Feature Extraction Using Topological Data Analysis for Machine Learning and Network Science Applications

Wei Guo

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2019

Reading Committee:

Ashis G. Banerjee, Chair

Linda Ng Boyle

Archis Ghate

Program Authorized to Offer Degree:
Industrial & Systems Engineering

University of Washington

Abstract

Feature Extraction Using Topological Data Analysis for
Machine Learning and Network Science Applications

Wei Guo

Chair of the Supervisory Committee:
Professor Ashis G. Banerjee
Industrial & Systems Engineering and Mechanical Engineering

Many real-world data sets can be viewed as a noisy sampling of an unknown high-dimensional topological space. The emergence and development of topological data analysis (TDA) over the last fifteen years or so provides a suite of tools to understand and exploit the topological structure of the underlying space from a multi-scale perspective that characterizes the shape of the data. This dissertation, thus, aims to leverage the shape information of data offered by the TDA tools to extract key features in machine learning and network science problems. We investigate a few TDA topics that are understudied following this line of research.

We first extend the application of TDA to the manufacturing systems domain. We apply a widely used TDA method, known as the Mapper algorithm, on two benchmark data sets for chemical process yield prediction and semiconductor wafer fault detection. The algorithm yields topological networks that capture the intrinsic clusters and connections among the clusters (i.e., subgroups) present in the data sets, which are difficult to detect using traditional methods. Key process variables (features) that best differentiate the subgroups of interest are subsequently identified through statistical tests.

Next we present a new method, referred as Sparse-TDA method, that integrates QR pivoting-based sparse sampling algorithm into vector-based TDA method to transform topological features into image pixels and identify discriminative pixel samples (features) in the

presence of noisy and redundant information. We demonstrate its advantage over a state-of-the-art kernel TDA method and L_1-regularized feature selection methods in terms of classification accuracy and training time on three challenging data sets pertaining to 3D meshes of synthetic and real human postures and textured images.

Finally, we propose a method that extends the persistence-based TDA that is typically used for characterizing shapes to general networks. We introduce the concept of the community tree, a tree structure established based on clique communities from the clique percolation method, to summarize the topological structures in a network from a persistence perspective. Furthermore, we develop efficient algorithms to construct and update community trees by maintaining a series of clique graphs in the form of spanning forests, in which each spanning tree is built on an underlying Euler Tour tree. With the information revealed by community trees and the corresponding persistence diagrams, our proposed approach is able to detect clique communities and keep track of the major structural changes during their evolution given a stability threshold. The results demonstrate its effectiveness in extracting useful structural insights for time-varying social networks.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY

SIMPLEX: a collection of vertices such that any subset, called a face, is also a simplex; a $(k-1)$-simplex consists of k vertices.

FEATURE: a broad term and its meaning is highly domain-dependent. For example, a feature refers to a process variable in manufacturing systems output prediction, while for multi-way classification, it refers to a pixel in the persistence images. For complex networks, it refers to a community that forms one of the primary structural components of the networks.

ACKNOWLEDGMENTS

First, I am deeply indebted to my adviser, Professor Ashis G. Banerjee, who walked me into the exciting field of topological data analysis and directed me to complete this work. Without his support, encouragement and endless patience, I would not have come thus far and accomplished this research. His instruction has not only strengthened my technical skills and writing ability, allowing me to conduct my research more effectively, but also forced me to further develop my approach to problem solving and identify import research directions.

I also owe thanks to our collaborators, Professor Steve L. Brunton and Dr. Krithika Manohar, Professor Yen-Chi Chen and Ruqian Chen, for their contribution to this work. The fruitful discussions with them have led to excellent ideas and enriched my thought on the topics presented in Chapters 4-5.

In addition, I want to thank Professors Linda Ng Boyle and Archis Ghate for taking the time to be the members of my thesis reading committee. I am also very grateful to them for deciding to give me an opportunity five years ago, which allows me to have a wonderful learning experience at University of Washington.

I want to thank to my parents and my husband Dan, for their unending love and confidence in me that always encourages me. Any success that I have today could not have been possible without them. I am also very thankful to my in-laws, Connie and Steve, who have always been supportive and understanding. It is the strong support from my family that enables me to achieve my goal.

Finally, I would like to express my gratitude to Boeing for their financial support. This work is largely funded by Boeing under Contract # SSOW-BRT-W0714-0004 and Contract # 2018-BCAPD-039, supervised by Dr. Tom Hogan and Dr. Agnes Blom-Schieber, respectively.

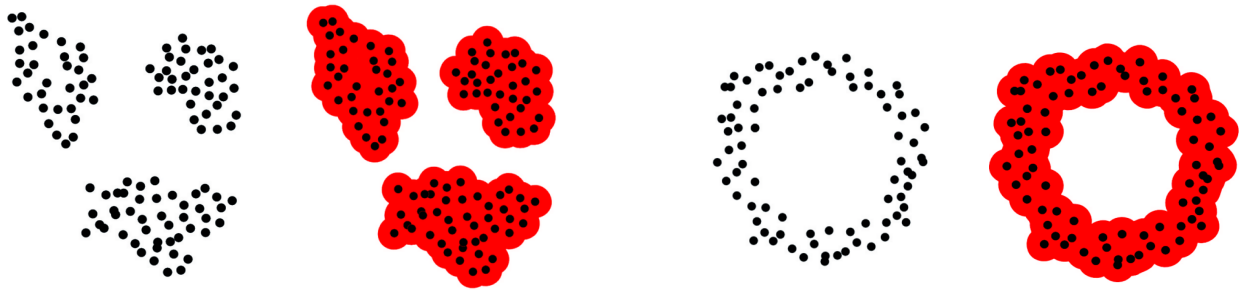
DEDICATION

To my parents and my husband who accompany me through this journey

Chapter 1

INTRODUCTION

It is a real challenge to glean valuable information from large, noisy and complex data. Traditional methods tend to rely on oversimplified assumptions to conduct data analysis. To fill this void, the field of topological data analysis (TDA) has since arisen with a strong theoretical base from algebraic topology and computational geometry aimed at providing powerful tools for a deeper understanding of the structure of data [23]. In TDA, we simply assume a data set is sampled from an unknown topological space which characterizes the "shape" of the data [135]. To explore the structure of data, a central idea of TDA is to study the underlying topological space instead so that matures tools from algebraic topology and computational geometry can be utilized to extract its topological and geometric features, which are, in turn, used to infer robust information about the structure of data qualitatively and/or quantitatively.



(a) X_1 and $X_1^{r_1}$ (b) X_2 and $X_2^{r_2}$

Figure 1.1: The structure of X_i in $\mathbb{R}^2$ reflected by its geometric features corresponds to the topological features of $X_i^{r_i}$, $i = 1, 2$. (a) Clusters in X_1 correspond to connected components in $X_1^{r_1}$. (b) A loop in X_2 corresponds to a cycle in $X_2^{r_2}$. Source: [86].

For example, given a finite set of points X in $\mathbb{R}^2$ (i.e., a point cloud), and let X^r be the union of balls each with radius r centered on a point in X. X^r can be then considered as an approximation of the topological space $\mathbb{X}$ underlying X. As shown in Figure 1.1, the geometric features of X_i, $i = 1, 2$, including clusters and loops, can be detected by studying the topological features of $X_i^{r_i}$ such as connected components and cycles. Thus, to study the structure of X using topology, we indeed study the topological features of X^r rather than those of X itself. In fact, computing the number of connected components and cycles from X_i with n points directly will offer no insight into the structure of X_i, resulting in only n connected components and 0 cycle [86].

1.1 Building Simplicial Complexes From Data

The goal of TDA now is to recover the topological space $\mathbb{X}$ underlying the data. To achieve this goal, a nature way is to connect nearby data points to create a skeleton approximation of $\mathbb{X}$. In TDA, the connectivity of data points is usually represented by simplicial complexes[1] that comprise a set of simplices. A simplicial complex can be viewed both as a topological space from which topological features can be inferred and a combinatorial representation that is suitable for effective computation [30].

Simplicial complexes are often built through the nerve of a cover. A cover of a data set, denoted by $\mathcal{U}$, is a collection of open sets $\{U_i\}_{i\in I}$ (I is the index set) such that every data point is included by at least one U_i. The nerve $N(\mathcal{U})$ of $\mathcal{U}$ is a simplicial complex constructed by collapsing each U_i into a vertex and creating a k-simplex for each $(k+1)$-way intersection of U_i's. Formally, a cover $\mathcal{U}$ is *good* if all U_i in the cover and all their non-empty finite intersections are contractible[2]. Typical examples of contractible spaces are the convex sets in $\mathbb{R}^n$. The nerve theorem in algebraic topology asserts that the nerve of a good cover $\mathcal{U}$ is homotopy equivalent to the union of U_i in the cover [112]. Since the union of U_i in the cover

[1] An exception is image data which consists of pixels (2D images), voxels or their higher-dimensional analogs. This type of data is more often represented by cubical complexes for computational efficiency [127].

[2] Intuitively, a set in $\mathbb{R}^n$ is said to be contractible if it can be continuously deformed to one of its points.

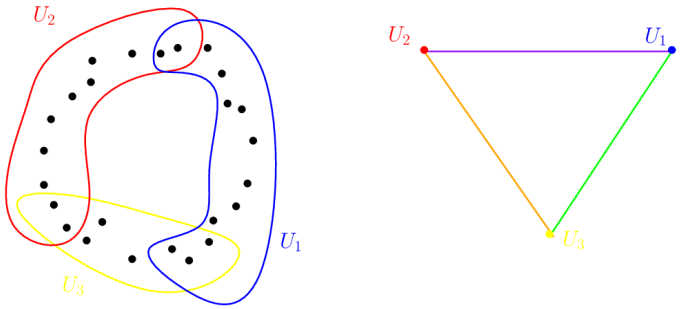


Figure 1.2: A cover of a data set in $\mathbb{R}^2$ and its nerve. Source: [30].

$\bigcup_{i\in I} U_i$ is an approximation of the unknown $\mathbb{X}$, the nerve theorem plays a fundamental role in TDA as it sheds light on approximating $\mathbb{X}$ using the nerve of covers.

1.2 Mapper Algorithm

One way to build covers is to apply a function to the data, which is a key step in the Mapper algorithm proposed by Singh et al. [116]. Inspired by the classical Morse theory, the Mapper algorithm is a highly successful tool in TDA for exploratory data analysis. In topology, Morse theory enables one to characterize the topology of a manifold via some functional level sets [96]. More specifically, given a manifold $\mathcal{M}$, when $h : \mathcal{M} \to \mathbb{R}$ is a smooth real-valued function (Morse function), topological transition of $\mathcal{M}$ is inferred from the level sets $h^{-1}(c)$ for some critical values c.

The Mapper algorithm extends this inference to the high dimensional data analysis with the assumption that high dimensional data resides on a low dimensional manifold or stratified space. Given a data set X and a well-chosen function $f : X \to \mathbb{R}^n$, the Mapper algorithm builds an overlapping cover $U = \{U_i\}_{i\in I}$ of $f(X)$ in a low dimensional space and then pulls it back to X via f^{-1}, resulting a collection of subsets $\{f^{-1}(U_i)\}_{i\in I}$ that defines a *pullback cover* of X. Next, a clustering algorithm is applied to each $f^{-1}(U_i)$, $i \in I$. The collection of these clusters is a *refinement* of $\{f^{-1}(U_i)\}_{i\in I}$, and the output of the Mapper algorithm is the nerve of the refined pullback cover of X [30].

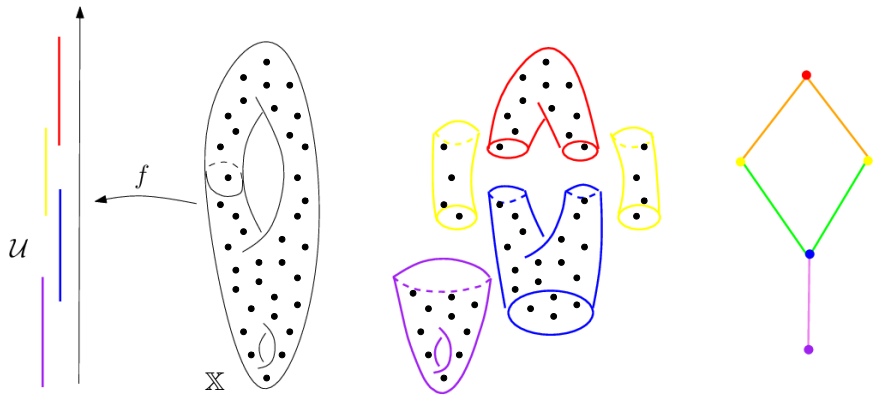


Figure 1.3: An example of the Mapper algorithm applied to a sampling of a double torus $\mathbb{X}$ in $\mathbb{R}^3$. Note that the refined pullback cover induced by the height function f and an interval cover $\mathcal{U}$ is not a good one since the purple cover element is not contractible. Despite the smaller cycle being missing, the larger one is reflected in the nerve. Source: edited from [30].

More generally, the Mapper algorithm can be defined in a topological space $\mathbb{X}$. The only difference from the definition above is the clustering step, where instead of partitioning each $f^{-1}(U_i)$ into clusters, we split each one into connected components. It is worth noting that there is little guarantee that the refined pullback cover of $\mathbb{X}$ induced by $(f,\mathcal{U})$ is a good cover. Even though the condition in the nerve theorem does not hold in general for the Mapper algorithm, the resulting nerve still preserves certain topology features of $\mathbb{X}$ and helps to point to areas of interest in the data set $X \subset \mathbb{X}$. Figure 1.3 gives an intuitive example of this scenario.

1.3 Persistent Homology

Another way to build covers is to generate the union of balls centered on the data points. Since balls are convex sets and the intersection of any finite number of convex sets is convex, the nerve of the cover is homotopy equivalent to the union of balls due to the nerve theorem. As r increases, the union of balls estimates the unknown $\mathbb{X}$ at multiple scales. When r is

sufficiently small, the simplicial complex taken from the nerve for a data set of n points consists of n 0-simplices. While r is sufficiently large, the simplicial complex turns to be a single $(n-1)$-simplex. The question now becomes if it is possible to find the optimal r that best captures the topology of the unknown $\mathbb{X}$ [59].

To answer this question, Edelsbrunner et al. introduces the concept of "persistence" as a measure to discern topological features of $\mathbb{X}$ [48]. The strategy is, rather than search for a fixed value of r, we examine a range of r starting from 0. Correspondingly, we obtain a nested sequence of simplicial complexes $K_1, \ldots, K_d$ such that $K_1 \subseteq \ldots \subseteq K_d$. During the process, known as a *filtration*, topological features appear and disappear at different scales that are referred to as the *birth* and *death* times of the features. From a geometric perspective, the topological features are interpreted as l-dimensional holes, e.g., connected components as 0-dimension holes, cycles as 1-dimensional holes and voids as 2-dimensional holes. See Figure 1.4 for an illustration. The lifetime of a topological feature is called persistence. The features that persist over a wide range of scales are considered as the true features of $\mathbb{X}$ while the short-lived ones are deemed as noise terms.

This framework is formalized as persistent homology, and it has become a predominant tool in TDA since its introduction [48, 136]. In practice, a more efficient way to generate a filtration is to consider the sublevel sets $S_t = f^{-1}([-\infty, t])$ of a descriptor function $f : \Omega \to \mathbb{R}$

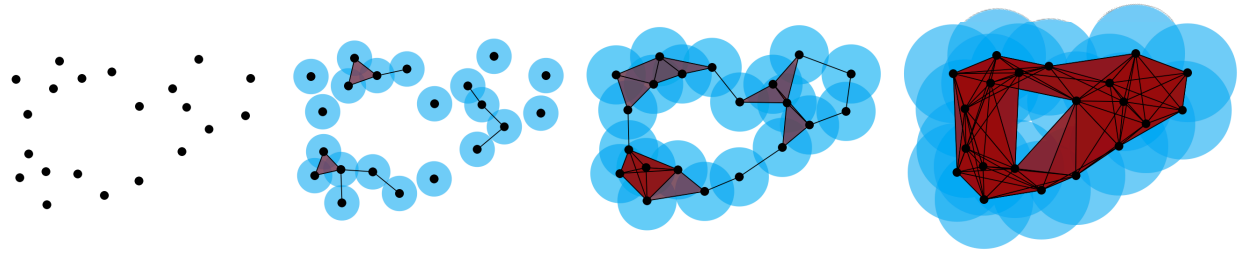


Figure 1.4: The filtration of a family of unions of balls and their corresponding nerves for a data set representing two loops. As r increases, the cycles appear and disappear. Source: `https://orbifold.net/default/what-is-persistent-homology/`.

defined on some domain Ω for $t \in \mathbb{R}$ based on the fact $S_i \subseteq S_j$ if $i \leq j$. Accordingly, persistent homology studies the topological changes of the sublevel sets $f^{-1}([-\infty, t])$ as t increases from $-\infty$ to ∞. If the function is the distance function $f(x) = \min_{y \in P} ||x - y||$ defined on $\mathbb{R}^n$ for a point cloud $P \in \mathbb{R}^n$, then the sublevel sets $f^{-1}([0, t])$ are union of balls with radius t around each point in P.

The topological information captured by persistent homology is commonly summarized in a persistent diagram (PD). A l-dimensional PD includes a countable multiset of points in $\mathbb{R}^2$, where each point (b, d) represents a l-dimensional hole that is born at time b and filled at time d, as well as the diagonal $\Delta = \{(b, b) \in \mathbb{R}^2 | b \in \mathbb{R}\}$ with points in Δ having infinite multiplicity. Figure 1.5 illustrates a filtration from a function $f : \mathbb{R} \to \mathbb{R}$, resulting a 0-dimensional PD.

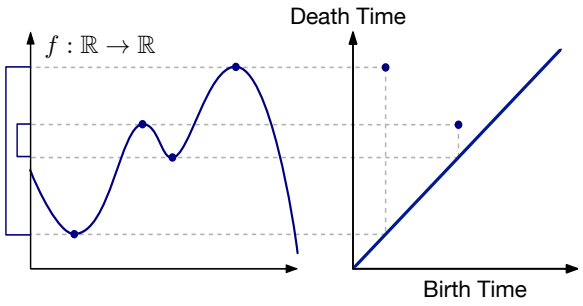


Figure 1.5: A function $f : \mathbb{R} \to \mathbb{R}$ (left) and its 0-dimensional PD (right). A connected component in the corresponding sublevel set is created (i.e., birth time) when t passes a new local minimum. As t increases and reaches a local maximum, it merges two connected components (i.e., death time) and is paired with the higher (younger) of the two local minima that represents the two components. The other minimum then represents the connected component resulting from the merger. Source: [109].

1.4 Community Tree

Establishing a filtration to track the evolution of topological features across the scales lies at the core of persistent homology. Jointly with researchers from UW Statistics and Mathematics, we extend this idea to the topological characterization of structural features in the form of communities (a.k.a. clusters or cohesive subgroups) in a network [32]. Generally speaking, a community is a subgraph such that "the number of internal edges is larger than the number of external edges" [55]. We adopt the description of a community from the well-known clique percolation method (CPM), where a community is defined as a k-clique chain, i.e., a union of k-cliques that can be reached from each other through a series of adjacent k-cliques [102]. Here a k-clique is a subgraph with k vertices connected to each other and two k-cliques are adjacent if they share $k-1$ vertices. Such a community is referred to as a k-clique community or k-community for short, denoted by $\mathcal{C}_k$, and k is called as its order. Note that by this definition, any arbitrary network G is a 1-community.

Since a k-clique itself is a $(k-1)$-clique chain, a k-community is also a $(k-1)$-community. This property allows any k-community $\mathcal{C}_k$ to grow a nested sequence of communities across different orders such that $\mathcal{C}_k \subseteq \mathcal{C}_{k-1} \subseteq \cdots \subseteq \mathcal{C}_1 = G$, which forms a filtration of G. As shown in Figure 1.6, the communities corresponding to a, b, d, e, f and the ones corresponding to c, d, e, f form two nested sequences starting from a and c, respectively. The multiple nested sequences originating from distinct communities during the filtration of G define a tree structure with the root node corresponding to G and the leaf nodes corresponding to the starting communities. Formally, let $\mathbb{C}_k = \{\mathcal{C}_{k,j} : j = 1, \cdots, J(k)\}$ be a collection of k-communities $\mathcal{C}_{k,j}$ in G, then the tree generated by the union of all communities at various orders $\mathbb{C} = \bigcup_{k\in\mathbb{N}} \mathbb{C}_k$ is called the community tree of G, denoted by $\mathcal{T}(G)$.

From a topological point of view, each nested sequence of communities records the evolution of the starting community as a connected component. Thus, the birth time of the connected component is defined as the order of the starting community. We say the starting community with a higher order is born earlier. When two connected components merge

at a certain order, the merging order is then defined as the death time of the connected component with a later birth time. We set the death time of the last remaining connected component, i.e., the one with the earliest birth time, to be 1. The birth and death time of each connected component in a community tree is also encoded in a PD[3]. A connected component with a longer persistence implies that its starting community behaves more robust against the changes occurring in the network.

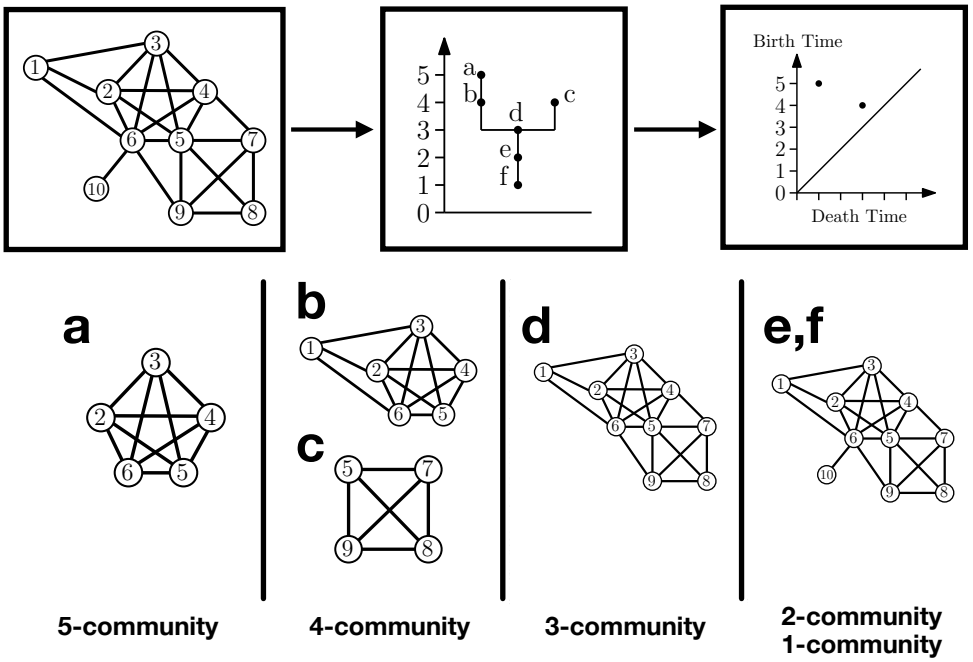


Figure 1.6: The construction of a community tree (top center) from a network (top left). The first connected component is born at $k = 5$ since it starts as a 5-community. At $k = 4$, the second connected component is born and then merged into the first connected component at $k = 3$. Accordingly, (1,5) and (3,4) are shown in the PD (top right) as the death and birth time for the two connected components. The alphabetical letter denotes the corresponding position of each community (bottom) in the community tree. Source: edited from [32].

[3]The x and y axes are labeled as death time and birth time, respectively, in this PD so that all the points corresponding connected components stay above the diagonal.

1.5 Outline

In this chapter, we first introduce the background of two prevalent tools in TDA from a standpoint of building covers. The Mapper algorithm builds one cover in a supervised manner, while persistent homology is motivated by building a sequence of covers to quantify topological features during a filtration to discern true features from noise. Following the strategy in persistent homology, we then introduce the concept of community trees aiming to measure the changes of communities for dynamic network analysis. Chapter 2 reviews the connection of the Mapper algorithm and persistent homology to machine learning and network science problems, where topological and geometric features are often transformed to new families of features for feature extraction. In Chapter 3, we initiate the application of the Mapper algorithm in the manufacturing systems in which key process variables (features) are usually difficult to detect using traditional methods. In Chapter 4, we switch our focus to persistent homology in the classification tasks, and present a method that realizes sparse selection of discriminative features for mesh objects including 3D shapes and images. Chapter 5 centers on developing efficient and scalable algorithms to construct and update a community tree, and exploring its use in community tracking and event detection on real-world networks. The last chapter summarizes the contribution and anticipated impact of the work in this dissertation, along with future research directions.

Chapter 2
LITERATURE REVIEW

This chapter concentrates on the practical aspects of the Mapper algorithm, persistent homology and related methods. It is beyond the scope of this chapter to provide an overview of TDA methods stemming from other theories, such as target tracking in sensor networks using sheaves and sheaf cohomology [36].

2.1 TDA in Machine Learning

In this section, we discuss the Mapper algorithm and persistent homology with more details, including their stability and the typical pipelines of the two approaches in solving machine learning problems.

2.1.1 Point Clouds and Mapper Algorithm

A data set consisting of N measurements with m features from practical applications is viewed as a point cloud of N points in $\mathbb{R}^m$ in TDA. Among various TDA methods, the Mapper algorithm is regarded well-suited for point cloud data. The Mapper algorithm does not seek to deliver a fully accurate representation of a data set, but rather allows one to rapidly obtain a qualitative understanding of how the data are organized on a large scale from its compressed output [116]. Given a data matrix $\mathbf{X} \in \mathbb{R}^{N\times m}$, the setup of the Mapper algorithm includes:

1. Set resolution parameters: a number of intervals l and overlap percentage p, where $p \in (0, 100)$.
2. Compute the pairwise distance matrix $\mathbf{D} = [d(\mathbf{x}_i, \mathbf{x}_j)] \in \mathbb{R}^{N\times N}$ based on the distance metric chosen.

3. Select a function $f : \mathbb{X} \to \mathbb{R}^n$ to stratify the data.

The function f is often referred to as the filter (or lens) function. The most crucial step in the Mapper algorithm is the selection of the filter function to "guide" a clustering algorithm on the high-dimensional data. A few common filter functions include Gaussian kernel density estimator, eccentricity filter, principal metric SVD filter, and eigenvectors of graph Laplacians. Moreover, we can take the projection found by dimensionality reduction/manifold learning techniques that maps the high-dimensional data to a low-dimensional space as the filter function. In addition, it should be noted that when N is too large, numerical optimization techniques are used.

As an illustrative example, the implementation of 1-D Mapper algorithm (i.e., the target parameter space of the filter function is $\mathbb{R}$) is described below. The algorithm is also summarized as a flow chart in Figure 2.1. After setup, the first step described in lines 2-3 is to divide the filter range and cover it with overlapped intervals so that the clustering algorithm in the ensuing step focuses on the local information of the data that is likely to be ignored by the clustering over the entire data. The second step described by lines 4-9 is to cluster the data in the original high dimensional space, where C_r represents the minimum number of points in any level set (subset) for clustering. The Mapper algorithm is not tied to any particular clustering algorithm. However, it is always required to estimate certain parameters (thresholds) in order to determine the number of clusters in every level set. Lines 10-19

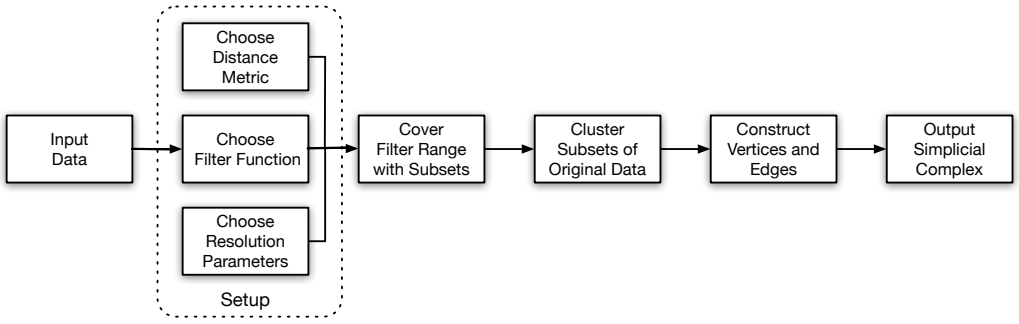


Figure 2.1: Framework of the Mapper algorithm for generating topological networks.

Algorithm 1 1-D Mapper algorithm

procedure MAPPER($\mathbf{D}$, f, $\mathbf{X}$, l, p)

Compute the filter value $f(\mathbf{x}_i)$, $i = 1, 2, \ldots, N$, and find the filter range $[f_{\min}, f_{\max}]$.

Cover the filter range by l intervals $\{U_1, U_2, \ldots, U_l\}$ of equal length L with $p\%$ overlap between successive intervals, where $U_i = [a_i, b_i]$, $i = 1, 2, \ldots, l$, and $a_1 = f_{\min}$, $b_l = f_{\max}$. Here, $L = (f_{\max} - f_{\min})/[(l-1)(1-p\%) + 1]$, $a_i = a_1 + (i-1)L(1-p\%)$, $b_i = a_i + L$.

Form the level sets $S_i = f^{-1}([a_i, b_i])$ for $i = 1, 2, \ldots, l$, i.e., for each point $\mathbf{x}_i \in S_i$, $a_i \leq f(\mathbf{x}_i) \leq b_i$. Let $|S_i|$ denote the number of points in S_i.

Let n be the total number of clusters. $n \leftarrow 0$.

for $i = 1, 2, \ldots, l$ **do**

  if $|S_i| \geq C_r$ **then**

    Apply a standard clustering algorithm to S_i. Let $S_{i,j}$ be the jth cluster in S_i and n_i be the number of clusters generated in S_i.

  $n \leftarrow n + n_i$.

Let A be the adjacency matrix. $A \leftarrow [0]_{n \times n}$.

for $i = 1, 2, \ldots, l-1$ **do**

  if $n_i > 0$ **and** $n_{i+1} > 0$ **then**

    for $j = 1, 2, \ldots, n_i$ **do**

      for $k = 1, 2, \ldots, n_{i+1}$ **do**

        if $S_{i,j} \cap S_{i+1,k} \neq \emptyset$ **then**

          $A(j,k) \leftarrow 1$, $A(k,j) \leftarrow 1$.

Construct a graph with one vertex $v_{i,j}$ for each cluster $S_{i,j}$. Let $V = \{v_{i,j}\}$.

Connect $v_{i,j}$ and $v_{i+1,k}$ with an edge for $i = 1, 2, \ldots, l-1$ when $A(j,k) = 1$. Let $E = \{\{v_{i,j}, v_{i+1,k}\} | A(j,k) = 1\}$.

Output the graph (i.e. 1-D simplicial complex) $G(V, E)$.

refer to the last step of the algorithm that any two clusters from neighboring level sets are

linked together if they have one or more common data points.

In the 1-D Mapper case, the output is a 1-D simplicial complex that comprises only vertices (0-simplex) and edges (1-simplex). More generally, if the target space is $\mathbb{R}^n$, higher simplices may appear in the output simplicial complex, such as triangular faces (2-simplex) whenever three clusters from neighboring level sets have nonempty intersections. Additionally, the resolution of the complex changes from coarse to fine as the number of intervals l increases. This change of resolution reflects the change in topology of the data set.

It is worth mentioning that the filter range is not necessarily covered by l overlapped intervals of equal length. In fact, the Mapper algorithm is highly parallelizable. To improve the efficiency of parallel computation, it is more convenient to decompose the filter range into l overlapped intervals wherein each interval contains the same number of points so that the running times would be similar for all the level sets.

2.1.2 Stability of Mapper Algorithm

The Mapper algorithm is highly sensitive to the choices of $\mathcal{U}$ and resolution parameters. Dey et al. briefly discussed the inherent instability of the Mapper algorithm and proposed a multiscale version of the algorithm that possesses the stability against perturbations of functions and a hierarchy of covers [44]. Carrière and Oudot further quantified the degree of the (in)stability of the 1-D Mapper algorithm to input fluctuations by constructing its multinerve variant as an auxiliary tool [25]. Based on this framework, Carrière et al. provided rules of thumb for automatically tuning the resolution parameters in the 1-D Mapper case [24]. Due to the limited usage of this result, it is still a common practice to test a wide range of parameters and select the most stable outputs whose shapes persist over a large-scale change.

2.1.3 Application of Mapper Algorithm to Feature Selection

The output graph of the Mapper algorithm contains the information of clusters at the local level and the interaction between these local clusters. Therefore, the principle of applying

the Mapper algorithm to feature selection is to recognize shapes in the resulting graph that encode the essential structural information of the data. Typical shapes of interest found in a graph are subgroups of clusters that display distinct patterns such as "loops" (continuous circular segments) and "flares" (or "tendrils"), as opposed to portions of the graph within which the local environment of each cluster is roughly identical. After a stable output is selected, standard statistical tests are performed among the subgroups of interest to identify the features that best differentiate them.

The Mapper algorithm has enjoyed immense popularity in fields such as bioinformatics and machine vision. For example, it has been used to reveal unique and subtle aspects of the folding patterns of RNA [131] and to unlock previously unidentified relationships in immune cell reactivity between patients with type-1 and type-2 diabetes [114]. Another influential example occurs in personalized breast cancer diagnosis, in which a novel subgroup of tumors with a unique mutational profile and 100% survival rate has been discovered [99]. Additionally, its deformation invariant property has been used to detect 3D objects from point cloud data with intrinsically different shapes [116]. For more details about the Mapper algorithm and concrete examples of real applications, we refer the reader to [89, 116].

2.1.4 Meshes and Persistent Homology

Unlike the Mapper algorithm, one appealing attribute of persistent homology is its ability to numerically characterize the shape of data. Not only does it count the existing topological features, but also provides their metric measurements. Persistent homology requires an efficiently computable filtration defined as a nested sequence of simplicial complexes. Although it emerges in an effort to reconstruct underlying spaces from point cloud data, the high computational cost from the standard algorithm [48, 136] for triangulation during the filtration prohibits its usage for this unorganized data type. Even though more fast algorithms have since been introduced [8, 9, 31, 42] and new combinatorial representations, such as witness complex and its "lazy" version [41], were proposed to reduce time complexity and memory consumption, it is hardly practical to compute persistent homology from massive

point clouds in most cases, especially towards real-time applications.

In fact, persistent homology is widely used to handle triangle meshes of geometric objects obtained from 3D scanning and quadrilateral or hexahedral meshes of image data. These meshes are simply examples of simplicial or cubical complexes. Persistent homology is then computed through sublevel set filtration of a descriptor function defined on the vertices of the mesh object.

2.1.5 Stability of Persistent Homology

A critical property of PDs is their stability with respect to input noise. A general metric associated with PDs is the *p-Wasserstein distance*, $1 \leq p \leq \infty$ [35]. The p-Wasserstein distance between the PDs of f and g is defined by

$$d_{W,p}(D(f), D(g)) = \left(\inf_{\gamma} \sum_{x} ||x - \gamma(x)||_{\infty}^{p} \right)^{\frac{1}{p}},$$

where the infimum is over all bijections $\gamma : D(f) \to D(g)$ and the sum is over all points $x \in D(f)$.

Let M be a compact triangulable metric space and $f, g : M \to \mathbb{R}$ be two tame Lipschitz functions with the corresponding PDs $D(f)$ and $D(g)$. It has been proven that assuming M satisfies a weak condition (see details in [35]), there exist constants $q \geq 1$ and C_L, which depend on M and the Lipschitz constants of f and g, such that

$$d_{W,p}(D(f), D(g)) \leq C_L ||f - g||_{\infty}^{1-\frac{q}{p}}, \ p \geq q.$$

This upper bound on $d_{W,p}(D(f), D(g))$ implies that a PD $D(f)$ is stable with respect to the p-Wasserstein distance under small perturbations of f.

Note that taking $p \to \infty$ yields another commonly-used metric, the *bottleneck distance* [34], i.e.,

$$d_B(D(f), D(g)) = \inf_{\gamma} \sup_{x} ||x - \gamma(x)||_{\infty}.$$

Correspondingly, the PDs satisfy

$$d_B(D(f), D(g)) \leq ||f - g||_{\infty}.$$

2.1.6 Vectorization and Kernel Methods for PDs

Previous studies on linking persistent homology to machine learning can be divided into two categories based on whether the persistence information of the topological features is used directly or indirectly. A representative application in the first category is 3D shape segmentation, where the persistence information is directly incorporated to determine the number of meaningful segments [29, 117]. Another direct use is presented in [57] for segmenting high resolution CT images through the restoration of topological handles (i.e., topological cycles).

In the second category, the persistence information is first transformed into another representation and then fed to machine learning algorithms. One seminal work is persistence landscape introduce by Bunenik [18], which converts a PD to a collection of continuous, piecewise linear functions as elements of a Hilbert space so that algorithms that require a Hilbert space structure can be directly applied for. Besides, a growth area of interest lies in establishing kernels for PDs in image/shape recognition and classification tasks, where the points in the PDs are transferred into a Hilber space through a feature map. Pachauri et al. [101] first computed a Gaussian kernel to estimate the density of points on a regular grid for each rasterized PD, and fed the discrete density estimation as a vector into an SVM classifier without any feature selection. However, their method did not establish the stability of the kernel-induced vector representation. Reininghaus et al. [109] then designed a stable multi-scale kernel for PDs motivated by scale-space theory. Similar to this work, Kusano et al. [82] proposed a stable persistence weighted Gaussian kernel, allowing one to control the effect of persistence. However, the computational complexity of both the kernel-based methods for calculating the Gram matrix is $O(m^2n^2)$ if there are n PDs for training and the PDs contain at most m points, which can be quite expensive for many practical applications.

To enable large-scale computations with PDs, recent methods have mapped each PD to a stable vector to allow direct use of vector-based learning methods. For example, Adams et al. [1] constructed vectors by discretizing the weighted sum of probability distributions centered at each point in transformed PDs. Carrière et al. [26] rearranged the entries of the

distance matrix between points in a PD and Bonis et al. [14] adopted a pooling scheme to construct the vectors.

2.2 TDA in Network Science

A great deal of work under this topic is devoted to the neuroscience studies with simplicial complexes being used to model brain functional networks. One direct benefit of using this representation is that it naturally incorporates polyadic interactions among the network vertices (neurons or brain regions) that regular network models, limited to the dyadic setting, fail to accurately reveal. This is particularly helpful for characterizing the coactivity patterns in neural data, e.g., simultaneously active traces of n neural units can now be encoded by a $(n-1)$-simplex rather than $n(n-1)/2$ edges to differentiate from the case where neural units are coactive in all pairwise combinations but never as a whole.

Different types of simplicial complexes have been constructed to detect meaningful structure from the activity and connectivity of neural data. The most common type is the clique complex [74], which is defined simply by substituting every clique of size n in a graph with a $(n-1)$-simplex. Giusti et al. [63] utilize this simplicial structure to measure the neural activity in rat hippocampal pyramidal cells during both spatial and non-spatial behavior. Other types of simplicial complexes, including independence complex [80], concurrence complex and its dual [38, 45], are often used when information cannot be directly encoded in a graph. These types of simplicial complexes often consider the relationships between two variables of interest, e.g., neural units and time, or coactivity in two separate regions [62]. In each case, the patterns of relationships between the two variables are recorded in a binary matrix, where the row and column each represents one variable. The simplicial complex is then formed by taking each row (or column) as a vertex and each column (or row) that corresponds to the coactivity patterns, as a simplex. Such simplicial complexes can be used to decipher high-order dependence between neuron units and recognize significant modular structure in dynamic human brain networks [7, 49].

Two typical needs in the neuroscience field is to evaluate the hierarchical structure in

weighted networks and temporal dynamics of neural processes. With constructed simplicial complexes, Giusti et al. [62] reviews on using persistent homology to build weighted simplex filtration [60] or temporal filtration [123] to meet these needs. Beyond the neuroscience community, the clique complex is also broadly utilized to construct other filtrations over a graph in a more general background, such as clique complex filtration [72], k-clique filtration [110], Vietoris-Rips filtration [77] and power filtration [11]. The generated PDs are often used to analyze the structural features of a wide variety of networks ranging from biological areas to infrastructural and social domains. We recommend [2] for a more comprehensive survey of key advances using TDA on complex networks.

Chapter 3

FEATURE SELECTION USING TDA FOR ACCURATE PREDICTION OF MANUFACTURING SYSTEM OUTPUTS

The work reported here has appeared in the special issue on "High Performance Computing and Data Analytics for Cyber Manufacturing" of the Journal of Manufacturing Systems, vol. 43, pp. 225-234, 2017. A preliminary version of the work also appeared in the Proceedings of IEEE International Symposium on Assembly and Manufacturing, pp. 31-36, 2016.

3.1 Background and Motivation

Sensors play an essential role in carrying out product feasibility assessment, yield enhancement, and quality control in modern manufacturing systems such as vehicle assembly, microprocessor fabrication, and pharmaceuticals development [124]. A large number of sensors of many different types are typically employed in such systems to measure a variety of process variables ranging from operating conditions and equipment states to material compositions and processing defects over extended time periods. Thus, the volume of acquired data is so vast and heterogeneous that the contribution of individual sensor measurements in predicting the overall system outputs gets obscured. This prediction is made more challenging by the fact that the measurements are often noisy and replete with missing or outlier values. Furthermore, there is significant redundancy among the sensor measurements, leading to the presence of numerous false correlations in the recorded data. It is, therefore, necessary to perform an analysis using statistical methods that are specifically suited to identifying and filtering out existing correlations in erroneous, heterogeneous, and high-dimensional data sets.

Historically, multivariate statistical process control (MSPC) methods, such as principal component analysis (PCA) and partial least-squares (PLS), have served as the dominant

mode of addressing this problem [91]. The common idea behind these methods is to define a new set of variables (known as *latent variables*) through linear combinations of the original variables that describe the sensor measurements. The set of latent variables may be reduced in some cases by performing subsequent dimensionality reduction techniques. However, these methods do not work particularly well when there are a large number of input process variables, and they share highly non-linear relationships with the system outputs that cannot be modeled using Gaussian distributions. The methods also encounter difficulties in removing the false correlations among the measurements particularly when they are erroneous. More recently, several non-linear prediction methods have been developed based on response surface fitting as well as kernelized and robust variants of the MSPC techniques [115, 111]. While these methods may achieve high prediction accuracy, they do not provide any direct way of quantifying the contribution or impact of the individual process variables.

Here, we present an alternative method to select the important variables that are subsequently used in both linear and non-linear prediction models. More specifically, we employ the Mapper algorithm to address the extraction of key process variables from a topological perspective [116]. As a powerful TDA tool, the Mapper algorithm enhances existing clustering techniques to discover hidden patterns. In particular, it clusters all the level sets of the data to generate a topological network that represents the inherent clusters and connections among the clusters in the actual data.

We first apply the Mapper algorithm on a benchmark chemical processing data set to predict product yield [64]. Specifically, the shape of the generated topological network is used to select key features that explain the observed differences in the process measurements in a statistically significant manner. Second, we investigate the role of individual process variables in causing wafer failures in another publicly available semiconductor manufacturing data set. Although it has been recognized that k-nearest neighbor methods can identify faulty wafers effectively [67, 68, 88, 134], the actual process variables that result in the wafer anomalies have never been identified. To this end, we demonstrate how the Mapper algorithm rapidly traces the causality hidden in this high-dimensional data set.

The rest of the chapter is organized as follows. Section 3.2 gives an overview of the general characteristics of manufacturing data and the types of predictor (feature) and response variables that are of interest to us. In Section 3.3, we review the application of Mapper algorithm to feature selection. We demonstrate the applicability of the Mapper algorithm for feature selection on two benchmark manufacturing data sets in Section 3.4. The effectiveness of the selected features is further assessed through predictive models. We conclude this chapter with remarks of this method in the manufacturing field.

3.2 Problem Formulation

In real-world manufacturing systems, data is collected using a large number of sensors that are affixed to or embedded within different machines and equipment, resulting in a high-dimensional body of heterogeneous data. The data is usually in the form of *time series measurements* of different process variables such as temperature, pressure, density, humidity, voltage, chemical or material composition including the relative proportions of various constituents of alloys or mixtures, material removal or deposition rate, number and severity of processed part flaws and defects, and so on. The sensors, thus, come in myriad forms ranging from thermocouples, pressure gauges, hydrometers, hygrometers, and voltmeters to optical cameras, spectrometers, laser scanners, and ultrasonic transducers.

Consequently, manufacturing sensor data is prone to noise terms, missing values, and outliers. These measurement errors depend on the sensitivity of the sensors to the operating conditions based on their underlying physical principles of actions. For example, it is not at all uncommon for temporary sensor hardware malfunction to result in missing values. A further problem is that of co-linearity, which is usually caused by partial redundancy in the sensor arrangement such as the placement of multiple sensors in close proximity to one another. The net result of these complications is that manufacturing systems are often "data-rich but information-poor".

Consequently, there is a strong need to effectively select a minimal number of process variables that primarily affect the output variables of interest such as product quality and

yield of a manufacturing system comprising several processes of varying types. As discussed earlier in Section 3.1, this form of selection facilitates process monitoring and diagnostics through targeted sensor data acquisition, storage, and processing. Even if it is cheap or convenient to manage data from all the sensors, knowing which measurements of what variables matter the most makes it feasible to rapidly regulate out-of-control processes or adapt them to manufacture high quality products at desired rates.

To formulate the problem mathematically, we suppose there are m process variables (features) and N sensor measurements recorded at different time instants. Each measurement is, thus, represented by an m-dimensional vector $\mathbf{x}_i \in \mathbb{R}^m$, $i = 1, 2, \ldots, N$. The data is then assembled into a matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]^T \in \mathbb{R}^{N \times m}$. Each column denotes a process variable, which is measured by one sensor operating alone or by the concurrent operation of several sensors that function in unison. The latter case is known as data fusion [52], which provides a wide range of sensed parameters, and is, hence, more reliable for data analysis.

In a batch process with batch length L, a 3-D data array $\bar{\mathbf{X}} \in \mathbb{R}^{N \times m \times L}$ is often unfolded batch-wise into a 2-D matrix $\mathbf{X} \in \mathbb{R}^{N \times mL}$. In this case, each measurement $\mathbf{x}_i \in \mathbb{R}^{mL}$ is a batch and each process variable is measured L times throughout the batch, hence, corresponding to L columns. For each row, the measurement is either spatially-sampled or temporally-sampled. For instance, in the semiconductor manufacturing environment, electronic wafer map data collected from in-line measurements are sampled spatially across the surface of the wafer for defect inspection [120]. Usually, there will also be one or more response variables to reflect the output quality or quantity. We write the output with r response variables into a matrix $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N]^T \in \mathbb{R}^{N \times r}$, where each response variable is represented by one column. Response variables are commonly seen as continuous variables denoting production yields or binary variables indicating pass or fail outcomes.

3.3 Technical Approach

In the chemical manufacturing process study, our choice of the filter function is the 2-D projection found by the multidimensional scaling (MDS) method. MDS in this case attempts

to embed the data such that the pairwise distances in the high-dimensional space are preserved in the 2-D Euclidean space. Accordingly, the 2-D embedding coordinates denoted by $\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_N$, are the minimizers of a loss function, σ, defined as

$$\sigma(\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_N) = \sum_{j=2}^{N} \sum_{i=1}^{j-1} (||\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j||_2 - d(\mathbf{x}_i, \mathbf{x}_j))^2. \tag{3.1}$$

Therefore, given a topological space $\mathbb{X}$, the filter function is specified as

$$f : \mathbb{X} \to f_1 \times f_2, \tag{3.2}$$

where f_1 and f_2 are coordinates of $\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_N$ along the 1^{st} and 2^{nd} dimension, respectively.

For the study of fault detection in the semiconductor manufacturing processes, we employ the 2-D projection found by the t-distributed stochastic neighboring (t-SNE) algorithm as the filter function [90]. t-SNE aims to preserve the joint probabilities p_{ij} that measure similarities between $\mathbf{x}_i$ and $\mathbf{x}_j$, $i, j = 1, 2, \ldots, N$, as much as possible in the 2-D space. Specifically, p_{ij} is defined as

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}, \tag{3.3}$$

where the conditional probability $p_{j|i}$ that represents the similarity of $\mathbf{x}_j$ to $\mathbf{x}_i$ is given by

$$p_{j|i} = \frac{\exp(-||\mathbf{x}_i - \mathbf{x}_j||^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-||\mathbf{x}_i - \mathbf{x}_k||^2 / 2\sigma_i^2)}. \tag{3.4}$$

Herein the variance of the Gaussian σ_i centered at $\mathbf{x_i}$ is determined by a predefined perplexity. On the other hand, the joint probability q_{ij} that reflects the similarity between 2-D embedding coordinates $\tilde{\mathbf{x}}_\mathbf{i}$ and $\tilde{\mathbf{x}}_\mathbf{j}$ is defined based on a heavy-tailed Student's t-distribution with one degree of freedom:

$$q_{ij} = \frac{(1 + ||\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j||^2)^{-1}}{\sum_{k \neq l} (1 + ||\tilde{\mathbf{x}}_k - \tilde{\mathbf{x}}_l||^2)^{-1}}, \tag{3.5}$$

such that dissimilar measurements in the m-D space are mapped far apart in the 2-D space. $\tilde{\mathbf{x}}_\mathbf{i}$, $i = 1, 2, \ldots, N$ are then determined by minimizing the Kullback-Leibler divergence be-

tween the joint probability distribution P in the m-D space and the joint probability distribution Q in the 2-D space,

$$D_{\mathrm{KL}}(P||Q) = \sum_{j=2}^{N} \sum_{i=1}^{j-1} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{3.6}$$

Likewise, the filter function in this case is given by

$$f : \mathbb{X} \to g_1 \times g_2, \tag{3.7}$$

where g_1 and g_2 are coordinates of $\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_N$ along the 1^{st} and 2^{nd} dimension, respectively.

The Mapper algorithm is largely concerned with the geometric features of the data, such as "loops" and "flares". Aside from these intriguing shapes, we also discern the trends in terms of the *output values* associated with each cluster in the graph rendered by the Mapper algorithm, such as which clusters contain several measurements from faulty samples in the case of anomaly detection. Furthermore, we distinguish the fundamental subgroups from artifacts by observing whether the shapes of the given subgroups remain consistent when the resolution parameters are varied over a wide range of values. After the fundamental subgroups of interest are detected, standard statistical tests, such as the Kolmogorov-Smirnov test and Student's t-test, are performed to identify the features that best distinguish the subgroups from one another. The final set of features thus selected are then fed into classification or regression models to perform a desired prediction task.

Thus, we end up addressing two main challenges in applying the Mapper algorithm to identify key features from manufacturing data. The first one pertains to a suitable selection of the filter function so as to map the high-dimensional data to a low-dimensional space where the data can be conveniently stratified. Unlike in the case of meshes or images, there is no well-established function, and we select the MDS projection method based on final output prediction quality. The second challenge is on varying the resolution parameters appropriately so that the fundamental subgroups are correctly distinguished from artifacts in the generated topological networks. Choice of a coarse granularity of variation leads to the appearance and disappearance of subgroups, whereas the use of very fine granularity

makes the process time-consuming. We vary the parameters in a simple way such that a majority of the subgroups, which are identified at a particular resolution, remain intact as the parameters change.

3.4 Results

In this section, we conduct two studies to show how to achieve feature selection using the Mapper algorithm. With selected features, the first study obtains accurate predictions of productivity for a chemical processing benchmark, and the second study reaches a high accuracy in fault classification for a semiconductor etch process.

3.4.1 Prediction of manufacturing productivity

The data is for a chemical process plant that is described in [81] and can be obtained from the R package "AppliedPredictiveModeling". The data set contains 176 measurements of biological materials for which 57 variables are measured, where there are 12 biological starting materials and 45 manufacturing process parameters (predictors). The starting material is generated from a biological unit and has a wide range of quality and characteristics. The manufacturing process parameters include temperature, drying time, washing time, and concentrations of by-products at various steps. The biological variables are used to gauge the quality of the raw material before processing but cannot be changed, whereas the manufacturing process parameters can be changed during operations. The measurements are not independent since partial measurements are produced from the same batch of biological starting materials. We aim to investigate the relationships between the predictors and the final pharmaceutical product yield, and develop a model to estimate the percentage yield of the manufacturing process.

Data preprocessing

As we want to maximize the level of automation in predicting manufacturing productivity for industrial applications, the data is preprocessed with a minimum amount of work. First, the

outliers in the data set are marked as missing values and the features with near-zero variances are discarded. During this step, BiologicalMaterial07 is removed. Second, we apply Box-Cox transformation to the data to eliminate distributional skewness, and scale each column of the data to zero mean and unit variance. The last step is to impute the missing values by the k-NN method with $k = 5$. Note that all of these steps can be handled automatically in the production environment.

Feature selection

To begin with, we choose Euclidean distance as the metric to represent the similarity between the measurements. In this work, much effort is spent on the suitable selection of the filter function due to the complex underlying structure of the data. Some commonly considered filter functions include the eccentricity function, linear and nonlinear projections such as PCA and Isomap. Regarding the quantity of interest and the purpose of the filter function, we use the response variable to "supervise" the stratification of the data. The output of the MDS method that reduces the data set to 2 dimensions is shown to provide the smoothest variations of the response values over the embedding coordinates, and is eventually chosen as the filter function.

In the next stage, each dimension is covered by 14 intervals of equal length with 80% overlap between any two successive intervals, leading to the filter range being divided into 196 level sets in all. Density-based spatial clustering of applications with noise (DBSCAN) method is subsequently employed for clustering in each level set, where the number of clusters is determined by the minimum number of measurements in a cluster and the maximum distance between two measurements in the same cluster [50].

As a result, we implement the steps above in Python[1] and obtain a topological network in the form of a simplicial complex as shown in Figure 3.1. Each cluster is represented by a node, and the node size is proportional to the number of measurements in the cluster based on

[1]Code adapted from https://github.com/MLWave/kepler-mapper

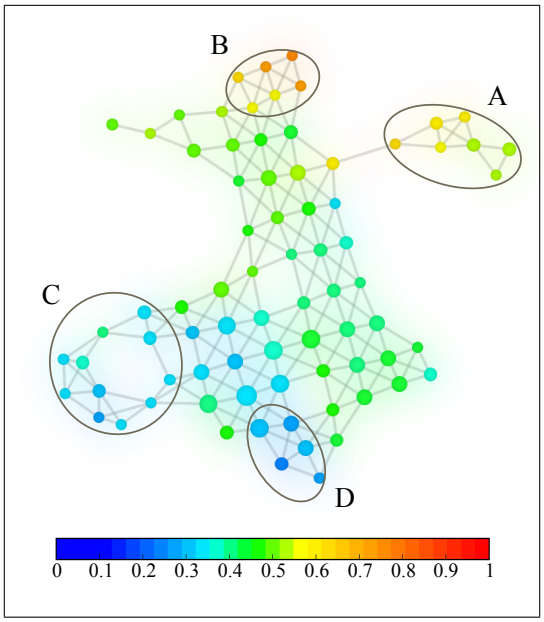


Figure 3.1: Topological network derived from the chemical processing data at a specified resolution. Each node is colored based on the average normalized yield value for the measurements in the node, where the normalized yield varies between 0 and 1. High and low yield subgroups are isolated from the rest of the network, where A and C are extracted as outer flares and B and D are extracted from the periphery of the network as suggested in [89].

a logarithmic scale. An edge is generated between any two nodes from neighboring level sets that have at least one measurement in common. We normalize the value of the product yield within the range 0-1, and color each node based on the average normalized yield value for the measurements in the node. As is seen in Figure 3.1, the shape of the data is captured by the generated topological network after iterating through multiple times at various resolution scales. The resolution is set at a large number of intervals and a high overlap percentage. A large number of intervals helps to uncover subtle aspects of the shape of the data rather than a blob, and a high overlap percentage seeks to have all nodes connected as a single network if possible. Thus, we are able to find out the subgroups of interest and acquire an overall structural information of how the data is distributed within the network. In this problem,

we are interested in the difference in patterns between the measurements with high and low yields. Notice that the high yields are separated into two subgroups, and the low yields are also bifurcated into two subgroups with different patterns. Therefore, two subgroups of measurements with high yield and two subgroups of measurements with low yield are extracted from the data as encircled in Figure 3.1.

Two-sample Kolmogorov-Smirnov (KS) test, which is sensitive to the difference in both location and shape of the empirical cumulative distributions of two groups, is then performed between subgroups A and C, A and D, B and C, and B and D over all the columns in the data matrix to identify the features that best discriminate between them. We record the largest KS-score and the associated p-value as well as the adjusted p-value among the four tests for each feature. The results are presented in Table 3.1 and further visualized in Figure 3.2. The p-values are adjusted using the well-established Benjamini-Hochberg (B-H) procedure [10, 132] that is commonly used to reduce the false discovery rate (FDR) when multiple features or variables are evaluated for statistical significance. The B-H adjustment provides greater flexibility at the expense of somewhat higher FDR as compared to the traditional Bonferroni correction method. This adjustment is, thus, better suited for our purpose as we want to identify *all* the process variables that may have an impact on the manufacturing system outputs. The most salient features are selected based on high KS-scores (> 0.9) and low p-values (< 0.05), where 11 of them are the measurements of manufacturing processes that can be controlled. Thus, the product yield should be improved by altering these steps in the process to have higher or lower values. We also note that the selection of the most salient features are not affected by the B-H procedure in this case.

Figure 3.3 examines the effects of the features on the product yield and probes the relationships between them. We color the same network nodes based on normalized feature values. The color of each node encodes the normalized feature value averaged across all the measurements in the node, with blue denoting a low value and red indicating a large value. We see that significant differences between the subgroups exist both for BiologicalMaterial06 and ManufacturingProcess13, both of which are selected in Table 3.1. Contrary to Fig-

Table 3.1: KS test to identify features that best differentiate between the subgroups.

Feature	KS-score	p-value	Adj. p-value	Feature	KS-score	p-value	Adj. p-value
B01	0.882	5.53e-7	2.21e-6	M18	0.882	1.93e-7	7.20e-7
B02	1	7.57e-8	1.06e-6	**M19**	1	1.95e-9	2.18e-8
B03	1	7.57e-8	1.06e-6	M20	0.778	1.12e-4	3.49e-4
B04	0.917	1.16e-6	9.28e-6	M21	0.598	0.002	0.004
B05	0.739	2.36e-5	6.01e-5	M22	0.203	0.821	0.901
B06	1	7.57e-8	1.06e-6	M23	0.369	0.142	0.204
B08	1	7.55e-9	8.46e-8	M24	0.539	0.007	0.012
B09	0.417	0.054	0.082	M25	0.787	5.22e-6	1.39e-5
B10	0.728	3.32e-5	8.09e-5	**M26**	0.941	2.08e-8	1.17e-7
B11	0.886	4.95e-7	2.22e-6	M27	0.717	4.64e-5	1.04e-4
B12	0.952	1.34e-8	9.39e-8	**M28**	1	1.95e-9	2.18e-8
M01	0.533	0.008	0.013	**M29**	1	1.95e-9	2.18e-8
M02	1	7.55e-9	8.46e-8	M30	0.768	2.15e-5	6.35e-5
M03	0.650	0.001	1.23e-3	**M31**	0.944	6.14e-8	3.82e-7
M04	1	1.88e-7	1.75e-6	**M32**	0.941	2.08e-8	1.17e-7
M05	0.647	5.95e-4	1.28e-3	M33	0.894	1.28e-7	5.50e-7
M06	0.722	4.32e-4	1.15e-3	M34	0.238	0.718	0.855
M07	0.261	0.521	0.635	M35	0.501	0.011	0.017
M08	0.314	0.259	0.345	M36	0.787	5.22e-6	1.39e-5
M09	0.944	1.08e-6	6.05e-6	M37	0.317	0.284	0.379
M10	0.833	2.64e-5	1.06e-4	M38	0.381	0.167	0.253
M11	0.886	4.95e-7	2.22e-6	M39	0.294	0.371	0.472
M12	0.667	0.001	0.003	M40	0.278	0.560	0.713
M13	1	1.88e-7	1.75e-6	M41	0.262	0.601	0.783
M14	0.692	9.71e-5	2.01e-4	M42	0.488	0.034	0.064
M15	0.905	8.40e-8	4.28e-7	M43	0.846	7.12e-7	2.21e-6
M16	0.690	0.001	0.002	M44	0.291	0.342	0.426
M17	0.833	2.64e-5	1.06e-4	M45	0.222	0.819	0.936

[a]B: BiologicalMaterial; M: ManufacturingProcess

[b]Key features characterized with high KS-score (> 0.9) and low adjusted p-value (< 0.05) are written in bold.

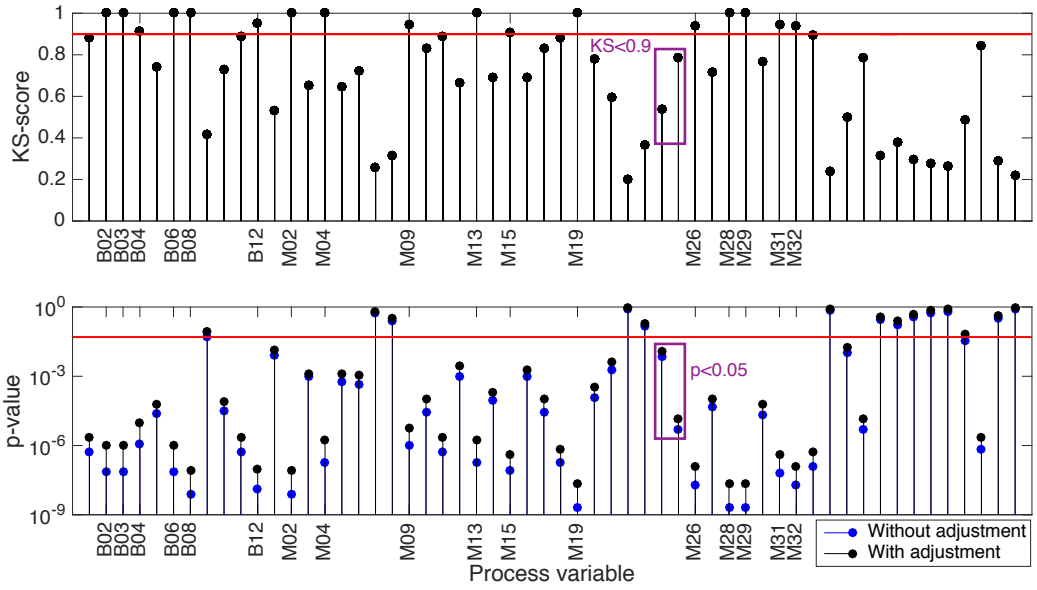
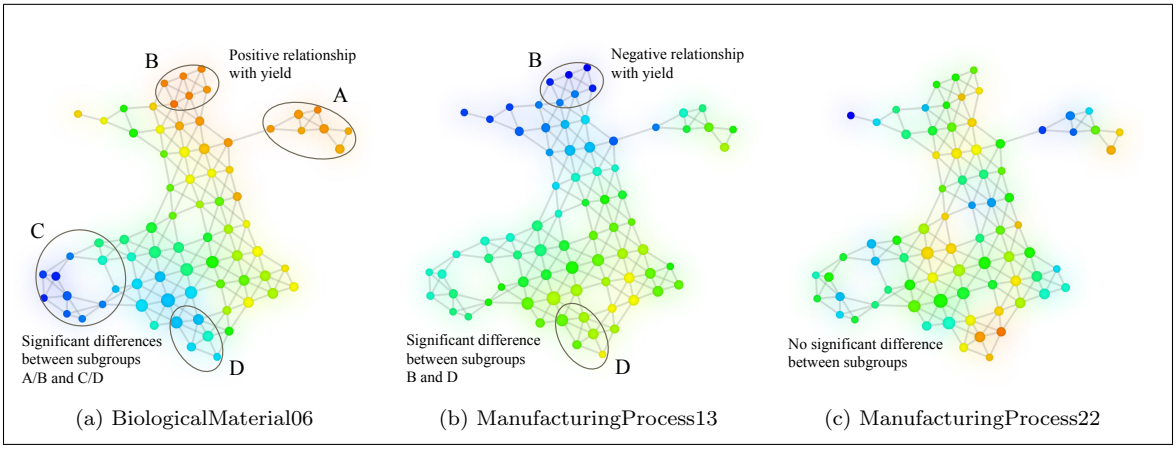


Figure 3.2: Key features (marked by x-axis tick labels) that best differentiate between the subgroups are identified by Kolmogorov-Smirnov tests as those which yield a high KS-score (> 0.9) and a low corresponding adjusted p-value (< 0.05).

ure 3.3(a)(b), an unselected feature ManufacturingProcess22 shows no significant difference between any of the subgroups in Figure 3.3(c). Meanwhile, on comparing with Figure 3.1, BiologicalMaterial06 shows a positive relationship with the yield, whereas ManufacturingProcess13 displays a negative relationship with the yield.

Predictive modeling

Four regression models, PLS, random forest (RF), cubist and Gaussian process with a Gaussian kernel (kGP), are chosen to predict the yield of the chemical manufacturing process. These models represent a linear model, a tree-based model, a rule-based model and a kernelized technique, respectively. We randomly split the entire data set into a training set and a testing set in 7:3 ratio. Parameters in each trained model are tuned to be optimal using 25 iterations of 10-fold cross-validated search over the parameter set. The trained models are



(a) BiologicalMaterial06 (b) ManufacturingProcess13 (c) ManufacturingProcess22

Figure 3.3: Topological networks colored based on different selection of features at the same resolution as in Figure 3.1. For every network, each node is colored based on the average normalized feature value of all the measurements included in the node, where the normalized feature value varies between 0 and 1.

then adopted to predict the percentage yield for the testing set.

Table 3.2 compares the prediction results and computation times between all the features and just the selected features for the models based on 30 runs. The prediction accuracy is evaluated by the root mean squared error (RMSE) and computation times are measured on a laptop with an Intel Core i5 (1.7 GHz) CPU and 4 GB RAM. We find that the models with selected features achieve comparable performance as the models with all the features. Especially, in the case of the PLS, RF and kGP models, selected features outperform all the features in terms of both training and testing errors, which highlights the efficacy of the selected features based on the Mapper algorithm. Meanwhile, the training times are reduced by about 30%∼60% for the RF and Cubist models using the selected features.

Table 3.3 compares the top features identified by different methods. Since there is almost no dominant feature due to the complexity of the data, the features identified by each method vary from each other. For the Mapper algorithm, the feature that overlap with the features identified by at least one of the other methods are highlighted. In fact, even though the

Table 3.2: Estimation errors and computation times for different models with all features and selected features

	Method	Errors (RMSE)		Computation Times (s)	
		Training	Testing	Training	Testing
All features	PLS	1.20±0.05	1.29±0.10	1.33±0.30	0.005±0.001
	RF	1.13±0.06	1.15±0.15	130±2.40	0.006±0.001
	Cubist	1.00±0.07	1.15±0.13	58.5±4.11	0.025±0.004
	kGP	1.21±0.04	1.25±0.11	8.14±0.43	0.002±9.4e-4
Selected features	PLS	**1.13±0.05**	**1.25±0.09**	**1.02±0.16**	**0.002±3.8e-4**
	RF	**1.11±0.06**	**1.13±0.15**	**91.2±2.53**	**0.005±8.9e-4**
	Cubist	1.05±0.10	1.20±0.08	**24.7±1.54**	**0.008±0.002**
	kGP	**1.19±0.05**	**1.22±0.11**	**6.24±0.33**	**0.001±6.3e-4**

other four methods have the ability to detect significant features, it is difficult for them to interpret how the yield is affected by these features. In contrast, the Mapper algorithm is well capable of unraveling the relationships between the features and product yield through easy and rapid visualization as shown in Figure 3.3.

3.4.2 Fault detection of semiconductor manufacturing process

In this study, the data set[2] is collected from an Al stack etch process performed on a commercial-scale Lam 9600 plasma etch tool at Texas Instrument Inc. [128]. The data consists of 108 normal wafers and 21 faulty wafers from three separate experiments (denoted as experiment numbers 29, 31, and 33) with 19 process variables for monitoring. Since two of the process variables, RF reflected power and TCP reflected power, remain almost zero during the entire batch, only 17 process variables are used for fault detection and diagnosis, as tabulated in Table 3.4. Moreover, one normal wafer and one faulty wafer are removed

[2]Available at http://software.eigenvector.com/Data/Etch/index.html.

Table 3.3: Top 17 important features identified by different methods

PLS	RF	Cubist	kGP	TDA
M32	M32	M32	M32	B02
M36	B06	M17	B06	**B03**
M17	M17	M31	M13	**B06**
M13	M31	B06	M17	**B08**
M09	B03	M13	M36	**M02**
M33	M13	M04	B03	**M04**
M06	M01	M21	M31	**M13**
B06	B08	B03	M33	M19
M12	B11	M09	M09	**M28**
B03	M39	M01	B04	**M29**
B04	B04	M20	M06	B12
B08	M20	M39	M29	**M09**
B01	B09	B04	M04	**M31**
B11	M06	M33	B11	M26
M31	M18	M02	M02	**M32**
M04	M11	M05	B01	**B04**
M28	M33	B10	M27	M15

[a]B02, B12, M30, M40 are excluded from the PLS, RF, Cubist or kGP model since these features are removed before models being trained due to their high correlation with other features.

[b]The important features given by PLS, RF, Cubist, kGP and the TDA method are ranked based on the weighted sums of the absolute regression coefficients, average impurity reduction, usage in the rule conditions, and KS-score in Table 3.1, respectively. Features with the same KS-score are ordered by their feature names. For the kGP method, a LOESS smoother is fitted to assess the relationship between each feature and the outcome. The importance of the features is ranked by their R^2 statistics.

[c]The ranking of feature importance varies somewhat with the training samples and the results in Table 3.3 are reported based on a certain choice of the samples.

from the data set due to a large amount of missing values. Finally, because the experiments were run several weeks apart from one another, the process shift and drift lead to different means and covariance structures in the data gathered in each of the three experiments.

Table 3.4: Process variables for semiconductor wafer fault detection

No.	Variables	No.	Variables
1	BCl_3 flow	10	RF power
2	Cl_2 flow	11	RF impedance
3	RF bottom power	12	TCP tuner
4	Endpoint A detector	13	TCP phase error
5	Helium chuck pressure	14	TCP impedance
6	Pressure	15	TCP top power
7	RF tuner	16	TCP load
8	RF load	17	Vat valve
9	RF phase error		

The faulty wafers were intentionally induced through the modification of several of the process variables: TCP power, RF power, pressure, BCl_3 or Cl_2 flow rate, and Helium chuck pressure. To simulate an actual sensor failure, readings from the corresponding sensor were intentionally adjusted using a bias term so that its mean value was equal to the original baseline value of the relevant process variable. For example, if the TCP power was modified from its normal baseline value of 350W to a value of 400W, the values of TCP power in the data set would be reset to a mean of 350W by adding a constant bias of –50W. Table 3.5 lists the induced faults associated with each faulty wafer in the three experiments. In general, the modification of any one of the process variables may generally be expected to result in changes to the remainder of them because of correlations which may exist between the process variables. In this work, our goal is to determine the process variables which are most affected by the induced faults and use this information to construct a classification model

Table 3.5: Induced faults and experiments associated with each faulty wafer

No.	Exp.	Fault names	No.	Exp.	Fault names
1	29	TCP power +50[a]	11	31	Cl_2 flow +5
2	29	RF power -12	12	31	BCl_3 flow -5
3	29	RF power +10	13	31	Pressure +2
4	29	Pressure +3	14	31	TCP power -20
5	29	TCP power +10	15	33	TCP power -15
6	29	BCl_3 flow +5	16	33	Cl_2 flow -10
7	29	Pressure -2	17	33	RF power -12
8	29	Cl_2 flow -5	18	33	BCl_3 flow +10
9	29	Helium chuck pressure	19	33	Pressure +1
10	31	TCP power +30	20	33	TCP power +20

[a]The addition term in each fault name represents an offset of the process variable from its normal baseline value during the batch. For example, "TCP power +50" means that the induced fault is an increase of 50 units in the TCP power.

for fault detection.

Data prepossessing

We follow a similar data prepossessing step as the aforementioned study. First, we remove the first five records to eliminate effects which due to initial fluctuations. To accommodate shorter batches, we retain 85 records in each batch to ensure that each batch record is of equal length. Next, the 3-D data array is unfolded batch-wise to a 2-D matrix, resulting in a total of 1445 features, i.e. each feature is considered to be a pairwise combination of a process variable and a batch record. Finally, each column of the 2-D matrix is scaled to zero mean and unit variance.

Feature selection

The etching process reflected in our data set is a typical nonlinear, multimodal process. For this reason, the filter function used to identify a 2-D embedding of the data is taken corresponding to that of t-SNE, a nonlinear dimensionality reduction method which, as previously mentioned, tends to map dissimilar measurements far apart in the low-dimensional space. The distance metric used between a given pair of 1445-dimensional data measurements is, therefore, taken to be the joint probability between the two, as defined in Eq. (3.3). The resolution is 24 intervals per dimension with 80% overlap between adjacent intervals and the DBSCAN method is once again used for subset clustering. Figure 3.4 shows that the generated topological network of the semiconductor data is separated into three subnetworks. This is consistent with the fact that the data sets collected from the three experiments have

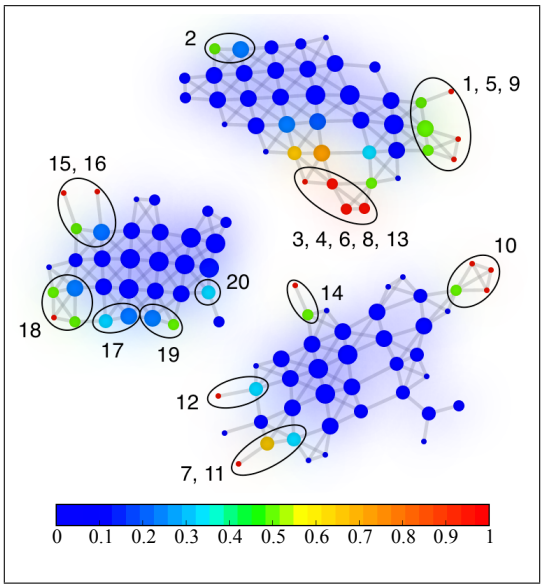
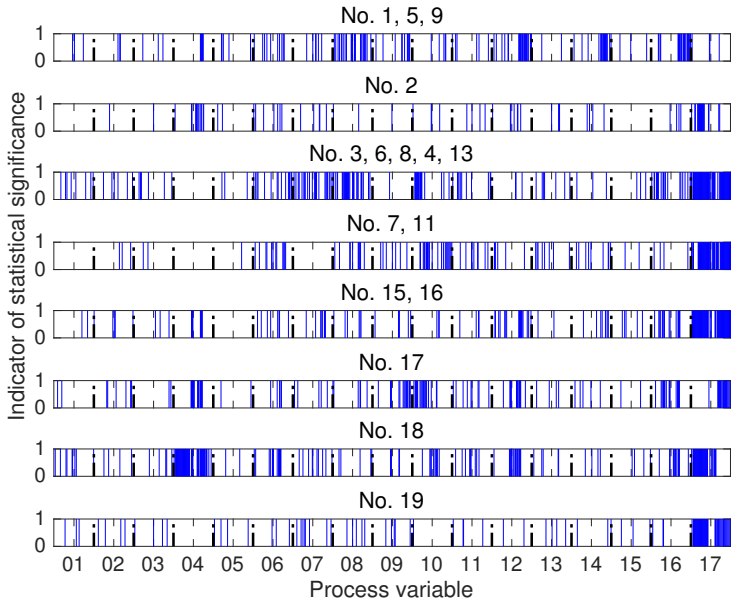


Figure 3.4: Topological network derived from the semiconductor data at a specified resolution. Each node is colored based on the average output for the measurements included in the node, where the output of a faulty wafer is 1 while the output of a normal wafer is 0. Subgroups that consist of nodes containing measurements of faulty wafers are extracted from each subnetwork.

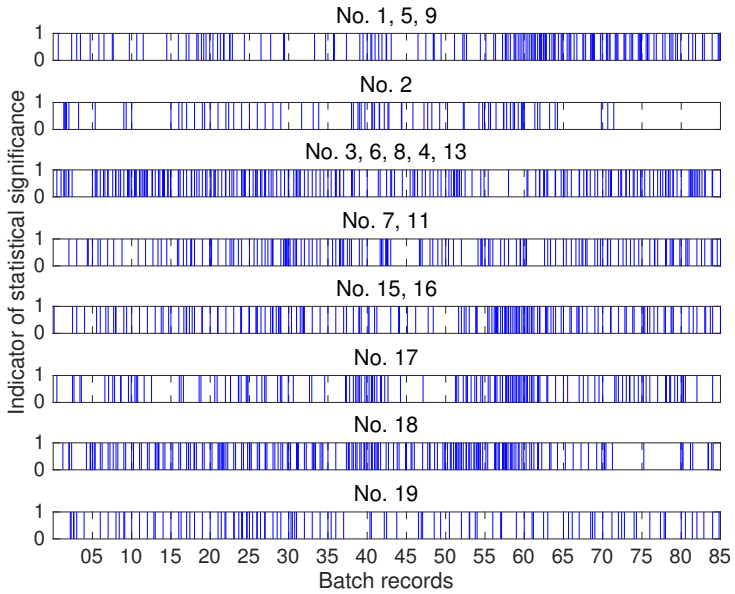
different means and somewhat different covariance structure. It is worth noting that faulty wafers 7 and 13 are two exceptions in the sense that each one is grouped with other wafers which originated from a different experiment.

We color each node based on the the average output values across all the measurements in the node. The output is either 0 or 1, representing a normal or a faulty wafer, respectively. As expected, measurements representing faulty wafers are positioned at the boundary regions of each subnetwork. We conjecture that this is because each faulty wafer was induced differently, giving rise to different behaviors in the wafer processing. We further identify subgroups consisting of nodes containing measurements of faulty wafers in Figure 3.4, as indicated by closed elliptical paths. Since the subgroups for faulty wafers 10, 12, 14, and 20 have extremely small sample size, they are excluded from the statistical tests for feature selection. For the rest of the subgroups, the Wilcoxon rank-sum tests are performed across all of the process variables throughout the batch. As a non-parametric alternative to the two-sample Student's t-test, the Wilcoxon rank-sum test is able to handle small sample size for non-normal distributions. These tests are conducted between each subgroup of faulty wafers and the nodes corresponding to normal wafers in the rest of its subnetwork, excluding those which belong to other subgroups of faulty wafers. The results of these tests are shown in Figure 3.5b, where they are organized by process variable in subfigure (a) and by batch record in subfigure (b).

By comparing the two rankings of the features, we find that statistically significant features ($p < 0.05$) are more concentrated within individual process variables than within individual batch records. For example, it is evident that process variable 17 (Vat valve) is strongly correlated with faulty wafers, while process variables 5 (Helium chuck pressure) has little impact on wafer failure. As in Section 3.4.1, we perform B-H procedure to adjust the p-values and count the occurrence of each statistically significant feature throughout the batch for every process variable. The results for both raw and adjusted p-values are shown in Figure 3.6. It is seen that the relative importance of the process variables remains more or less the same after B-H adjustment, especially for the first eight process variables. Hence,



(a) Features ordered by process variables



(b) Features ordered by batch records

Figure 3.5: Wilcoxon rank-sum test to identify the features that best differentiate between faulty wafers and normal wafers. The features are ordered by (a) process variables and (b) batch records, respectively. Statistically significant features ($p < 0.05$) have values of 1 as represented by the blue lines.

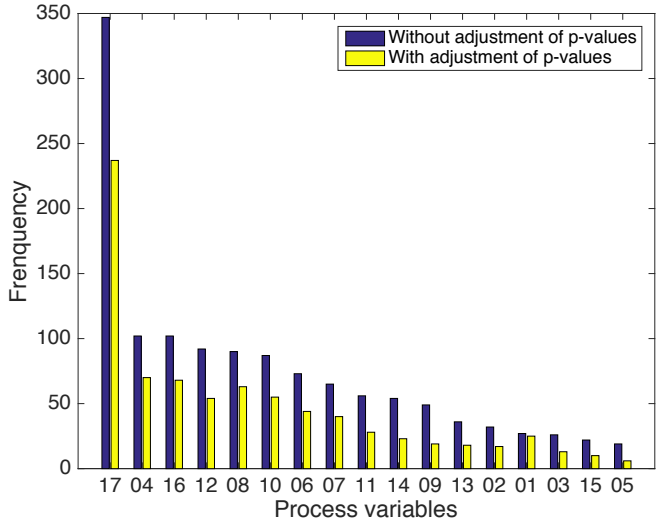


Figure 3.6: Counts of statistically significant features in terms of differentiating between faulty and normal wafers for each process variable.

we only select the first eight process variables for fault classification.

Predictive modeling

To build a fault detection classifier, we first compute the column means throughout the batch for each variable and use them for the new feature values. The transformed data is then randomly split into a training set and a testing set in the ratio of 7:3, where each set maintains the same proportion of normal and faulty wafers. The standard soft margin C-support vector machine (SVM) classifier with a Gaussian kernel, as implemented in `LIBSVM` [27], is employed for fault classification. The cost factor C and the variance σ of the Gaussian kernel are tuned using 10-fold cross-validation on the training set using an iterative grid search. We start a coarse grid search with exponentially growing sequences of C and γ first, thereafter proceeding with finer grid searches in the vicinity of the optimal region yielded by the previous grid search. Each grid search includes a total of 50 pairs of (C, γ) values which are used to apply the training model. To illustrate the performance of the fault

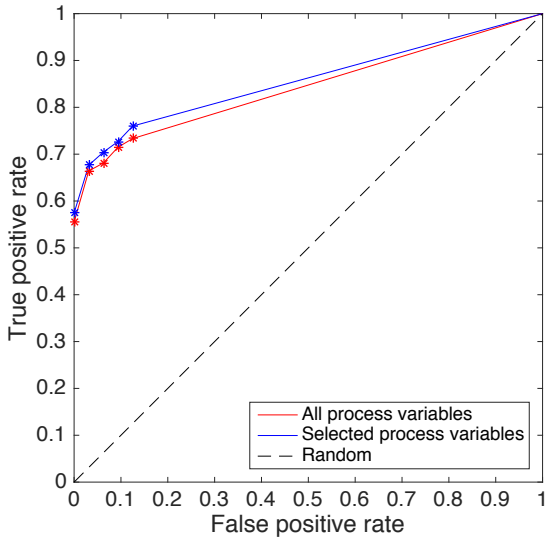


Figure 3.7: ROC curves of Gaussian kernel SVM classifiers on the data with all process variables and with selected process variables.

classifiers, receiver operating characteristic (ROC) curves for the testing set with all process variables and with selected variables are reported in Figure 3.7. As seen in Figure 3.7, the fault classifier with the eight selected process variables outperforms the classifier which uses all process variables, indicating the effectiveness of the former variables in predicting wafer failure. Meanwhile, about 18% reduction in the computational time is achieved from ∼1.1s to ∼0.9s of each run.

3.5 Discussion

In this chapter, we expand the application of TDA to the manufacturing process studies. The Mapper algorithm is applied to the predictive analysis of a chemical manufacturing process data set for yield prediction and a semiconductor etch process data set for fault detection. We show that it facilitates the analysis of the impact of each process variable on system outputs through direct visualization. Key process variables or features that impact the system outcomes are selected by analyzing the network shapes. We then use predictive

models to evaluate the impact of the selected features. Results show that the models achieve at least the same level of high prediction accuracy as with all the process variables, thereby, providing a way to carry out process monitoring and control in a more cost-effective manner.

Chapter 4

SPARSE REALIZATION OF TDA FOR MULTI-WAY CLASSIFICATION

The work reported here has appeared in IEEE Transactions on Knowledge and Data Engineering, vol. 30, no. 7, pp. 1403-1408, 2018. The sparse sampling work (included here for the sake of completeness) is done by Prof. Steven Brunton from the Department of Mechanical Engineering and Krithika Manohar from the Department of Applied Mathematics at the University of Washington.

4.1 Background and Motivation

Multi-way or multi-class classification, where the goal is to correctly predict one out of K classes for any data sample, poses one of the most challenging problems in supervised machine learning. However, a large number of real-world sensing problems in a variety of domains such as computer vision, robotics and remote diagnostics, do consist of multiple classes. Examples include human face recognition for surveillance, object detection for mobile robot navigation, and critical equipment condition monitoring for preventive maintenance. The number of classes in these problems often exceeds ten and sometimes goes up to a hundred depending on the complexity of the sensed system or environment and the number and types of sensor modalities.

While a whole host of techniques such as artificial neural networks, decision trees, naïve Bayes, nearest neighbors, and SVMs have been successfully applied for binary classification problems, extensions of these techniques have had mixed success in addressing multi-way classification problems with more than a few classes. Other approaches involving hierarchical classification or transformation to binary classification have not been particularly successful

either. The success rates diminish further in the absence of a large number of data samples for each of the labeled classes. The primary reason is that all of these methods encounter difficulties in selecting the right set of distinguishing features among the different classes.

Recent research has started investigating completely new techniques for multi-way classification that attempt to better understand the structure of the underlying high-dimensional sample space. One such class of techniques is TDA. TDA represents the unknown sample space in the form of persistent shape descriptors that are coordinate free and deformation invariant. Thus, the descriptors define topological features and yield insights regarding suitable feature selection.

Another critical tool facilitating multi-way classification is the feature-driven sparse sampling of high-dimensional data. Observations are typically sparse in a transform basis of informative features, thus, measurements can be optimally chosen to enhance discriminating features in the data. This permits heavily subsampled inputs for downstream classifiers, which drastically reduces the burdens of sample acquisition, processing and storage without sacrificing performance. In the context of image classification using linear discriminant analysis, Brunton et al. [17] use convex L_1 optimization to identify sparse pixel locations that map into the discriminating subspaces in PCA coordinates. Recent advances in model order reduction employ fast matrix pivoting schemes to sample PCA libraries for sparse classification of dynamical regimes in physical systems [113, 94].

Here, we bring together the two research areas of TDA and sparse sampling in the context of multi-way classification. In particular, we leverage sparse sampling for optimal feature selection once the features are extracted using a vectorization TDA method in challenging computer vision problems. The problems comprise three benchmark data sets pertaining to 3D meshes of synthetic and real human postures and textured images, respectively. We call our new method the Sparse-TDA algorithm and show that it achieves comparable accuracy as the TDA method with significantly lower training times. Thus, our method opens up a new direction in making online multi-way classification practically feasible.

The rest of the chapter is organized as follows. We first summarize a state-of-the-art

kernel-based TDA method to be used for later comparison in Section 4.2. Our Sparse-TDA method is described in the following section after outlining the QR factorization-based optimal sparse feature selection algorithm. Experimental results are discussed next in Section 4.4 followed by concluding remarks in Section 4.5.

4.2 Kernel-Based TDA Method for Multi-Way Classification

Given a set $\mathcal{X}$, $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a *kernel* if there exists a Hilbert space $\mathcal{H}$, called *feature space*, and a map $\Phi : \mathcal{X} \to \mathcal{H}$ such that

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle_{\mathcal{H}} \tag{4.1}$$

for all $x, x' \in \mathcal{X}$. In machine learning, a kernel represents a similarity measure between the samples, and Φ is called its *feature map*. A kernel satisfying Eq. (4.1) is also symmetric and positive definite [71].

4.2.1 Multi-scale kernel TDA

In [109], Reininghaus et al. devise the *persistence scale space* kernel on the set of PD $\mathcal{D}$ as a multi-scale kernel via a feature map $\Phi_\sigma : \mathcal{D} \to L_2(\Omega)$, where $\Omega = \{x = (x_1, x_2) \in \mathbb{R}^2 : x_2 \geq x_1\}$ denotes the space above the diagonal. Given a PD $D \in \mathcal{D}$, the feature map Φ_σ is the solution of a heat diffusion problem with a Dirichlet boundary condition on the diagonal:

$$\Phi_\sigma(D) = \frac{1}{4\pi\sigma} \sum_{y \in D} \Big(e^{-\frac{||x-y||^2}{4\sigma}} - e^{-\frac{||x-\bar{y}||^2}{4\sigma}} \Big), \tag{4.2}$$

where $\bar{y} = (b, a)$ is the mirror image of $y = (a, b)$ across the diagonal. The map then yields the kernel $k_\sigma : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$ in a closed form as

$$\begin{aligned} k_\sigma(F, G) &= \langle \Phi_\sigma(F), \Phi_\sigma(G) \rangle_{L_2(\Omega)} \\ &= \frac{1}{8\pi\sigma} \sum_{\substack{y \in F \\ z \in G}} \Big(e^{-\frac{||y-z||^2}{8\sigma}} - e^{-\frac{||y-\bar{z}||^2}{8\sigma}} \Big) \end{aligned} \tag{4.3}$$

for $\sigma > 0$ and $F, G \in \mathcal{D}$, which has been shown to be 1-Wasserstein stable. Further, note that because the summation in Eq. (4.3) is carried out over all pairwise combinations of

the points in the PDs F and G, evaluation of the kernel requires $O(|F||G|)$ time, where $|F|$ and $|G|$ denote the number of points in F and G, respectively. Experiments on benchmark data sets show that this method greatly outperformed an alternative approach based on persistence landscape [18], a popular statistical treatment of TDA.

4.3 Sparse-TDA Method

We now introduce a vector representation of a PD, termed a persistence image (PI), presented in [1]. Since our Sparse-TDA method will combine PI-based TDA with sparse sample selection, we first summarize the sparse sampling method before describing the combination.

4.3.1 Optimized Sparse Sample Selection

Vectorized PIs sparsely encode topological structure within a few key pixel locations containing nonzero entries. Sampling these PIs at critical pixel locations is often sufficient for training downstream classifiers at a fraction of the runtime required for full PIs. To determine these PI indices, we use a pixel sampling method based on powerful low-rank matrix approximations. First, we arrange the PI vectors from all the training classes into columns of a matrix $\mathbf{X}$ and compute its truncated singular value decomposition to obtain the dominant PI variation patterns (principal components) $\mathbf{U}_r$

$$\mathbf{X} \approx \mathbf{U}_r \mathbf{\Sigma}_r \mathbf{V}_r^T. \tag{4.4}$$

The SVD truncation parameter r determines the number of pixel samples and is chosen according to the optimal singular value threshold [58]. We then discretely sample the PI principal components using the pivoted QR factorization, an efficient greedy alternative to expensive convex optimization methods. QR pivoting is the workhorse behind discrete sampling for underdetermined least squares problems [20], polynomial interpolation [119], and more recently, model order reduction [46] and sensor placement [93]. The pivoting procedure optimizes a row permutation $\mathbf{\Pi}$ of the principal components that is numerically

well-conditioned by factoring $\mathbf{U}_r^T$ into unitary and upper-triangular matrices $\mathbf{Q}$ and $\mathbf{R}$

$$\mathbf{U}_r^T \mathbf{\Pi}^T = \mathbf{QR}. \tag{4.5}$$

The final step converts a given PI, $\mathbf{x}$, into a sparsely sampled PI, $\tilde{\mathbf{x}} = \mathbf{\Pi}_r \mathbf{x}$, where the first r permutation indices correspond to the selected pixel locations.

4.3.2 Combining Sparse Sample Selection with Persistence Images

Let $\mathcal{D} = \{D_i \mid i = 1, \ldots, n\}$ be a training set of PDs. To construct a PI from a given PD D_i [1], D_i is first transformed from birth-death coordinates to birth-persistence coordinates. Let $T : \mathbb{R}^2 \to \mathbb{R}^2$ be the linear transformation,

$$T(x, y) = (x, y - x). \tag{4.6}$$

A persistence surface $\rho_{D_i} : \mathbb{R}^2 \to \mathbb{R}$ on $T(D_i)$ is defined by

$$\rho_{D_i}(z) = \sum_{u \in T(D_i)} f(u) g_u(z) \tag{4.7}$$

where $f : \mathbb{R}^2 \to \mathbb{R}$ is a non-negative weighting function that is zero along the horizontal axis, continuous, and piecewise differentiable; $g_u : \mathbb{R}^2 \to \mathbb{R}$ is a probability function with mean $u = (u_x, u_y) \in \mathbb{R}^2$ and variance σ^2.

In our experiments, the linear weighting (LW) function is

$$f(u) = \frac{u_y}{u_y^*}, \tag{4.8}$$

where $u_y^* = \max_{i=1,\ldots,n} \max_{u \in D_i} u_y$. The form of the nonlinear weighting (NW) function is inspired by the weighting function used in [82] and chosen as

$$f(u) = \arctan(c u_y). \tag{4.9}$$

where $c = (\text{median}_{i=1,\ldots,n} \text{median}_{u \in D_i} u_y)^{-1}$. We choose g_u to be the Gaussian distribution, i.e.,

$$g_u(z) = \frac{1}{2\pi\sigma^2} e^{-[(z_x - u_x)^2 + (z_y - u_y)^2]/2\sigma^2}. \tag{4.10}$$

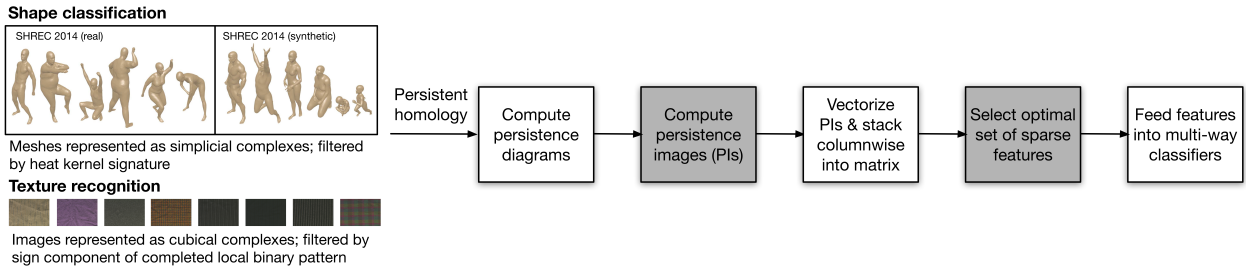


Figure 4.1: Pipeline of Sparse-TDA method for multi-way classification. Our contribution lies in linking persistence images with sparse sample selection, where computational speed-up is realized in both steps (gray panels).

where $z = (z_x, z_y)$. Then the PI, a matrix of pixel values, is obtained by calculating the integral of ρ_{D_i} on each grid box from discretization,

$$I(\rho_{D_i}) = \iint \rho_{D_i}(z_x, z_y) dz_x dz_y. \tag{4.11}$$

PI has also been proven to be 1-Wasserstein stable. Assume that the number of desired features (i.e., pixel samples) is s. Applying the sparse sampling method on $\mathbf{X}$, we obtain the row indices of s optimal pixel locations and the sparsely sampled PIs $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n \in \mathbb{R}^s$ for the downstream classifiers. Figure 4.1 shows the complete pipeline of our method.

4.4 Results

We now discuss the performance of our Sparse-TDA method on three benchmark computer vision data sets. The data sets are explained first, followed by illustrations of the selected features, and quantitative comparisons of our method with the L1-SVM feature selection method using the same PIs and the multi-scale kernel TDA method. The illustrations and comparison results show the usefulness of the method on challenging multi-way classification problems.

4.4.1 Data Sets

For shape classification, SHREC'14 synthetic and real data sets are used, given in the format of triangulated 3D meshes [106]. The synthetic set contains meshes from five males, five females and five children in 20 different poses, while the real set consists of 20 males and 20 females in 10 different poses.

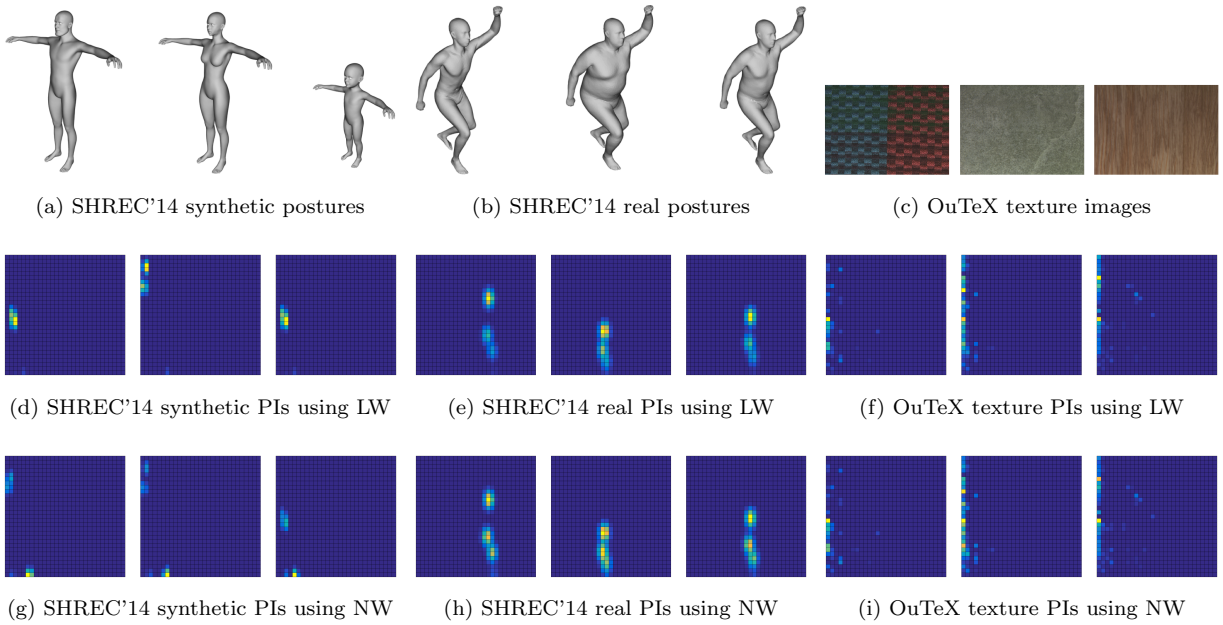
For texture recognition, we use the Outex_TC_00000 data set [100]. This data set contains 480 images equally categorized into 24 classes and provides 100 predefined 50/50 training/testing splits. During preprocessing, we downsample the original images to 32×32 pixel images as done in the multi-scale kernel TDA method.

4.4.2 Feature Selection

We first follow the same procedure performed in the multi-scale kernel TDA method to obtain the PDs. For SHREC'14 data sets, we compute the heat kernel signature [121] on the surface mesh of each object and then compute the 1-dimensional PDs using Dipha[1]. For the OuTeX data set, we take the sign component of the completed local binary pattern operator [65] as the descriptor function. Then we generate the 0-dimensional PDs from the filtration of its rotation-invariant version with $P = 8$ neighbors and radius $R = 1$.

To generate the PIs, we set the grid resolution to be 30×30 for all three data sets. In fact, the classification accuracy is fairly robust to the choice of resolution [1]. We also set σ to be 0.2, 0.0001 and 0.02 for SHREC'14 synthetic, SHREC'14 real and OuTex data sets, respectively. Figure 4.2 shows representative PIs for three different classes in all of our benchmark data sets. Noticeable differences are observed among the PIs for each of the three data sets, although the differences are most pronounced for the SHREC'14 synthetic data set, reasonably clear for the SHREC'14 synthetic data set, and less evident for the OuTeX data set. These differences in the pixel values of the PIs form the distinguishing class features from which an optimal set is selected by QR pivots. Figure 4.3 measures the effect of varying the

[1]https://github.com/DIPHA/dipha



(a) SHREC'14 synthetic postures (b) SHREC'14 real postures (c) OuTeX texture images

(d) SHREC'14 synthetic PIs using LW (e) SHREC'14 real PIs using LW (f) OuTeX texture PIs using LW

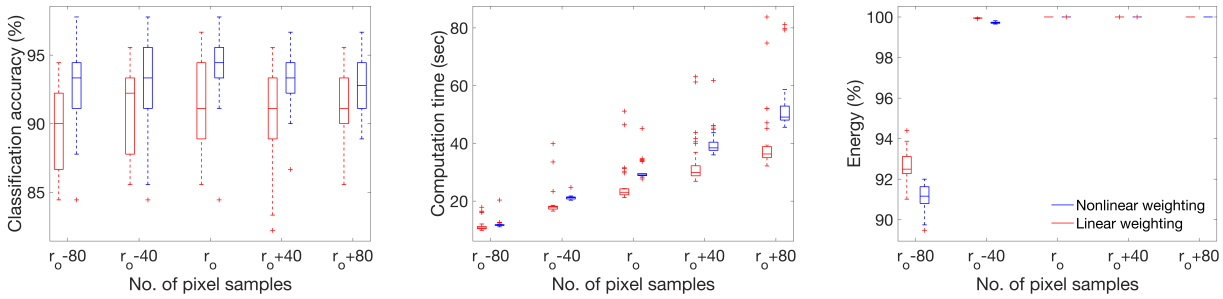
(g) SHREC'14 synthetic PIs using NW (h) SHREC'14 real PIs using NW (i) OuTeX texture PIs using NW

Figure 4.2: Representative classes and corresponding persistence images (PIs) for various benchmark data sets using two Sparse-TDA variants.

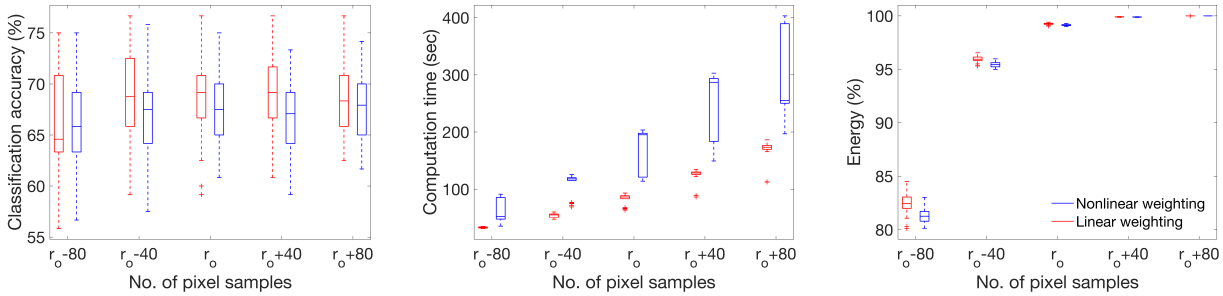
number of pixel samples determined by the SVD truncation parameter in Eq. (4.4) on classification accuracy and performance (see below for detailed settings). As expected, classifier training time increases with additional samples. The accuracy, however, improves until the number of samples equals the optimal SVD truncation parameter $s = r_o$, after which limited additional information is available. Beyond this value, accuracy tapers off, which is consistent with the percentage of PI variance (energy) captured by the truncated SVD. For this reason, s is selected as r_o for our Sparse-TDA method in the following simulations. In the case of the L1-SVM method, a sparse solution is generated by L1 regularization during the training phase of a linear classifier. No feature selection is involved for the kernel TDA method.

4.4.3 Classification Performance

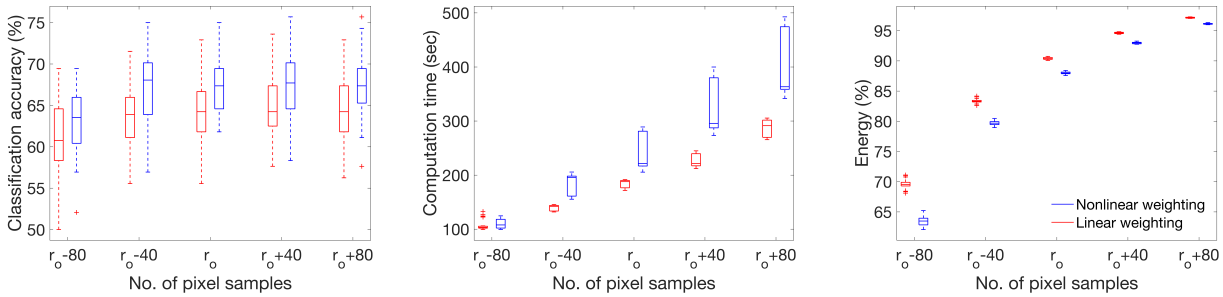
We feed the reduced feature vectors for training into a soft margin C-SVM classifier with a radial basis function (RBF) kernel, implemented in LIBSVM [27], for each data set. The



(a) SHREC'14 Synthetic. Training size: 210. Linear weighting: $r_o = 100 \sim 103$. Nonlinear weighting: $r_o = 99 \sim 104$.



(b) SHREC'14 Real. Training size: 280. Linear weighting: $r_o = 121 \sim 126$. Nonlinear weighting: $r_o = 122 \sim 126$.



(c) OuTeX Texture. Training size: 336. Linear weighting: $r_o = 112 \sim 116$. Nonlinear weighting: $r_o = 108 \sim 112$.

Figure 4.3: Comparison of classification accuracy, training time, and energy among different selections of desired pixel samples for the benchmark data sets using two Sparse-TDA methods. The results are based on 30 runs with a 70/30 training/testing split.

cost factor C and kernel parameter γ are tuned based on a grid search using 10-fold cross-validation on the training data. We start a coarse grid search with exponentially growing sequences of C and γ first, thereafter proceeding with finer grid searches in the vicinity of

the optimal region yielded by the previous grid search. Each grid search includes a total of 50 pairs of (C, γ) values which are used to apply the training model to the sparsely sampled PIs of the test set. For the L1-SVM method, since only the cost factor needs to be trained, it is then tuned 10 times using the same scheme as described above with the implementation in LIBLINEAR [53]. Results are reported based on 30 runs for each case with the exception of those presented for the OuTex data set in Tables 4.1-4.2, which are based on 100 runs.

Table 4.1 compares the classification accuracy of both the variants of the L1-SVM and our Sparse-TDA method with the multi-scale kernel TDA method. Note that the two SHREC'14 data sets are partitioned into 70/30 training/testing samples, whereas the OuTeX set is partitioned into 50/50 training/testing samples. The number of samples for each class is approximately the same in all the training sets. Consistent with the differences observed in the PIs among the classes, both the variants of our method perform slightly better than the kernel TDA method for the SHREC'14 real data set. On the other hand, our method is marginally worse than the kernel TDA method for both the SHREC'14 synthetic and OuTeX data sets, even though the accuracy increases slightly using nonlinear weighting. Both the L1-SVM variants are, however, inferior to the other methods by varying degrees for all the data sets.

Table 4.1: Comparison of classification accuracy (in %) with the same training and test data set split as in the original multi-scale kernel TDA article

Method		SHREC'14 Synthetic	SHREC'14 Real	OuTeX Texture
L1-SVM	LW	89.6 ± 2.3	63.9 ± 4.9	55.1 ± 3.5
	NW	92.1 ± 2.5	63.9 ± 4.4	57.4 ± 3.6
Sparse-TDA	LW	91.5 ± 3.0	$\mathbf{68.8 \pm 4.2}$	62.6 ± 2.7
	NW	94.0 ± 2.6	67.8 ± 3.6	66.0 ± 2.4
Kernel TDA		$\mathbf{97.8 \pm 2.0}$	65.3 ± 4.5	$\mathbf{69.2 \pm 2.4}$

Table 4.2 provides a comparison of the three methods in terms of the SVM-based classifier

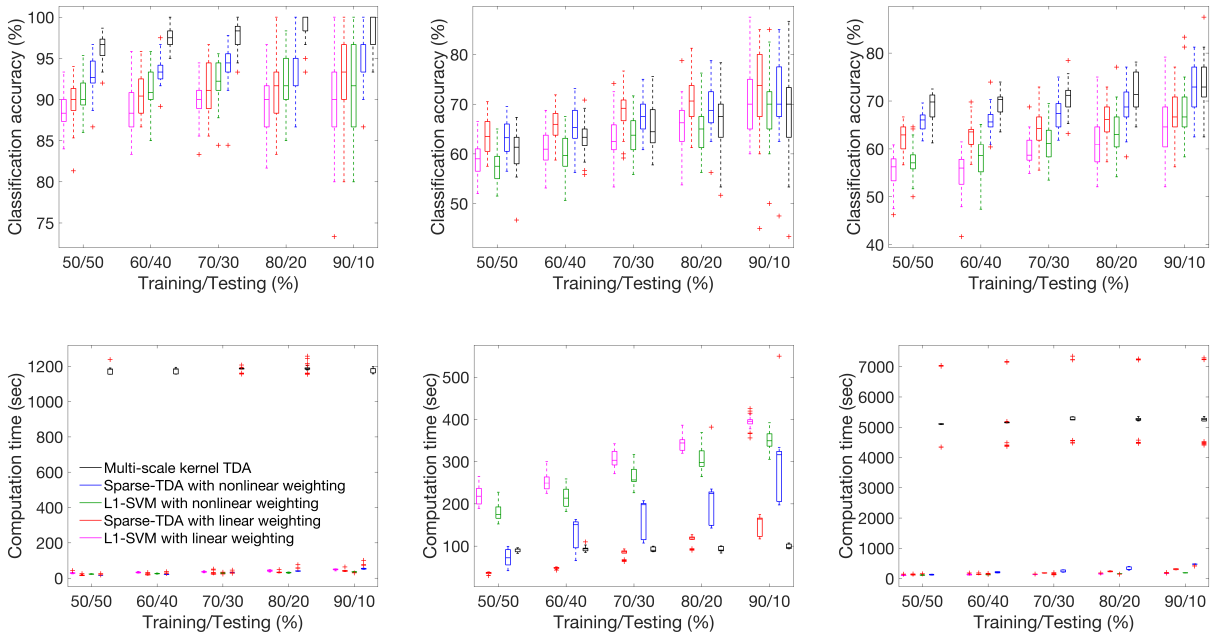
training time as measured on a laptop with a 2.4 GHz Intel Core i5 CPU and 4 GB RAM. In the case of the L1-SVM and Sparse-TDA methods, the training time starts from the computation of the PIs. Not surprisingly, both our method variants are usually much faster than kernel TDA as they use smaller sets of selected features. As expected, the reduction in training time is greater with linear weighting than with nonlinear weighting. In fact, the Sparse-TDA method with linear weighting achieves about 46X speed-up for the SHREC'14 synthetic data set and roughly 45X speed-up for the OuTeX data set. However, there is no consistent speed-up for the SHREC'14 real data set owing to the fact that there are only 4-5 points in each PD, rendering the training of the kernel TDA method exceptionally fast. In contrast, there are 38-294 points and 127-299 points in each PD for the SHREC'14 synthetic and OuTeX data sets, respectively. On the other hand, the L1-SVM method is not consistently fast due to the non-differentiability of the L1-regularized form, which leads to more difficulties in solving the optimization problem during training. For example, the training time of the L1-SVM method with linear weighting is more than three times as much as that of our counterpart method.

Table 4.2: Comparison of classifier training time (in s) with the same training and test data set split as in the original multi-scale kernel TDA article

Method		SHREC'14 Synthetic	SHREC'14 Real	OuTeX Texture
L1-SVM	LW	35.4 ± 2.9	305 ± 20.9	$\mathbf{106 \pm 14.8}$
	NW	28.5 ± 1.9	267 ± 21.0	113 ± 15.9
Sparse-TDA	LW	$\mathbf{25.6 \pm 7.0}$	$\mathbf{82.2 \pm 9.3}$	120 ± 9.7
	NW	30.5 ± 3.5	171 ± 41.4	131 ± 11.5
Kernel TDA		1182 ± 12.0	92.3 ± 5.1	5457 ± 979

Figure 4.4 shows the trends in improving the classification accuracy and reducing the classifier training time, respectively, as a function of increasing training/testing split for all the benchmark data sets. Consistent with the results reported in Table 4.1, our classification

accuracy is marginally inferior to that of the kernel TDA method for the SHREC'14 synthetic and OuTeX data sets. However, both our method variants marginally outperform the kernel method for the most challenging SHREC'14 real data set. The training time trends are also very similar to the results presented earlier in Table 4.2, with more than an order of magnitude reduction for the SHREC'14 synthetic and the OuTeX data sets, and comparable values for the SHREC'14 real set. The increase in classifier training times with higher training/test splits is, however, slightly more for both our method variants as compared to the kernel TDA method due to the selection of more pixel samples as training size increases. Overall, we observe that for each of the benchmark data sets, at least one of our Sparse-TDA variants outperforms the kernel TDA method either in terms of classification accuracy or classifier training time. Moreover, Sparse-TDA outperforms both the L1-SVM variants in

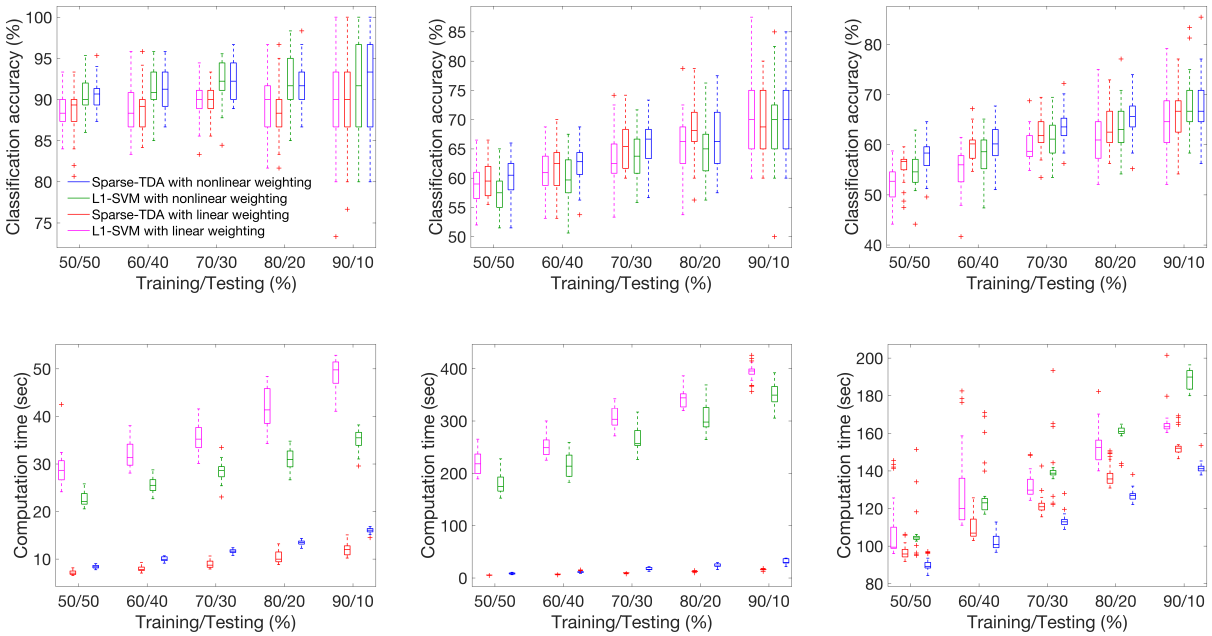


(a) SHREC'14 Synthetic (b) SHREC'14 Real (c) OuTeX Texture

Figure 4.4: Comparison of classification accuracy and training time among L1-SVM, Sparse-TDA, and state-of-the-art multi-scale kernel TDA methods for various training-testing partition ratios.

terms of classification accuracy for all the data sets, and achieves comparable computation time for the SHREC'14 synthetic and the OuTeX data sets, and a substantial reduction for the SHREC'14 real set.

We run additional experiments with an L1-SVM feature selection method and an L_2-regularized linear SVM classifier for our method. The L_2-regularized linear SVM is now trained exactly in the same way as the L1-SVM method and implemented using LIBLINEAR. As seen in Figure 4.5, Sparse-TDA outperforms the L1-SVM variants with respect to both classification accuracy and computation time in almost every case. These results further demonstrate the efficiency and effectiveness of QR sampling in our method, and also show the advantage of separating sampling and classification to have greater flexibility in choosing



(a) SHREC'14 Synthetic (b) SHREC'14 Real (c) OuTeX Texture

Figure 4.5: Comparison of classification accuracy and training time between L1-SVM and Sparse-TDA with a L_2-regularized linear SVM classifier for various training-testing data set partition ratios. The results are based on 30 runs in each case.

the downstream classifiers. In these experiments, the differentiability of the L_2 regularization form helps to solve the optimization problem more easily than the L_1 regularization form during training, resulting in higher accuracy and shorter training time.

4.5 Discussion

In this chapter, we present a new method, referred as the Sparse-TDA algorithm, that provides a sparse realization of a TDA algorithm. More specifically, we combine optimized sparse sampling based on pivoted QR factorization with a state-of-the-art TDA method. Instead of persistence diagrams, we use a vector-based representation of persistent homology, called persistence images, with two different weighting functions to extract the topological features.

The results are promising on three benchmark multi-way classification problems pertaining to 3D meshes of human posture recognition, both for real and synthetic shapes, and image texture detection. Our method gives similar classification accuracy and substantial reduction in training times as compared to a kernel TDA method that was earlier evaluated on these data sets. It also provides better accuracy and similar training times as compared to popular SVM classifiers. Such performance is, therefore, expected to lay the foundation for online adaptation of TDA on challenging data sets with a large number of classes in response to changes in the availability of training samples

In the future, we would like to further improve the accuracy of the Sparse-TDA method by designing our own weighting function for the persistence images. We would also like to come up with theoretical performance guarantees based on the characteristics of the data sets, particularly the training sample size for each individual class. Last but not the least, we plan to show the effectiveness of our method on other hard classification problems arising in robot visual perception and human face recognition.

Chapter 5

EFFICIENT COMMUNITY DETECTION IN LARGE-SCALE DYNAMIC NETWORKS USING TDA

The work reported here is based on a collaborative research with Prof. Yen-Chi Chen from the Department of Statistics and Ruqian Chen from the Department of Mathematics at the University of Washington. The theoretical background of this work (included here for the sake of completeness) has appeared in the 31^{st} Conference on Neural Information Processing Systems (NIPS) workshop on Synergies in Geometric Data Analysis [32].

5.1 Background and Motivation

The need to understand the dynamical and functional behavior of real-world networked systems has initiated extensive investigation of network structures over the past decade. Meanwhile, the topological analysis in the recent neuroscience studies has demonstrated that it is able to extract intrinsic information from neural networks that is practically impossible to extract using other less recent techniques of network theory [63, 37].

There is no universal definition of a community (a.k.a. cluster or cohesive subgroup) in network theory. A loose definition is that a community is a subgraph such that "the number of internal edges is larger than the number of external edges" [55]. One pioneering work in community detection was proposed by Girvan and Newman in 2002 [61]. The authors developed an algorithm in which the communities were isolated by the successive removal of the identified inter-community edges. Since then, many new techniques, such as spin models, random walks, optimization, synchronization, as well as traditional clustering methods have been presented [108, 133, 98, 3, 22].

Most of the aforementioned methods deliver standard partitions in which each vertex is

assigned to a single community. However, vertices are often shared between communities in real-world networks. Therefore, detecting overlapping communities has received a lot of attention in the recent past. The first and the most popular algorithm is the clique percolation method (CPM) proposed by Palla et al. [102]. It is based on the assumption that the internal edges of a community are likely to form cliques due to their high densities. Thus, a community is defined as a k-clique chain, i.e., a union of all the k-cliques that can be reached from each other through a series of adjacent k-cliques, where a k-clique refers to a maximal clique with k vertices and two k-cliques are adjacent if they share $k-1$ vertices. In terms of implementation, the first step of this method is to find all the maximal cliques in the network. The second step is to generate the pairwise clique overlap matrix and extract the k-clique matrix from it. The extracted matrix represents the adjacency matrix of the cliques with size k. Overlapping communities are then detected by finding all the connected components in the adjacency matrix.

In the CPM, k is a predefined input parameter that may render information loss by leaving a considerable fraction of vertices and edges out of the communities. Moreover, without a prior structural information of the network, it is difficult for one to choose the value of k to identify meaningful communities. To preserve the structural information as much as possible, we propose to convert a network to a topological representation in the form of a k-clique based community tree to summarize the community structure in the network at each order. The evolutionary analysis of the community structure will then be transformed to track the topological changes of the k-clique based community tree over time. To this end, as opposed to the CPM for static networks, we develop an incremental algorithm that allows us to update community trees with incremental changes in a network to keep track of evolving communities.

Furthermore, we address the robustness of communities with respect to vertex and/or edge updates. In a fast-changing network, many of vertex and/or edge updates over a given time window may not lead to any significant change in the global community structure. Thus, it is unnecessary to incrementally update the evolving communities for each time

window. For this reason, we adopt a metric defined from a previous study [32] to assess the topological change in community trees, and integrate it into the incremental algorithm to further improve the computational efficiency.

Following this framework, the rest of this chapter is organized as follows. In Section 5.2, we first review the stability of community trees. Section 5.3 outlines the implementation aspects for building and updating a community tree from a undirected, unweighted network. The overall algorithm, named as Dynamic CPM, is presented in Section 5.4. Experimental results on a diverse collection of social network datasets are discussed in Section 5.5 followed by a discussion of our findings in Section 5.6.

5.2 Stability of Community Tree

5.2.1 Stability Theory

Chen et al. [32] defines the bottleneck distance between two community trees as the bottleneck distance between their corresponding PDs, and proposes to use this metric to quantify how the two trees differ. To prove community trees are stable with respect to this metric, Chen et al. further introduces a quantity named *star number*. Mathematically, the *addition star number* (ASN) of G_1 and G_2 is given by

$$ASN(G_2, G_1) = \min\{|V_0| : \nu(e) \cap V_0 \neq \emptyset \ \ \forall e \in E(G_2) \backslash E(G_1)\},$$

where V_0 is a collection of vertices and $\nu(e)$ represents the two vertices of e. The *removal star number* (RSN) can be defined in a similar manner. The sum of RSN and ASN is the *total star number* (TSN). Figure 5.1 provides an example of computing bottleneck distance of community trees and the TSN.

$TSN(G_2, G_1)$ attributes the change from G_1 to G_2 to a specified number of vertices. Moreover, it has been proved that the difference between two community trees is bounded above by their TSN, i.e.,

$$d_B(\mathcal{T}(G_1), \mathcal{T}(G_2)) \leq TSN(G_2, G_1).$$

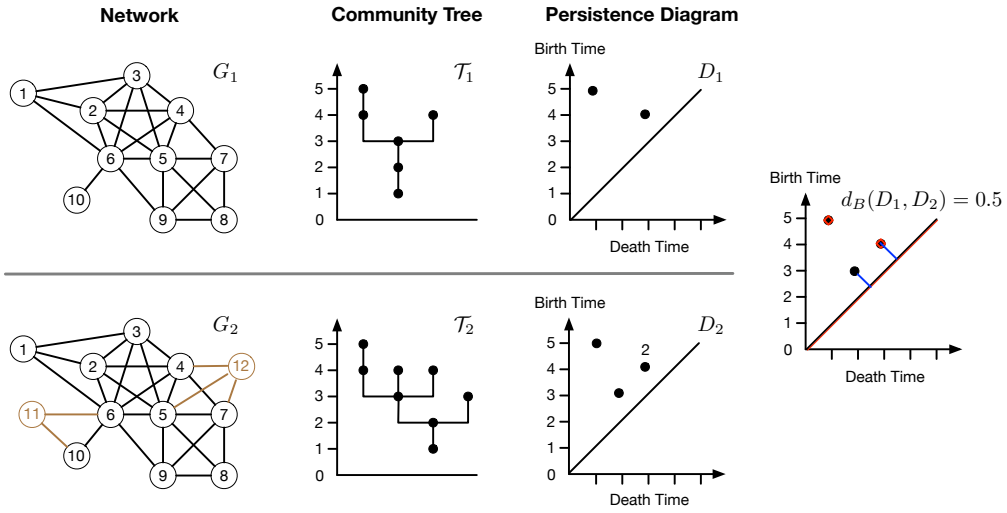


Figure 5.1: The distance between two community trees. The community tree $\mathcal{T}_2$ and the corresponding PD D_2 are updated as v_{11}, v_{12} and multiple edges are added (colored in brown) from G_1 to G_2. The PD in the right panel shows an optimal matching between the points in D_1 and D_2, which indicates $d_B(D_1, D_2) = 0.5$, i.e., $d_B(\mathcal{T}_1, \mathcal{T}_2) = 0.5$. On the other hand, $TSN = ASN = 2$ since all the added edges are incident to v_{11} and v_{12}.

This result guarantees that one can infer information about the changes to a community tree from the TSN. Thus, rather than directly update the community tree, we compute the TSN first to estimate the topological change in community structures between the two networks.

5.2.2 Algorithm for the upper bound of TSN

Although computing the TSN has been proved to be NP-complete in [32], we can circumvent this issue by calculating an upper bound for the TSN. Given a temporal network for a duration of time, we usually take all the vertices and edges up to a particular time t and create a graph G_t for evolutionary analysis. Thus, the change from G_{t-1} to G_t is now an incremental case where vertices and edges are only added to G_{t-1}. Accordingly, computing

the upper bound of TSN is reduced to computing the upper bound of ASN.

Algorithm 2 Compute an upper bound on the TSN in the incremental case

1: **procedure** TSN-UPPER-BOUND(G_+^Δ)
2: $\tau \leftarrow |\text{FIND-VC}(G_+^\Delta)|$ ▷ Use Bar-Yehuda and Even Algorithm to obtain 2-OPT local ratio for minimum vertex cover.
3: **return** τ

Here we employ an approximation algorithm to obtain the upper bound τ of ASN. As shown in Algorithm 2, we first compute an edge-difference graph $G_+^\Delta = (V(G_t), E(G_t) \backslash E(G_{t-1}))$[1] as the input. Bar-Yehuda and Even's greedy algorithm [4] is then used to find a vertex cover for G_+^Δ. The solution given by this algorithm is guaranteed to be within 2 times the optimum solution. Hence, we use the size of this cover as an upper bound for ASN. The worst-case runtime for Bar-Yehuda and Even's algorithm is $O(\max(|E_{t-1}|, |E_t|))$. Therefore, Algorithm 2 also has a worst case runtime of $O(\max(|E_{t-1}|, |E_t|))$.

5.3 Clique Graph and Euler Tour Tree

In this framework, we transform a network G into a community tree through an auxiliary structure, termed as weighted *clique graph* (CG). In a weighted CG, each node represents an MC found in the network and the edge weight denotes the number of vertices shared by the two corresponding MCs. Since the presence of any single vertex in G does not affect the resulting $\mathcal{T}(G)$, we ignore these MCs of size 1 when generating a weighted CG. Let $\mathcal{G}$ be a weighted CG that comprises all the MCs of size $s \geq 2$, and define $\mathcal{G}_i$ as the subgraph of $\mathcal{G}$ composed of CG nodes representing the MCs of size $s \geq i+1$ and edges with weights $w_{e_\mathcal{G}} \geq i$. Note that $\mathcal{G}_1 = \mathcal{G}$.

To perform the updates on the community tree efficiently as G changes, we maintain a spanning forest (a spanning tree for each connected component (CC)) of $\mathcal{G}_i$, denoted by

[1] $(V(G), E(G))$ and $(V(G_t), E(G_t))$ are henceforth denoted by (V, E) and (V_t, E_t), respectively, for simplicity.

$\mathcal{F}_i$ for $i = 1, \ldots, \omega(G) - 1$. We refer to the edges in $\mathcal{F}_i$ as the *tree edges* and maintain an adjacency list of the tree edges for every node at each level i. Moreover, since the birth time of a CC in $\mathcal{F}_i$ equals the size of the largest MC included in the CC, we choose this MC as the *representative MC* for the CC and use it as the CC's "label". If there are multiple MCs with the same maximum size, without loss of generality, the MC with the minimum ID is selected as the representative MC. Accordingly, we record the death time of the CC as the death time of the representative MC and use the ID of the representative MC as the CC's ID. We also name the CG node corresponding to the representative MC as *representative CG node*. See Figure 5.2 for an example.

Each spanning tree in the forest is built on an underlying *Euler Tour (ET) Tree* data structure, introduced by Henzinger and King [69]. The ET tree is constructed based on the Euler tour of the spanning tree, which is essentially a depth-first traversal of the spanning tree and ends at the node at which it starts. Each ET tree is often stored as a balanced binary search tree (BST). The data structure was later modified by Tarjan [122] to better support operations on the tree nodes such as changing node values. Therefore, we adopt Tarjan's version of the data structure here. In this version, the Euler tour of a spanning tree is a sequence of arcs (directed edges) over the spanning tree with one "loop" arc per node visited by the tour, i.e, each edge (u, v) results in two arcs (u, v) and (v, u), while each node v corresponds to a single loop arc (v, v). See Figure 5.3 for a simple illustration.

We implement the Euler tour of a spanning tree with a splay tree [118], which is a self-adjusting BST where a node is always splayed to the root through a series of rotations when accessed. Each node in the spanning tree holds a pointer to its corresponding node, referred to as *ET node*, in the splay tree. Particularly, the ET node corresponds to the representative CG node is referred to as *representative ET node*. Meanwhile, each ET node in the splay tree also stores pointers to its parent, right and left child. With this representation, the following operations are supported in $O(\log n)$ amortized time using $O(n)$ space, where n is the number of nodes in the spanning tree(s) involved in the operation [122].

- CONNECTED(u, v): Return if u and v are in the same spanning tree.

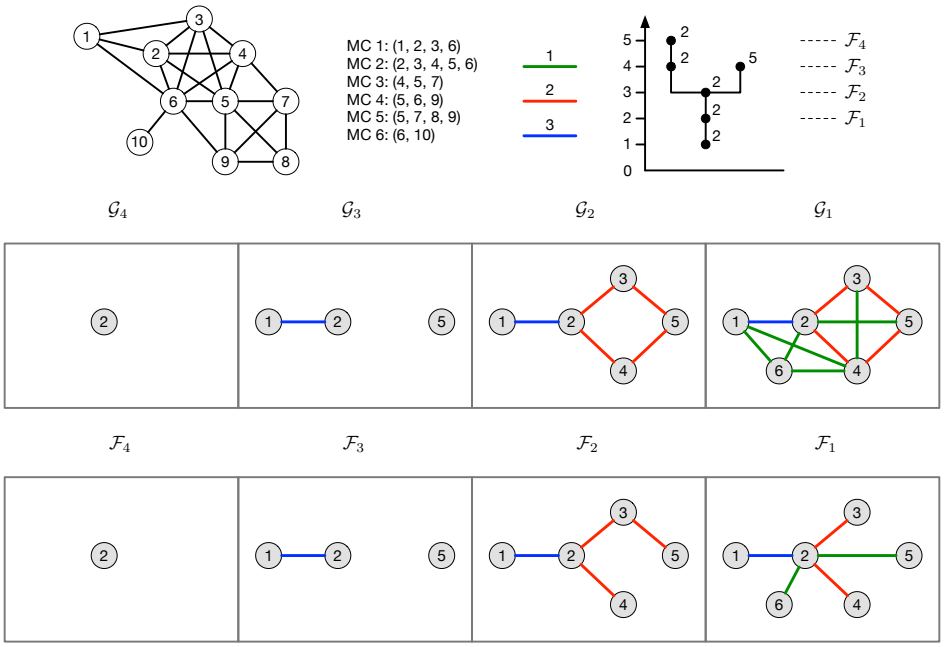


Figure 5.2: An example of the sequence of weighted clique graphs $\mathcal{G}_i$ and the corresponding spanning forests $\mathcal{F}_i$, $i = \omega(G) - 1, \ldots, 1$, for a given network G. The green, red and blue lines in $\mathcal{G}_i$ and $\mathcal{F}_i$ represent the edges with an edge weight of 1, 2 and 3, respectively. The connectivity information in $\mathcal{F}_i$ is summarized in the community tree $\mathcal{T}(G)$ at order $i + 1$. For example, the representative CG nodes in $\mathcal{F}_3$ are 2 and 5, while the representative CG node in $\mathcal{F}_2$ is 2.

- LINK(u, v): If u and v are in different spanning trees, insert an edge (u, v) connecting the trees together.
- CUT(u, v): Delete the edge (u, v) from a spanning tree, splitting the tree into two trees.
- ADD-VAL(v, α): Add α to the value of each node in a spanning tree containing node v.

CONNECTED is implemented by using the FIND-ROOT operation in the splay tree. Since linking two trees and cutting a tree each amounts to a fixed set of splitting and concatenation operations on Euler tours, LINK and CUT are realized by a constant number of SPLIT and JOIN operations of the splay tree [122].

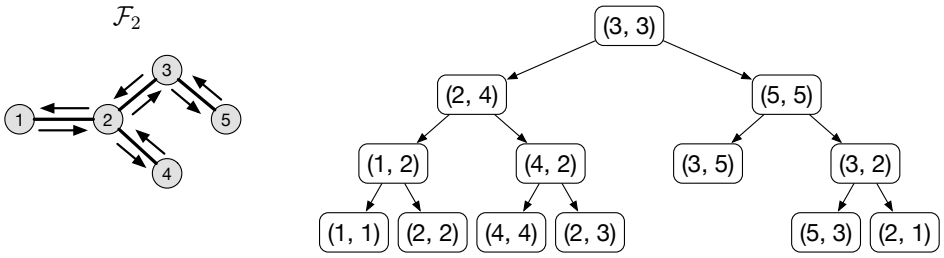


Figure 5.3: An example of the Euler tour of a spanning tree in $\mathcal{F}_2$ (left) and one possible representation of this Euler tour as a balanced BST keyed by the index in the tour (right). In this example, the Euler tour is $(1,1)$-$(1,2)$-$(2,2)$-$(2,4)$-$(4,4)$-$(4,2)$-$(2,3)$-$(3,3)$-$(3,5)$-$(5,5)$-$(5,3)$-$(3,2)$-$(2,1)$. A spanning tree with n nodes will be represented by a balanced BST with $2(n-1)+n$ nodes.

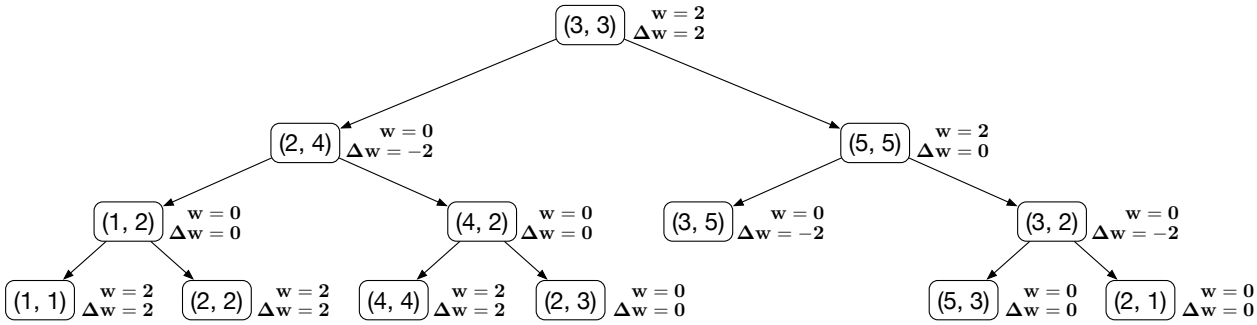


Figure 5.4: An example of an ET tree for implementing the ADD-VAL operation. It is sufficient to only maintain $\Delta w(x)$ for each node x to find $w(x)$ in the ET tree.

To handle the ADD-VAL operation efficiently, we store the value for each node of the splay tree implicitly. Specifically, let $w(x)$ be the value associated with a splay tree node x. Rather than storing $w(x)$ at x, we store the difference $\Delta w(x)$ between $w(x)$ and the value of its parent, i.e.,

$$\Delta w(x) = \begin{cases} w(x) & \text{if } x \text{ is the root the splay tree} \\ w(x) - w(p(x)) & \text{if } x \text{ is a nonroot, where } p(x) \text{ is the parent of } x \end{cases} \tag{5.1}$$

When x is the root, then $w(x)$ itself is assigned to $\Delta w(x)$. In this manner, for each node x, $w(x)$ is computed by summing Δw over all the ancestors of x. It also implies that adding α to $\Delta w(x)$ amounts to adding α to the values of all the descendants of x. Thus, ADDVAL(v, α) is realized by simply adding α to the node value Δw of the root node of the splay tree that v's corresponding ET node belongs to. $\Delta w(x)$ is updated in each rotation in $O(1)$ time during a splay step. One can check [118] for details.

In our case, for a loop node, $w(x)$ is defined as the ID of the representative ET node of the ET tree that the loop node belongs to. This affiliation links each CG node in a spanning tree to the corresponding representative CG node, which facilitates the need to update the representative CG node in the new spanning tree whenever a tree edge is inserted between two CG nodes. For a nonloop node, $w(x)$ is assigned an arbitrary value of 0 since it can be simply ignored. An example is given in Figure 5.4 and more details can found in Algorithm 10.

5.4 Dynamic CPM

We now present the overall method of our paper, which we call the Dynamic CPM. As laid out in Algorithm 3[2], it essentially computes the sequence of community trees for a given sequence of complex networks that vary over time. We then directly obtain the network communities simply by scanning through each order of the computed trees one-by-one.

There are two distinct phases in the Dynamic CPM: *initialization* and *update if necessary*. When a graph, corresponding to a previously non-analyzed network, is fed to the method for the first time, Algorithm 4 is used to yield the initial community tree. Subsequently, as new graphs, which are modified forms of the initial graph with vertex and edge insertions, are provided as inputs, we first use our TSN bound result on the bottleneck distance between any two successive graphs to decide whether new community trees should be computed. For efficiency, we can use Algorithm 2 to approximate the TSN bound. If we decide to compute new community trees, then Algorithm 12 is used to update the trees without recomputing

[2]The notations for Algorithm 3 and the following algorithms are given in Table A.1.

Algorithm 3 Calculate sequence of community trees for dynamic networks that are represented as time-varying undirected, unweighted graphs $G_0, G_1, \ldots, G_T$

procedure DYNAMIC-CPM($G_0, G_1, \ldots, G_T, l$)
  $\mathcal{T}_0, \mathcal{M}_0, \mathcal{T}_0, m_0 \leftarrow$ BUILD-CT(G_0) ▷ Algorithm 4
  $Q \leftarrow \emptyset$, $S \leftarrow 0$, $t' \leftarrow 0$
  for $t = 1, \ldots, T$ **do**
    $G_+^\Delta \leftarrow (V_t, E_t \backslash E_{t-1})$
    $\tau_t \leftarrow$ TSN-UPPER-BOUND(G_+^Δ) ▷ Algorithm 2
    if $t \leq l$ **then**
      PUSH-BACK(Q, τ_t)
      $S \leftarrow S + \tau_t$
    else
      if $\tau_t \geq S/l$ **then**
        $V_+^\Delta \leftarrow V_t \backslash V_{t'}$, $E_+^\Delta \leftarrow E_t \backslash E_{t'}$
        $\mathcal{T}_t, \mathcal{M}_t, \mathcal{T}_t, m_t \leftarrow$ UPDATE-CT($G_{t'}, V_+^\Delta, E_+^\Delta, \mathcal{T}_{t'}, \mathcal{M}_{t'}, \mathcal{T}_{t'}, m_{t'}$) ▷ Algorithm 12
        $t' \leftarrow t$, $\mathcal{T}_{t'} \leftarrow \mathcal{T}_t$, $\mathcal{M}_{t'} \leftarrow \mathcal{M}_t$, $m_{t'} \leftarrow m_t$
      else
        $\mathcal{T}_t \leftarrow \mathcal{T}_{t-1}$ ▷ Retain previous community tree as the network has not changed substantially (from a topological perspective)
      $\tau_{t-l} \leftarrow$ POP-FRONT(Q)
      PUSH-BACK(Q, τ_t)
      $S \leftarrow S - \tau_{t-l} + \tau_t$
  return $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_T$

them from scratch. We present the initial community tree construction algorithm and the update algorithm (along with all their sub-routines) in the Appendix.

It is worth noting the importance of the update condition (line 11) in Algorithm 3. We are typically interested only in communities with strong (topological) persistence rather than communities that are short-lived and change even for minor modifications to the networks. In fact, we do not want to update the detected communities in such scenarios. Therefore, the choice of update condition provides a measure of control over the persistence of the communities in addition to rendering our community tracking method more efficient. Here, we update $\mathcal{T}_t$ if τ_t is no less than a moving average of length l. In the following experiments, the length l is selected to be 3.

Figure 5.7 provides a realistic scenario where the Dynamic CPM works well. In this example, we recompute the community trees at t_4 and t_7 because their respective TSN bounds are greater than the moving averages from previous windows. Indeed, the change from G_0 to G_4 is non-trivial as a new 5-community is born and a large number of lower order communities emerge. From G_4 to G_7, three more 5-communities appear along with more lower order communities being created. Thus, it makes sense to recompute the community tree. On the other hand, taking the set of 3-communities as an example, the unchanged community trees at order 3 well approximate the real 3-communities found by the CPM as is shown in Figure 5.6.

5.5 Results

We first briefly describe the network datasets that are used to evaluate our algorithm below. Although all the datasets are originally collected in the format of (sender, receiver, timestamp) tuples as directed networks, directionality is ignored in our analysis.

- Email networks [email-Enron [78]; email-Eu-core [105]]: Each vertex is an email address and an edge (v_i, v_j) means that person v_i sent or received an e-mail to or from person v_j. email-Enron records the communication among Enron employees mostly from

Table 5.1: Summary statistics of datasets

Dataset	# vertices	# edges	# periods	(V_t, E_t)
email-Enron	182	2097	30	(150± 31, 1,150± 674)
bitcoin-au	1288	6236	24	(739 ± 390, 3,164 ± 2,024)
college-msg	1899	13,838	23	(1,787 ± 93, 12,847 ± 923)
email-Eu-core	893	12,556	51	(823 ± 56, 8,542 ± 2,758)
short-msg	44,090	52,222	41	(30,209 ± 9,485, 30,952 ± 12,816)
Facebook-wall	45,813	183,412	37	(16,797 ± 11,819, 59,711 ± 50,379)

November 1998 to June 2002[3], while email-Eu-core contains email data over 1 year from members of four different departments at a large European research institution.

- Bitcoin network [bitcoin-au [79]]: Each vertex is an active user ID and an edge represents a Bitcoin payment transferred between two user IDs. The bitcoin-au dataset consists of all transactions made from September 2010 to December 2013.
- Online social networks [college-msg [104]; Facebook-wall [126]]: Each vertex is a user who interacts with others through an online platform. The college-msg network, spanning over 193 days, originates from an online community for students at University of California, Irvine, and an edge (v_i, v_j) means user v_i sent or received a private message to or from user v_j. The Facebook-wall dataset is derived from the Facebook New Orleans networks ranging from September 2004 to January 2009, where an edge (v_i, v_j) indicates user v_i (or v_j) posted on user v_j's (or v_i's) wall.
- Short message correspondence [short-msg [129]]: Each vertex is a mobile phone user and an edge (v_i, v_j) represents user v_i sent or received a short message (SM) to or from user v_j. The short-msg dataset spans over 338 days.

More detailed information about our datasets is summarized in Table 5.1. The second

[3]This dataset is downloaded from `http://www.cis.jhu.edu/~parky/Enron/`. The other datasets can be found from the links in `http://snap.stanford.edu/temporal-motifs/data.html`

and third column refers to the number of vertices and edges at the end of the observation period, respectively. From (V_t, E_t), we observe that the college-msg dataset is a relatively slow-growing network while the Face-wall dataset is a fast-growing one. This is consistent with the values shown in the second column from Table 5.2, as less updates are needed for the college-msg dataset while more updates are performed for the Facebook-wall one. We also notice the corresponding values for the other four datasets are about 0.5. This is because for steadily growing networks, the probably that the TSN bound is larger than the average TSN bound from the previous window is 0.5. Thus, for networks with fairly steady growth, the fraction of periods at which CT is updated should be around 0.5.

Table 5.2: Summary statistics relative to the evolution of networks

Dataset	TSN bound	Fraction of periods at which CT is updated
email-Enron	31 ± 16	0.5
bitcoin-au	76 ± 27	0.5
college-msg	75 ± 73	0.22
email-Eu-core	106 ± 38	0.45
short-msg	944 ± 493	0.56
Facebook-wall	2,981 ± 2,092	0.78

Figure 5.5 compares the computation time, including building and updating community trees using the Dynamic CPM to the cumulative computation time for orders above 1 using original CPM as measured on a desktop with a 2.2 GHz Intel Xeon processor and 32 GB RAM. It can be seen that our algorithm reduces the computation time significantly on the two largest datasets, short-msg and Facebook-wall, while there is no significant computation time difference between the two methods for the other four datasets.

This observation is further verified by the statistical test in Table 5.3. We further evaluate three quantities that are often used to measure small world effect on these networks. As shown in Table 5.3, the average clustering coefficients of the short-msg and Facebook-wall

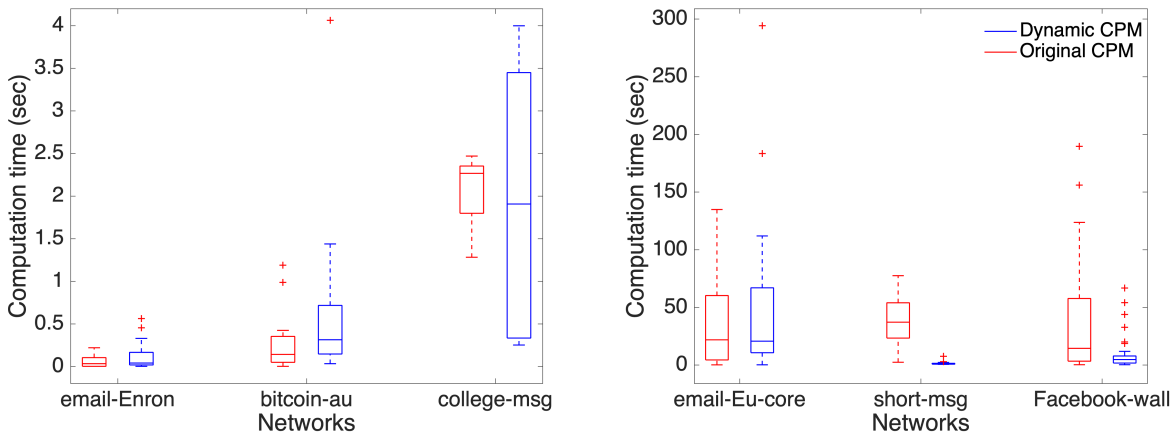


Figure 5.5: Comparison of computation time between the original CPM and our algorithm for six network datasets. The networks are divided into two groups based on their computation time for display purposes only.

datasets are considerately smaller than other datasets, and their average shortest path lengths and effective diameters [85] are noticeable larger than the others. All these values reflect that the communities in short-msg and Facebook-wall networks are less close-knit than the other networks over time. In fact, the short-msg and Facebook-wall networks can be classified as small world networks, as their average shortest path lengths scale with $\ln V_t$. Meanwhile, the other networks are regarded as ultra-small world since they have smaller average shortest path lengths that scale with $\ln \ln V_t$ [5]. The corresponding community trees for small world networks tend to have more branches, especially at lower orders. The highest order of these trees are also much smaller than their counterparts for the ultra-small networks. This structural characteristics implies that the Link and/or Cut operations during the insertion and/or deletion of CG edges are more often performed between splay trees of small sizes, as opposed to the scenario for ultra-small networks where large splay trees are more frequently involved during these operations. This explains why our algorithm shows more strength for small world networks than ultra-small ones.

Table 5.3: Wilcoxon rank-sum test for computation time difference between the original CPM and our algorithm, and measures for the degree of clustering in networks

Dataset	p-value	Average clustering coefficient	Average shortest path length	Effective diameter (90th percentile)
email-Enron	0.39	0.44 ± 0.08	2.49 ± 0.38	3.12 ± 0.56
bitcoin-au	0.16	0.34 ± 0.02	2.82 ± 0.15	3.40 ± 0.23
college-msg	0.84	0.11 ± 0.002	3.05 ± 0.12	3.61 ± 0.15
email-Eu-core	0.61	0.36 ± 0.03	2.89 ± 0.29	3.44 ± 0.46
short-msg	**3.9e-9**	**0.045 ± 0.011**	**9.70 ± 1.66**	**13.2 ± 2.25**
Facebook-wall	**0.02**	**0.097 ± 0.025**	**6.20 ± 1.44**	**7.68 ± 2.05**

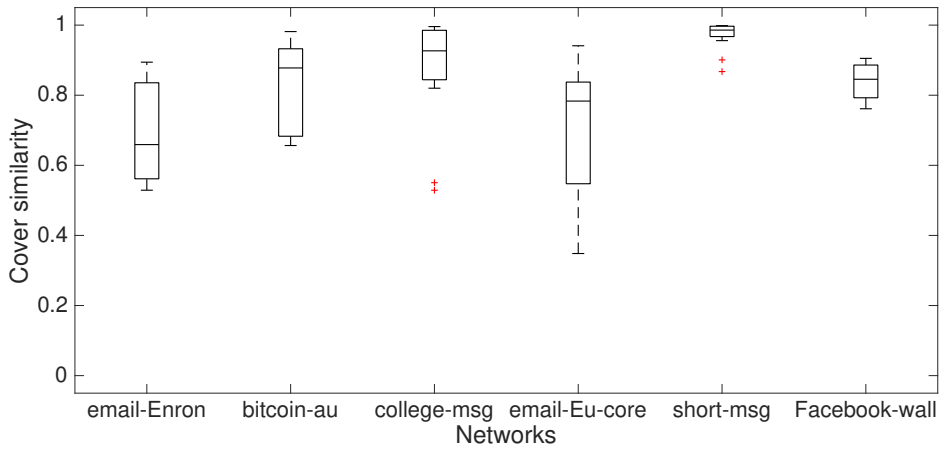


Figure 5.6: Distribution of similarities measure by NMI between a set of 3-communities found in a network up to a particular time and the cover found by the previous update of the network for our datasets.

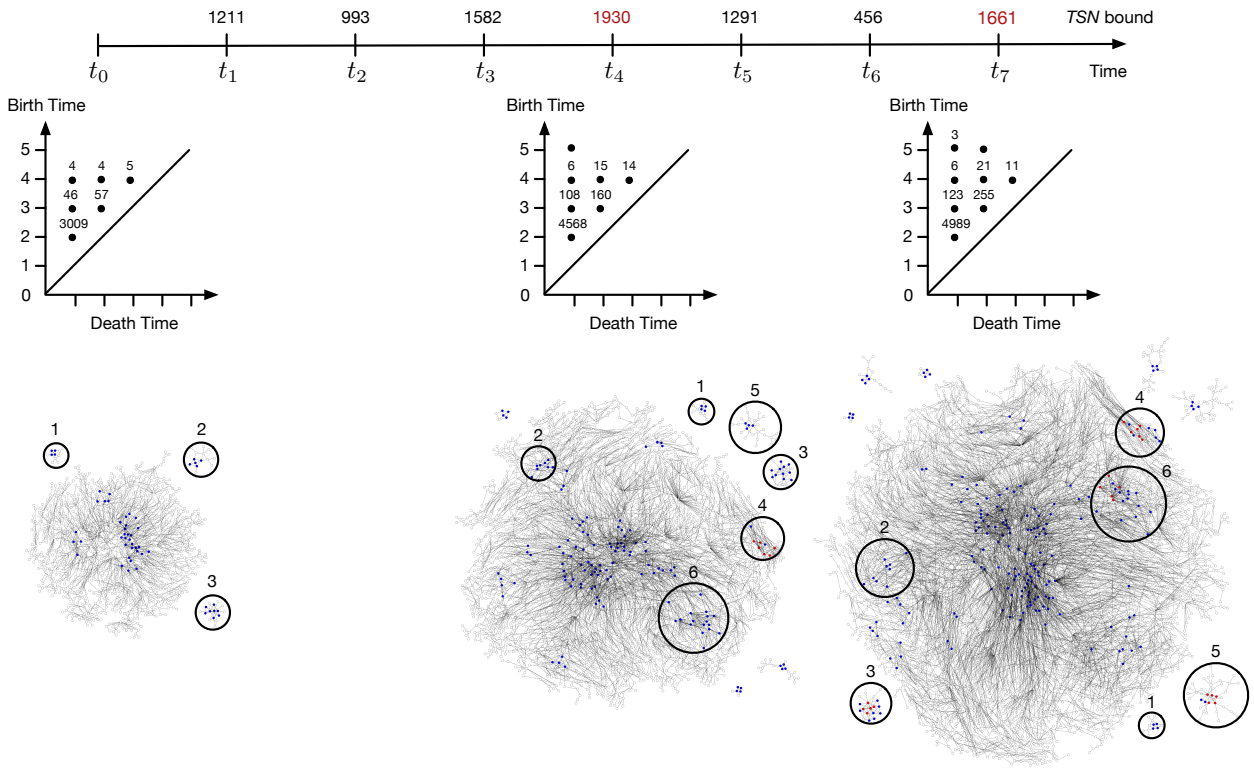


Figure 5.7: Network evolution for the short-msg dataset. The PDs correspond to the community trees at t_0, t_4, t_7, respectively. The number attached above to a point represents the multiplicity of this point in the PDs. The graphs shown at the bottom are a collection of CCs (subgraphs) where the set of 4-communities (blue) and 5-communities (red) belong to.

Figure 5.6 evaluate the similarities between the network cover[4] from an unchanged community tree at order 3 and the real set of 3-communities by t_i for all the datasets. The evaluation metric is chosen to be normalized mutual information (NMI), which is a measure of similarities between the partitions based on information theory. This metric is proven to be reliable for comparing the real communities and the found ones when evaluating community detection algorithms [39]. Here we adopt a widely-used version, proposed by Lancichinetti et al. [84], for comparative analysis of overlapping communities. It can be seen that the mean

[4]A set of communities is often referred to as a cover in the literature

values of the cover similarity for small-world networks are roughly above 0.85, while those vary from around 0.65 to 0.9 among ultra-small world networks.

In Figure 5.7, the short-msg dataset serves an example to demonstrate the evolutionary network analysis using our algorithm. The community tree is initially built at t_0 and updated at t_4 and t_7 base on the rule in our algorithm. The corresponding PDs indicate that the community tree is featured with a large number of lower order branches throughout the duration. The generated community tree by our algorithm is able to track communities over time and identify their appearance and disappearance by their labels. In this example, the highest order of $\mathcal{T}_0$ is 4 and there are four CCs at t_0. From t_0 to t_4, a newborn 5-community CC_4 appears, and it is not evolved from any of the 4-communities at t_0. CC_1 and CC_3 stay isolated from others, but CC_2 has been merged to the central CC at t_4. From t_4 to t_7, three more 5-communities are born, where one of them is CC_3 and the other two, CC_5 and CC_6 are evolved from newly 4-communities at t_4. Interestingly, CC_1 still remains disconnected to any other CCs.

5.6 Discussion

In this chapter, we develop an algorithm called the Dynamic CPM that can efficiently detect communities in large-scale dynamic networks. The experiments show that our algorithm reduces the computation time significantly compared to the original CPM, especially for small world networks. The network cover of 3-communities from unchanged community trees also maintain relatively high similarities to the real covers. Furthermore, since the community tree encodes the evolutionary information of communities, our algorithm can naturally track similar communities over time, which often helps to determine fundamental structures of dynamic networks. In addition, the concise tree structure also records when communities appear, disappear, split or merge, which allows us to identify the occurrence of critical events.

Chapter 6

CONCLUSIONS

This dissertation explores the utilization of TDA tools in extracting key features regarding different types of input data including point clouds, meshes and networks, through three themed chapters. TDA delves directly into the structure of data to avoid information loss or introducing unwanted bias due to overly simplified assumptions from traditional methods. The obtained shape and structural information helps to grasp how data is organized locally and globally, which leads us to look into whether particular subgroups of the data are selectively responsive to different features (process variables) in order to enhance the interpretability of feature attribution in manufacturing systems in Chapter 3. On the other hand, the rich source of information offered by TDA motivates us to build a pipeline towards the incorporation of sparse sampling to extract discriminative features faithfully and efficiently in redundant contexts in Chapter 4. More emphasis has been put on the computational aspects of TDA in a temporal setting in Chapter 5, as we develop a framework for analyzing structure features using the notion of communities in dynamic networks based on the established community tree representation.

6.1 Contribution

Chapter 3 shows that the Mapper algorithm adds a new perspective to the traditional means of feature selection and provide critical insights hidden in the complex data. Through direct visualization, we generate an abstract view of the data to facilitate a better understanding of the casual relationships between the features and manufacturing system outputs. The contributions of the work are summarized below:

- To the best of our knowledge, we successfully demonstrate the value of any TDA

method in the manufacturing systems domain for the first time.

- We effectively detect structural information present in manufacturing systems data, which is highly valuable as it allows identification of subgroups of interest for targeted hypothesis testing with respect to the differences in the observed patterns.
- We demonstrate that just using the identified features with the most significant causal relationships provides a similarly high level of prediction accuracy as achieved with the complete set of features but with substantially reduced training times.

The procedure demonstrated in Chapter 4 can be considered as a specific example of a more general, modular data processing architecture, where extracted topological features are vectorized and a sparse subset are selected for efficient downstream classification. The Sparse-TDA and L1-SVM procedures are both instances in this overarching framework. Compared to L1-SVM, the benefit of the Sparse-TDA procedure is that it allows the feature extraction and sparse sampling techniques to be varied independently, enabling the inclusion of constraints, tailored cost functions, and highly optimized algorithms. As we have shown in this work, the separation of sampling and classification steps ultimately improves classification accuracy while decreasing computation in comparison with multi-scale kernel TDA and leading L1-SVM variants.

The construction and update algorithms for community trees in Chapter 5 can be comfortably adapted to cope with direct and weighted networks by replacing the CPM with its directed and weighted versions defined in [54, 103]. In other words, the work in Chapter 5 provides the core component of a unified framework for analyzing community structures in general dynamic networks based on the community tree representation.

6.2 Anticipated Impact

The results in Chapter 3 open a feasible path for efficient manufacturing process monitoring and control, especially in complex systems with a large number of process variables. The compact visual presentation of the data makes the attribution of the individual process variables to the outputs easily observable. Furthermore, the Mapper algorithm overcomes

the issue that well separated data in the high-dimensional space may overlap in a low-dimensional projection after dimensionality reduction by clustering data in the original high dimensions. This operation will assist to identify subtle activities, e.g., potential anomalies in the manufacturing systems, and allow for effective root cause analysis in the diagnose processes. By combining the Mapper algorithm with existing machine learning techniques, there is also the possibility of developing a practically useful method that is well-suited for analyzing high-dimensional, heterogeneous manufacturing data in general.

The work in Chapter 4 paves the way for online TDA on streaming data. In this context, the first step is to determine whether the addition of new data necessitates the computation of new samples. Each additional set of streaming vectorized topological features can be approximated by the old principal components (PCs) and samples to determine if an SVD-QR update is needed. One way is to check whether the resulting approximation error distribution is significantly larger than the approximation to the old data. If this is the case, then the PCs can be updated using a rank-one incremental update of the old SVD [15]. For updating a pivoted QR factorization, one can leverage a randomized sampling of pivots [47] to accelerate computation, which is also scalable to parallel architectures.

The framework in Chapter 5 is expected to lay the basis for a TDA route to the study of multilayer networks, particularly in fragility analysis. Multilayer networks exhibit a more realistic fashion to characterize a wide variety of real-world networks [73], where intra-layer edges encode different or related types of interactions, and dynamical processes traverse along both intra-layer and inter-layer edges. In fact, a temporal network is a special type of multilayer network where each time instant is mapped into a different layer. Since random failures or targeted attacks tend to cause cascading effects (the failure of one node will recursively provoke the failure of connected nodes) on the network functionality and the magnitude of such effects is directly related to the topology of the network, investigating community-based topological structures and the resilience of multilayer networks will help to improve infrastructural design in relevant applications so as to make them more robust to critical failure modes.

6.3 Future Work

While substantial progress has been made by TDA in a large number of of areas with input data given in the format of mesh surfaces or networks, current TDA methods often encounter computational difficulties in attempts to acquire quantitative information from massive point clouds. This bottleneck of TDA has motivated me to seek solutions from a broader spectrum of methods in geometric learning. Recently, Qi et al. [107] proposed a deep neural network architecture on point clouds, referred to as PointNet, that is able to learn point set features efficiently and robustly. In fact, there has been an increasing interest in the past few years in numerous fields such as computer vision, medical imaging, biochemistry, and network analysis [43, 95, 97] in applying deep neural models to non-Euclidean geometric data due to its computational efficiency, where it has since been termed geometric deep learning (GDL). My principal interest is the capacity of GDL methods to deal with signals defined over a dynamically changing structure or shape. Here, examples of signals include the geometric coordinates of users on each vertex in a social network, and time-dependent gene expression data defined on a regulatory network. The study of these problems can be used for detecting abnormal activities and identifying influential spreaders in evolving networks.

BIBLIOGRAPHY

[1] Henry Adams, Tegan Emerson, Michael Kirby, Rachel Neville, Chris Peterson, Patrick Shipman, Sofya Chepushtanova, Eric Hanson, Francis Motta, and Lori Ziegelmeier. Persistence images: A stable vector representation of persistent homology. *Journal of Machine Learning Research*, 18(1):218–252, 2017.

[2] Mehmet E. Aktas, Esra Akbas, and Ahmed El Fatmaoui. Persistence homology of networks: methods and applications. *Applied Network Science*, 4(1):61, 2019.

[3] Alex Arenas, Albert Diaz-Guilera, and Conrad J. Pérez-Vicente. Synchronization reveals topological scales in complex networks. *Physical Review Letters*, 96(11):114102, 2006.

[4] Reuven Bar-Yehuda and Shimon Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2(2):198–203, 1981.

[5] Albert-László Barabási et al. *Network science*. Cambridge University Press, 2016.

[6] Maxime Barrault, Yvon Maday, Ngoc Cuong Nguyen, and Anthony T. Patera. An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations. *Comptes Rendus Mathematique*, 339(9):667–672, 2004.

[7] Danielle S. Bassett, Nicholas F. Wymbs, Mason A. Porter, Peter J. Mucha, Jean M. Carlson, and Scott T. Grafton. Dynamic reconfiguration of human brain networks during learning. *Proceedings of the National Academy of Sciences*, 108(18):7641–7646, 2011.

[8] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Clear and compress: Computing persistent homology in chunks. In *Topological Methods in Data Analysis and Visualization III*, pages 103–117. Springer, 2014.

[9] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Distributed computation of persistent homology. In *2014 Proceedings of the 16th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 31–38. SIAM, 2014.

[10] Yaov Benjamini and Yosef Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society, Series B*, 57(1):289–300, 1995.

[11] Sebastian Benzekry, Jack A. Tuszynski, Edward A. Rietman, and Giannoula L. Klement. Design principles for cancer therapy guided by changes in complexity of protein-protein interaction networks. *Biology Direct*, 10(1):32, 2015.

[12] Ginestra Bianconi. Interdisciplinary and physics challenges of network theory. *Europhysics Letters*, 111(5):56001, 2015.

[13] Stefano Boccaletti, Ginestra Bianconi, Regino Criado, Charo I Del Genio, Jesús Gómez-Gardenes, Miguel Romance, Irene Sendina-Nadal, Zhen Wang, and Massimiliano Zanin. The structure and dynamics of multilayer networks. *Physics Reports*, 544(1):1–122, 2014.

[14] Thomas Bonis, Maks Ovsjanikov, Steve Oudot, and Frédéric Chazal. Persistence-based pooling for shape pose recognition. In *International Workshop on Computational Topology in Image Context*, pages 19–29, 2016.

[15] Matthew Brand. Incremental singular value decomposition of uncertain data with missing values. In *European Conference on Computer Vision*, pages 707–720. Springer, 2002.

[16] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[17] Bingni W. Brunton, Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. Sparse sensor placement optimization for classification. *SIAM Journal on Applied Mathematics*, 76(5):2099–2122, 2016.

[18] Peter Bubenik. Statistical topological data analysis using persistence landscapes. *Journal of Machine Learning Research*, 16(1):77–102, 2015.

[19] Sergey V. Buldyrev, Roni Parshani, Gerald Paul, Eugene H. Stanley, and Shlomo Havlin. Catastrophic cascade of failures in interdependent networks. *Nature*, 464(7291):1025–1028, 2010.

[20] Peter Businger and Gene H. Golub. Linear least squares solutions by Householder transformations. *Numerische Mathematik*, 7(3):269–276, 1965.

[21] Tony Cai and Xiaodong Li. Robust and computationally feasible community detection in the presence of arbitrary outlier nodes. *Annals of Statistics*, 43(3):1027–1059, 2015.

[22] Andrea Capocci, Vito D. P. Servedio, Guido Caldarelli, and Francesca Colaiori. Detecting communities in large networks. *Physica A: Statistical Mechanics and its Applications*, 352(2):669–676, 2005.

[23] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

[24] Mathieu Carriere, Bertrand Michel, and Steve Oudot. Statistical analysis and parameter selection for mapper. *Journal of Machine Learning Research*, 19(1):478–516, 2018.

[25] Mathieu Carrière and Steve Oudot. Structure and stability of the one-dimensional mapper. *Foundations of Computational Mathematics*, 18(6):1333–1396, 2018.

[26] Mathieu Carrière, Steve Oudot, and Maks Ovsjanikov. Stable topological signatures for points on 3D shapes. In *Proceedings of Eurographics Symposium on Geometry Processing*, pages 1–12, 2015.

[27] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27, 2011.

[28] Saifon Chaturantabut and Danny C. Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing*, 32(5):2737–2764, 2010.

[29] Frédéric Chazal, Leonidas J. Guibas, Steve Oudot, and Primoz Skraba. Persistence-based clustering in Riemannian manifolds. *Journal of the ACM (JACM)*, 60(6):41, 2013.

[30] Frédéric Chazal and Bertrand Michel. An introduction to topological data analysis: fundamental and practical aspects for data scientists. *arXiv preprint arXiv:1710.04019*, 2017.

[31] Chao Chen and Michael Kerber. Persistent homology computation with a twist. In *Proceedings 27th European Workshop on Computational Geometry*. Citeseer, 2011.

[32] Ruqian Chen, Yen-Chi Chen, Wei Guo, and Ashis G. Banerjee. A note on community trees in networks. *arXiv preprint arXiv:1710.03924*, 2017.

[33] Moo K. Chung, Peter Bubenik, and Peter T. Kim. Persistence diagrams of cortical surface data. In *International Conference on Information Processing in Medical Imaging*, pages 386–397, 2009.

[34] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37(1):103–120, 2007.

[35] David Cohen-Steiner, Herbert Edelsbrunner, John Harer, and Yuriy Mileyko. Lipschitz functions have L_p-stable persistence. *Foundations of Computational Mathematics*, 10(2):127–139, 2010.

[36] Justin Curry. Sheaves, cosheaves and applications. *arXiv preprint arXiv:1303.3255*, 2013.

[37] Carina Curto. What can topology tell us about the neural code? *Bulletin of the American Mathematical Society*, 54(1):63–78, 2017.

[38] Carina Curto and Vladimir Itskov. Cell groups reveal structure of stimulus space. *PLoS Computational Biology*, 4(10):e1000205, 2008.

[39] Leon Danon, Albert Diaz-Guilera, Jordi Duch, and Alex Arenas. Comparing community structure identification. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(09):P09008, 2005.

[40] Apurba Das, Michael Svendsen, and Srikanta Tirthapura. Incremental maintenance of maximal cliques in a dynamic graph. *arXiv preprint arXiv:1601.06311*, 2016.

[41] Vin De Silva and Gunnar Carlsson. Topological estimation using witness complexes. *Eurographics Symposium on Point-Based Graphics*, pages 157–166, 2004.

[42] Vin De Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. Persistent cohomology and circular coordinates. *Discrete & Computational Geometry*, 45(4):737–759, 2011.

[43] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[44] Tamal K. Dey, Facundo Mémoli, and Yusu Wang. Multiscale mapper: Topological summarization via codomain covers. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 997–1013. SIAM, 2016.

[45] Clifford H. Dowker. Homology groups of relations. *Annals of mathematics*, pages 84–95, 1952.

[46] Zlatko Drmac and Serkan Gugercin. A new selection operator for the discrete empirical interpolation method—improved a priori error bound and extensions. *SIAM Journal on Scientific Computing*, 38:A631–A648, 2015.

[47] Jed A. Duersch and Ming Gu. Randomized QR with column pivoting. *SIAM Journal on Scientific Computing*, 39(4):C263–C291, 2017.

[48] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. *Discrete & Computational Geometry*, 28(4):511–533, 2002.

[49] Steven P. Ellis and Arno Klein. Describing high-order statistical dependence using concurrence topology, with application to functional mri brain data. *Homology, Homotopy and Applications*, 16(1):245–264, 2014.

[50] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[51] Tim S. Evans. Clique graphs and overlapping communities. *Journal of Statistical Mechanics: Theory and Experiment*, 2010(12):P12037, 2010.

[52] Fazel Famili, W-M Shen, Richard Weber, and Evangelos Simoudis. Data pre-processing and intelligent data analysis. *International Journal on Intelligent Data Analysis*, 1(1), 1997.

[53] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[54] Illés Farkas, Dániel Ábel, Gergely Palla, and Tamás Vicsek. Weighted network modules. *New Journal of Physics*, 9(6):180, 2007.

[55] Santo Fortunato and Darko Hric. Community detection in networks: A user guide. *Physics Reports*, 659:1–44, 2016.

[56] Marcio Gameiro, Yasuaki Hiraoka, Shunsuke Izumi, Miroslav Kramar, Konstantin Mischaikow, and Vidit Nanda. A topological measurement of protein compressibility. *Japan Journal of Industrial and Applied Mathematics*, 32(1):1–17, 2015.

[57] Mingchen Gao, Chao Chen, Shaoting Zhang, Zhen Qian, Dimitris Metaxas, and Leon Axel. Segmenting the papillary muscles and the trabeculae from high resolution cardiac CT through restoration of topological handles. In *International Conference on Information Processing in Medical Imaging*, pages 184–195. Springer, 2013.

[58] Matan Gavish and David L. Donoho. The optimal hard threshold for singular values is $4/\sqrt{3}$. *IEEE Transactions on Information Theory*, 60(8):5040–5053, Aug. 2014.

[59] Robert Ghrist. Barcodes: The persistent topology of data. *Bulletin of the American Mathematical Society*, 45(1):61–75, 2008.

[60] Cedric E. Ginestet, Thomas E. Nichols, Ed T. Bullmore, and Andrew Simmons. Brain network analysis: separating cost from topology using cost-integration. *PloS One*, 6(7):e21570, 2011.

[61] Michelle Girvan and Mark E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[62] Chad Giusti, Robert Ghrist, and Danielle S. Bassett. Two's company, three (or more) is a simplex. *Journal of computational neuroscience*, 41(1):1–14, 2016.

[63] Chad Giusti, Eva Pastalkova, Carina Curto, and Vladimir Itskov. Clique topology reveals intrinsic geometric structure in neural correlations. *Proceedings of the National Academy of Sciences*, 112(44):13455–13460, 2015.

[64] Wei Guo and Ashis G. Banerjee. Toward automated prediction of manufacturing productivity based on feature selection using topological data analysis. In *Proceedings of IEEE International Symposium on Assembly and Manufacturing*, pages 31–36, 2016.

[65] Zhenhua Guo, Lei Zhang, and David Zhang. A completed modeling of local binary pattern operator for texture classification. *IEEE Transactions on Image Processing*, 19(6):1657–1663, 2010.

[66] Tanja Hartmann, Andrea Kappes, and Dorothea Wagner. Clustering evolving networks. In *Algorithm Engineering*, pages 280–329. Springer, 2016.

[67] Q. Peter He and Jin Wang. Fault detection using the k-nearest neighbor rule for semiconductor manufacturing processes. *IEEE Transactions on Semiconductor Manufacturing*, 20(4):345–354, 2007.

[68] Q. Peter He and Jin Wang. Large-scale semiconductor process fault detection using a fast pattern recognition-based method. *IEEE Transactions on Semiconductor Manufacturing*, 23(2):194–200, 2010.

[69] Monika Rauch Henzinger and Valerie King. Randomized dynamic graph algorithms with polylogarithmic time per operation. In *Proceedings of the 27th annual ACM symposium on Theory of computing*, pages 519–527. ACM, 1995.

[70] Yasuaki Hiraoka, Takenobu Nakamura, Akihiko Hirata, Emerson G Escolar, Kaname Matsue, and Yasumasa Nishiura. Hierarchical structures of amorphous solids characterized by persistent homology. *Proceedings of the National Academy of Sciences*, 113(26):7035–7040, 2016.

[71] Thomas Hofmann, B Schölkopf, and Alexander J. Smola. Kernel methods in machine learning. *Annals of Statistics*, pages 1171–1220, 2008.

[72] Danijela Horak, Slobodan Maletić, and Milan Rajković. Persistent homology of complex networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(03):P03034, 2009.

[73] Lucas G. Jeub, Michael W. Mahoney, Peter J. Mucha, and Mason A. Porter. A local perspective on community structure in multilayer networks. *Network Science*, pages 1–20, 2017.

[74] Jakob Jonsson. *Simplicial complexes of graphs*, volume 3. Springer, 2008.

[75] Siddharth Joshi and Stephen Boyd. Sensor selection via convex optimization. *IEEE Transactions on Signal Processing*, 57(2):451–462, 2009.

[76] Brian Karrer, Elizaveta Levina, and Mark E. J. Newman. Robustness of community structure in networks. *Physical Review E*, 77(4):046119, 2008.

[77] Arshi Khalid, Byung Sun Kim, Moo K. Chung, Jong Chul Ye, and Daejong Jeon. Tracing the evolution of multi-scale functional networks in a mouse model of depression using persistent brain network homology. *NeuroImage*, 101:351–363, 2014.

[78] Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In *European Conference on Machine Learning*, pages 217–226. Springer, 2004.

[79] Dániel Kondor, Márton Pósfai, István Csabai, and Gábor Vattay. Do the rich get richer? an empirical analysis of the bitcoin transaction network. *PloS One*, 9(2):e86197, 2014.

[80] Dimitry Kozlov. *Combinatorial algebraic topology*, volume 21. Springer Science & Business Media, 2007.

[81] Max Kuhn and Kjell Johnson. *Applied predictive modeling*. Springer, 2013.

[82] Genki Kusano, Yasuaki Hiraoka, and Kenji Fukumizu. Persistence weighted Gaussian kernel for topological data analysis. In *Proceedings of International Conference on Machine Learning*, 2016.

[83] Roland Kwitt, Stefan Huber, Marc Niethammer, Weili Lin, and Ulrich Bauer. Statistical topological data analysis - a kernel perspective. In *Advances in Neural Information Processing Systems*, pages 3070–3078, 2015.

[84] Andrea Lancichinetti, Santo Fortunato, and Janos Kertesz. Detecting the overlapping and hierarchical community structure in complex networks. *New journal of physics*, 11(3):033015, 2009.

[85] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007.

[86] Michael Lesnick. Studying the shape of data using topology. *Institute for Advanced Studies (IAS) letter*, 2013.

[87] Chunyuan Li, Maks Ovsjanikov, and Frédéric Chazal. Persistence-based structural recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1995–2002, 2014.

[88] Yuan Li and Xinmin Zhang. Diffusion maps based k-nearest-neighbor rule technique for semiconductor manufacturing process fault detection. *Chemometrics and Intelligent Laboratory Systems*, 136:47–57, 2014.

[89] Pek Y Lum, Gurjeet Singh, Alan Lehman, Tigran Ishkanov, Mikael Vejdemo-Johansson, Muthu Alagappan, John Carlsson, and Gunnar Carlsson. Extracting insights from the shape of complex data using topology. *Scientific Reports*, 3, 2013.

[90] Laurens Van Der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[91] John F. MacGregor and Theodora Kourti. Statistical process control of multivariate processes. *Control Engineering Practice*, 3(3):403–414, 1995.

[92] Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Learning communities in the presence of errors. *arXiv preprint arXiv:1511.03229*, 2015.

[93] Krithika Manohar, Bingni W. Brunton, J. Nathan Kutz, and Steven L. Brunton. Data-driven sparse sensor placement for reconstruction: Demonstrating the benefits of exploiting known patterns. *IEEE Control Systems Magazine*, 38(3):63–86, 2018.

[94] Krithika Manohar, Steven L. Brunton, and J. Nathan Kutz. Environment identification in flight using sparse approximation of wing strain. *Journal of Fluids and Structures*, 70:162–180, 2017.

[95] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on Riemannian manifolds. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 37–45, 2015.

[96] John Milnor. *Morse theory*. Princeton University Press, 1963.

[97] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. *arXiv preprint arXiv:1611.08402*, 2016.

[98] Mark E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69(6):066133, 2004.

[99] Monica Nicolau, Arnold J. Levine, and Gunnar Carlsson. Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. *Proceedings of the National Academy of Sciences*, 108(17):7265–7270, 2011.

[100] Timo Ojala, Topi Mäenpää, Matti Pietikäinen, Jaakko Viertola, Juha Kyllönen, and Sami Huovinen. OuTeX - New framework for empirical evaluation of texture analysis algorithms. In *Proceedings of the 16th International Conference on Pattern Recognition*, pages 701–706, 2002.

[101] Deepti Pachauri, Chris Hinrichs, Moo K. Chung, Sterling C. Johnson, and Vikas Singh. Topology-based kernels with application to inference problems in Alzheimer's disease. *IEEE Transactions on Medical Imaging*, 30(10):1760–1770, 2011.

[102] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005.

[103] Gergely Palla, Illes J. Farkas, Peter Pollner, Imre Derényi, and Tamás Vicsek. Directed network modules. *New Journal of Physics*, 9(6):186, 2007.

[104] Pietro Panzarasa, Tore Opsahl, and Kathleen M. Carley. Patterns and dynamics of users' behavior and interaction: Network analysis of an online community. *Journal of the American Society for Information Science and Technology*, 60(5):911–932, 2009.

[105] Ashwin Paranjape, Austin R. Benson, and Jure Leskovec. Motifs in temporal networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 601–610. ACM, 2017.

[106] David Pickup et al. SHREC'14 track: Shape retrieval of non-rigid 3D human models. In *Proceedings of Eurographics Workshop on 3D Object Retrieval*, pages 101–110, 2014.

[107] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

[108] Jörg Reichardt and Stefan Bornholdt. Detecting fuzzy community structures in complex networks with a Potts model. *Physical Review Letters*, 93(21):218701, 2004.

[109] Jan Reininghaus, Stefan Huber, Ulrich Bauer, and Roland Kwitt. A stable multi-scale kernel for topological machine learning. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 4741–4748, 2015.

[110] Bastian Rieck, Ulderico Fugacci, Jonas Lukasczyk, and Heike Leitte. Clique community persistence: A topological visual analysis approach for complex networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):822–831, 2017.

[111] Roman Rosipal and Leonard J. Trejo. Kernel partial least squares regression in reproducing kernel Hilbert space. *Journal of Machine Learning Research*, 2:97–123, 2001.

[112] Joseph J. Rotman. *An introduction to algebraic topology*. Springer Science & Business Media, 2013.

[113] Syuzanna Sargsyan, Steven L. Brunton, and J. Nathan Kutz. Nonlinear model reduction for dynamical systems using sparse sensor locations from learned libraries. *Physical Review E*, 92(3):033304, 2015.

[114] Ghanashyam Sarikonda et al. CD8 T-cell reactivity to islet antigens is unique to type 1 while CD4 T-cell reactivity exists in both type 1 and type 2 diabetes. *Journal of Autoimmunity*, 50:77–82, 2014.

[115] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[116] Gurjeet Singh, Facundo Mémoli, and Gunnar Carlsson. Topological methods for the analysis of high dimensional data sets and 3D object recognition. In *Eurographics Symposium on Point-Based Graphics*, pages 91–100, 2007.

[117] Primoz Skraba, Maks Ovsjanikov, Frédéric Chazal, and Leonidas Guibas. Persistence-based segmentation of deformable shapes. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 45–52, 2010.

[118] Daniel D. Sleator and Robert E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM (JACM)*, 32(3):652–686, 1985.

[119] Alvise Sommariva and Marco Vianello. Computing approximate Fekete points by QR factorizations of Vandermonde matrices. *Computers & Mathematics with Applications*, 57(8):1324–1336, 2009.

[120] Chao-Ton Su, Taho Yang, and Chir-Mour Ke. A neural-network approach for semiconductor wafer post-sawing inspection. *IEEE Transactions on Semiconductor Manufacturing*, 15(2):260–266, 2002.

[121] Jian Sun, Maks Ovsjanikov, and Leonidas Guibas. A concise and provably informative multi-scale signature based on heat diffusion. In *Proceedings of the Eurographics Symposium on Geometry Processing*, pages 1383–1392, 2009.

[122] Robert E. Tarjan. Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm. *Mathematical Programming*, 78(2):169–177, 1997.

[123] Dane Taylor, Florian Klimm, Heather A. Harrington, Miroslav Kramár, Konstantin Mischaikow, Mason A. Porter, and Peter J. Mucha. Topological data analysis of contagion maps for examining spreading processes on networks. *Nature Communications*, 6:7723, 2015.

[124] Jiri Tlusty. *Manufacturing processes and equipment*. Prentice Hall, 2000.

[125] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363:28–42, 2006.

[126] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. On the evolution of user interaction in facebook. In *Proceedings of the 2nd ACM workshop on Online social networks*, pages 37–42. ACM, 2009.

[127] Hubert Wagner, Chao Chen, and Erald Vuçini. Efficient computation of persistent homology for cubical data. In *Topological Methods in Data Analysis and Visualization II*, pages 91–106. Springer, 2012.

[128] Barry M. Wise, Neal B. Gallagher, Stephanie W. Butler, Daniel D. White, and Gabriel G. Barna. A comparison of principal component analysis, multiway principal component analysis, trilinear decomposition and parallel factor analysis for fault detection in a semiconductor etch process. *Journal of Chemometrics*, 13(3-4):379–396, 1999.

[129] Ye Wu, Changsong Zhou, Jinghua Xiao, Jürgen Kurths, and Hans J. Schellnhuber. Evidence for a bimodal distribution in human communication. *Proceedings of the National Academy of Sciences*, 107(44):18803–18808, 2010.

[130] Yanyan Xu, James Cheng, and Ada Wai-Chee Fu. Distributed maximal clique computation and management. *IEEE Transactions on Services Computing*, 9:110–122, 2016.

[131] Yuan Yao, Jian Sun, Xuhui Huang, Gregory R. Bowman, Gurjeet Singh, Michael Lesnick, Leonidas J. Guibas, Vijay S. Pande, and Gunnar Carlsson. Topological methods for exploring low-density states in biomolecular folding pathways. *Journal of Chemical Physics*, 130(14):144115, 2009.

[132] Daniel Yekutieli and Yaov Benjamini. Disovering the false discovery rate. *Journal of Statistical Planning and Inference*, 82(1-2):171–196, 1999.

[133] Haijun Zhou. Distance, dissimilarity index, and network community structure. *Physical Review E*, 67(6):061901, 2003.

[134] Zhe Zhou, Chenglin Wen, and Chunjie Yang. Fault detection using random projections and k-nearest neighbor rule for semiconductor manufacturing processes. *IEEE Transactions on Semiconductor Manufacturing*, 28(1):70–79, 2015.

[135] Afra Zomorodian. Topological data analysis. *Advances in Applied and Computational Topology*, 70:1–39, 2012.

[136] Afra Zomorodian and Gunnar Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, 2005.

Appendix A

SUPPLEMENTARY MATERIALS FOR EFFICIENT COMMUNITY DETECTION IN LARGE-SCALE DYNAMIC NETWORKS USING TDA

A.1 Design of Data Structure

We assign each vertex v with a unique ID, denoted by $v.id$, and sort each MC in an increasing order by the vertex ID. Let $T(v)$ be a **trie** that stores the set of MCs of G starting with root v. That is, for any MC $M \in T(v)$, $v.id = \min\{u.id : u \in M\}$. Each MC, represented by a root-to-leaf path, is also affiliated with a unique ID in an increasing order and stored at the leaf node. Figure A.1 provides an example of tries rooted at several vertices of a graph. This form of tree is a space-optimized data structure. It only stores the vertices in the shared root-to-node subpath between the MCs once, and, therefore, uses less space than a hash table with the nodes as keys and MCs as values. It does not have to be binary. It also supports search, insertion, and deletion operations in time of the order of the number of elements k in the operation set. Specifically, the functions used in the following algorithms relevant to a trie include:

- INITIALIZE-TRIE(v, m): Initialize a trie $T(v)$ with ID m
- TRIE-ADD$(T(v), M, m)$: Add the MC M with ID m to $T(v)$
- TRIE-REMOVE$(T(v), M)$: Remove the MC M from $T(v)$
- TRIE-GET-ID$(T(v), M)$: Obtain the ID of M from $T(v)$

To support efficient querying, we record the following information for each node x in $T(v)$:

- *key*: The ID of vertex u that node x represents.
- *value*: The value is initiated as -1. If x is a leaf node, then it stores the ID of the MC.

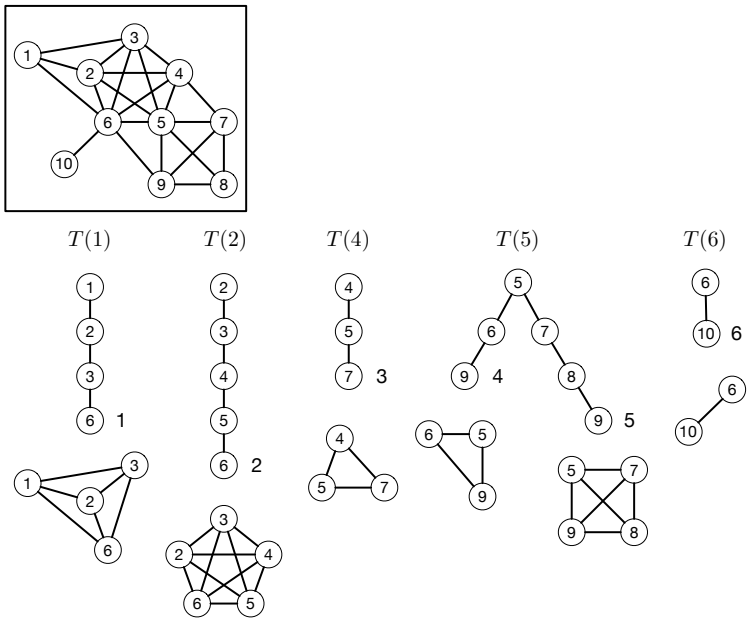


Figure A.1: Tries for the network shown in Figure 5.2. For $i = 1, 2, \ldots, 6$, each root-to-leaf path in $T(i)$ represents an MC whose lowest labeled vertex is i. For example, there are two MCs containing vertex 5, where all the other vertices have labels greater than 5. One tree path is an MC of size 3 containing vertices $5, 6, 9$. The other path is an MC of size 4 comprising vertices $5, 7, 8, 9$. Each MC's ID is stored at the leaf node of its trie.

- *parent*: The parent node of x.
- *child*: A hash table that stores the children nodes of x.
- *next*: The next node of x in the linked list stored in *nodeList*.

We also define the following data fields of $T(v)$:

- *root*: The root node of $T(v)$.
- *nodeList*: A hash table that maps the ID of a certain vertex u to a linked list of nodes in $T(v)$ that represent u in each entry; this list is required as the same vertex x may appear multiple times in different branches of $T(v)$ [130].

To manage the collections of $T(v)$ for all the vertices $v \in V(G)$, we create a hashtable $\mathcal{T}$ with each $v.id$ as the key. The three operations on the hash table are listed below:

- HT-GET($\mathfrak{T}, v.id$): Obtain $T(v)$ from $\mathfrak{T}$ through $v.id$
- HT-ADD($\mathfrak{T}, (v.id, T(v))$): Add $(v.id, T(v))$ to $\mathfrak{T}$
- HT-DELETE($\mathfrak{T}, v.id$): Delete $(v.id, T(v))$ from $\mathfrak{T}$ through $v.id$
- HT-ISKEY($\mathfrak{T}, v.id$): True if $v.id$ is a key in $\mathfrak{T}$

The set of MCs of G is computed using the pivoting-based depth-first search algorithm presented in [125], which has been shown to be worst-case optimal with a runtime of $O(3^{n/3})$ for an n-vertex graph. Table A.1 summarizes the symbols used in this Appendix.

Table A.1: Summary of symbols and descriptions

Symbol	Description
M	Maximal clique (MC)
E_{ab}	ET tree node formed of CG nodes a and b
E_M	ET tree node corresponding to M
E_R	Root node of an ET tree
$\mathcal{G}_i$	Weighted CG at level i
$L(v)$	Set of the vertices adjacent to v whose IDs are smaller than v's ID
m	ID of M
$\mathcal{M}(G)$	Set of MCs in the graph G
$\omega(G)$	Maximum size of MCs in G
M	MC object based on M
$\mathfrak{M}$	Hash table with the (key, value) pair as (m, M) for each entry
n_M	Array of CG nodes with each element representing an M in $\mathcal{G}_i$
P	PD represented by an array of birth and death time pairs of communities $\mathcal{C}$
$T(v)$	Trie that stores the set of MCs starting with root v
$\mathfrak{T}$	Hash table with the (key, value) pair to be $(v.id, T(v))$, $v \in V$ for each entry
$\mathcal{T}$	Community tree of G represented by an array of size $\omega(G)$

A.2 Initial Community Tree Construction Algorithm

It has been explained in Section 5.3 that building a network G_0's community tree amounts to generating its corresponding spanning forests and ET trees at each level. We divide the generation process into two steps, as is given in Algorithm 5 and illustrated in Figure A.2. Note that all the operations related to spanning trees include the operations involved with ET trees if necessary.

Specifically, the first step is to initialize the sequence of the CGs (lines 3-15 in Algorithm 5), i.e., place each existing MC of size $|M| \geq 2$ as a CG node to each associated CG. Thus, for each $M \in \mathcal{M}(G_0)$ with $|M| \geq 2$, we initialize a MC object M given its ID m and its size $|M|$ first, and M becomes the initial representative MC of the single-MC component. As aforementioned in Section 5.3, the birth and death time of a CC is recorded as those of its representative MC. Hence, each M in Algorithm 6 possesses the following data fields: *id*, *visited*, *birthT*, *deathT*, *child*, and holds a pointer $\mathsf{M}.pCG$ to an array of CG nodes. In our pseudo-code, any variable that represents an array or object is treated as a *pointer* to the data representing the array or object. For each CG node $\mathsf{M}.pCG[i]$, $i = |M| - 1, \ldots, 1$, its attributes include *id*, *size* and *adjTrEdges* that represent the MC's ID and size, as well as

Algorithm 4 Build the initial $\mathcal{T}$ for a given unweighted, undirected graph G_0

1: **procedure** BUILD-CT(G_0)
2: $\mathcal{T} \leftarrow \emptyset$, $\mathcal{M} \leftarrow \emptyset$, $\mathcal{T}' \leftarrow \emptyset$, $m_0 \leftarrow 0$
3: $\mathcal{M}(G_0) \leftarrow$ LIST-MCS(G_0) ▷ Algorithm in [125]
4: $\omega(G_0) \leftarrow \max_{M \in \mathcal{M}(G_0)} |M|$
5: Initialize an empty array of size $\omega(G_0)$ as $\mathcal{T}$
6: $\mathcal{T}, \mathcal{M}, \mathcal{T}, m \leftarrow$ ADD-MCS($G_0 \mathcal{M}(G_0), \mathcal{T}, \mathcal{M}, \mathcal{T}, \mathcal{T}', m_0$) ▷ Algorithm 5
▷ Extract the birth and death time for nodes in the community tree
7: $\mathcal{M}, P \leftarrow$ GENERATE-PD($\mathcal{M}, \mathcal{T}$) ▷ Algorithm 11
8: **return** $\mathcal{T}, \mathcal{M}, \mathcal{T}, P, m$

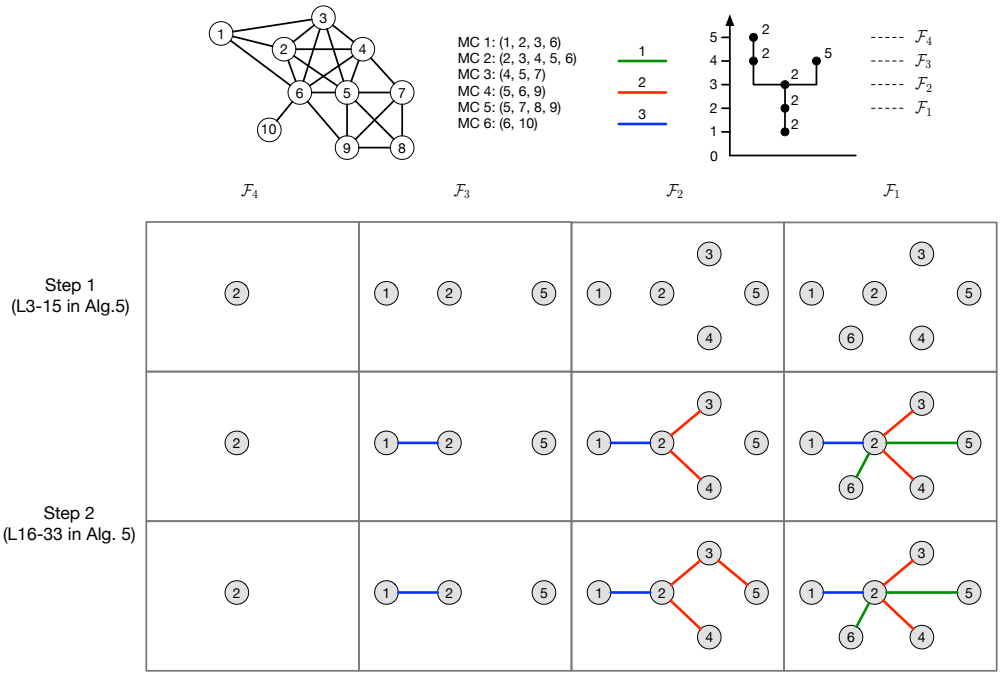


Figure A.2: An example of building a community tree. To construct a community tree, each existing MC of size $|M| \geq 2$ is first laid as a single-node spanning tree in $\mathcal{F}_{|M|-1}, \ldots, \mathcal{F}_1$, respectively (top row). Starting with CG node 2 arbitrarily, we then build connections between node 2 and its neighbors, i.e., nodes 1, 3, 4, 5 and 6. For each unvisited neighbor, we insert a tree edge with weight $w_{e_\mathcal{G}}$ in $\mathcal{F}_i$, $i \leq w_{e_\mathcal{G}}$ between the two nodes if they are from different spanning trees (middle row). Node 2 is marked as a visited one after all the connections. We repeat this process for node 3, and thus, add a tree edge to node 5 in $\mathcal{F}_2$ (bottom row). No more connections are made after we go through the other nodes.

the adjacency lists of the tree edges, respectively. Furthermore, each CG node also has a pointer to its ET node in the corresponding ET tree. For each ET node E_{ab}, in turn, formed of two CG nodes a and b, the node value affiliated with E_{ab} is:

- *dRepID*: Defined by Equation (5.1), where $w(x)$ is the ID of the representative ET node of the ET tree that E_{ab} belongs to if E_{ab} is a loop node, and $w(x)$ is assigned to be 0 if E_{ab} is a nonloop node. Thus, for a single-node ET tree, $E_{ab}.dRepID = a.id$ if

Algorithm 5 Add each MC in $\mathcal{M}^{new}$ with its corresponding CG nodes and ET nodes

1: **procedure** ADD-MCS($G, \mathcal{M}^{new}, \mathfrak{T}, \mathfrak{M}, \mathcal{T}, \mathcal{T}', m_0$)
2: $m \leftarrow m_0$
▷ Initialize the CGs at each order
3: **for each** $M \in \mathcal{M}^{new}$ **do**
4: $m \leftarrow m + 1$
5: $v^* \leftarrow \text{argmin}_{v \in M}\, v.id$
6: $T(v^*) \leftarrow$ HT-GET($\mathfrak{T}, v^*.id$)
7: **if** $T(v^*) =$ NULL **then**
8: $T(v^*) \leftarrow$ INITIALIZE-TRIE(v^*, m)
9: $\mathfrak{T} \leftarrow$ HT-ADD($\mathfrak{T}, (v^*.id, T(v^*))$)
10: $T(v^*) \leftarrow$ TRIE-ADD($T(v^*), M, m$)
11: **if** $|M| \geq 2$ **then**
12: $\mathsf{M} \leftarrow$ INITIALIZE-MC($|M|, m$) ▷ Algorithm 6
13: $\mathfrak{M} \leftarrow$ HT-ADD($\mathfrak{M}, (m, \mathsf{M})$)
14: **for** $i = |M|, \ldots, 2$ **do**
15: $\mathcal{T}[i] \leftarrow$ HT-ADD($\mathcal{T}[i], (m, \mathsf{M}.pCG[i-1].pET)$)
▷ Generate spanning forests by connecting the nodes based on their edge weights
16: **for each** $M \in \mathcal{M}^{new}$ **do**
17: $v^* \leftarrow \text{argmin}_{v \in M}\, v.id$
18: $T(v^*) \leftarrow$ HT-GET($\mathfrak{T}, v^*.id$)
19: $m \leftarrow$ TRIE-GET-ID($T(v^*), M$)
20: $\mathsf{M} \leftarrow$ HT-GET($\mathfrak{M}, m$)
21: $\mathcal{N}(M) \leftarrow$ GET-NEIGHBOR-MCS($G, M, m, \mathfrak{T}$) ▷ Algorithm 7
22: **if** $\mathcal{N}(M) \neq \emptyset$ **then**
23: **for each** $(m', M') \in \mathcal{N}(M)$ **do**
24: $\mathsf{M}' \leftarrow$ HT-GET($\mathfrak{M}, m'$)

```
25:             if M′.visited = TRUE and m′ > m_0 then
26:                 continue
27:             else
28:                 if min(|M|, |M′|) = 2 then
29:                     T ← INSERT-CG-EDGES(M.pCG, M′.pCG, T, T′, 1)  ▷ Algorithm 9
30:                 else
31:                     w_eG ← INTERSECT(M, M′)
32:                     T ← INSERT-CG-EDGES(M.pCG, M′.pCG, T, T′, w_eG)
33:         M.visited = TRUE
34:     return ℑ, ℳ, T, m
```

E_{ab} is a loop node, while $E_{ab}.dRepID = 0$ if E_{ab} is a nonloop one.
Apart from these node values, E_{ab} also stores three pointers to its parent, right and left child. To gain direct access to each M, we create a hash table $\mathcal{M}$ in which the (key, value) pair for each entry is (m, M). For a similar reason, we let $\mathcal{T}$ be an array of size $\omega(G_0)$ representing the community tree in which each element $\mathcal{T}[i]$ is a hash table with entries $(m, \mathsf{M}.pCG[i-1].pET)$ for each representative MC M at order i for $i \geq 2$ (lines 14-15 in Algorithm 5). $\mathcal{T}[i]$ gets updated after a tree edge is added in $\mathcal{F}_{i-1}$.

The second step is to generate each $\mathcal{F}_i$ by connecting CG nodes based on their edge weights $w_{e_\mathcal{G}}$ (lines 16-33 in Algorithm 5). To avoid the expensive intersection tests against every other MC for a given $M \in \mathcal{M}(G_0)$ with $|M| \geq 2$, we first identify a set that consists of pairs of a neighboring MC M' and its ID m', denoted by $\mathcal{N}(M)$ (Algorithm 7). Here, a 'neighboring' MC of M is an MC that shares at least one vertex with M. We insert a tree edge between $\mathsf{M}.pCG[w_{e_\mathcal{G}}]$ and $\mathsf{M}'.pCG[w_{e_\mathcal{G}}]$ in $\mathcal{F}_{w_{e_\mathcal{G}}}$ as long as $\mathsf{M}.pCG[w_{e_\mathcal{G}}]$ and $\mathsf{M}'.pCG[w_{e_\mathcal{G}}]$ are not reachable, and repeat the insertion procedure in $\mathcal{F}_i, i < w_{e_\mathcal{G}}$, until $\mathsf{M}.pCG[i]$ and $\mathsf{M}'.pCG[i]$ are found in the same spanning tree. Since the maximum possible number of vertices shared by M and M' is $\min(|M|, |M'|) - 1$, we have $1 \leq w_{e_\mathcal{G}} \leq \min(|M|, |M'|) - 1$.

Algorithm 6 Initialize an MC M given its ID m and its size $|M|$

procedure INITIALIZE-MC($|M|, m$)
    $\mathsf{M}.id \leftarrow m$
    $\mathsf{M}.visited \leftarrow$ FALSE
    $\mathsf{M}.birthT \leftarrow |M|$
    $\mathsf{M}.deathT \leftarrow -1$
    $\mathsf{M}.child \leftarrow \emptyset$
    $\mathsf{M}.pCG \leftarrow$ SET-CG-NODE($|M|, m$)
    return M

function SET-CG-NODE($|M|, m$) ▷ Initialize a CG node in $\mathcal{G}_{|M|-1}, \ldots, \mathcal{G}_1$, respectively
    Initialize an array of size $|M| - 1$ as n_M
    for $i = 1, \ldots, |M| - 1$ **do**
        $n_M[i].id \leftarrow m$ ▷ $n_M[i]$ represents the CG node in $\mathcal{G}_i$
        $n_M[i].size \leftarrow |M|$
        $n_M[i].adjTrEdges \leftarrow \emptyset$ ▷ Adjacency list of tree edges
        $n_M[i].pET \leftarrow$ NULL
    for $i = 1, \ldots, |M| - 1$ **do**
        $n_M[i].pET \leftarrow$ SET-ET-NODE($n_M[i], n_M[i]$) ▷ Point to its ET node
    return n_M

function SET-ET-NODE(a, b) ▷ Create an ET node with CG nodes a and b
    $E_{ab}.CG_1 \leftarrow a$, $E_{ab}.CG_2 \leftarrow b$
    $E_{ab}.parent \leftarrow$ NULL, $E_{ab}.right \leftarrow$ NULL, $E_{ab}.left \leftarrow$ NULL
    if $a.id = b.id$ **then** ▷ If the ET node corresponds to a node in CG
        $E_{ab}.dRepID \leftarrow a.id$
    else ▷ If the ET node corresponds to a tree edge in CG
        $E_{ab}.dRepID \leftarrow 0$
    return E_{ab}

Algorithm 7 Find the neighboring MCs given a MC M and its ID m

1: **procedure** GET-NEIGHBOR-MCS($G, M, m, \mathcal{T}$)
2: $\{v_1, \ldots, v_{|M|}\} \leftarrow M$, $\mathcal{N}(M) \leftarrow \emptyset$
▷ Output all the MCs except M in $T(u)$ for $u \in M$
3: **for** $j = 1, \ldots, |M|$ **do**
4: $T(v_j) \leftarrow$ HT-GET($\mathcal{T}, v_j.id$)
5: $\mathcal{N}(v_j) \leftarrow$ OUTPUT-MC($T(v_j).root, T(v_j)$) ▷ Algorithm 8
6: $\mathcal{N}(M) \leftarrow \mathcal{N}(M) \cup \mathcal{N}(v_j)$
7: $\mathcal{N}(M) \leftarrow$ HT-DELETE($\mathcal{N}(M), m$)
▷ Output all the MCs in $T(u)$ for $u \in \tilde{L}(v_j)$ that contains v_j, $j = 1, \ldots, |M|$
8: **for** $j = 1, \ldots, |M|$ **do**
9: $L(v_j) \leftarrow$ GET-LOWER-NEIGHBORS($G, v_j.id$)
10: $\tilde{L}(v_j) \leftarrow L(v_j) \backslash \{v_i : v_i \in M, v_i.id < v_j.id\}$
11: **for each** $u \in \tilde{L}(v_j)$ **do**
12: $T(u) \leftarrow$ HT-GET($\mathcal{T}, u.id$)
13: $x \leftarrow$ HT-GET($T(u).nodeList, v_j.id$)
14: **while** $x \neq$ NULL **do**
15: $\mathcal{N}(x) \leftarrow$ OUTPUT-MC($x, T(u)$)
16: $\mathcal{N}(M) \leftarrow \mathcal{N}(M) \cup \mathcal{N}(x)$
17: $x \leftarrow x.next$
18: **return** $\mathcal{N}(M)$

Therefore, the intersection tests are only performed between M and its neighboring MCs when $\min(|M|, |M'|) > 2$. For the special case when $\min(|M|, |M'|) = 2$, $w_{e_{\mathcal{G}}}$ is 1; hence, no intersection test is required. All the MCs are initially *unvisited*, and one is marked as *visited* once it finishes connections with all its unvisited neighboring MCs to avert duplicate calculations.

Finally, we record the death time of the representative MCs from the updated $\mathcal{T}$ and the

Algorithm 8 Output all root-to-leaf paths that contain a node x in $T(v)$

```
1: function OUTPUT-MC(x, T(v))
2:     M ← ∅, 𝒩(x) ← ∅                          ▷ 𝒩(x) is defined as a global variable
3:     p ← x
4:     while p ≠ NULL do
5:         M ← M ∪ {p.key}
6:         p ← p.parent
7:     M ← REVERSE(M)                            ▷ Reverse the array of M
8:     SEARCH-DOWN(M, x, T(v))
9:     return 𝒩(x)
10:
11: function SEARCH-DOWN(M, x, T(v))
12:     if x.child ≠ ∅ then
13:         𝒩(x) ← 𝒩(x) ∪ {(x.value, M)}
14:         return
15:     else
16:         for each (y.key, y) ∈ x.child do
17:             M ← M ∪ {y.key}
18:             SEARCH-DOWN(M, y, T(v))
19:             M ← DELETE-LAST(M)                ▷ Delete the last element in M
```

corresponding PD is derived through a tree traversal (Algorithm 11).

In Algorithm 7, we obtain $\mathcal{N}(M)$ for M by first including the MCs from $T(u)$ for $u \in V(M)$ (lines 3-7). For adding neighboring MCs that do not start with $u \in V(M)$, it suffices to only consider the MCs present in $T(u)$ in $u \in \tilde{L}(v_j)$, $\tilde{L}(v_j) \leftarrow L(v_j) \backslash \{v_i : v_i \in M, v_i.id < v_j.id\}$ for $j = 1, \ldots, |M|$, that contains v_j (lines 8-17).

Algorithm 8 is straightforward. To output all the paths that contain a node x in $T(v)$,

Algorithm 9 Add an edge between $n_{M_1}[i]$ and $n_{M_2}[i]$ with weight $w_{e_\mathcal{G}}$ in $\mathcal{F}_i$, $i = w_{e_\mathcal{G}}, \ldots, 1$

procedure INSERT-CG-EDGES($n_{M_1}, n_{M_2}, \mathcal{T}, \mathcal{T}', w_{e_\mathcal{G}}$)
  $e \leftarrow$ SET-CG-EDGE($n_{M_1}, n_{M_2}, w_{e_\mathcal{G}}$)
  for $i = w_{e_\mathcal{G}}, \ldots, 1$ **do**
    if ETT-CONNECTED($n_{M_1}[i].pET, n_{M_2}[i].pET$) = TRUE **then**
      break
    else
      ETT-SPLAY($n_{M_1}[i].pET$)
      ETT-SPLAY($n_{M_2}[i].pET$)
      $\mathcal{T} \leftarrow$ UPDATE-REP($n_{M_1}[i].pET, n_{M_2}[i].pET, \mathcal{T}, \mathcal{T}', i$) ▷ Algorithm 10
      ADD-TREE-EDGE($n_{M_1}, n_{M_2}, e_\mathcal{G}, i$)
  return $\mathcal{T}$

function SET-CG-EDGE($n_{M_1}, n_{M_2}, w_{e_\mathcal{G}}$)
  ▷ Initialize an edge with $w_{e_\mathcal{G}}$ inserted between CG nodes representing M_1 and M_2
  $e_\mathcal{G}.CG_1 \leftarrow n_{M_1}$, $e_\mathcal{G}.CG_2 \leftarrow n_{M_2}$
  for $i = 1, \ldots, w_{e_\mathcal{G}}$, **do**
    $e_\mathcal{G}.pET_1[i] \leftarrow$ NULL, $e_\mathcal{G}.pET_2[i] \leftarrow$ NULL
  return $e_\mathcal{G}$

function ADD-TREE-EDGE($n_{M_1}, n_{M_2}, e_\mathcal{G}, i$)
  $n_{M_1}[i].adjTEdges \leftarrow$ PUSH-BACK($n_{M_1}[i].adjTEdges, e_\mathcal{G}$)
  $n_{M_2}[i].adjTEdges \leftarrow$ PUSH-BACK($n_{M_2}[i].adjTEdges, e_\mathcal{G}$)
  $e_\mathcal{G}.pET_1[i] \leftarrow$ SET-ET-NODE($n_{M_1}[i], n_{M_2}[i]$) ▷ Store the pointers in the edge
  $e_\mathcal{G}.pET_2[i] \leftarrow$ SET-ET-NODE($n_{M_2}[i], n_{M_1}[i]$)
  ETT-LINK($n_{M_1}[i].pET, n_{M_2}[i].pET$)

Algorithm 10 Update $\mathcal{T}$ before a tree edge insertion

1: **procedure** UPDATE-REP($E_{R_1}, E_{R_2}, \mathcal{T}, \mathcal{T}', i$)
2: $r_1 \leftarrow E_{R_1}.dRepID$, $r_2 \leftarrow E_{R_2}.dRepID$
3: **if** $r_1 \neq r_2$ **then**
4: $s_1 \leftarrow$ GET-REP-SIZE($r_1, \mathcal{T}, \mathcal{T}', i$)
5: $s_2 \leftarrow$ GET-REP-SIZE($r_2, \mathcal{T}, \mathcal{T}', i$)
6: **if** $(s_1 > s_2)$ **or** $(s_1 = s_2$ **and** $r_1 < r_2)$ **then**
7: $\mathcal{T} \leftarrow$ UPDATE-REP-ET-NODE($E_{R_2}, r_2, r_1, \mathcal{T}, i$)
8: **else**
9: $\mathcal{T} \leftarrow$ UPDATE-REP-ET-NODE($E_{R_1}, r_1, r_2, \mathcal{T}, i$)
10: **return** $\mathcal{T}$
11:
12: **function** GET-REP-SIZE($r, \mathcal{T}, \mathcal{T}', i$)
13: **if** HT-ISKEY($\mathcal{T}[i], r$)=TRUE **then**
14: $E_M \leftarrow$ HT-GET($\mathcal{T}[i], r$)
15: $s \leftarrow E_M.CG_1.size$
16: **else**
17: $s \leftarrow$ HT-GET($\mathcal{T}'[i], r$)
18: **return** s
19:
20: **function** UPDATE-REP-ET-NODE($E_R, r_1, r_2, \mathcal{T}, i$)
21: $E_R.dRepID \leftarrow r_2$
22: **if** HT-ISKEY($\mathcal{T}[i], r_1$) = TRUE **then**
23: $\mathcal{T}[i] \leftarrow$ HT-DELETE($\mathcal{T}[i], r_1$)
24: **return** $\mathcal{T}$

Algorithm 11 Obtain the birth and death time of the nodes from $\mathcal{T}$ and save this information in P

procedure GENERATE-PD($\mathcal{M}, \mathcal{T}$)
  $E_{M'} \leftarrow \text{argmax}_{(m,E_M)\in\mathcal{T}[2]}(E_M.CG_1.size)$
  $\mathcal{T}[1] \leftarrow (E_{M'}.CG_1.id, E_{M'})$
  for $i = |\mathcal{T}| - 1, \ldots, 1$ **do**
    $\mathcal{M} \leftarrow$ RECORD-DEATH($\mathcal{M}, \mathcal{T}, i$)
  $\mathsf{M}' \leftarrow$ HT-GET($\mathcal{M}, E_{M'}.CG_1.id$)
  $\mathsf{M}'.deathT \leftarrow 1$
  $P \leftarrow \emptyset$
  PREORDER(P, M')
  return $\mathcal{M}, P$

function RECORD-DEATH($\mathcal{M}, \mathcal{T}, i$)
  $D \leftarrow \mathcal{T}[i+1] \backslash \mathcal{T}[i]$
  if $D \neq \emptyset$ **then**
    for each $(m, E_M) \in D$ **do**
      $\mathsf{M} \leftarrow$ HT-GET($\mathcal{M}, m$)
      $\mathsf{M}.deathT \leftarrow i$
      for each $(m', E_{M'}) \in \mathcal{T}[i]$ **do**
        if $i = 1$ **or** ETT-CONNECTED($\mathsf{M}.pCG[i-1].pET, E_{M'}$) = TRUE **then**
          $\mathsf{M}' \leftarrow$ HT-GET($\mathcal{M}, m'$)
          $\mathsf{M}'.child \leftarrow$ PUSH-BACK($\mathsf{M}'.child, \mathsf{M}$)
          break
  return $\mathcal{M}$

we first add x itself and all the nodes above x to each path, and then walk down the paths below x recursively in case x is a shared node.

```
24: function PREORDER(P, M)
25:   ▷ Visit each node in the community tree through a preorder traversal starting from M
26:     if M = NULL then
27:         return
28:     else
29:         P ← P ∪ {(M.deathT, M.birthT)}
30:         while M.child ≠ ∅ do
31:             M′ ← POP-BACK(M.child)
32:             PREORDER(P, M′)
```

Algorithm 9 provides the implementation details of inserting tree edges. First, a CG edge $e_{\mathcal{G}}$ is initialized with endpoints and an array of pairs of null pointers. If $e_{\mathcal{G}}$ is inserted in $\mathcal{F}_i$, in addition to adding $e_{\mathcal{G}}$ to the adjacency lists of tree edges of $n_{M_1}[i]$ and $n_{M_2}[i]$, we also create two ET nodes that $e_{\mathcal{G}}.pET_1[i]$ and $e_{\mathcal{G}}.pET_2[i]$ are directed to when linking two ET trees (lines 22-24). Notice the ET nodes corresponding to the CG nodes to be connected are splayed to the roots of their respective ET trees in preparation for the linkage of two trees (lines 7-8). Meanwhile, we also need to update the representative ET node of the new tree.

In Algorithm 10, let E_{R_1}, E_{R_2} be the respective root nodes of two trees to be connected, and r_1, r_2 be the values of the representative ID of the two trees. Since E_{R_1}, E_{R_2} are both loop nodes, and each loop node in an ET tree is designed to carry the representative ET node's ID in the difference form of $dRepID$, we have $r_1 = E_{R_1}.dRepID$ and $r_2 = E_{R_2}.dRepID$. Due to this fact, we can directly identify the representative ET node and determine the size of the representative CG node s_i for each spanning tree (lines 14-15). If $s_1 > s_2$, or $s_1 = s_2$ and $r_1 < r_2$, we add $(r_1 - r_2)$ to $E_{R_2}.dRepID$ before the merger to ensure that the IDs of all the loop nodes in the new ET tree rooted E_{R_1} at are the same as r_1. We then remove the representative ET node with ID r_2 from $\mathcal{T}[i]$. The operation (line 9) is done the other way round in case the if condition is not true. We discuss the role of $\mathcal{T}'$ in Section A.3.

In Algorithm 11, we first set the only element in $\mathcal{T}[1]$ to be the element including the oldest representative ET node in $\mathcal{T}[2]$ (lines 2-3). We then record the death time for each M whose corresponding element (m, E_M) disappears in $\mathcal{T}[i]$, if any (lines 13-22). For each $(m', E_{M'}) \in \mathcal{T}[i]$, if E_M is in the same ET tree as $E_{M'}$ (line 19), it implies the CC represented by M has been merged into the CC represented by M', i.e., M becomes the *child* of M' (line 21). A special case is that for the only remaining element in $\mathcal{T}[1]$, the death time of its corresponding MC, named *root representative MC*, is set to 1. Finally, we output the persistence diagram as an array of pairs of every representative MC's death and birth time through a preorder traversal starting from the root representative MC (lines 26-32).

A.3 Community Tree Update Algorithm

Algorithm 12 Update $\mathcal{T}$ after multiple insertions of vertices and edges from G_{t-1} to G_t

1: **procedure** UPDATE-CT($G_{t-1}, V_+^\Delta, E_+^\Delta, \mathfrak{T}, \mathcal{M}, \mathcal{T}, m_{t-1}$)
2: $G'_{t-1} \leftarrow G_{t-1} + V_+^\Delta$
3: **for** $i = 1, \ldots, |\mathcal{T}|$ **do**
4: **for each** $(m, E_M) \in \mathcal{T}[i]$ **do**
5: $\mathcal{T}'[i] \leftarrow$ HT-ADD($T'[i], (m, E_M.CG_1.size)$)
6: $\mathcal{M}^{new} \leftarrow$ LIST-NEW-MCS(G'_{t-1}, E_+^Δ) ▷ Algorithm 3 in [40]
7: $\omega \leftarrow \max_{M \in \mathcal{M}^{new}} |M|$
8: $\mathcal{M}^{del} \leftarrow$ LIST-SUBSUMED-MCS($E_+^\Delta, \mathfrak{T}, \mathcal{M}^{new}$) ▷ Algorithm 4 in [40]
9: $\mathfrak{T}, \mathcal{M}, \mathcal{T} \leftarrow$ REMOVE-MCS($\mathcal{M}^{del}, \mathfrak{T}, \mathcal{M}, \mathcal{T}$) ▷ Algorithm 13
10: **if** $\omega > |\mathcal{T}|$ **then** ▷ Expand $\mathcal{T}$ with empty elements at the end to reach a size of ω
11: $\mathcal{T} \leftarrow$ RESIZE($\mathcal{T}, \omega$)
12: Initialize an empty array of size $|\mathcal{T}|$ as $\mathcal{T}'$
13: $\mathfrak{T}, \mathcal{M}, \mathcal{T}, m_t \leftarrow$ ADD-MCS($G'_t, \mathcal{M}^{new}, \mathfrak{T}, \mathcal{M}, \mathcal{T}, \mathcal{T}', m_{t-1}$)
14: $\mathcal{M}, P \leftarrow$ GENERATE-PD($\mathcal{M}, \mathcal{T}$)
15: **return** $\mathfrak{T}, \mathcal{M}, \mathcal{T}, P, m_t$

The variations of G are simply captured through insertions of edges and vertices. As Algorithm 12 and Figure A.3 demonstrate, the updates of $\mathcal{T}$ and P from G_{t-1} to G_t are conducted through the following steps: insert each single vertex in $V_+^\Delta = V_t \backslash V_{t-1}$, where no change of

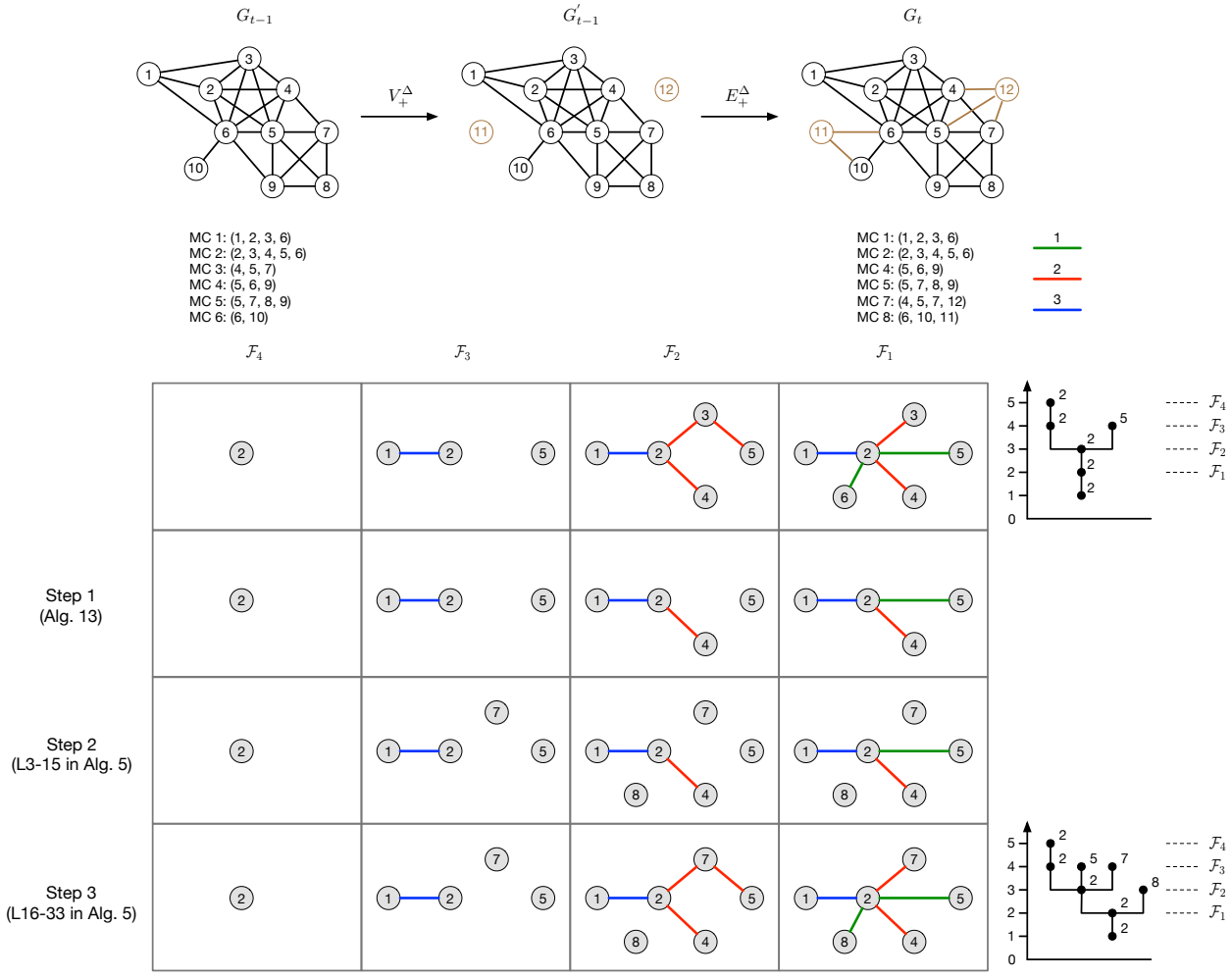


Figure A.3: An example of updating a community tree. $\mathcal{T}$ does not change from G_0 to G'_0 since only single vertices are added. From G'_0 to G_1, the CG nodes corresponding to MCs 3 and 6 are removed along with their incident edges before those corresponding to MCs 7 and 8 are added and connected to others. $\mathcal{T}$ is simultaneously updated due to tree edge insertions.

Algorithm 13 Update CGs after removing all the MCs in $\mathcal{M}^{del}$

1: **procedure** REMOVE-MCS($\mathcal{M}^{del}, \mathcal{T}, \mathcal{M}, \mathcal{T}$)
2: **for each** $M \in \mathcal{M}^{del}$ **do**
3: $v^* \leftarrow \text{argmin}_{v \in M}\, v.id$
4: $T(v^*) \leftarrow$ HT-GET($\mathcal{T}, v^*.id$)
5: $m \leftarrow$ TRIE-GET-ID($T(v^*), M$)
6: $\mathsf{M} \leftarrow$ HT-GET($\mathcal{M}, m$)
7: DELETE-CG-EDGES(M) ▷ Algorithm 14
8: **for** $i = |M|, \ldots, 2$ **do**
9: **if** HT-ISKEY($\mathcal{T}[i], m$) = TRUE **then**
10: $\mathcal{T}[i] \leftarrow$ HT-DELETE($\mathcal{T}[i], m$)
11: $\mathcal{M} \leftarrow$ HT-DELETE($\mathcal{M}, m$)
12: DELETE(M) ▷ Delete M as an object and the corresponding CG, ET nodes at each order
13: $T(v^*) \leftarrow$ TRIE-REMOVE($T(v^*), M$)
14: **return** $\mathcal{T}, \mathcal{M}, \mathcal{T}$

$\mathcal{T}$ occurs, and then add the edges in $E_+^\Delta = E_t \backslash E_{t-1}$.

Algorithm 12 then enumerates the new MCs $\mathcal{M}^{new}$ and the subsumed MCs[1] $\mathcal{M}^{del}$ due to the insertion of edges in E_+^Δ. The CG nodes corresponding to each $M \in \mathcal{M}^{del}$ are then removed from $\mathcal{F}_{|M|-1}, \ldots, \mathcal{F}_1$, as given in Algorithm 13, before those corresponding to the new MCs are inserted in the CGs. The removal of a CG node implies the deletion of all the incident tree edges (Algorithm 14). As the reverse operation of ADD-TREE-EDGE, DELETE-TREE-EDGE removes the tree edge $e_\mathcal{G}$ from the tree adjacency list of the corresponding CG nodes, and splits the ET tree into two by cutting out two arcs, $e_\mathcal{G}.pET_1[i]$ and $e_\mathcal{G}.pET_2[i]$, in the tours. After all the tree edges incident to $\mathsf{M}.pCG$ are discarded, we additionally remove

[1]We determine if a clique is a subsumed MC through the function TRIE-GET-ID. If it returns -1, then the clique is not a subsumed MC; otherwise, it is.

Algorithm 14 Delete CG edges incident to the CG node corresponding to M at each order

```
1: procedure DELETE-CG-EDGES(M)
2:     for i = M.birthT − 1, ..., 1 do
3:         while n_M[i].adjTrEdges ≠ ∅ do
4:             e_G ← POP-BACK(n_M[i].adjTrEdges)
5:             n_{M_1} ← e_G.CG_1, n_{M_2} ← e_G.CG_2
6:             DELETE-TREE-EDGE(n_{M_1}, n_{M_2}, e_G, i)
7:             if i = 1 then
8:                 DELETE(e_G)
9:
10: function DELETE-TREE-EDGE(n_{M_1}, n_{M_2}, e_G, i)
11:     n_{M_1}[i].adjTrEdges ← REMOVE-FROM-LIST(n_{M_1}[i].adjTrEdges, n_{M_1}[1].id, n_{M_2}[1].id)
12:     n_{M_2}[i].adjTrEdges ← REMOVE-FROM-LIST(n_{M_2}[i].adjTrEdges, n_{M_1}[1].id, n_{M_2}[1].id)
13:     ETT-CUT(e_G.pET_1[i], e_G.pET_2[i])
14:
15: function REMOVE-FROM-LIST(L, m_1, m_2)
16:     for i = 1, ..., |L| do
17:         n_{M_1} ← L[i].CG_1, n_{M_2} ← L[i].CG_2
18:         if n_{M_1}[1].id = m_1 and n_{M_2}[1].id = m_2 then
19:             L ← L\L[i]
20:             return L
```

E_M from $T[i]$ if M is a representative MC, erase M from $\mathcal{M}$ and delete it, along with the corresponding isolated CG and ET nodes (lines 8-12 in Algorithm 13).

Notice that we do not need to find the representative ET node for the separate tree after the split in Algorithm 14. For each removed $M \in \mathcal{M}^{del}$, it is subsumed by a $M \in \mathcal{M}^{new}$ that will reconnect with the neighbors of the removed MC and be the candidate of the representative MC of the new CC after the reconnection. Thus, we only need to update

the representative MC during ADD-MCs. Back to Algorithm 10, the case when $r_1 = r_2$ implies that both ET trees are cut from the same tree and still retain the representative ID from the original tree. Thus, no update needs to be performed before the reconnection. We create $\mathcal{T}'$ to store the sizes of the representative MCs at each order in G_{t-1} (lines 3-5 in Algorithm 12). When $r_1 \neq r_2$, if the subsumed MC was the representative MC and removed from $\mathcal{T}[i]$ beforehand, we obtain the size of the representative MC from $\mathcal{T}'$ instead.