: OLSH index, $\mathbb{O}$, a top-$k$ query $q$
    Output: result set $R$

1: $TempRes = \emptyset$
2: for $1 \leq i \leq k$ do
3:     $Res_i$ = searchLSHOnionLayer($q$, $\mathbb{O}$, $i$, $k - i + 1$)
4:     $TempRes$= pickBest($TempRes \cup Res_i$, $k$)
5: end for
6: return $R = TempRes$
```

Algorithm 4 Algorithm For Computing max_score For Monotone Functions

```
    Algorithm get_maxscore ($M = (l_1, h_1, l_2, h_2, ..., l_d, h_d), q$)

1: Initiate a point $p$ whose coordinates are not decided yet
2: for $i = 1$ to $d$ do /* examine each dimension in turn */
3:     if $q$ is increasinly monotone on this dimension then
4:         the $i$-th coordinate of $p$ is set to $h_i$
5:     else the $i$-th coordinate of $p$ is set to $l_i$
6:     end if
7: end for
8: return $q(p)$
```

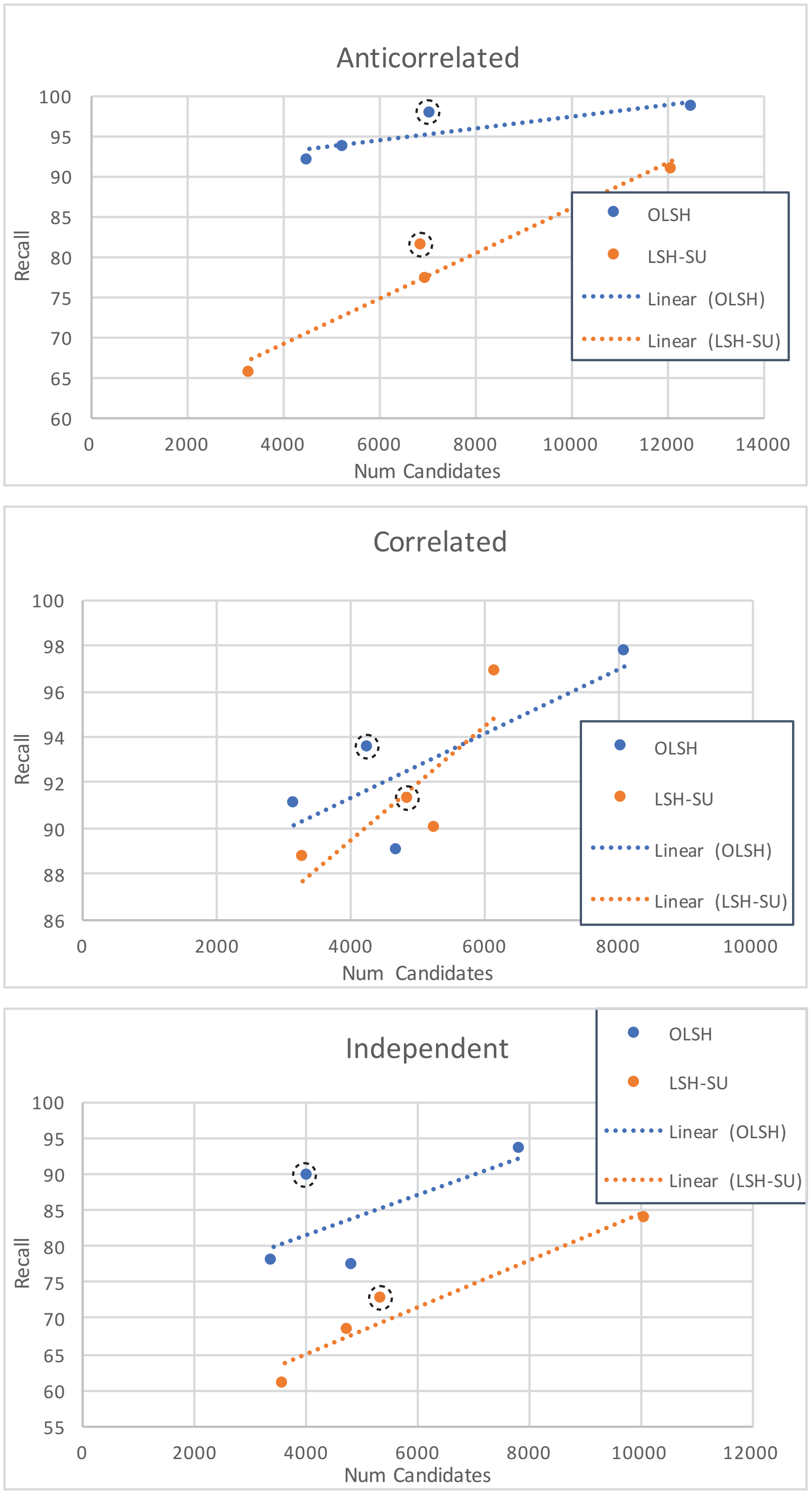
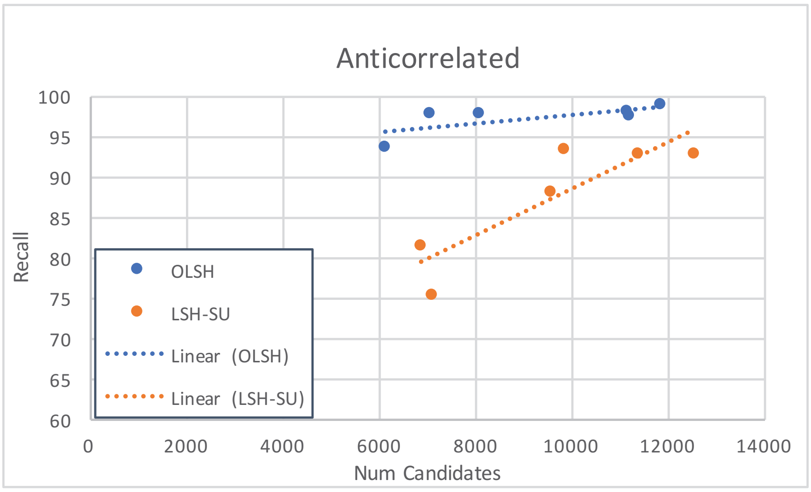
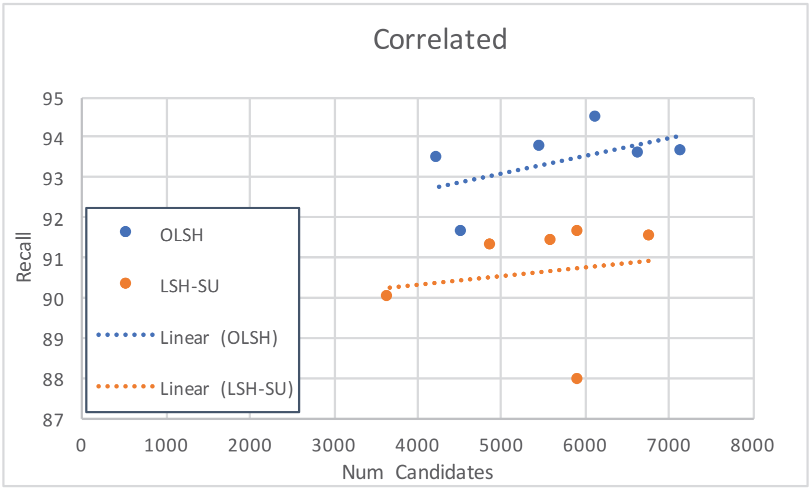


Figure 5.3: Recall vs. # of candidates for different κ assignment strategies – $\kappa = log_2(|O_i|)$ based solutions are highlighted (200K objects, target average redundancy 2)





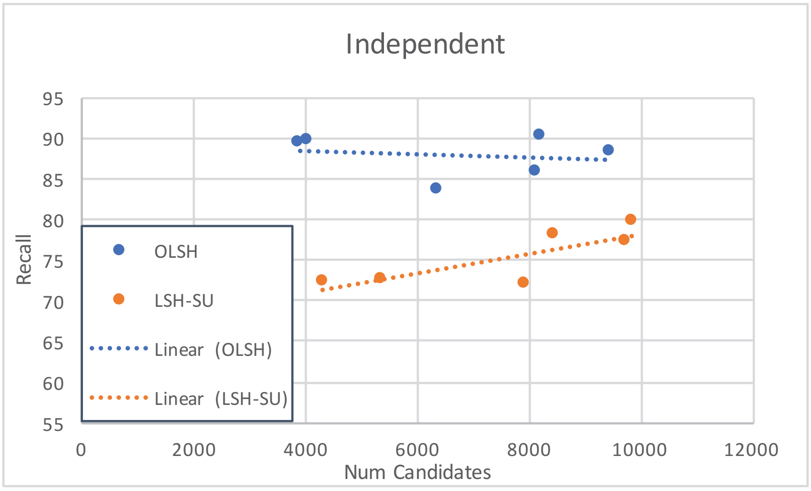


Figure 5.4: Recall vs. # of candidates for different data sizes – each dot corresponds to a different data size (200K objects, target average redundancy 2)

Algorithm 5 Algorithm BRS

```
    **Algorithm get_maxscore** $(M = (l_1, h_1, l_2, h_2, ..., l_d, h_d), q)$

1: Initiate the candidate heap $H$ /*$H$ takes entires in the form (REntry, key) and
   manages them in descending order of key (REntry is an entry in RTree)*/
2: Initiate a result set $S$ with size $k$
3: load the root of RTree
4: for each entry e in the root do
5:     $e.maxscore = get\_maxscore(e.MBR, q)$
6:     insert $(e, e.maxscore)$ into $H$
7: end for
8: while (S contains less than $k$ objects) do
9:     $he$ = de-heap(H)
10:    if $he$ is a leaf entry then
11:         add $he$ to $S$
12:        if $S$ contains $k$ tuples then
13:            return $S$
14:        end if
15:    else
16:        for every entry $e$ in $he$.childnode do
17:            e.maxscore=$get\_maxscore(e.MBR, q)$
18:            insert(e, e.maxscore) into H
19:        end for
20:    end if
21: end while
22: return $q(p)$
```

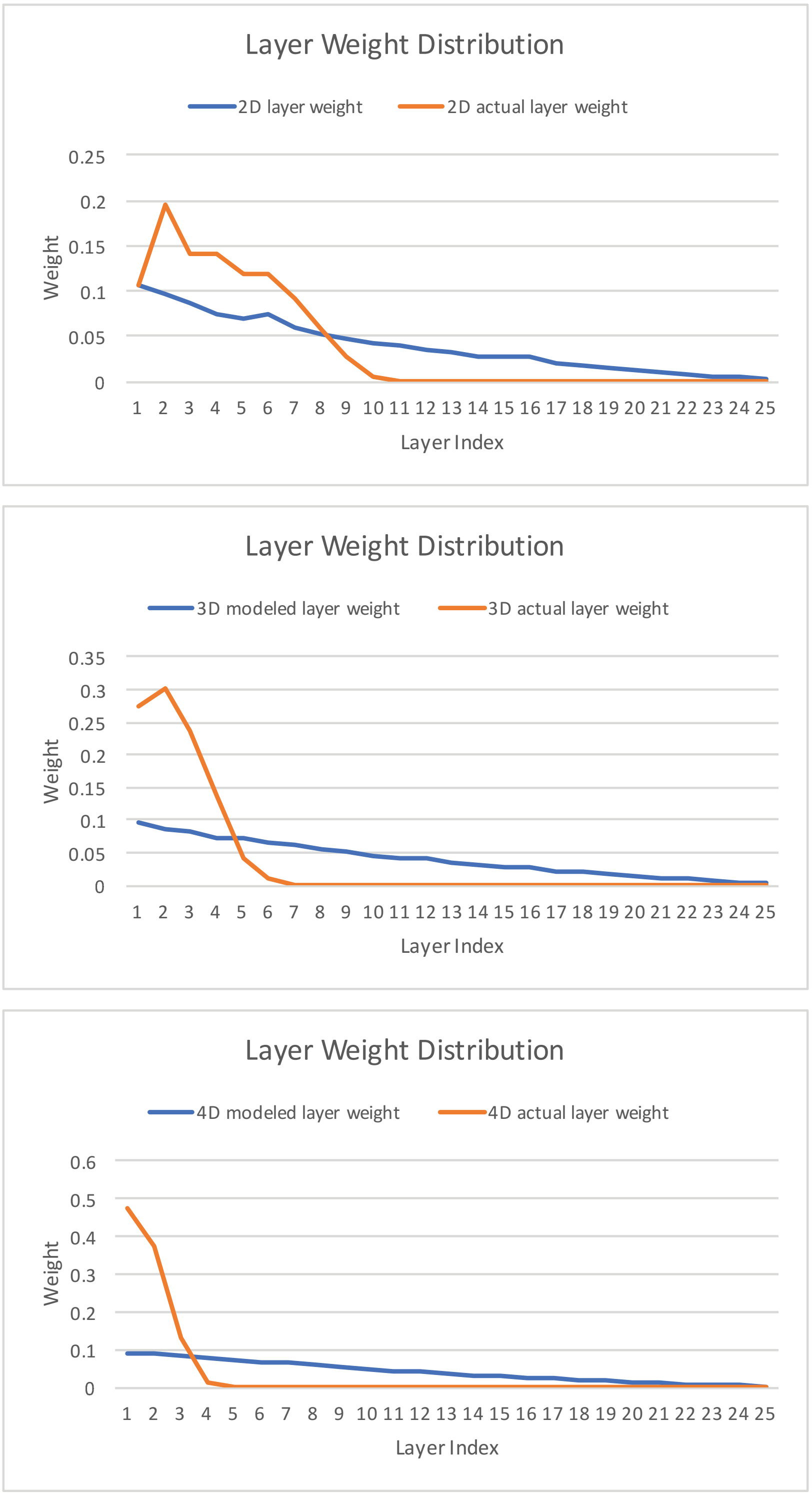


Figure 5.5: Actual Layer Weight vs. Learned Layer Weight Distribution

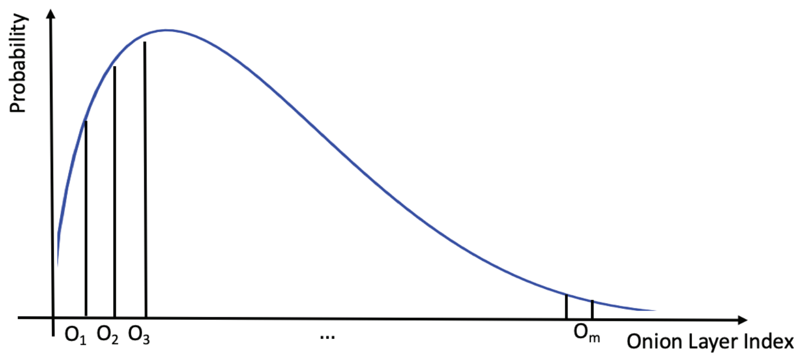


Figure 5.6: Slicing Beta Distribution For Layer Weight Learning

Chapter 6

PLSH: PARTITION-LSH FOR APPROXIMATE TOP-K QUERY PROCESSING

Previously I have presented OLSH, a technique based on Onion technique which first creates layers then apply resource allocation to minimize the total error across all the onion-layers with Locality Sensitive Hashing (LSH) indexing structure applied among each onion-layer created. This approach, however, faces a crucial challenge that Onion layer creation process is computationally infeasible when the dimensionality of data increases, i.e. dimension reaches 100. To address this challenge with performance guarantee in presence of high-dimensional data set, I propose *PLSH*: a partition based LSH approach that first paritions data into different bins based on their l_2 norm; afterwards, within each bin, I apply LSH bucket indexing structure for fast top-k retrival tasks. Each bin would contribute k candidates and a re-ranking process is applied afterwards. In this case, the importance or weight of each bin will be revised compared to the previous OLSH approach such that resource allocation process can be executed given a limited hashing resources.

6.1 PLSH for Approximate top-k Processing

In this section, we discuss the proposed *PLSH* index structure to *approximately* solve top-k queries. At the highest level, PLSH first partitions data based on their l_2 norm values then learn the importance of each partition from sample queries. Distinct from prior works, to reduce the amount of data access, each data within each partition is further organized into (inner product based) LSH layers, referred to as h-layers.

In LSH based retrieval schemes, there is an inherent trade-off between available hashing resources and the accuracy. Since basic LSH does not provide error guarantees for top-k retrieval results (but instead targets accuracy guarantees based on distances), in this section we discuss how to control this trade-off using an accuracy model that relates target with the number of h-layers created. Furthermore, we complement the proposed PLSH index structure with a partition-aware (PA) resource allocation strategy, which takes into account the distribution of data across different partitions based on sample queries with the number k, of the results required, and the user's target accuracy, to allocate available hashing resources among the partitions.

The following table depicts the symbols used within this manuscript.

Table 6.1: Symbols Used in Analysis

Symbol	**Description**
$\mathcal{D}$	input dataset
$\mathcal{Q}$	sample query set
$\mathcal{K}$	hash code set
m	num of partitions
P	partition set
$\vec{w}$	partition weight vector
k	results of interest
B	hash resource budget
$\mathcal{L}$	hash layers
dim	dimensionality of data
$Beta(\alpha, \beta)$	beta distribution
$cdf(i)$	cumulative density function at index i given a $Beta(\alpha, \beta)$

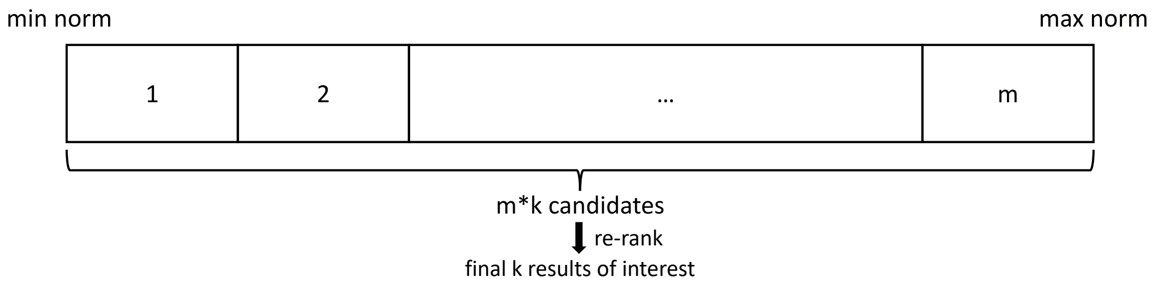


Figure 6.1: Norm Partition Framework for Top-k Retrieval

6.1.1 Norm Based Partition

For each data entry from a dataset D, $\forall d_i \in D, i = 1, 2, \dots, n$ is firstly partitioned into m chunks based on l_2 norm value $||d_i|| = \sqrt{\sum_{j=1}^{dim} d_j}$. Considering both norm and angle information, to answer top-k results for a given query, each partition would contribute up to k candidates, hence $m \times k$ in total.

The larger norm range a partition represents, the more likely the candidates calculated from it will be contributed to the final result set of top-k query. Meanwhile, since the angle between a query and a data entry also contains crucial information for ranking a dot product value, it makes sense to take both l_2 norm and the angle in multi-dimensional space into consideration when assigning weight to different partitions. Without prior knowledge, we design a framework to learn partition weight on the fly using sample queries, which will be discussed in the following section.

6.1.2 Partition Importance Learning

In this section we discuss how to calculate the weight of each partition, based on which the distribution of hash resources can be realized.

Given a dataset $d_i \in D, i = 1, 2, \dots, n$ and number of desired partitions m, let $norm_{max}$ and $norm_{min}$ denote the max and min norm of the norm, we adopt equal-width partition strategy such that each chunk will cover the data norm range in

Algorithm 6 PLSH Index Building

```
    **Input: database $D$, # of partitions $m$, sample queries set $\mathcal{Q}$**
    **Output:** $P$, $\vec{w}$
1: $\mathcal{P}$ = NormBinPartition($D$, $m$)
2: $\vec{w}$ = LearnPartitionWeight($D$, $m$, $\mathcal{P}$)
3: $\mathcal{K}$ = AssignPerPartitionHashCounts($\mathcal{P}$, $m$, $\mathcal{H}$)
4: $\mathcal{L}_0$ = SolveNonLinear($\mathcal{P}$, $\mathcal{K}$, $m$, $B$, $\mathcal{H}$)
5: $\mathcal{L}$ = RevisePartitions($\mathcal{P}$, $m$, $B$, $\mathcal{K}$, $\mathcal{L}_0$, $\mathcal{H}$)
6: **return** $\mathbb{P}$ = computeHashTables($m$, $\mathcal{P}$, $\mathcal{K}$, $\mathcal{L}$, $\mathcal{H}$)
```

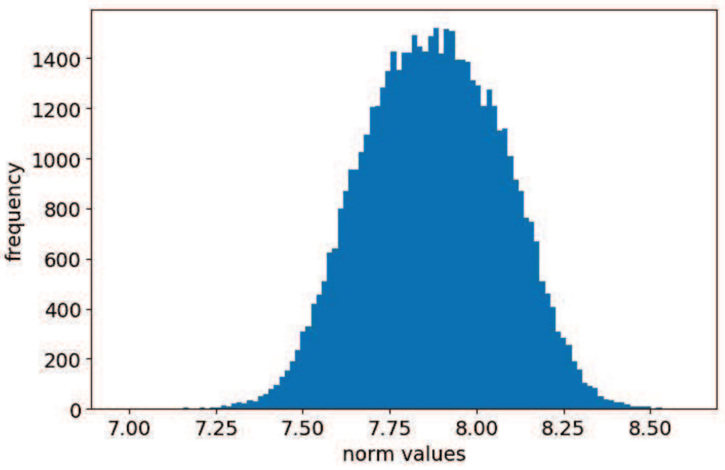


Figure 6.2: Yahoo!Music Data Norm Distribution (192D)

$\dfrac{norm_{max} - norm_{min}}{m}$. Without loss of generality, we assume that data is normalized within $[0, 1]$ along each dimension, hence $norm_{min} = 0$ and $norm_{max} = \sqrt{dim}$. Let $p_1, p_2, \ldots, p_m$ denote the m partitions, we have $norm_{p_i} - norm_{p_{i-1}} = norm_{p_{i+1}} - norm_{p_i}$.

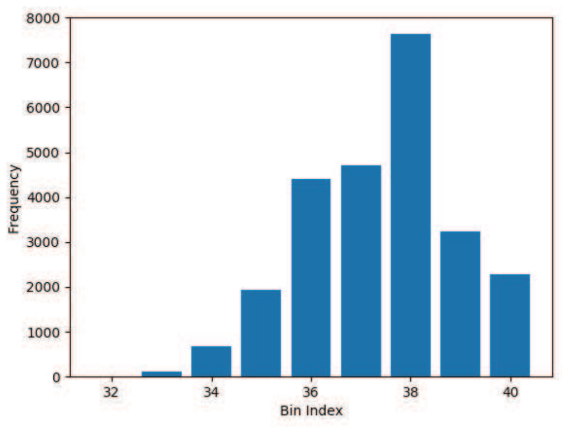
To learn the partition importance, one intuition is that partitions which contain more data should have large weight for resource allocation since it has larger probability to generate top-k query result candidates. The results of dot product, however, also considers *angle* between query and data. For instacne, as shown in Figure 6.2, data that resides in the long tail of the norm distribution can contribute to the top-k result set considering both norm and angle information even though data cardinality is relatively smaller. Moreover, partition weight can also be query dependent such that top-k results are generated from a specific partition of data hence making weight-assigning process non-trivial. To learn the importance for each partition, I first execute sample queries to decide the contribution of each partition to the final top-k result sets. One example is shown in Figure 6.3, where I first partition data into 40 bins, then execute the sample queries to calculate top-25 result distribution. Note that:

- Each partition $p_i, i = 1, 2, \ldots, m$ generates k candidates using LSH indexing scheme then re-ranking those $m \times k$ candidates to generate final k results to return

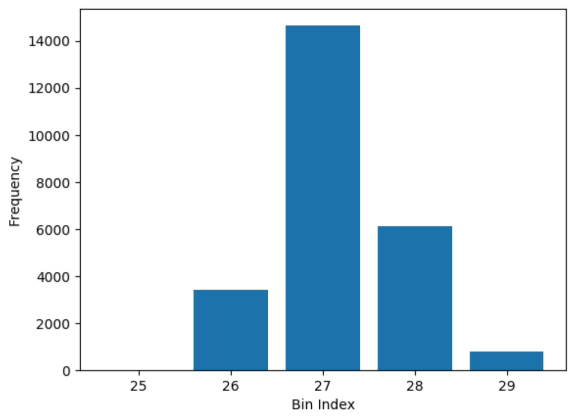
- Let $\mathcal{Q}$ denote the sample query set, to learn the distribution one would expect $||\mathcal{Q}\| \times k$ data elements as output results as each $q_i, i = 1, 2, \ldots \|\mathcal{Q}\|$ outputs k results of interest.

As shown in Figure 6.3, those returned results of interest can be modeled using Beta-Distribution [1] : values within each bin represents the number of results that belong to the norm range. With the distribution information calculated in Figure 6.3, I then proceed to calculate the parameters α, β for Beta-Distribution. To make the model learning process meaningful, sufficient amount of queries should be included in $\mathcal{Q}$. Afterwards, we leverage Beta-Distribution to calculate weight of each partition for hash resource allocation and in the following sections I will introduce different strategies to compute the weight for each p_i.

[1]https://en.wikipedia.org/wiki/Beta_distribution



(a) Yahoo!Music 192D Top-25



(b) 100D Top-25

Figure 6.3: Ground Truth Bin Partition Distribution

Equal-Width Weight Calculation

Given $Beta(\alpha, \beta)$ learned from previous step, we can now proceed to assign weight or importance for each partition p_i to intelligently allocate hash resources. It is straight foward to adopt *equal wdith (EW)* stategy from the parition approach where that each partition represents the same norm range interval. Let *cdf* denote the cumulative density function for $Beta(\alpha, \beta)$ and the weight w_i for partition p_i can be calculated as shown in Equal 6.1:

$$w_i = cdf(i + 1) - cdf(i) \tag{6.1}$$

Essentially it represents that a straightforward approach to compute weight $\vec{w}$ for partition P.

Equal-Depth Cardinality (EDC) Weight Calculation

One of the alternative strategy for weight computation is to partition the data based on l_2 norm such that each partition $p_i, i = 1, 2, \ldots, m$ contains the same cardinality, namely *equal depth cardinality (EDC)* approach. The intuition is to "slice" the curve of the learned $Beta(\alpha, \beta)$ with equal cardinality in each partition p_i, the weight or importance of p_i is different since the different portions of the data share various contribution to the final top-k result set. By leveraging EW partition, let $norm_{EW}$ denote the output of EW partition approach such that each partition represents the same norm range interval, the *EDC* partition strategy is described as shown in Algorithm 7. We first sort the data based on its norm values, then uniformly split (UniSplit) data into m partitions. From line 5 to line 14, the algorithm computes the data norm boundaries of P_{EDC} from UniSplit *w.r.t* those from $norm_{EW}$ so that the weight values w_{EDC} can be computed using *cdf* functionality from $Beta(\alpha, \beta)$. Essentially *EDC* leverages *EW* to learn weight for each $p_i, i = 1, 2, \ldots, m$ and repartition the data.

Equal-Depth Probability (EDP) Weight Calculation

Additionally, we also provide equal depth weight learning strategy *w.r.t* probability of each chunk: data is re-partitioned into different chunk such that each chunk of data represents equal probability interval *w.r.t* $Beta(\alpha, \beta)$ learned in the previous steps. The algorithm is described as Algorithm 8:

More specifically, the algorithm first computes and sorts norm of $\mathcal{D}$. This is the re-partition strategy that each chunk shares the same weight from the $Beta(\alpha, \beta)$, hence each chunk of data would contribute $1/m$ amount of weight given m total chunks. Line 7 to 15 shows how to re-partition data using *prob_step*: it first identifies the $norm_{index}$ index given $Beta(\alpha, \beta)$ and cumulative probability. Afterwards it rounds the $norm_{index}$ to get the integer index to retrieve the norm bound from $norm_{EW}$, and identify current norm of interest at line 11. It re-partitions the data based on current norm of interest, then updates parameters for next round of iterations.

EDC and *EDP* partition strategies can be shown in Figure 6.4 so that:

- For EDC, the cardinality within each parition should be the same: let $card_i$

denote the data elements that in the $i-th$ partition and we have $card_1 = card_2 = card_3, \ldots, = card_m$

- For EDP, the probability $w.r.t$ probability densitity function of $Beta(\alpha, \beta)$ is the same, such that $p_2 - p_1 = p3 - p2 = \ldots, = p_m - p_{m-1}$

With the partition strategies discussed above, let $p_i \subseteq \mathcal{D}$ be an partition (or p-partition). PLSH organizes the data on this p-partition into LSH data buckets, in a way that provides approximate search for objects matching the query, q. More specifically, for i^{th} p-partition, we pick two parameters, κ_i and L_i, that control the precision and recall, respectively, for that partition. As discussed in Section 2.2, concatenating κ_i independently selected hash functions will help reduce the number of false positives. More specifically, the data in p_i are partitioned into buckets using the combined hash function. We refer to each such grouping of κ_i hash functions as a hash-layer (or h-layer) of the corresponding partition . In order to reduce the misses, we then create L_i many such h-layers and union the results obtained from each of the corresponding buckets, the idea being that if a result is missed by an individual h-layer, it is less likely that it will be missed simultaneously by multiple h-layers.

Definition 19 (PLSH Index) *Let us consider a data set, D, indexed using PLSH for top-k search for $k \leq m$. As before, let us denote the set of data points in the i^{th} partition (p-partition) as P_i. Let us further denote the number of hash layers (h-layers) and the number of hashes per h-layer for the i^{th} partition as L_i and κ_i as before.*

We represent this PLSH index with a 5-tuple $\langle m, \mathcal{P}, \mathcal{K}, \mathcal{L}, \mathcal{H} \rangle$, where $\mathcal{P} = \{P_1, \ldots, P_m\}$, $\mathcal{K} = \{\kappa_1, \ldots, \kappa_m\}$, $\mathcal{L} = \{L_1, \ldots, L_m\}$, and $\mathcal{H}$ is a suitable hash family. $\diamond$

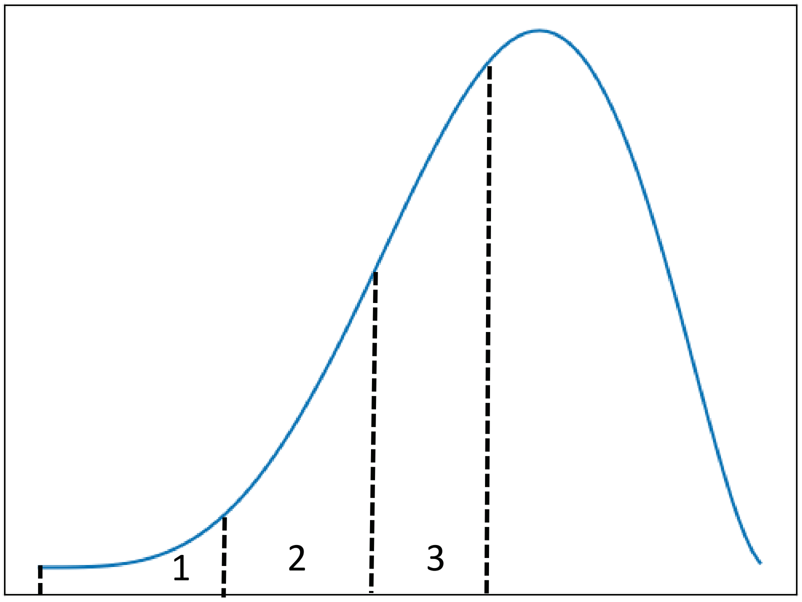


Figure 6.4: Slicing Beta Distribution for Partition Weight Learning

As we have seen in Section 2.2, the hash family, $\mathcal{H}$, used for mapping the data into hash buckets needs to be compatible with the underlying distance/similarity function. LSH families have been explored for different distances or similarities, including l_p distance (LSH with $p-$stable distributions), Hamming distance, Jaccard coefficient, and so on Indyk and Motwani (1998); Datar *et al.* (2004); Neyshabur and Srebro (2015); Shrivastava and Li (2014); Huang *et al.* (2018). Since, for top-k queries considered in this paper, the degree of match between the query, q, and the data point, p, is defined based on their inner (or dot) product, for PLSH we need a hashing scheme that supports inner product search.

6.1.3 Accuracy/Resource Trade-offs in PLSH

In traditional LSH, the collision probability for a given hash layer is computed as ψ^κ, where ψ is the collision probability (say for a given distance/similarity target) and κ is the number of hash functions concatenated in that layer. This is because hash functions are selected independently and, for any object to appear in the result of a given hash layer, it needs to collide with the query for all κ hash functions in the layer. The collision rate for the entire LSH index (consisting of L hash layer) is, then, $1-(1-\psi^{\kappa})^{L}$, since the only way a data object will fail to appear in the result is if it does not collide with the query for any of the L layers.

In the case of PLSH, however, not all partitions contribute to the result set of a top-k query the same way. We therefore need to formulate the accuracy measure in a way that takes into account the contribution of different partitions to the final result:

- As discussed earlier, the larger norm and the samller angle a data element has towards a given query q, the more probability it will be returned in the final result list. To answer top-k query, we partition data into m partitions and within each partition we calculate k candidates

- The algorithm then learns the importance of each partition after executing sample queries $\mathcal{Q}$: the more data elements returned as top-k results reside in a given parititon, the more weight or importance that partition would have, hence more hashing resources should be allocated to that partition.

We next study the accuracy resource trade-offs in PLSH design.

Resource Consumption

Let us consider a data set, D, indexed using an PLSH index $\mathbb{P} = \langle m, \mathcal{P}, \mathcal{K}, \mathcal{L}, \mathcal{H} \rangle$ as formalized in Definition 19. The data elements corresponding to an partition have to be indexed once for each of the corresponding hash layers. Consequently, the total resource consumption of the OLSH index, $\mathbb{P}$, can be computed as

$$resource(\mathbb{P}) \sim \sum_{i=1}^{m} L_i \times |P_i|.$$

Error Rate

To compute the error rate for PLSH under a given parameter setting, $\mathbb{P} = \langle m, \mathcal{P}, \mathcal{K}, \mathcal{L}, \mathcal{H} \rangle$, we need to consider that the different onion layers (a) are allocated potentially different amount of resources and (b) have different contributions to the result set of the top-k query.

In particular, to answer top-k queries using m partition, we have seen that each one would contribute k candidates, as shown in Figure 6.1. Additionally, based on the ground truth top-k distribution across those partitions, example shown as Figure 6.3, the parititons with more data should have more weight or importance, hence more hashing resources should be allocated to them. More specifically, we have quantified those weights by leveraing three partitions strategies, namely EW, EDC and EDP, described in Algorithm 7 and Algorithm 8, hence we have the weight w_i for each partition P_i. Thus we can proceed to quantify the total error rate across all partitions for top-k query processing:

Let us consider a data set partitioned into m partitions. Given a top-k query, the likelihood of the partition i to produce top-k results can be represented as w_i.

Having quantified the likelihood of the P_i to produce the resulsts for a given top-k query, q, we can then compute the expected error rate for a given PLSH index $\mathbb{P} = \langle m, \mathcal{P}, \mathcal{K}, \mathcal{L}, \mathcal{H} \rangle$ for this query as follows:

$$err(\mathbb{P}, k) = \sum_{i=1}^{m} \left(w_i \times \left((1 - \psi_i^{\kappa_i})^{L_i} \right) \right),$$

where ψ_i is the per-hash function success probability for the hash functions used to index the i^{th} onion layer.

6.1.4 PLSH Design Criteria - Resource Allocation Revised

Given the resource and accuracy trade-offs formalized in the above section, we can then restate the PLSH index design criteria as follows: Let D be a data set. An PLSH index, $\mathbb{P} = \langle m, \mathcal{P}, \mathcal{K}, \mathcal{L}, \mathcal{H} \rangle$, to answer top-$k$ queries for this data set must satisfy the following:

$$\begin{aligned} &\text{minimize} && err(\mathbb{P}, m) \\ &\text{subject to} && resource(\mathbb{P}) \leq resource_{max}, \end{aligned}$$

where $resource_{max}$ is the maximum resource allocated for the PLSH index. In addition, we would like to keep the number, $cand(\mathbb{P}, k)$, of candidate results produced during the processing of top-k queries ($k \leq m$) as low as possible.

6.1.5 Overview of PLSH

In Algorithms 6 through 10, we present the pseudo-codes for construction and use of the PLSH index structures for top-k query processing.

Constructing PLSH

Algorithms 6 and 9 present the outline of the process through which we create an PLSH index for a given data set D. As we see in Algorithm 6, the process takes as input, in addition to a data set, the maximum rank to be supported as well as a suitable hash family. The first step of the algorithm is to extract the m partitions, the top-k search algorithm is suitably modified to deal with this case. In the second step, we compute the per-layer hash counts for each partition, based on the corresponding number of data elements.

In the third step, we solve the optimization problem formulated in the previous subsection. Similarly to OLSH, after obtaining an initial design $\mathbb{P}_0 =$

$\langle m, \mathcal{P}, \mathcal{K}, \mathcal{L}_0, \mathcal{H} \rangle$, we incrementally improve the design by allocating new hash layers to different partitions in a way that maximally improves the accuracy. We achieve this by computing, for each partition P_i, a per-layer resource demand term

$$demand(P_i) = |P_i|$$

and a degree of potential for improvement

$$potential_0(P_i) = err(\mathbb{P}_0) - err(\mathbb{P}_0^{(i\ddagger+1)}),$$

where $L_{i,0}$ is the hash layer assignment for partition P_i based on the initial solution $\mathbb{P}_0$ and $err(\mathbb{P}_0^{(i\ddagger+1)})$ is the error one would observe if the hash layer assignment for P_i is incremented by one:

$$err(\mathbb{P}_0^{(i\ddagger+1)}) = \sum_{i=1}^{m} \left(w_i \times \left((1 - \psi_i^{\kappa_i})^{L_{i,0}+1} \right) \right).$$

Given these two terms, for improvement, we select the layer i with the largest potential among those layers with resource demands that are less than the available resources,

$$demand(P_i) \leq resource_{max} - resource(\mathbb{P}_0),$$

and we increase the number of hash layers assigned to this onion layer by one. This gives us a revised resource allocation, $\mathbb{P}_1 = \langle n, \mathcal{P}, \mathcal{K}, \mathcal{L}_1, \mathcal{H} \rangle$. Similar to OLSH, the process is repeated (as shown in Algorithm 9) until the resources have been used up such that no more improvements is possible.

Top-k Search with PLSH

The outline of the top-k search process is given in Algorithm 10. For each partition, the algorithm fetches top-k results as each partition can potentially contribute to k elements to the final result set given a query q. The final result is than obtained by

Recall									
	Anticorrelated			**Correlated**			**Independent**		
Count(M)	EW	EDC	EDP	EW	EDC	EDP	EW	EDC	EDP
0.1	9.52	99.9	80.59	33.86	99.7	39.99	5.49	98.54	89.98
0.5	9.48	99.1	93.79	12.15	98.3	51.53	3.00	94.92	93.37
1	9.44	96.39	93.66	21.89	99.5	99.93	4.86	90.99	89.27
2	4.68	97.32	92.14	38.68	96.43	93.66	2.12	90.48	89.85

Table 6.2: Total amount of hash used for different top-k queries (100D target average redundancy 2, index designed for up to top-25 retrieval)

combining partial results from the m partitions and selecting the best k out of them. The overall process is analogous to that of OLSH, with modifications on resource allocation learned from *beta distribution* over different partitions and candidate set generation during search process.

Sample Results

As shown in the sample result, by leveraing hash redundancy 2, learning the beta distribution to assign layer weight, we can achieve high recall accuracy for top-k data retrieval. In general, when each partition contains equal amount of data (EDC partition strategy) output performs the others since the other two strategies face data skewness: to fill equal norm range width or equal probability slicing, the imbalanced data distribution would undermine the indexing and retrieval process when assigning hash budget.

6.2 Conclusion

As shown above, I proposed a novel framework Partition-LSH index structure to approximately solve top-k inner product queries, which is analogous to OLSH but to handle the situation where the dimensionality of data is high, i.e. 100 such that creating convex hull or Skyline layers is computationally infeasible. I proceed to partition the data based on its l_2 norm values and learn the importance of each partition to allcoate limited hash resources. The result has shown effectiveness of parition-aware resource allocation strategy for high dimensional data, which shows data indexing scheme with learned properties can significantly improve query accuracy.

Algorithm 7 EDC Partition

```
    Input: $\mathcal{D}$, $m$, $Beta(\alpha,\beta)$, $norm_{EW}$
    Output: $P_{EDC}$, $\vec{w_{EDC}}$
1:  $NormList$ = CalculateNorm($D$)
2:  $SortedNormList$ = SortNorm(NormList)
3:  $P_{EDC}, norm_{P_{EDC}}$ = UniSplit($\mathcal{D}$, $m$)
4:  $EDC\_list = \emptyset$
5:  for $1 \leq i \leq m$ do
6:      $max_{cur} = max(norm_{P_{EDC_i}})$
7:      for $1 \leq i \leq m$ do
8:          if $norm_{EW_i} \leq max_{cur} \leq norm_{EW_{i+1}}$ then
9:              $norm\_range = norm_{EW_{i+1}} - norm_{EW_i}$
10:             $norm_i = norm_{EW_i} + max_{cur}/(norm\_range)$
11:         end if
12:         add $norm_i$ into $EDC\_list$
13:     end for
14: end for
15: for $1 \leq i \leq m$ do
16:     $w_{EDC_i} = cdf(EDC\_list(i+1)) - cdf(EDC\_list(i))$
17: end for
18: return $P_{EDC}, \vec{w_{EDC}}$
```

Algorithm 8 EDP Partition

```
    Input: $\mathcal{D}$, $m$, $Beta(\alpha, \beta)$, $norm_{EW}$
    Output: $P_{EDP}$, $\vec{w}$
 1: $NormList$ = CalculateNorm($D$)
 2: $SortedNormList$ = SortNorm(NormList)
 3: prob_step = $\frac{1}{m}$
 4: cur_prob = prob_step
 5: $P_{EDP} = \emptyset$
 6: $pre_{norm} = 0$
 7: for $1 \leq i \leq m$ do
 8:     $p_i$ = Identify_Partition_Index($Beta(\alpha, \beta), m, cur\_prob$)
 9:     $p_{low} = floor(p_i), p_{high} = ceil(p_i)$
10:     $low_{norm} = EW_{p_{low}}, high_{norm} = EW_{p_{high}}$
11:     $target_{norm} = low_{norm} + (p_i - p_{low}) * (high_{norm} - low_{norm})$
12:     $P_{EDP_i} = FindDataIndex(pre_{norm} \leq NormList \leq target_{norm})$
13:     $\vec{w_i}$ = cur_prob
14:     cur_prob = cur_prob + prob_step
15: end for
16: return $P_{EDP}, \vec{w}$
```

Algorithm 9 ReviseLayers

```
Input: Partition $\mathcal{P}$, # of partitions $m$, budget $B$, per-layer hash counts, $\mathcal{K}$, initial layer assignment $\mathcal{L}_0$, hash family, $\mathcal{H}$
Output: Revised layer assignment, $\mathcal{L}$
1: resources = ComputeResources($\mathcal{P}$, $\mathcal{L}_0$)
2: $\mathcal{L} = \mathcal{L}_0$
3: BestLayer = $\infty$
4: Available = $B-$ resources
5: while (Available > 0) and (BestLayer $\neq \perp$) do
6:     BestPotential = -1
7:     BestLayer = $\perp$
8:     Used = 0
9:     for $1 \leq i \leq m$ do
10:        $potential_i$ = ComputePotential($\mathcal{P}$, $m$, $\mathcal{K}$, $\mathcal{L}$, $\mathcal{H}$)
11:        $demand_i$ = Size($P_i$)
12:        if ($demand_i \leq$ Available) and ($potential_i >$ BestPotential) then
13:            BestPotential = $potential_i$
14:            BestLayer = $i$
15:            Used = $demand_i$
16:        end if
17:    end for
18:    if BestLayer $\neq \perp$ then
19:        $\mathcal{L}$ = incrementPartition($\mathcal{L}$, BestLayer)
20:        Available = Available - Used
21:    end if
22: end while
23: return L
```

Algorithm 10 Top-K Search

```
    Input: PLSH index, $\mathbb{P}$, a top-$k$ query $q$
    Output: result set $R$
1: $TempRes = \emptyset$
2: for $1 \le i \le k$ do
3:     $Res_i$ = searchLSHPartitions($q$, $\mathbb{P}$, $i$, $k - i + 1$)
4:     $TempRes$= pickBest($TempRes \cup Res_i$, $k$)
5: end for
6: return $R = TempRes$
```

Chapter 7

CONCLUSION

The main goal of this dissertation is to design and develop efficient indexing scheme and algorithm framework for multi-dimensional data so that experts can better interpret data for decision making. Particularly, I look at two different models: a) multi-variate time series, b) high dimensional data indexing and querying processing with limited computational resources. More specifically, for multi-variate time series, it can be interrelated with multi-resolution which includes motion and gesture data. In thesis, I presented a metadata-enriched multi-variate time series model, where a dependency/correlation model relates the individual variates to each other. Recognizing that multi-variate temporal features can be extracted more effectively by simultaneously considering, at multiple scales, differences among individual variates along with the dependency/correlation model that relates them, I further developed algorithms to detect robust multi-variate temporal (RMT) features that are multi-resolution, local, and invariant against various types of noise. Later on I leverage RMT algorithm to build system framework to detect multi-variate time series features for predicating the diffusion process of diseases and the impact of natural disasters.

In tems of high dimensional data indexing and query processing, I presented a novel Onion-LSH(OLSH) index sturcutre to approximately solve top-k inner product queries, which combines the Onion-based data layering with angular LSH-based indexing for efficient data access. As there is a trade-off between the available hashing resources and the accuracy, I propose a top-k based accuracy model, with a layer-aware resource allocation strategy which takes into account the distribution of data and the layer contribution to the final result set. I also show the alternative layer indexing by using Skyline, as SLSH. Moreoever, as OLSH and SLSH leverages the fixed amount of hash budget and minimizes the total amount of error, I also proposed to *reverse* the optimization process, with the total error as input and minimize the total hash resource across k layers. Last but not least, I also proposed an algorithm when dimensionality is high i.e. 100, a situation where generating convex hull for Onion indexing or Skyline layers creation is computational infeasible. I adopted the partition strategy based on l_2 norm of data elements then proceed to learning the weight of each partition, to achieve partition-aware resource allocation intelligently leveraging the total hash resources available. Experimental results with real and synthetic benchmarks showed that that the proposed OLSH technique achieves the target accuracy rates within given resource budget under different scenarios. The results have shown that SLSH follows a similar pattern as OLSH, both of which indicate that by learning the structure of input data, one can achieve top-k query retrieval effectively and efficiently. In a nutshell, results have shown the effectiveness and efficiency of data indexing scheme with learned properties of data.

REFERENCES

"Exact euclidean lsh", `http://www.mit.edu/~andoni/LSH/`, accessed: 2018-03-09 (1999).

"GleamViz", "`http://www.gleamviz.org/simulator/`" (2019).

"Socioeconomic data and applications center (sedac)", `http://sedac.ciesin.columbia.edu.`, accessed: 10 May 2016 (2019).

"The swine flu outbreak and its global economic impacts", `http://www.brookings.edu/research/interviews/2009/05/04-swine-flu-mckibbin`, accessed: 10 May 2016 (2019).

Abubakar, I., P. Gautret, G. W. Brunette, L. Blumberg, D. Johnson, G. Poumerol, Z. A. Memish, M. Barbeschi and A. S. Khan, "Global perspectives for prevention of infectious diseases associated with mass gatherings", The Lancet infectious diseases (2012).

Aggarwal, C. C., "On effective classification of strings with wavelets", in "Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining", pp. 163–172 (ACM, 2002).

Balcan *et al.*, D., "Seasonal transmission potential and activity peaks of the new influenza a(h1n1): a monte carlo likelihood analysis based on human mobility", BMC Medicine **7**, 45 (2009).

Barber, C. B., D. P. Dobkin and H. Huhdanpaa, "The quickhull algorithm for convex hulls", ACM Transactions on Mathematical Software (TOMS) **22**, 4, 469–483 (1996).

Batal, I., D. Fradkin, J. Harrison, F. Moerchen and M. Hauskrecht, "Mining recent temporal patterns for event detection in multivariate time series data", in "Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining", pp. 280–288 (ACM, 2012).

Bawa, M., T. Condie and P. Ganesan, "Lsh forest: Self-tuning indexes for similarity search", in "Proceedings of the 14th International Conference on World Wide Web", WWW '05, pp. 651–660 (ACM, New York, NY, USA, 2005), URL `http://doi.acm.org/10.1145/1060745.1060840`.

Bemdt, D. J. and J. Clifford, "Using dynamic time warping to find patterns in time series", (1994).

Blei, D. M. and J. D. Lafferty, "Dynamic topic models", in "Proceedings of the 23rd international conference on Machine learning", pp. 113–120 (ACM, 2006).

Borzsony, S., D. Kossmann and K. Stocker, "The skyline operator", in "Data Engineering, 2001. Proceedings. 17th International Conference on", pp. 421–430 (IEEE, 2001).

Candan, K. S., R. Rossini, M. L. Sapino and X. Wang, “sdtw: Computing dtw distances using locally relevant constraints based on salient feature alignments”, PVLDB **5**, 11, 1519–1530 (2012a).

Candan, K. S., R. Rossini, X. Wang and M. L. Sapino, “sdtw: computing dtw distances using locally relevant constraints based on salient feature alignments”, Proceedings of the VLDB Endowment **5**, 11, 1519–1530 (2012b).

Candan, K. S. and M. L. Sapino, *Data management for multimedia retrieval* (Cambridge University Press, 2010a).

Candan, K. S. and M. L. Sapino, *Data Management for Multimedia Retrieval* (Cambridge University Press, New York, NY, USA, 2010b), ISBN-10:0521887399, ISBN-13: 978-0521887397, May 31, 2010.

Castro, N. and P. Azevedo, “Multiresolution motif discovery in time series”, in “Proceedings of the 2010 SIAM international conference on data mining”, pp. 665–676 (SIAM, 2010).

Chang, Y.-C., L. Bergman, V. Castelli, C.-S. Li, M.-L. Lo and J. R. Smith, “The onion technique: indexing for linear optimization queries”, in “ACM Sigmod Record”, vol. 29, pp. 391–402 (ACM, 2000).

Chen, L., *Similarity Search over Time Series and Trajectory Data*, Ph.D. thesis, University of Waterloo (2005).

Chen, L. and R. Ng, “On the marriage of lp-norms and edit distance”, in “VLDB”, (2004).

Chen, Y., E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen and G. Batista, “The ucr time series classification archive”, `www.cs.ucr.edu/~eamonn/time_series_data/` (2015).

Chowell, G., P. W. Fenimore, M. A. Castillo-Garsow and C. Castillo-Chavez, “Sars outbreaks in ontario, hong kong and singapore: the role of diagnosis and isolation as a control mechanism”, Journal of theoretical biology (2003).

Chung, F.-L., T. C. Fu, R. Luk and V. Ng, “Flexible time series pattern matching based on perceptually important points”, (2001).

Colizza, V., A. Barrat, M. Barthelemy, A. Valleron and A. Vespignani, “Modeling the worldwide spread of pandemic influenza: baseline case and containment interventions”, PLoS Comput Biol **4**, 1 (2007).

Datar, M., N. Immorlica, P. Indyk and V. S. Mirrokni, “Locality-sensitive hashing scheme based on p-stable distributions”, in “Proceedings of the twentieth annual symposium on Computational geometry”, pp. 253–262 (ACM, 2004).

De Silva, A., R. J. Hyndman and R. Snyder, “The vector innovations structural time series framework: a simple approach to multivariate forecasting”, Statistical Modelling **10**, 4, 353–374 (2010).

Deodhar *et al.*, S., “An interactive, web-based high performance modeling environment for computational epidemiology”, ACM TMIS **5**, 2, 7:1–7:27 (2014).

Ding, H., G. Trajcevski, P. Scheuermann, X. Wang and E. Keogh, “Querying and mining of time series data: Experimental comparison of representations and distance measures”, VLDB pp. 1542–1552 (2008).

Eichler, M., “Granger causality and path diagrams for multivariate time series”, Journal of Econometrics (2006).

EmitLab-ASU, “Rmt code”, Available upon request (2017).

Esling, P. and C. Agon, “Time-series data mining”, ACM Computing Surveys (CSUR) **45**, 1, 12 (2012).

Fagin, R., A. Lotem and M. Naor, “Optimal aggregation algorithms for middleware”, in “PODS”, pp. 102–113 (2001).

Fagin, R., A. Lotem and M. Naor, “Optimal aggregation algorithms for middleware”, Journal of computer and system sciences **66**, 4, 614–656 (2003).

Ferguson, N., D. Cummings, S. Cauchemez, C. Fraser, S. Riley, A. Meeyai, S. Iamsirithaworn and D. Burke, “Strategies for containing an emerging influenza pandemic in southeast asia”, Nature **534**, 7046 (2005).

Gan, J., J. Feng, Q. Fang and W. Ng, “Locality-sensitive hashing scheme based on dynamic collision counting”, SIGMOD ’12 (2012).

Gao, J., H. Jagadish, B. C. Ooi and S. Wang, “Selective hashing: Closing the gap between radius search and k-nn search”, KDD ’15 (2015).

Germann, T., K. Kadau, I. Longini Jr. and C. Macken, “Mitigation strategies for pandemic influenza in the united states”, Natl Acad Sci U S A **103**, 15 (2006).

Gionis, A., P. Indyk and R. Motwani, “Similarity search in high dimensions via hashing”, in “Proceedings of the 25th International Conference on Very Large Data Bases”, VLDB ’99, pp. 518–529 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999), URL `http://dl.acm.org/citation.cfm?id=645925.671516`.

Harris, C. and M. Stephens, “A combined corner and edge detector.”, Fourth Alvey Vision Conference pp. 147–151 (1988).

Harvey, A. and S. Koopman, “Multivariate structural time series model”, in “System Dynamics in Economic and Financial Models”, pp. 269–296 (John Wiley and Sons, 1997).

Heo, J.-S., J. Cho and K.-Y. Whang, “The hybrid-layer index: A synergic approach to answering top-k queries in arbitrary subspaces”, in “Data Engineering (ICDE), 2010 IEEE 26th International Conference on”, pp. 445–448 (IEEE, 2010).

Huang, Q., G. Ma, J. Feng, Q. Fang and A. K. H. Tung, "Accurate and fast asymmetric locality-sensitive hashing scheme for maximum inner product search", in "Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &#38; Data Mining", KDD '18 (2018).

Ilyas, I. F., G. Beskales and M. A. Soliman, "A survey of top-k query processing techniques in relational database systems", ACM Computing Surveys (CSUR) **40**, 4, 11 (2008).

Indyk, P. and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality", in "Proceedings of the thirtieth annual ACM symposium on Theory of computing", pp. 604–613 (ACM, 1998).

Ji, X., J. Bailey and G. Dong, "Mining minimal distinguishing subsequence patterns with gap constraints", in "KAIS", (2007).

Jin, Y. and B. Prabhakaran, "Knowledge discovery from 3d human motion streams through semantic dimensional reduction", ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) **7**, 2, 9 (2011a).

Jin, Y. and B. Prabhakaran, "Knowledge discovery from 3d human motion streams through semantic dimensional reduction", ACM Transactions on Multimedia Computing, Communications and Applications **7**, 2 (2011b).

Joly, A. and O. Buisson, "A posteriori multi-probe locality sensitive hashing", in "Proceedings of the 16th ACM International Conference on Multimedia", MM '08 (2008).

Keogh, E., "Exact indexing of dynamic time warping", in "VLDB", pp. 406–417 (2002).

Keogh, E., K. Chakrabarti, M. Pazzani and S. Mehrotra, "Dimensionality reduction for fast similarity search in large time series databases", Knowledge and information Systems **3**, 3, 263–286 (2001).

Keogh, E. and C. A. Ratanamahatana, "Exact indexing of dynamic time warping", KAIS (2005).

Khan, K., J. Arino, W. Hu, P. Raposo, J. Sears, F. Calderon, C. Heidebrecht, M. Macdonald, J. Liauw, A. Chan *et al.*, "Spread of a novel influenza a (h1n1) virus via global airline transportation", New England journal of medicine (2009).

Kim, D. and B. Prabhakaran, "Motion fault detection and isolation in body sensor networks", Pervasive and Mobile Computing **7**, 6, 727–745, URL `https://doi.org/10.1016/j.pmcj.2011.09.006` (2011a).

Kim, D.-J. and B. Prabhakaran, "Motion fault detection and isolation in body sensor networks", Pervasive and Mobile Computing **7**, 6, 727–745 (2011b).

Kolda, T. G. and B. W. Bader, "Tensor decompositions and applications", SIAM review **51**, 3, 455–500 (2009).

Kruskal, J. B., “An overview of sequence comparison: Time warps, string edits, and macromolecules”, SIAM Review **25**, 2, 201–237 (1983).

Krzanowski, W., “Between-groups comparison of principal components”, Journal of the American Statistical Assoc. (1979).

Li, C., S. Zheng and B. Prabhakaran, “Segmentation and recognition of motion streams by similarity search”, ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) **3**, 3, 16 (2007a).

Li, C., S. Q. Zheng and B. Prabhakaran, “Segmentation and recognition of motion streams by similarity search”, ACM Trans. Multimedia Comput. Commun. Appl. **3**, 3, URL `http://doi.acm.org/10.1145/1236471.1236475` (2007b).

Li, L., B. A. Prakash and C. Faloutsos, “Parsimonious linear fingerprinting for time series”, Proceedings of the VLDB Endowment **3**, 1-2, 385–396 (2010).

Li, T., S. Ma and M. Ogihara, “Wavelet methods in data mining”, in “Data Mining and Knowledge Discovery Handbook”, pp. 603–626 (Springer, 2005).

Lin, J., E. Keogh, S. Lonardi and B. Chiu, “A symbolic representation of time series, with implications for streaming algorithms”, in “Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery”, pp. 2–11 (ACM, 2003a).

Lin, J., E. Keogh, S. Lonardi and B. Chiu, “A symbolic representation of time series, with implications for streaming algorithms”, in “Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery”, DMKD ’03, pp. 2–11 (ACM, 2003b).

Liu, S., Y. Garg, K. S. Candan, M. L. Sapino and G. Chowell-Puente, “Notes2: Networks-of-traces for epidemic spread simulations”, in “Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence”, (2015).

Liu, S., S. Poccia, K. S. Candan, G. Chowell and M. L. Sapino, “epidms: data management and analytics for decision-making from epidemic spread simulation ensembles”, The Journal of infectious diseases **214**, suppl_4, S427–S432 (2016).

Liu, S., S. R. Poccia, K. S. Candan, M. L. Sapino and X. Wang, “Robust multi-variate temporal features of multi-variate time series”, ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) (2018).

Liu, Y., J. Cui, Z. Huang, H. Li and H. T. Shen, “Sk-lsh: An efficient index structure for approximate nearest neighbor search”, Proc. VLDB Endow. **7**, 9, 745–756, URL `http://dx.doi.org/10.14778/2732939.2732947` (2014).

Longini Jr.*et al.*, I., “Containing pandemic influenza at the source”, Science **309**, 5737 (2005).

Lowe, D. G., “Object recognition from local scale-invariant features”, in “Computer vision, 1999. The proceedings of the seventh IEEE international conference on”, vol. 2, pp. 1150–1157 (Ieee, 1999a).

Lowe, D. G., “Object recognition from local scale-invariant features”, in “Proceedings of the International Conference on Computer Vision”, ICCV ’99 (1999b).

Lowe, D. G., “Distinctive image features from scale-invariant keypoints”, International journal of computer vision **60**, 2, 91–110 (2004a).

Lowe, D. G., “Distinctive image features from scale-invariant keypoints”, Int. J. Comput. Vision **60**, 2, 91–110 (2004b).

Lowe, D. G., “Distinctive image features from scale-invariant keypoints”, Int. J. Comput. Vision **60**, 2 (2004c).

Lv, Q., W. Josephson, Z. Wang, M. Charikar and K. Li, “Multi-probe lsh: Efficient indexing for high-dimensional similarity search”, in “Proceedings of the 33rd International Conference on Very Large Data Bases”, VLDB ’07, pp. 950–961 (VLDB Endowment, 2007), URL `http://dl.acm.org/citation.cfm?id=1325851.1325958`.

Mehta, S., R. Nallusamy, R. V. Marawar and B. Prabhakaran, “A study of dwt and svd based watermarking algorithms for patient privacy in medical images”, in “Healthcare Informatics (ICHI), 2013 IEEE International Conference on”, pp. 287–296 (IEEE, 2013a).

Mehta, S., R. Nallusamy, R. V. Marawar and B. Prabhakaran, “A study of DWT and SVD based watermarking algorithms for patient privacy in medical images”, in “ICHI’13”, pp. 287–296 (2013b), URL `https://doi.org/10.1109/ICHI.2013.41`.

Merler, S. and M. Ajelli, “The role of population heterogeneity and human mobility in the spread of pandemic influenza.”, Proc Biol Sci. **277**, 1681 (2014).

Merler, S., M. Ajelli, A. Pugliese and N. Ferguson, “Determinants of the spatiotemporal dynamics of the 2009 h1n1 pandemic in europe: implications for real-time modelling.”, PLoS Comput Biol. **7**, 9 (2011).

Mills, T. C. and T. C. Mills, *Time series techniques for economists* (Cambridge University Press, 1991).

Mocap, “Cmu mocap data set”, Http://mocap.cs.cmu.edu/ (2001).

Mohammad, Y. and T. Nishida, “Constrained motif discovery in time series”, New Generation Computing **27**, 4, 319 (2009).

Mörchen, F., “Time series feature extraction for data mining using dwt and dft”, (2003).

Mossong, J., N. Hens, M. Jit, P. Beutels, K. Auranen, R. Mikolajczyk, M. Massari, S. Salmaso, G. S. Tomba, J. Wallinga, J. Heijne, M. Sadkowska-Todys, M. Rosinska and W. J. Edmunds, “Social contacts and mixing patterns relevant to the spread of infectious diseases”, PLoS Med **5**, 3 (2008).

Motwani, R., A. Naor and R. Panigrahi, “Lower bounds on locality sensitive hashing”, in “Proceedings of the Twenty-second Annual Symposium on Computational Geometry”, SCG ’06, pp. 154–157 (ACM, New York, NY, USA, 2006), URL `http://doi.acm.org/10.1145/1137856.1137881`.

Nagarkar, P. and K. S. Candan, “PSLSH: an index structure for efficient execution of set queries in high-dimensional spaces”, in “CIKM”, pp. 477–486 (ACM, 2018).

Neyshabur, B. and N. Srebro, “On symmetric and asymmetric lshs for inner product search”, in “Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37”, ICML’15, pp. 1926–1934 (JMLR.org, 2015), URL `http://dl.acm.org/citation.cfm?id=3045118.3045323`.

Papadias, D., Y. Tao, G. Fu and B. Seeger, “An optimal and progressive algorithm for skyline queries”, in “Proceedings of the 2003 ACM SIGMOD international conference on Management of data”, pp. 467–478 (ACM, 2003).

Papadias, D., Y. Tao, G. Fu and B. Seeger, “Progressive skyline computation in database systems”, ACM Transactions on Database Systems (TODS) **30**, 1, 41–82 (2005).

Papadimitriou, S., J. Sun and C. Faloutsos, “Streaming pattern discovery in multiple time-series”, in “Proceedings of the 31st international conference on Very large data bases”, pp. 697–708 (VLDB Endowment, 2005).

Patel, P., E. Keogh, J. Lin and S. Lonardi, “Mining motifs in massive time series databases”, in “Proceedings of IEEE International Conference on Data Mining”, ICDM ’02, pp. 370–377 (2002), December.

Peng, J., H. Wang, J. Li and H. Gao, “Set-based similarity search for time series”, in “Proceedings of the 2016 International Conference on Management of Data”, pp. 2039–2052 (ACM, 2016a).

Peng, J., H. Wang, J. Li and H. Gao, “Set-based similarity search for time series”, in “Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016”, edited by F. Özcan, G. Koutrika and S. Madden, pp. 2039–2052 (ACM, 2016b), URL `http://doi.acm.org/10.1145/2882903.2882963`.

Perng, C.-S., H. Wang, S. R. Zhang and D. S. Parker, “Landmarks: a new model for similarity-based pattern querying in time series databases”, in “Proceedings of 16th International Conference on Data Engineering (Cat. No. 00CB37073)”, pp. 33–42 (IEEE, 2000).

Poccia, S. R. and Y. Garg, “On the effectiveness of distance measures for similarity search in multi-variate sensory data: Eectiveness of distance measures for similarity search”, in “ICMR’17”, pp. 489–493 (2017).

Poccia, S. R., M. L. Sapino, X. C. Sicong Liu, Y. Garg, S. Huang, J. H. Kim, X. Li, P. Nagarkar and K. S. Candan, "SIMDMS: Data management and analysis to support decision making through large simulation ensembles", in "EDBT'17", pp. 582–585 (2017).

Rakthanmanon, T., B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria and E. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping", in "KDD", (2012).

Rakthanmanon, T. and E. Keogh, "Fast shapelets: A scalable algorithm for discovering time series shapelets", in "proceedings of the 2013 SIAM International Conference on Data Mining", pp. 668–676 (SIAM, 2013).

Sakoe, H. and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition", in "Acoustics, Speech and Signal Processing, IEEE Transactions on", (1978).

Salton, G. and M. J. McGill, *Introduction to Modern Information Retrieval* (1983).

Sanguansat, P., "Multiple multidimensional sequence alignment using generalized dynamic time warping", **8**, 10, 668–678 (2012).

Shrivastava, A. and P. Li, "Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips)", in "Advances in Neural Information Processing Systems", pp. 2321–2329 (2014).

Shuai, L., C. Li, X. Guo, B. Prabhakaran and J. Chai, "Motion capture with ellipsoidal skeleton using multiple depth cameras", IEEE Trans. Vis. Comput. Graph. **23**, 2, 1085–1098, URL `https://doi.org/10.1109/TVCG.2016.2520926` (2017).

Silva, A., R.J.Hyndman and R.Snyder, "The vector innovations structural time series framework: A simple approach to multivariate forecasting.", Statistical Modelling **10**, 4, 353–374 (2010).

STEM, "Spatiotemporal epidemiological modeler", Https://www.eclipse.org/stem/ (2016).

Sun, Y., W. Wang, J. Qin, Y. Zhang and X. Lin, "Srs: Solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index", Proc. VLDB Endow. **8**, 1, 1–12, URL `http://dx.doi.org/10.14778/2735461.2735462` (2014).

Tan, K.-L., P.-K. Eng and B. C. Ooi, "Efficient progressive skyline computation", in "VLDB", vol. 1, pp. 301–310 (2001).

Theobald, M., G. Weikum and R. Schenkel, "Top-k query evaluation with probabilistic guarantees", in "Proceedings of the Thirtieth international conference on Very large data bases-Volume 30", pp. 648–659 (VLDB Endowment, 2004).

Van den Broeck, W., C. Gioannini, B. Gonçalves, M. Quaggiotto, V. Colizza and A. Vespignani, "The gleamviz computational tool, a publicly available software to explore realistic epidemic spreading scenarios at the global scale", BMC infectious diseases (2011).

Vlachos, M., M. Hadjieleftheriou, D. Gunopulos and E. Keogh, "Indexing multidimensional time-series", The VLDB Journal **15**, 1, 1–20 (2006).

Wang, J., H. T. Shen, J. Song and J. Ji, "Hashing for similarity search: A survey", CoRR **abs/1408.2927**, URL `http://arxiv.org/abs/1408.2927` (2014a).

Wang, X. and K. S. Candan, "Relevant shape contour snippet extraction with metadata supported hidden markov models", in "Proceedings of the ACM International Conference on Image and Video Retrieval", pp. 430–437 (ACM, 2010).

Wang, X., K. S. Candan and M. L. Sapino, "Leveraging metadata for identifying local, robust multi-variate temporal (rmt) features", in "Data Engineering (ICDE), 2014 IEEE 30th International Conference on", pp. 388–399 (IEEE, 2014b).

Wang, X., K. S. Candan and M. L. Sapino, "Leveraging metadata for identifying local, robust multi-variate temporal (RMT) features", in "Data Engineering (ICDE), 2014 IEEE 30th International Conference on", pp. 388–399 (IEEE, 2014c).

Wang, X., J. Lin, P. Senin, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen and S. Frankenstein, "Rpm: Representative pattern mining for efficient time series classification.", in "EDBT", pp. 185–196 (2016a).

Wang, X., J. Lin, P. Senin, T. Oates, S. Gandhi, A. P. Boedihardjo, C. Chen and S. Frankenstein, "RPM: representative pattern mining for efficient time series classification", in "Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016.", pp. 185–196 (2016b), URL `https://doi.org/10.5441/002/edbt.2016.19`.

Wu *et al.*, J., "School closure and mitigation of pandemic (h1n1) 2009, hong kong", Emerg Infect Dis **16**, 3 (2010).

Yang, K. and C. Shahabi, "A pca-based similarity measure for multivariate time series", in "MMDB", pp. 65–74 (ACM, 2004).

Yankov, D., E. Keogh, J. Medina, B. Chiu and V. Zordan, "Detecting time series motifs under uniform scaling", in "Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining", pp. 844–853 (ACM, 2007).

Ye, L. and E. Keogh, "Time series shapelets: a new primitive for data mining", in "Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining", pp. 947–956 (ACM, 2009).

Zhang, W., K. Gao, Y.-d. Zhang and J.-t. Li, “Data-oriented locality sensitive hashing”, in “Proceedings of the International Conference on Multimedia”, MM ’10, pp. 1131–1134 (ACM, New York, NY, USA, 2010), URL `http://doi.acm.org/10.1145/1873951.1874168`.