University of Oxford


# Topological Data Analysis of Temporal Networks

by

Dimitri Lozeve

St Catherine's College

A dissertation submitted in partial fulfilment of the requirements for the degree of Master of Science in Statistical Science

*Department of Statistics,*
*24–29 St Giles, Oxford, OX1 3LB*

September 2018

## Abstract

Temporal networks are a mathematical model to represent interactions evolving over time. As such, they have a multitude of applications, from biology to physics to social networks. The study of dynamics on networks is an emerging field, with many challenges in modelling and data analysis.

An important issue is to uncover meaningful temporal structure in a network. We focus on the problem of periodicity detection in temporal networks, by partitioning the time range of the network and clustering the resulting subnetworks.

For this, we leverage methods from the field of topological data analysis and persistent homology. These methods have begun to be employed with static graphs in order to provide a summary of topological features, but applications to temporal networks have never been studied in detail.

We cluster temporal networks by computing the evolution of topological features over time. Applying persistent homology to temporal networks and comparing various approaches has never been done before, and we examine their performance side-by-side with a simple clustering algorithm. Using a generative model, we show that persistent homology is able to detect periodicity in the topological structure of a network.

We define two types of topological features, with and without aggregating the temporal networks, and multiple ways of embedding them in a feature space suitable for machine-learning applications. In particular, we examine the theoretical guarantees and empirical performance of kernels defined on topological features.

Topological insights prove to be useful in statistical learning applications. Combined with the recent advances in network science, they lead to a deeper understanding of the structure of temporal networks.

# *Acknowledgements*

I would like to thank my supervisors, Dr Heather Harrington, Dr Renaud Lambiotte, and Dr Mason Porter, from the Mathematical Institute, for their continuous support and guidance from the very beginning. They have allowed me to pursue my interests in networks and topological data analysis while providing me with resources, ideas, and motivation. They remained available to answer my questions and listen to my ideas, and provided invaluable feedback at every stage of the project.

I would also like to thank Dr Steve Oudot from École polytechnique, who was the first to introduce me to the field of topological data analysis, which led me to the original idea for the project. He was also very helpful during the project, giving me advice and updates on recent advances in persistent homology.

I also want to acknowledge the students and staff of the Department of Statistics and St Catherine's college, who always provided a stimulating work environment, along with excellent discussions.

Finally, my thanks go to my family and friends for their interest in my project, because trying to explain it to people not acquainted with the topic was, and remains, the best way to clarify my ideas and organise my thoughts.

# *Contents*

# *List of Figures*



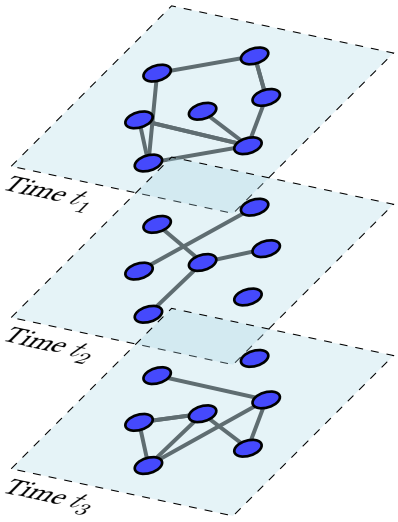
# *List of Algorithms*

# 1 Introduction

## 1.1 Temporal networks analysis

Networks are one of the most important mathematical concepts developed in the last few centuries. They allow the representation of interconnected data and complex systems. As such, the concept has been applied to wide variety of problems, in biology and neuroscience, physics, computer networks, and social science. In this context, network science has emerged as a discipline of its own, combining ideas and challenges from multiple fields of study [51].

An emerging trend in network science is the study of dynamics on networks [30, 32, 60]. Real-world systems, such as brains or social groups, tend to evolve over time, and these changing networks have given birth to the new field of network dynamics, where edges can reconfigure over time. Mathematical modelling of temporal connectivity patterns remain a difficult problem [5]. Recent advances in applied mathematics have led to may concurrent representations, multilayer networks [40] being one of the most important.




Temporal networks bring new challenges in size, shape, and complexity of data analysis, but also new opportunities with the development of new empirical methods and theoretical advances. One of these advances is the development of generative models that can be used to infer the dynamic mechanisms taking place in real-world systems [8, 25, 58].

Moreover, network theory naturally exposes many links with topology. The purpose of networks lies in the representation of *structure*, while topology is the study of spaces and *connectedness*. As topological methods gain traction in data science and statistical learning, they are also applied to more complex data representations, including networks [33, 59, 68]. Topological features naturally complement more traditional network statistics by focusing on mesoscale structure.

## 1.2 Related work

Topological data analysis (TDA) is a recent field [11]. It was originally focused on point cloud data, with only a recent shift towards network data [33]. Various methods have been developed, the main one being the weight-rank clique filtration (WRCF) [59]. Other examples of application of TDA to networks using WRCF can be found in [52].

There has also been attempts to map the nodes of a network to points in a metric space. For instance, the shortest-path distance between nodes can be used to compute pairwise distances in the network. Note that for this method to work properly, the network has to be connected. Many methods can be used to build a simplicial complex from a directed or undirected network [33, 38].

The main starting point for this project was the introduction of TDA for the study of temporal networks in [61]. In this dissertation, topological methods are introduced to classify temporal

networks randomly generated by different models. The objective of this study was to uncover the temporal structure of a network in order to inform its partitioning into "snapshots". Different methods to partition a network were compared for the first time, and topological features appeared to be relevant for distinguishing various temporal distributions.

Finally, there has been an increasing interest in using the topological structure of a dataset as an additional source of information for a statistical learning model. This has led to the development of topological descriptors suitable for use in various learning contexts. Previous work on vectorizations and kernels on topological features will be useful in the analysis of the structure of temporal networks.

## 1.3 Contributions

The main contributions of this work are threefold:

- We make an attempt at temporal partitioning networks and clustering the subnetworks, with immediate application for detecting periodicity. Sliding windows and persistent homology have been used in the context of periodicity detection before [56, 57], but never in the context of temporal networks.
- In general, topological methods have never been thoroughly studied on temporal network data. The work in [61] is the first to introduce the topic, but computation was limited due to the lack of available libraries. Here, we introduce recent (from the last 2–3 years) state-of-the-art topological methods and adapt them to temporal networks.
- Various methods to use topological features in a statistical learning context and their trade-offs are exposed. The mathematical background and practical considerations are leveraged to compare them in the context of machine learning.
- Finally, different topological approaches are compared. There are different ways to build a simplicial filtration on a network, and different manners of measuring distances between the outputs of persistent homology in the context of machine learning. These different methods are compared here with the objective of periodicity detection in temporal networks.

# 2 Graphs and Temporal Networks

## 2.1 Definition and basic properties

In this section, we introduce the notion of temporal networks (or temporal graphs). This is a complex notion, with many concurrent definitions and interpretations.

After clarifying the notations, we restate the standard definition of a non-temporal graph.

*Notation.*
- $\mathbb{N}$ is the set of non-negative natural numbers $0, 1, 2, \dots$
- $\mathbb{N}^*$ is the set of positive integers $1, 2, \dots$
- $\mathbb{R}$ is the set of real numbers. $\mathbb{R}_+ = \{x \in \mathbb{R} \mid x \geq 0\}$, and $\mathbb{R}_+^* = \{x \in \mathbb{R} \mid x > 0\}$.

**Definition 2.1** (Graph)**.** A *graph* is a couple $G = (V, E)$, where $V$ is a set of *nodes* (or *vertices*), and $E \subseteq V \times V$ is a set of *edges*. A *weighted graph* is defined by $G = (V, E, w)$, where $w : E \mapsto \mathbb{R}_+^*$ is called the *weight function*.

We also define some basic concepts that we will need later to build simplicial complexes on graphs.

**Definition 2.2** (Clique)**.** A *clique* is a set of nodes where each pair is adjacent. That is, a clique $C$ of a graph $G = (V, E)$ is a subset of $V$ such that for all $i, j \in C, i \neq j \implies (i, j) \in E$. A clique is said to be *maximal* if it cannot be augmented by any node, such that the resulting set of nodes is itself a clique.

Temporal networks can be defined in the more general framework of *multilayer networks* [40]. However, this definition is much too general for our simple applications, and we restrict ourselves to edge-centric time-varying graphs [16]. In this model, the set of nodes is fixed, but edges can appear or disappear at different times.

In this study, we restrict ourselves to discrete time stamps. Each interaction is taken to be instantaneous.

**Definition 2.3** (Temporal network)**.** A *temporal network* is a tuple $G = (V, E, \mathcal{T}, \rho)$, where:
- $V$ is a set of nodes,
- $E \subseteq V \times V$ is a set of edges,
- $\mathbb{T}$ is the *temporal domain* (often taken as $\mathbb{N}$ or any other countable set), and $\mathcal{T} \subseteq \mathbb{T}$ is the *lifetime* of the network,
- $\rho : E \times \mathcal{T} \mapsto \{0, 1\}$ is the *presence function*, which determines whether an edge is present in the network at each time stamp.

The *available times* of an edge are the set $\mathcal{I}(e) = \{t \in \mathcal{T} : \rho(e, t) = 1\}$.

Temporal networks can also have weighted edges. In this case, it is possible to have constant weights (edges can only appear or disappear over time, and always have the same weight), or time-varying weights. In the latter case, we can set the domain of the presence function to be $\mathbb{R}_+$ instead of $\{0, 1\}$, where by convention a 0 weight corresponds to an absent edge.

**Definition 2.4** (Additive and dismantling temporal networks)**.** A temporal network is said to be *additive* if for all $e \in E$ and $t \in \mathcal{T}$, if $\rho(e,t) = 1$, then for all $t' > t, \rho(e,t') = 1$. An additive network can only gain edges over time.

A temporal network is said to be *dismantling* if for all $e \in E$ and $t \in \mathcal{T}$, if $\rho(e,t) = 0$, then for all $t' > t, \rho(e,t') = 0$. An dismantling network can only lose edges over time.

## 2.2 Network statistics

To analyse networks, network statistics are used. These are low-dimensional summaries of important properties of a graph. Some of them focus on local features, while some others concentrate on global aspects. Note that the following only applies for graphs, and cannot be used directly on temporal networks.

These definitions are taken from the reference work by Newman [51].

The first network statistics try to determine which vertices are the most *central*, which are the most "important" in the network.

**Definition 2.5** (Local clustering coefficient)**.** The *local clustering coefficient* of a vertex $v$ is defined as

$$C(v) = \frac{\sum_{u,w \in \mathcal{V}} a_{u,v} a_{w,v} a_{u,w}}{\sum_{u,w \in \mathcal{V}, u \neq w} a_{u,v} a_{w,v}}.$$

The *average clustering coefficient* is

$$\overline{C} = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} C(v).$$

**Definition 2.6** (Global clustering coefficient)**.** The *global clustering coefficient* or *transitivity* is

$$C = \frac{3 \times \text{number of triangles}}{\text{number of connected triples}}.$$

Another interesting summary is the average shortest path between vertices.

**Definition 2.7** (Path)**.** A *path* between two vertices $v_0$ and $v_n$ is a sequence of vertices $(v_0, v_1, \dots, v_n)$ such that every consecutive pair of vertices $(v_i, v_{i+1})$ is connected by an edge.

The *length* of a path is the number of edges traversed along the path. The distance $l(u,v)$ between vertices $u$ and $v$ is defined as the length of the shortest path between $u$ and $v$.

**Definition 2.8** (Average shortest path length)**.** The *Average shortest path length* is defined as

$$l = \frac{1}{|\mathcal{V}|\left(|\mathcal{V}| - 1\right)} \sum_{u,v \in \mathcal{V}, u \neq v} l(u,v).$$

Many other centrality measures exist, the most well-known being the eigenvector centrality, Katz centrality, and PageRank. See chapter 7 of [51] for more details.

# 3 Topological Data Analysis and Persistent Homology

## 3.1 Basic constructions

### 3.1.1 Homology

Our goal is to understand the topological structure of a metric space. For this, we can use *homology*, which consists of associating a vector space $H_i(X)$ to a metric space $X$ and a dimension $i$. The dimension of $H_i(X)$ gives us the number of $i$-dimensional components in $X$: the dimension of $H_0(X)$ is the number of path-connected components in $X$, the dimension of $H_1(X)$ is the number of holes in $X$, and the dimension of $H_2(X)$ is the number of voids.

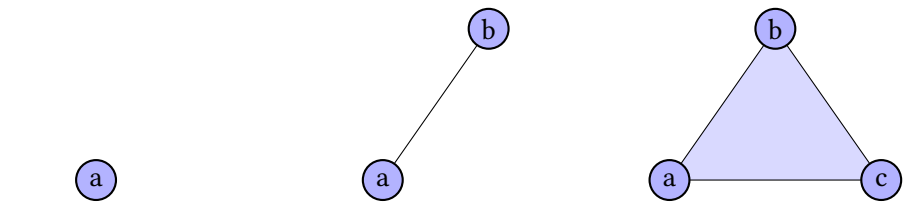
Crucially, these vector spaces are robust to continuous deformation of the underlying metric space (they are *homotopy invariant*). However, computing the homology of an arbitrary metric space can be extremely difficult. It is necessary to approximate it in a structure that would be both combinatorial and topological in nature.

### 3.1.2 Simplicial complexes

To understand the topological structure of a metric space, we need a way to decompose it in smaller pieces that, when assembled, conserve the overall organisation of the space. For this, we use a structure called a *simplicial complex*, which is a kind of higher-dimensional generalization of a graph.

The building blocks of this representation is the *simplex*, which is the convex hull of an arbitrary set of points. Examples of simplices include single points, segments, triangles, and tetrahedrons (in dimensions 0, 1, 2, and 3 respectively).

**Definition 3.1** (Simplex)**.** A *$k$-dimensional simplex* $\sigma = [x_0, \dots, x_k]$ is the convex hull of the set $\{x_0, \dots, x_k\} \in \mathbb{R}^d$, where $x_0, \dots, x_k$ are affinely independent. $x_0, \dots, x_k$ are called the *vertices* of $\sigma$, and the simplices defined by the subsets of $\{x_0, \dots, x_k\}$ are called the *faces* of $\sigma$.




(a) Single vertex (b) Segment (c) Triangle

Figure 3.1: Examples of simplices.

We then need a way to meaningfully combine these basic building blocks so that the resulting object can adequately reflect the topological structure of the metric space.

**Definition 3.2** (Simplicial complex)**.** A *simplicial complex* is a collection $K$ of simplices such that:

- any face of a simplex of $K$ is a simplex of $K$
- the intersection of two simplices of $K$ is either the empty set, or a common face, or both.

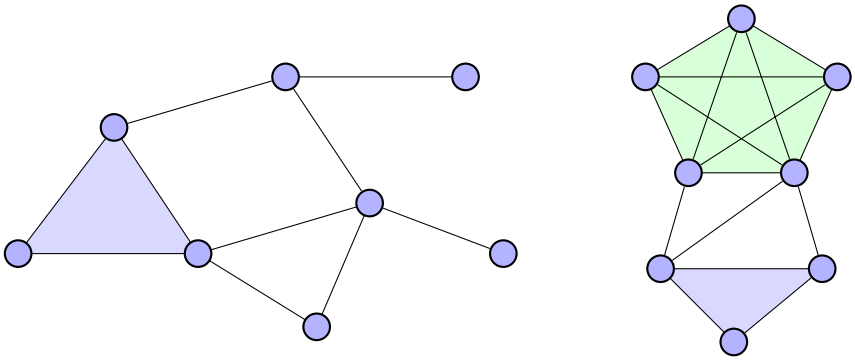


Figure 3.2: Example of a simplicial complex that has two connected components, two 3-simplices, and one 5-simplex.

The notion of simplicial complex is closely related to that of a hypergraph. One important distinction lies in the fact that a subset of a hyperedge is not necessarily a hyperedge itself.

### 3.1.3 Simplicial homology

Using these definitions, we can define homology on simplicial complexes [18, 23]. In this section, we restrict to homology with coefficients in $\mathbb{Z}_2$, the field with two elements.

**Definition 3.3** ($k$-chains)**.** Let $K$ be a finite simplicial complex, and $p$ a non-negative integer. The space of *$k$-chains* $C_p(K)$ of $K$ is the set of formal sums of $p$-simplices of $K$. More precisely, it is the $\mathbb{Z}_2$-vector space spanned by the $p$-simplices of $K$.

Since the coefficients of $C_p(K)$ are in $\mathbb{Z}_2$, a $p$-chain is simply a finite collection of $p$-simplices. The sum of two $k$-chains is the symmetric difference of the two chains, i.e. the collection of $p$-simplices that belong to either, but not both, of the chains.

**Definition 3.4** (Boundary)**.** The *boundary* of a $p$-simplex $\sigma$ is the $(p-1)$-chain

$$\partial_p(\sigma) := \sum_{\tau \in K_{p-1},\, \tau \subset \sigma} \tau,$$

where $K_{p-1}$ is the set of $(p-1)$-simplices of $K$.

As the $p$-simplices form a basis of $C_p(K)$, $\partial_p$ can be extended into a linear map from $C_p(K)$ to $C_{p-1}(K)$, called the *boundary operator*. The elements of the kernel $\mathrm{Ker}(\partial_p)$ are called the *$p$-cycles* of $K$. The image $\mathrm{Im}(\partial_p)$ is the space of *$p$-boundaries* of $K$.

**Lemma 3.1.** *The image of $\partial_{p+1}$ is a subset of the kernel of $\partial_p$.*

*Proof.* The boundary of a boundary is always empty. To see this, consider the boundary of a $(p+1)$-simplex $\sigma$. The boundary of $\sigma$ consists of all $p$-faces of $\sigma$. The boundary of this boundary will contain each $(p-1)$-face of $\sigma$ twice, and since $1+1=0$ in $\mathbb{Z}_2$, we have that

$$\partial_p \circ \partial_{p+1} \equiv 0.$$

This implies directly that $\mathrm{Im}(\partial_{p+1}) \subset \mathrm{Ker}(\partial_p)$. □

**Definition 3.5** (Homology)**.** For any $p \in \mathbb{N}$, the *$p$-th (simplicial) homology group* of a simplicial complex $K$ is the quotient vector space

$$H_p(K) := \mathrm{Ker}(\partial_p)/\mathrm{Im}(\partial_{p+1}).$$

The dimension $\beta_p(K)$ of $H_p(K)$ is called the *$p$-th Betti number* of $K$.

Let us close this overview of simplicial homology by a look at induced maps. Let $K$ and $L$ be two simplicial complexes and $f : K \mapsto L$ a simplicial map between them. Since $f$ maps linearly each simplex of $K$ to a simplex of $L$, we can extend it to map a chain of $K$ to a chain of $L$ of the same dimension. If $c = \sum a_i \sigma_i$ is a $p$-chain in $K$, we can define $f_\#(c) = \sum a_i \tau_i$, where $\tau_i = f(\sigma_i)$ if it has dimension $p$ and $\tau_i = 0$ if $f(\sigma_i)$ has dimension less than $p$.

$$\begin{array}{ccccccc}
\cdots & \xrightarrow{\partial_{p+1}} & C_p(K) & \xrightarrow{\partial_p} & C_{p-1}(K) & \xrightarrow{\partial_{p-1}} & \cdots \\
 & & \downarrow{\scriptstyle f^p_\#} & & \downarrow{\scriptstyle f^{p-1}_\#} & & \\
\cdots & \xrightarrow{\partial_{p+1}} & C_p(L) & \xrightarrow{\partial_p} & C_{p-1}(L) & \xrightarrow{\partial_{p-1}} & \cdots
\end{array}$$

Figure 3.3: Induced maps and boundary operators.

**Proposition 3.2.** *$f_\#$ commutes with the boundary operator:*

$$f_\# \circ \partial_K = \partial_L \circ f_\#.$$

*Proof.* Consider $f^p_\# : C_p(K) \mapsto C_p(L)$, and let $c = \sum a_i \sigma_i \in C_p(K)$. If $f(\sigma_i)$ has dimension $p$, then all $(p-1)$-faces of $\sigma_i$ map to the corresponding $(p-1)$-faces of $\tau_i$, and the proposition follows. On the other hand, if $f(\sigma_i)$ has dimension less than $p$, then the $(p-1)$-faces of σf $\sigma_i$ map to simplices of dimension less than $p-1$, with the possible exception of exactly two $(p-1)$-faces whose images coincide and cancel each other. So we have that $\partial_L(f_\#(\sigma_i)) = f_\#(\partial_K(\sigma_i)) = 0$. (See [23] for details.) □

**Corollary 3.3.** *$f_\#$ maps cycles to cycles, and boundaries to boundaries.*

Therefore, $f_\#$ defines a map over quotients, called the *induced map on homology*

$$f^p_* : H_p(K) \mapsto H_p(L).$$

**Proposition 3.4.** *$f \mapsto f^p_*$ is a functor:*

- *if $f = \mathrm{id}_X$, then $f^p_* = \mathrm{id}_{H_p(X)}$,*
- *if $X \xrightarrow{f} Y \xrightarrow{g} Z$, then $(g \circ f)_* = g_* \circ f_*$.*

The derivation of simplicial homology in this section used the field $\mathbb{Z}_2$. It is however possible to define homology over any field. The definition of the boundary operator needs to be adapted to ensure that the Lemma 3.1 remains true, even when $1 \neq -1$. In this dissertation, we consider only persistent homology on $\mathbb{Z}_2$, as it is the field used in our implementation. It is important to note however that changing the field of the vector spaces can affect the homology and therefore the topological features detected [74].

### 3.1.4 Filtrations

If we consider that a simplicial complex is a kind of "discretization" of a subset of a metric space, we realise that there must be an issue of *scale*. For our analysis to be invariant under small perturbations in the data, we need a way to find the optimal scale parameter to capture the adequate topological structure, without taking into account some small perturbations, nor ignoring some important smaller features.

To illustrate this, let us take the example of the Čech complex, one of the most important tools to build a simplicial complex from a metric space.

**Definition 3.6** (Nerve)**.** Let $\mathcal{S} = (S_i)_{i\in I}$ be a non-empty collection of sets. The *nerve* of $\mathcal{S}$ is the simplicial complex whose vertices are the elements of $I$ and where $(i_0, \dots, i_k)$ is a $k$-simplex if, and only if, $S_0 \cap \cdots \cap S_k \neq \emptyset$.

**Definition 3.7** (Čech complex)**.** Let $X$ be a point cloud in an arbitrary metric space, and $\varepsilon > 0$. The *Čech complex* $\check{C}_\varepsilon(X)$ is the nerve of the set of $\varepsilon$-balls centred on the points in $X$.

An example construction of a Čech complex is represented on Figure 3.4. The simplicial complex depends on the value of $\varepsilon$. To adequately represent the topological structure of the underlying point cloud, it is necessary to consider all possible values of $\varepsilon$ in order to capture all the topological features.

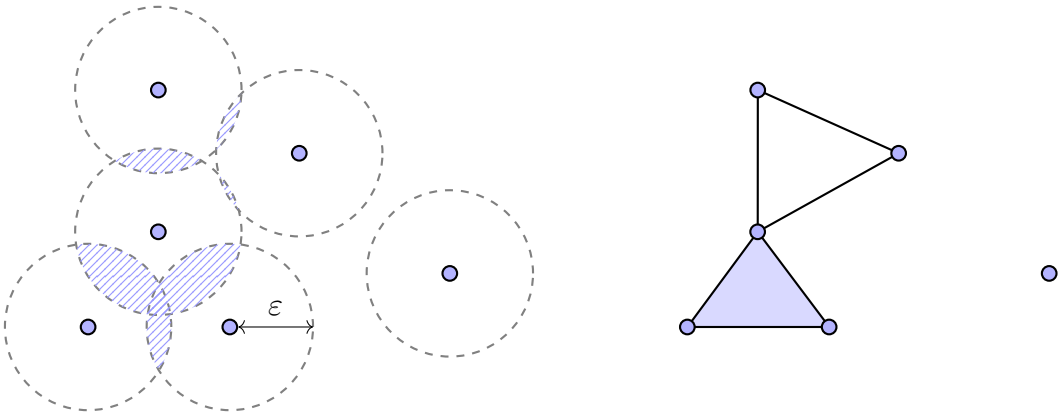



Figure 3.4: Example of a point cloud (left), and the corresponding Čech complex at level $\varepsilon$ (right). Dashed circles represent the $\varepsilon$-balls used to construct the simplicial complex.

This is the objective of *filtered simplicial complexes*.

**Definition 3.8** (Filtration)**.** A *filtered simplicial complex*, or simply a *filtration*, $K$ is a sequence $(K_i)_{i\in I}$ of simplicial complexes such that:

- for any $i, j \in I$, if $i < j$ then $K_i \subseteq K_j$,
- $\bigcup_{i\in I} K_i = K$.

To continue the example of Čech filtrations, one can build a sequence of simplicial complexes for each value of $\varepsilon > 0$. Due to their construction, Čech complexes on a point cloud $X$ respect the essential inclusion property:

$$\forall \varepsilon, \varepsilon' > 0, \quad \varepsilon < \varepsilon' \implies \check{C}_\varepsilon(X) \subseteq \check{C}_{\varepsilon'}(X).$$

## 3.2 Persistent Homology

We can now compute the homology for each step in a filtration. This leads to the notion of *persistent homology* [11, 74], which gives all the information necessary to establish the topological structure of a metric space at multiple scales.

**Definition 3.9** (Persistent homology)**.** The *$p$-th persistent homology* of a simplicial complex $K = (K_i)_{i \in I}$ is the pair $(\{H_p(K_i)\}_{i \in I}, \{f_{i,j}\}_{i,j \in I, i \leq j})$, where for all $i \leq j$, $f_{i,j} : H_p(K_i) \mapsto H_p(K_j)$ is induced by the inclusion map $K_i \mapsto K_j$.

By functoriality (Proposition 3.4), $f_{k,j} \circ f_{i,k} = f_{i,j}$. Therefore, the functions $f_{i,j}$ allow us to link generators in each successive homology space in a filtration.

Because each generator corresponds to a topological feature (connected component, hole, void, and so on, depending on the dimension $p$), we can determine whether it survives in the next step of the filtration. We say that $x \in H_p(K_i)$ is *born* in $H_p(K_i)$ if it is not in the image of $f_{i-1,i}$. We say that $x$ *dies* in $H_p(K_j)$ if $j > i$ is the smallest index such that $f_{i,j}(x) = 0$. The half-open interval $[i, j)$ represents the lifetime of $x$. If $f_{i,j}(x) \neq 0$ for all $j > i$, we say that $x$ lives forever and its lifetime is the interval $[i, \infty)$.

The couples of intervals (birth time, death time) depends on the choice of basis for each homology space $H_p(K_i)$. However, by the Fundamental Theorem of Persistent Homology [74], we can choose basis vectors in each homology space such that the collection of half-open intervals is well-defined and unique. This construction is called a *barcode* [11].

## 3.3 Topological summaries: barcodes and persistence diagrams

Although it contains relevant topological information, the persistent homology defined in the previous section cannot be used directly in statistical methods. *Topological summaries* are a compact representation of persistent homology as elements of a metric space. This is particularly useful in the context of statistical analysis, e.g. when one needs to compare the output of a given dataset to a null model.

One possible approach is to define a space in which we can project barcodes and study their geometric properties. One such space is the space of *persistence diagrams* [23].

**Definition 3.10** (Multiset)**.** A *multiset* $M$ is the couple $(A, m)$, where $A$ is the *underlying set* of $M$, formed by its distinct elements, and $m : A \mapsto \mathbb{N}^*$ is the *multiplicity function* giving the number of occurrences of each element of $A$ in $M$.

**Definition 3.11** (Persistence diagrams)**.** A *persistence diagram* is the union of a finite multiset of points in $\overline{\mathbb{R}}^2$ with the diagonal $\Delta = \{(x, x) \mid x \in \mathbb{R}^2\}$, where every point of $\Delta$ has infinite multiplicity.

One adds the diagonal $\Delta$ for technical reasons. It is convenient to compare persistence diagrams by using bijections between them, so persistence diagrams must have the same cardinality.

In some cases, the diagonal in the persistence diagrams can also facilitate comparisons between diagrams, as points near the diagonal correspond to short-lived topological features, so they are likely to be caused by small perturbations in the data.

One can build a persistence diagram from a barcode by taking the union of the multiset of (birth, death) couples with the diagonal $\Delta$. Figure 3.5 summarises the entire pipeline.

One can define an operator dgm as the first two steps in the pipeline. It constructs a persistence diagram from a subset of a metric space, via persistent homology on a filtered complex.

We can now define several distances on the space of persistence diagrams.

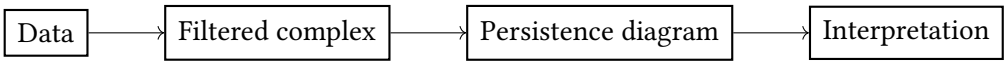



Figure 3.5: Persistent homology pipeline.

**Definition 3.12** (Wasserstein distance)**.** The *p-th Wasserstein distance* between two diagrams $X$ and $Y$ is

$$W_p[d](X,Y) = \inf_{\phi:X\mapsto Y}\left[\sum_{x\in X} d\left(x,\phi(x)\right)^p\right]$$

for $p \in [1,\infty)$, and:

$$W_\infty[d](X,Y) = \inf_{\phi:X\mapsto Y}\sup_{x\in X} d\left(x,\phi(x)\right)$$

for $p=\infty$, where $d$ is a distance on $\mathbb{R}^2$ and $\phi$ ranges over all bijections from $X$ to $Y$.

**Definition 3.13** (Bottleneck distance)**.** The *bottleneck distance* is defined as the infinite Wasserstein distance where $d$ is the uniform norm: $d_B = W_\infty[L_\infty]$.

The bottleneck distance is symmetric, non-negative, and satisfies the triangle inequality. However, it is not a true distance, as one can come up with two distinct diagrams with bottleneck distance 0, even on multisets that do not touch the diagonal $\Delta$.

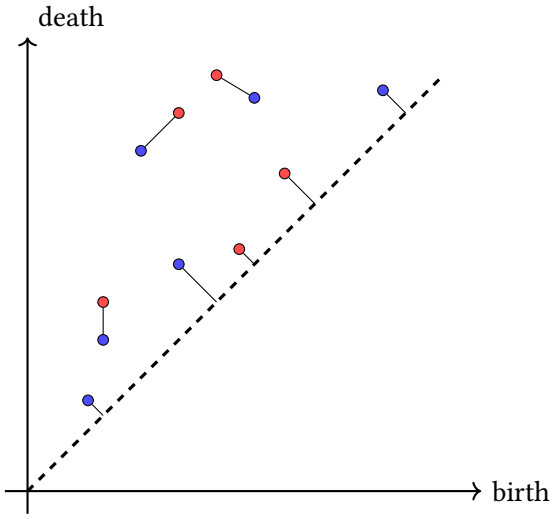



Figure 3.6: Bottleneck distance between two diagrams.

## 3.4 Stability

One of the most important aspects of topological data analysis is that it is *stable* with respect to small perturbations in the data. More precisely, the second step of the pipeline in Figure 3.5 is Lipschitz with respect to a suitable metric on filtered complexes and the bottleneck distance on persistence diagrams [19, 20]. First, we define a distance between subsets of a metric space [53].

**Definition 3.14** (Hausdorff distance)**.** Let $X$ and $Y$ be subsets of a metric space $(E,d)$. The *Hausdorff distance* is defined by

$$d_H(X,Y) = \max\left[\sup_{x\in X}\inf_{y\in Y} d(x,y), \sup_{y\in Y}\inf_{x\in X} d(x,y)\right].$$

We can now give an appropriate stability property [19, 20].

**Proposition 3.5.** *Let $X$ and $Y$ be subsets in a metric space. We have*

$$d_B(\mathrm{dgm}(X), \mathrm{dgm}(Y)) \leq d_H(X, Y).$$

## 3.5 Algorithms and implementations

Many algorithms have been developed to compute persistent homology. The first one developed, and by far the most commonly used is the so-called standard algorithm, introduced for the field $\mathbb{Z}_2$ in [22], and for general fields in [74]. This algorithm operates on the sequentially on the column of a boundary matrix. Its complexity is therefore cubic in the number of simplices in the worst case. It has been proven that this bound is hard [47].

Many algorithms have since been developed to deliver heuristic speed-ups in the case of sparse matrices. There are both sequential algorithms, such as the dual algorithm [66, 67], and algorithms that introduce parallelism in the computation, such as the distributed algorithm [7].

These algorithms have been implemented in many publicly-available implementations in the last few years. For a complete review and benchmarks of these implementations, see [52]. Here, we focus on implementations that provide a Python interface and implement common data structures, such as filtrations and persistence diagrams. State-of-the-art libraries include Ripser [6], DIPHA [62], GUDHI [44], and Dionysus [46]. GUDHI and Dionysus are under active development, with new versions released recently, exposing a complete Python API and implementing various algorithms, including multifield persistence and cohomology.

In this project, Dionysus 2 has been selected for its ease of use, good documentation, and good performance [52]. In this project, we only use persistent homology over the field $\mathbb{Z}_2$. Dionysus is also one of the few libraries to implement zigzag persistence (section 4.2).

# 4 Topological Data Analysis on Networks

## 4.1 Persistent homology for networks

We now consider the problem of applying persistent homology to network data. An undirected network is already a simplicial complex of dimension 1. However, this is not sufficient to capture enough topological information; we need to introduce higher-dimensional simplices. If the network is connected, one method is to project the nodes of a network onto a metric space [52], thereby transforming the network data into a point-cloud data. For this, we need to compute the distance between each pair of nodes in the network (e.g. with the shortest-path distance).

Various methods to project nodes onto a metric space (called *graph embeddings*) are available [24]. These mapping try to preserve the structure of the network as much as possible, e.g. by ensuring that neighbours in the network are neighbours in the metric space (according to the distance in that space), and vice-versa. A few methods worth mentioning in this area are *spectral methods*, which define the mapping according to the eigenvectors of a matrix constructed from the graph. These methods have the advantage of minimizing a well-defined criterion, which often admits an exact solution, and can often be computed exactly [24]. They include kernel principal components analysis, multidimensional scaling, Markov diffusion maps and Laplacian eigenmap. Other methods are *latent space methods*, which produce an embedding using a physical analogy, such as spring networks and attractive forces. These methods are often used for graph drawing (i.e. embedding in 2 or 3-dimensional spaces), but can only be approximated for large networks [24].

Using these graph embeddings, one can get a point cloud in a Euclidean space, and build a simplicial complex using one of the various methods developed for point clouds. One such example is the Čech complex (subsection 3.1.4).

Another common method, for weighted networks, is called the *weight rank-clique filtration* (WRCF) [59], which filters a network based on weights. The procedure works as follows:

1. Consider the set of all nodes, without any edge, to be filtration step 0.
2. Rank all edge weights in decreasing order $\{w_1, \dots, w_n\}$.
3. At filtration step $t$, keep only the edges whose weights are larger than or equal to $w_t$, thereby creating an unweighted graph.
4. Define the maximal cliques of the resulting graph to be simplices.

At each step of the filtration, we construct a simplicial complex based on cliques; this is called a *clique complex* [73]. The result of the algorithm is itself a filtered simplicial complex (Definition 3.8), because a subset of a clique is necessarily a clique itself, and the same is true for the intersection of two cliques.

This leads to one of the possibilities for applying persistent homology to temporal networks. One can apply WRCF on a network, obtaining a filtered complex, to which we can then apply persistent homology.

This method can quickly become very computationally expensive, as finding all maximal cliques (e.g. using the Bron–Kerbosch algorithm) is a complicated problem, with an optimal computational complexity of $\mathcal{O}(3^{n/3})$ [70]. In practice, one often restrict the search to cliques of dimension less than or equal to a certain bound $d_M$. With this restriction, the new simplicial

complex is homologically equivalent to the original: they have the same homology groups up to $H_{d_M-1}$.

## 4.2 Zigzag persistence

The persistent homology methods exposed in the previous sections operate on filtrations which are nested sequences of simplicial complexes:

$$\cdots \longrightarrow K_{i-1} \longrightarrow K_i \longrightarrow K_{i+1} \longrightarrow \cdots,$$

where each $\longrightarrow$ represents an inclusion map.

As we have seen in the previous section, filtrations can be built on networks. Computing persistent homology therefore relies on aggregating temporal networks, and then building a sequence of nested simplicial complexes orthogonal to the time dimension.

Another approach would be to use the existing temporal sequence in the network to build the filtration. The issue in this case is that the sequence is no longer nested, as edges can be added or removed at each time step (except for additive or dismantling temporal networks, see Definition 2.4). The development of *zigzag persistence* [12, 13] solves this issue by introducing a novel way to compute persistent homology on sequences of complexes that are no longer nested:

$$\cdots \longleftrightarrow K_{i-1} \longleftrightarrow K_i \longleftrightarrow K_{i+1} \longleftrightarrow \cdots,$$

where each $\longleftrightarrow$ represents an inclusion map oriented forwards or backwards.

To build this sequence from a temporal network, one can build a clique complex at each time step. Edge additions and deletions will translate to simplex additions and deletions in the sequence of simplicial complexes. More details of this implementation is provided in subsubsection 6.2.4.2.

Note that zigzag persistence is related to the more general concept of *multi-parameter persistence* [14, 21], where simplicial complexes can be filtered with more than one parameter. It is an active area of research, especially as the fundamental theorem of persistent homology is no longer valid with more than one parameter, and there are significant challenges in the visualization of "barcodes" for 2-parameter persistent homology [52].

The complexity of the zigzag algorithm is cubic in the maximum number of simplices in the complex [13], which is equivalent to the worst-case complexity of the standard algorithm for persistent homology (section 3.5). In practice however, zigzag computation tend to be much longer than their standard counterparts. Computing zigzag persistence on a temporal network is more costly than computing persistent homology on the weight rank clique filtration of the aggregated graph.

The library Dionysus [46] is the only one to implement zigzag persistence at the time of this writing. As implementation of the zigzag algorithm is not straightforward [13, 45], Dionysus was the most logical option for the topological study of temporal networks.

# 5 Persistent Homology for Machine-Learning Applications

The output of persistent homology is not directly usable by most statistical methods. For example, barcodes and persistence diagrams, which are multisets of points in $\overline{\mathbb{R}}^2$, are not elements of a metric space in which one can perform statistical computations.

The distances between persistence diagrams defined in section 3.3 allow one to compare different outputs. From a statistical perspective, it is possible to use a generative model of simplicial complexes and to use a distance between persistence diagrams to measure the similarity of our observations with this null model [2]. This would effectively define a metric space of persistence diagrams. It is even possible to define some statistical summaries (means, medians, confidence intervals) on these spaces [49, 71].

The issue with this approach is that metric spaces do not offer enough algebraic structure to be amenable to most machine-learning techniques. Many of these methods, such as principal-components analysis (PCA) and support vector machines (SVMs) require a Hilbert structure on the feature space [15, 19]. Equipped with this structure, one can then define common operations such as addition, average or scalar product on features, which then facilitate their use in machine learning. One of the most recent development in the study of topological summaries has been to find mappings between the space of persistence diagrams and Banach spaces[1, 10, 42, 43]. (The definitions of common topological structures can be found in Appendix A.)

## 5.1 Vectorization methods

The first possibility is to build an explicit feature map. Each persistence diagram is projected into a vector of $\mathbb{R}^n$, on which one can then build a suitable Hilbert structure.

The main examples in this category are persistence landscapes [10] and persistence images [1].

### 5.1.1 Persistence landscapes

Persistence landscapes [10] give a way to project barcodes to a space where it is possible to add them meaningfully. It is then possible to define means of persistence diagrams, as well as other summary statistics.

The function mapping a persistence diagram to a persistence landscape is *injective*, but no explicit inverse exists to go back from a persistence landscape to the corresponding persistence diagram. Moreover, a mean of persistence landscapes does not necessarily have a corresponding persistence diagram.

**Definition 5.1** (Persistence landscape)**.** The persistence landscape of a diagram $D = \{(b_i, d_i)\}_{i=1}^n$ is the set of functions $\lambda_k : \mathbb{R} \mapsto \mathbb{R}$, for $k \in \mathbb{N}$, such that

$$\lambda_k(x) = k\text{-th largest value of } \{f_{(b_i,d_i)}(x)\}_{i=1}^n,$$

(and $\lambda_k(x) = 0$ if the $k$-th largest value does not exist), where $f_{(b,d)}$ is a piecewise-linear function defined by:

$$f_{(b,d)} = \begin{cases} 0, & \text{if } x \notin (b,d), \\ x - b, & \text{if } x \in (b, \frac{b+d}{2}), \\ -x + d, & \text{if } x \in (\frac{b+d}{2}, d)\,. \end{cases}$$

Moreover, one can show that persistence landscapes are stable with respect to the $L^p$ distance, and that the Wasserstein and bottleneck distances are bounded by the $L^p$ distance [10]. We can thus view the landscapes as elements of a Banach space in which we can perform the statistical computations.

### 5.1.2 Persistence images

Persistence images [1] consist in a convolution of the persistence diagram with a probability distribution, followed by a discretization of the resulting distribution in order to obtain a finite-dimensional vector. Most of the following section is derived from the original paper [1].

**Definition 5.2** (Persistence surface)**.** Let $B$ be a persistence diagram, and $T : \mathbb{R}^2 \mapsto \mathbb{R}^2$ the linear transformation $T(x, y) = (x, y - x)$. Let $\phi_u : \mathbb{R}^2 \mapsto \mathbb{R}$ be a differentiable probability density function with mean $u \in \mathbb{R}^2$, and $f$ a non-negative weighting function which is zero along the horizontal axis, continuous, and piecewise differentiable.

The *persistence surface* associated to $B$ is the function $\rho_B : \mathbb{R}^2 \mapsto \mathbb{R}$ such that

$$\rho_B(z) = \sum_{u \in T(B)} f(u)\phi_u(z).$$

Then, one needs to reduce the persistence surface to a finite-dimensional vector by discretizing a subdomain of $\rho_B$ and integrating it over each region.

**Definition 5.3** (Persistence image)**.** Let $\rho_B$ be the persistence surface of a persistence diagram $B$. We fix a grid on the plane with $n$ cells (called *pixels*). The *persistence image* of $B$ is the collection of pixels, where for each cell $p$,

$$I(\rho_B)_p = \iint_p \rho_B \, \mathrm{d}y \, \mathrm{d}x.$$

There are three parameters:

- the resolution of the grid overlaid on the persistence surface,
- the probability distribution, which is often taken as a Gaussian distribution centred on each point (one still needs to choose an appropriate variance),
- the weight function, which must be zero on the horizontal axis (which corresponds to the diagonal $\Delta$ before transformation by the function $T$), continuous, and piecewise differentiable in order for the stability results to hold. Generally, weighting functions are taken non-decreasing in $y$ in order to weight points of higher persistence more heavily.

All of these choices are non-canonical, but the classification accuracy on most tasks seem to be robust to the choice of resolution and variance of the Gaussian distribution [72].

It is also important to note that points with infinite persistence are ignored by the weighting function $f$. Persistence images are therefore not suitable in applications where these features can be important to consider.

Persistence images are stable with respect to the 1-Wasserstein distance between persistence diagrams (and with respect to the $L^1$, $L^2$, and $L^\infty$ distances between images) [1].

In practice, persistence images are interesting because they project persistence diagrams in a Euclidean space. Compared to persistence landscape, one can apply a broader range of machine-learning techniques. It has also been observed that persistence images outperform performance landscapes in many classification tasks, with a comparable computational efficiency [1].

## 5.2 Kernel-based methods

The other possibility is to define feature maps *implicitly* by building kernels on persistence diagrams. Such a kernel allows to use a wide range of kernel-based machine-learning methods.

Let us recall the general framework of kernel methods [48, 65].

**Definition 5.4** (kernel)**.** A function $k : X \times X \mapsto \mathbb{R}_+$ on a non-empty set $X$ is a *kernel* if there exist a Hilbert space $\mathcal{H}$ and a map $\phi : X \mapsto \mathcal{H}$ such that

$$\forall x, y \in X,\ k(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathcal{H}}.$$

The Hilbert space $\mathcal{H}$ is called the *feature space* and the function $\phi$ is called the *feature map*.

As inner products are positive definite, so are kernels, since they are inner products on feature maps.

**Definition 5.5** (Reproducing kernel)**.** Let $\mathcal{H}$ be a Hilbert space of functions from a non-empty set $X$ to $\mathbb{R}$. A function $k : X \mapsto \mathbb{R}$ is called a *reproducing kernel* of $\mathcal{H}$ if it satisfies:

- $\forall x \in X,\ k(\cdot, x) \in \mathcal{H}$,
- $\forall x \in X, \forall f \in \mathcal{H},\ \langle f, k(\cdot, x) \rangle_{\mathcal{H}} = f(x)$.

Note that every reproducing kernel is a kernel, with feature space $\mathcal{H}$ and feature map $\phi : x \mapsto k(\cdot, x)$. In this case, $\phi$ is called the *canonical feature map*: the features are not explicited as vectors of $\mathbb{R}^n$, but as functions on $X$.

If $\mathcal{H}$ has a reproducing kernel, it is called a *reproducible kernel Hilbert space* (RKHS). The important result here is the *Moore-Aronszajn theorem* [9]: for every positive definite function $k$, there exists a unique RKHS with kernel $k$.

We can now build a feature space with a Hilbert structure without defining explicitly the feature map. Defining a kernel, i.e. any positive definite function, on persistence diagrams is enough to guarantee the existence of a unique RKHS with the adequate structure to perform machine-learning tasks.

The following sections will define some relevant kernels.

### 5.2.1 Sliced Wasserstein kernel

The sliced Wasserstein kernel is a new kernel on persistence diagrams introduced by Carrière et al. in [15]. The general idea is to intersect the plane by lines going through the origin, and projecting the points of the persistence diagrams onto these lines, computing the distance between the diagrams as the distance between measures on the real line. These distances are then integrated over all the possible lines passing through the origin.

The formal definition (taken from [15]) relies on the *1-Wasserstein distance* between measures on $\mathbb{R}$.

**Definition 5.6** (1-Wasserstein distance)**.** Lt $\mu$ and $\nu$ be two non-negative measures on $\mathbb{R}$ such that $\mu(\mathbb{R}) = \nu(\mathbb{R})$. The 1-Wasserstein distance between $\mu$ and $\nu$ is

$$\mathcal{W}(\mu, \nu) = \inf_f \int_{\mathcal{R}} f(x) \left[\mu(\mathrm{d}x) - \nu(\mathrm{d}x)\right],$$

where $f$ is 1-Lipschitz.

One can now define formally the sliced Wasserstein kernel.

**Definition 5.7** (Sliced Wasserstein kernel)**.** Let $\mathbb{S}_1$ be the $L_2$-distance sphere in $\mathbb{R}^2$. Given $\theta \in \mathbb{S}_1$ let $L(\theta)$ be the line $\{\lambda\theta : \lambda \in \mathbb{R}\}$, and $\pi_\theta$ the orthogonal projection onto $L(\theta)$. Let $\pi_\Delta$ be the orthogonal projection on the diagonal.

Let $D_1$ and $D_2$ be two persistence diagrams, and let

$$\mu_1^\theta = \sum_{p \in D_1} \delta_{\pi_\theta(p)} \qquad \text{and} \qquad \mu_{1\Delta}^\theta = \sum_{p \in D_1} \delta_{\pi_\theta \circ \pi_\Delta(p)},$$

and similarly for $\mu_2^\theta$ and $\mu_{2\Delta}^\theta$.

The sliced Wasserstein distance is defined as

$$SW(D_1, D_2) = \frac{1}{2\pi} \int_{\mathbb{S}_1} \mathcal{W}(\mu_1^\theta + \mu_{2\Delta}^\theta,\ \mu_2^\theta + \mu_{1\Delta}^\theta)\, \mathrm{d}\theta.$$

One can show that $SW$ is negative definite [15]. The function $k_{SW}$ defined as

$$k_{SW}(D_1, D_2) = \exp\left(-\frac{SW(D_1, D_2)}{2\sigma^2}\right)$$

is therefore a valid kernel, called the *sliced Wasserstein kernel*.

**Stability** It can be shown that the sliced Wasserstein kernel is *equivalent* to the 1-Wasserstein distance between persistence diagrams (Definition 3.12). (For a definition of metric equivalence, see Appendix A.)

**Approximate computation** In practice, $k_{SW}$ can be approximated by sampling $M$ directions between $-\pi/2$ and $\pi/2$. For each direction $\theta_i$ and for each persistence diagram $D$, one computes the scalar products between the points of the diagram and $\theta_i$, and sorts them into a vector $V_{\theta_i}(D)$. The $L_1$-distance between the vectors corresponding to each diagram is then averaged over the samples directions:

$$SW_M(D_1, D_2) = \frac{1}{M} \sum_{i=1}^{M} \left\| V_{\theta_i}(D_1) - V_{\theta_i}(D_2) \right\|_1.$$

The complete approximate computation is detailed in algorithm 1. It has a complexity of $\mathcal{O}(MN\log(N))$, where $N$ is an upper bound on the cardinality of the persistence diagrams.

### 5.2.2 Persistence scale-space kernel

The persistent scale-space kernel (PSSK) [43, 63] is another kernel on persistence diagrams. The following overview is summarised from [63]. The general idea is to represent a diagram $D$ as a sum of Dirac deltas centred on each point of $D$. This representation is a natural projection onto the sapce of functionals, which has a Hilbert structure.

However, this representation does not take into account the distance of the points of $D$ to the diagonal. This is important since points closed to the diagonal represent short-lived features, and are therefore more likely to be noise. Do take this into account, the sum of Dirac deltas is taken as the initial condition of a heat diffusion on the half-plane above the diagonal, with a null boundary condition on the diagonal itself.

This leads to the definition of the embedding as the solution of partial differential equation, which admit an explicit solution in the form of a positive definite kernel between persistence diagrams. This kernel also depends on a scale parameter, which allows to control the robustness of the embedding to noise.

**Algorithm 1:** Approximate computation of the sliced Wasserstein kernel.

**Input:** $D_1 = \{p_1^1, \dots, p_{N_1}^1\}, D_2 = \{p_1^2, \dots, p_{N_1}^2\}, M$
**Output:** $SW$
Add $\pi_\Delta(D_1)$ to $D_2$ and vice-versa
$SW \leftarrow 0$
$\theta \leftarrow -\pi/2$
$s \leftarrow \pi/M$
**for** $i \leftarrow 1$ **to** $M$ **do**
  Store the products $\langle p_k^1, \theta \rangle$ in an array $V_1$
  Store the products $\langle p_k^2, \theta \rangle$ in an array $V_2$
  Sort $V_1$ and $V_2$ in ascending order
  $SW \leftarrow SW + s\|V_1 - V_2\|_1$
  $\theta \leftarrow \theta + s$
**end**
$SW \leftarrow SW/\pi$

This kernel also comes with stability guarantees, as it is Lipschitz-continuous with respect to the 1-Wasserstein distance. It is also fast, as the distance between two diagrams $D_1$ and $D_2$ can be computed in $\mathcal{O}(|D_1||D_2|)$, where $|D|$ is the number of points in the diagram, or approximated in $\mathcal{O}(|D_1| + |D_2|)$ with bounded error. In practice, empirical tests show that the persistence scale-space kernel significantly outperforms the persistence landscapes in shape classification tasks [63].

### 5.2.3 Persistence weighted Gaussian kernel

The persistence weighted Gaussian kernel (PWGK) [42] is actually a family of kernels on persistence diagrams. Given a diagram $D$, one can define a measure $\mu_D^w := \sum_{x \in D} w(x)\delta_x$, where $\delta_x$ is the Dirac delta centred on $x$. The weight function $w$ can be chosen to give more weight to points farther from the diagonal. One example is $w(x) := \arctan\left(C(\text{death}(x) - \text{birth}(x))^p\right)$, with $C > 0$ and $p \in \mathbb{N}^*$.

Then, given a kernel $k$ and the corresponding RKHS $\mathcal{H}_k$,

$$\mu_D^w \mapsto \sum_{x \in D} w(x)k(\cdot, x)$$

is an embedding of $\mu_D^w$ in $\mathcal{H}_k$. The persistence weighted gaussian kernel is obtained by choosing $k$ as the Gaussian kernel $k_G(x, y) := \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$.

The PWGK is stable with respect to the bottleneck distance [42], and allows for efficient computation. If the persistent diagrams contain at most $n$ points, computation of the kernel involves $\mathcal{O}(n^2)$ evaluations of the kernel $k$. Similarly to the PSSK, an approximation is possible in $\mathcal{O}(n)$.

Experimental results on shape classification with SVMs show a significant improvement in accuracy over the PSSk, persistence images, and persistent landscapes [42].

## 5.3 Comparison

Every vectorization exposed in the previous sections are injective and stable with respect to some distance in the space of persistence diagrams. None of them, however, are surjective, and no explicit inverse exists.

Only one of these methods preserves the metric on the space of persistence diagrams: the sliced Wasserstein kernel, due to its equivalence to the 1-Wasserstein distance, as mentioned in subsection 5.2.1. As such, it is considered as the state-of-the-art in kernel embeddings of persistence diagrams.

There are two broad classes of applications that require different kinds of vectorization methods. On the one hand, if one needs to go back from the feature space to the diagram space, the best bet is an embedding that preserves distances, such as the sliced Wasserstein kernels, or has strong stability guarantees, such as the persistent weighted Gaussian kernel. These embeddings are best for distance-based methods, such as multidimensional scaling or nearest neighbours algorithms.

On the other hand, getting insights from individual points of a diagram, in order to recover information about individual topological features (such as cycles, holes, or voids), is a much harder, less well-studied problem. For instance, to recover the topological features of the mean of persistence diagrams, one would need to fit one of the vectorization methods on the mean. For this, persistence landscapes or images seem better suited.

This project focuses on clustering of networks. As such, conservation of the metric and stability is extremely important. Due to the theoretical guarantees, we will focus on the sliced Wasserstein kernel, which is also significantly easier to implement in its approximate version than the PSSK (which uses random Fourier features [63]) and the PWGK (which uses the fast Gauss transform [42]).

# 6 Temporal partitioning of networks

## 6.1 Problem statement

### 6.1.1 Data

Temporal networks represent an active and recent area of research. The additional dimension adds complexity to the study of graphs. As such, many methods that work well with graphs fail in the context of temporal networks.

Temporal networks are much more difficult to visualize, which makes it harder to uncover patterns directly [32]. Moreover, there are many issues in data collection. Complex interaction networks where each edge can be either present or absent at each time step grow exponentially in size with the number of nodes and the total data collection time [32]. Empirical temporal networks also tend to exhibit oversampling and noise, due to the nature of the measurements. For instance, proximity networks can record an interaction between two individuals if they walk close to each other without actually interacting. New advances try to take into account these limitations of data collection [50, 69].

In this study, we will consider temporal networks with *contact* interactions. In this context, interactions between nodes are supposed to have a duration of 0, and *oversampling* is used to represent a long interaction. For instance, in a network sampled every 5 seconds, an interaction lasting for 30 seconds will be recorded in 6 consecutive time steps.

### 6.1.2 Sliding windows

One possible solution to the study temporal networks is a partitioning of the time scale using *sliding windows*.

**Definition 6.1** (Temporal partitioning)**.** Let $G = (V, E, \mathcal{T}, \rho)$ a temporal network, and let $C = (c_1, \dots, c_n)$ be a cover of $\mathcal{T}$ by non-empty intervals of $\mathbb{N}$.

Then the sequence of temporal networks $(G_1, \dots, G_n)$, where $G_i = (V, E, c_i, \rho_i)$ and $\rho_i(e, t) = \rho(e, t)\mathbb{1}_{t \in c_i}$, is a *temporal partitioning* of $G$.

The partitioning is *uniform* if all intervals of $C$ have the same length. This length is called the *temporal resolution* of the partitioning.

In this project, we will only consider uniform partitioning of a finite temporal domain, where the intersection of two consecutive intervals have the same length. This intersection length is called the *overlap*.

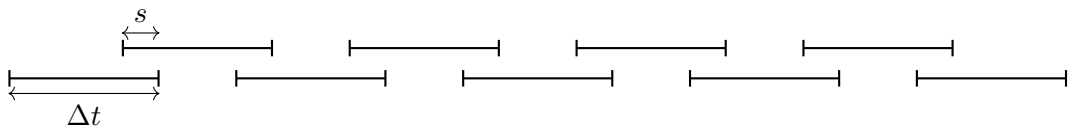



Figure 6.1: Uniform temporal partitioning with resolution $\Delta t$ and overlap $s$.

The choice of temporal resolution and overlap have a significant effect on the results of the analysis [41, 64, 69]. Different tasks may require specific parameters. A large resolution can overlook a significant pattern, while small overlap may cut through significant features, divided between two consecutive intervals.

### 6.1.3 Classification

After partitioning the temporal network, it is possible to run any kind of classification task on the resulting sequence of subnetworks.

If labels are available on each subnetwork, it is possible to run some supervised learning tasks. In the more common case of unsupervised learning, there are many possibilities, including the clustering of all subnetworks, or the detection of change points, where the structure of the network change fundamentally [55].

In this dissertation, we focus on unsupervised clustering of the subnetworks in order to detect periodicity in the original temporal network. Most machine-learning algorithms cannot take temporal networks directly as inputs. It is thus necessary to *vectorize* these networks, i.e. to project them onto a metric space with a structure suitable to the algorithm used. For instance, one could use traditional statistical summaries of networks (section 2.2), or the topological methods and their vectorizations discussed in the previous chapters.

The choice of vectorization depends on the choice of the clustering algorithm itself. Some machine-learning techniques, such as support vector machines, require a Hilbert structure on the input space [15, 28], while some, like $k$-nearest neighbours or agglomerative clustering, only require a metric space [28]. The feature space will therefore restrict the set of clustering algorithms available.

### 6.1.4 Applications

The persistent homology pipeline can be used to determine different properties of temporal networks. This study focuses on determining the *periodicity* of a temporal network. By clustering the subnetworks obtained by partitioning the temporal domain into sliding windows, it is possible to determine if a temporal network is periodic in its topological structure, and if so, to estimate its period.

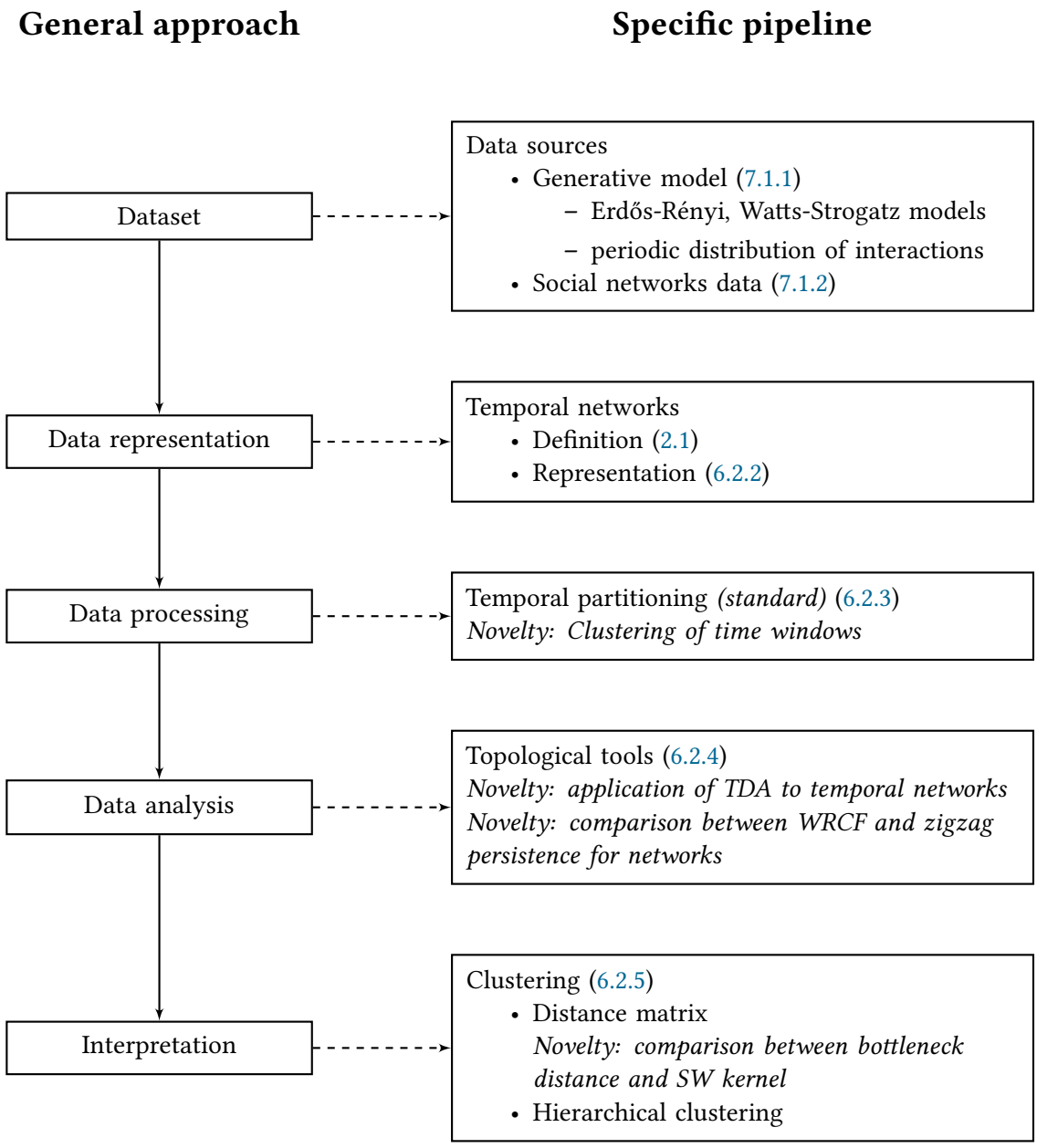
## 6.2 The analysis pipeline

### 6.2.1 General overview

The analysis pipeline consists in several steps:

1. Load the data: temporal networks are often distributed as *interaction lists*. In these files, each line consists of two nodes and a timestamp, and thus represents one contact interaction. One can reconstruct the temporal network by extracting all timestamps of a given edge and adding them as an edge property. It is then easy to extract a subnetwork within a specific time interval.
2. Interaction networks are sometimes directed. In these cases, it is necessary to transform the network into an undirected one, as most method (particularly topological methods, such as WRCF and zigzag persistence) only work on undirected networks.
3. Using the methods discussed in subsection 6.2.3, the temporal domain is segmented into sliding windows and a list of subnetworks can be generated.
4. Features are extracted from each subnetwork. These features can be constructed from different kinds of persistent homology on networks, as discussed in subsection 6.2.4.

Figure 6.2: Overview of the analysis pipeline. New approaches introduced in this study are highlighted in *italics*.



5. Depending on the methods used, the feature space is equipped with a metric that can make it a Hilbert space or a simple metric space. In any case, a distance matrix representing pairwise distances between each subnetwork is computed.
6. Hierarchical clustering is applied to the distance matrix.

The whole analysis pipeline is summarised in Figure 6.2.

### 6.2.2 Data representation

The data is represented in the algorithms as multigraphs. Each edge is associated to a timestamp (an integer). Two nodes can be linked by multiple edges, each one of them representing a time at which the edge is present.

This representation allows for easy filtering, as one can extract a temporal network in a given time interval by keeping only the edges whose timestamp is included in the interval. One can also build the underlying aggregated graph by "collapsing" multiple edges into a single one.

It is important to note that the nodes of the network are completely static and always present. This follows the temporal networks model adopted in Definition 2.3.

### 6.2.3 Sliding windows

As mentioned in subsection 6.1.2, we consider temporal networks whose temporal domain is a finite interval of $\mathbb{N}$.

For a temporal resolution $\Delta t$ and an overlap $s$, we compute the temporal partitioning as follows.

1. Compute the length of the temporal domain $\mathcal{T}$.
2. Segment it into $N$ sliding windows of length $\Delta t$ with an overlap $s$.
3. Each subnetwork in the sequence contains only the interactions appearing during the corresponding sliding window.

**Algorithm 2:** Temporal partitioning of network with sliding windows.

**Input:** Graph $G$, resolution res
**Output:** List of subnetworks windows
times $\leftarrow$ list of timestamps in $G$
window_length $\leftarrow$ res $\times$ (max(times) $-$ min(times))
**for** $i \leftarrow 0$ **to** $1/\text{res} - 1$ **do**
  windows $[i] \leftarrow$ subnetwork of $G$ containing all nodes, and edges whose timestamp is in $[\min(\text{times}) + \text{window\_length} \times i, \min(\text{times}) + \text{window\_length} \times (i+1)]$
**end**

### 6.2.4 Topological analysis

The major novelty in this analysis is to introduce topological features in temporal network analysis. The idea is that the techniques introduced in chapter 3 will reveal additional structure in networks that is not captured by traditional methods, and is relevant for detecting periodicity or other important properties of temporal networks.

Here, two different approaches are presented and compared. One is focusing on the topology of the aggregated graph, using weight-rank clique filtration, while the other leverages the temporal dimension by using the more recent advances in generalized persistence, specifically zigzag persistence.

#### 6.2.4.1 Aggregated graph persistence homology

The first possibility to introduce topological features into the feature map is to use weight-rank clique filtration (section 4.1) on the aggregated static graphs.

For this, we associate to each edge in the network a weight corresponding to the number of time steps in which it is present. For an edge $e$ and a time interval $c_i$ (keeping the notations of Definition 6.1), the weight associated to $e$ is

$$w(e) = \sum_{t \in c_i} \rho(e, t).$$

The resulting graph is called the *aggregated graph* of the temporal network on the time interval $c_i$. This graph being weighted, it is possible to compute persistence diagrams using weight-rank clique filtration (the algorithm is exposed in section 4.1).

#### 6.2.4.2 Zigzag persistence

The main drawback of WRCF persistence is the loss of any kind of temporal information in the network. Three nodes can be detected as being very close to one another even though their contacts might have been in separate time steps. We can avoid aggregating the temporal networks by using *generalised persistence*, specifically zigzag persistence as exposed in section 4.2.

In practice, zigzag persistence is more computationally expensive than WRCF persistence [13], and leads to lower number of topological features at every dimension. Aggregating networks tend to artificially create a lot of cliques that do not appear in the original temporal network.

To compute zigzag persistence, the algorithm needs the *maximal simplicial complex*, i.e. the union of all simplicial complexes in the sequence. In the case of temporal networks, this is the set of maximal cliques in the aggregated graph. Zigzag persistence can then be computed from the list of times when each simplex enters or leaves the maximal filtration. The following algorithm determines these times:

1. Determine the maximal simplicial complex by computing the cliques in the aggregated graph.
2. For each time step $t$:
   - Keep only the edges present at this time step (i.e. the edges $e$ such that $\rho(e, t) = 1$).
   - Compute all the cliques in this network.
3. For each clique in the maximal simplicial complex, determine where it is present and where it is absent in the sequence of lists of cliques.
4. Finally, determine the transition times in the presence arrays.

This computation can be quite costly for large networks, even before starting the main zigzag algorithm. Clique-finding is indeed an NP-complete problem [39]. This is thus by far the most computationally expensive step of the analysis pipeline, and is also more expensive than WRCF persistence.

### 6.2.5 Clustering

#### 6.2.5.1 Distance matrix

In order to cluster the subnetworks obtained by temporal partitioning of the original network, one needs to introduce a notion of distance between the topological features. Since the output of the previous step in the analysis pipeline take the form of persistence diagrams, two options are possible: a standard measure of distance between diagrams (see section 3.3), or one of the vectorization or kernel-based methods exposed in chapter 5.

One of the main contributions of this study is to compare the performance of the bottleneck distance (Definition 3.13) and of the sliced Wasserstein kernel (subsection 5.2.1) in the context of network clustering.

A distance matrix is obtained by computing pairwise distances between each pair of subnetworks obtained during the temporal partitioning step. One important remark is that the distances considered compute distances between persistence diagrams. However, persistence homology returns a *sequence* of such persistence diagrams for each subnetwork, each diagram in the sequence corresponding to topological features of a specific dimension. For the purposes of clustering, 0-dimensional features are not extremely interesting since they correspond to connected components, and 2 or 3-dimensional diagrams are often nearly empty except for very large subnetworks. It is therefore appropriate to restrict our analysis to 1-dimensional diagrams, which represent a good compromise. This is consistent with what has been done for point cloud data classification [15].

The bottleneck distance gives the space of persistence diagram a metric-space structure. Meanwhile, the sliced Wasserstein kernel gives the space a structure of a Hilbert space, which can be required for several machine-learning techniques, such as support-vector machines (SVMs) or principal components analysis (PCA).

For the implementation, we use the approximate computation of the sliced Wasserstein kernel (algorithm 1) sampled along 10 directions, which is actually faster in practice than the computation of the bottleneck distance. For the computation of the bottleneck distance, the diagram points that go to infinity are excluded. According to the definition of the bottleneck distance, if two diagrams do not have the same number of infinite points, the distance is automatically infinity, which does not work well in clustering algorithms. Moreover, this does not interfere with the comparison between the bottleneck distance and the sliced Wasserstein kernel, since infinite points are ignored by the kernel anyway.

#### 6.2.5.2 Hierarchical clustering

To simplify the interpretation of the analysis and the comparison between the different approaches, the clustering algorithm used is *hierarchical clustering* [28].

The main advantage is that it does not require knowing in advance the number of clusters that one is looking for. The only input is the dissimilarity matrix, obtained from a single metric. It is necessary here to use an algorithm that does not require the observations themselves, as in this case they take the form of a persistence diagram instead of a numeric vector. Moreover, kernel-based methods are not applicable to the bottleneck distance since it does not confer a Hilbert structure to the space of persistence diagram. By contrast, hierarchical clustering only requires a valid measure of distance.

The hierarchical representation (or *dendrogram*) is also especially useful in the context of periodicity detection, since periodicity can appear at various levels of the hierarchy.

Hierarchical clustering is performed in a bottom-up way, also called *agglomerative clustering*. Starting from the distance matrix, with each observation in its own cluster, the algorithm merges rows and columns at each step of the clustering, updating the distances between the new clusters. To do that, it needs to compute the distance between two clusters. Several approaches are possible to compute the distance between two clusters $A$ and $B$ using a metric $d$:

- Complete linkage: the distance between $A$ and $B$ is the maximum distance between their elements
$$\max \{d(x, y) : x \in A, y \in B\}$$
- Single linkage: using the minimum distance
$$\min \{d(x, y) : x \in A, y \in B\}$$

- Average linkage: using the mean distance between the elements

$$\frac{1}{|A||B|}\sum_{x\in A}\sum_{y\in B} d(x,y).$$

The implementation used is taken from the library Scikit-Learn [54].

# 7 Results and Discussion

## 7.1 Data

### 7.1.1 Generative model for periodic temporal networks

In order to detect periodicity, one can generate a random temporal network with a periodic structure.

We first build a random Erdős-Rényi graph. Starting from this base graph, we generate a temporal stream for each edge independently. This generative model is inspired by previous work on periodic temporal networks [61].

For each edge, we generate a sequence of times in a predefined time range $T$. For this, we choose uniformly at random a number of interactions $n$ in $[0, T/2]$. We then generate at random a sequence of $n$ times in $[0, T]$ from a density

$$f(t) = \sin(ft) + 1,$$

where $f$ is the frequency. The times are then sorted.

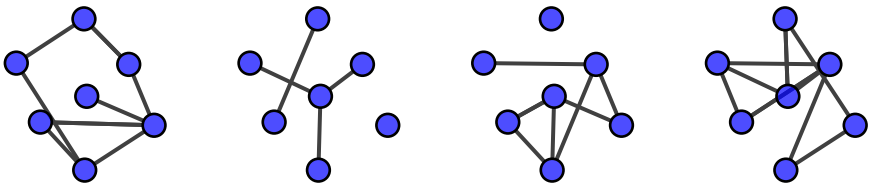


Figure 7.1: Example of a random temporal network generated by algorithm 3.

**Algorithm 3:** Random temporal network generation.

```
Input: nodes, edge_prob, time_range, frequency
Output: network
basegraph ← ErdősRényi(nodes, edge_prob)
network ← network with no edges and the vertices of basegraph
for e ∈ basegraph.edges do
    times ← random_edge_presences(time_range, frequency)
    for t ∈ times do
        Add (e.source, e.target, t) to network
    end
end
```

The complete method to generate a random network is summarised in algorithm 3. The function `random_edge_presences` returns a sequence of periodic times. An example of a small random network can be found on Figure 7.1.

### 7.1.2 Datasets

The SocioPatterns dataset [36] has been collected during the INFECTIOUS exhibition at the Science Gallery in Dublin, Ireland from April 17th to July 17th, 2009. During this event, a radio-frequency identification (RFID) device was embedded into each visitor's badge (as part of an interactive exhibit). RFID devices exchange radio packets when they are at a close range from each other (between 1 m and 1.5 m), in a peer-to-peer fashion. The data collection process is described in detail in [17].

The devices are configured so that face-to-face interactions between two individuals is accurately recorded with a probability of 99% over a period of 20 s, which is an appropriate time scale to record social interaction. False positives are also extremely rare as RFID devices have a very limited range and multiple radio packet exchanges are required to record an interaction.

The event in Dublin recorded more than 230,000 contact interactions between more than 14,000 visitors. The data is made available both in the form of daily-aggregated static networks [34] and as a list of contact interactions (each element being a timestamp and two nodes IDs) [35].

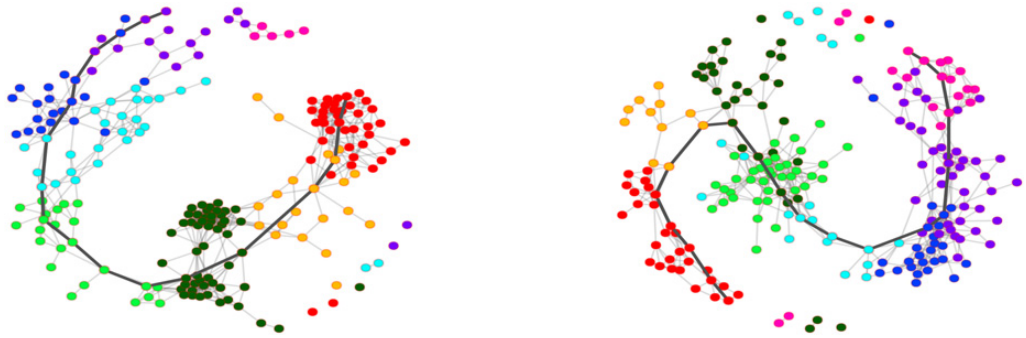


Figure 7.2: Aggregated networks for two different days of the SocioPatterns dataset. Nodes are colored from red to purple according to their arrival time. (Source: [36].)

The interactions times of the SocioPatterns dataset show that there are limited interactions between visitors entering the exhibition more than one hour apart (see Figure 7.2). A consequence of this is that the network diameter of the daily aggregated graphs connects visitors entering the venue at successive times, as can be seen in the figure.

Another interesting properties of these interactions is their lengths. Most of the interaction last less than one minute, as can be expected in the context of visitors in a museum exhibition. The distribution of the interaction durations shows broad tails, decaying slightly faster than a power law [36].

The temporal network has also been used in a variety of contexts from percolation analysis and dynamical spreading to community detection [36]. These studies have confirmed that topological criteria detect efficiently the edges acting as bridges between communities [26, 31].

Many empirical temporal networks exhibit periodic patterns [30]. Many papers have explored traditional network statistics and methods to uncover cyclic behaviour in various datasets, mainly telecommunication networks [3, 4, 29, 37].

Visualizations show significant variations in the patterns of the daily aggregated graphs between weekdays and weekends (see the SocioPatterns poster in appendix). This project will attempt to apply the topological methods on an empirical dataset to try to detect periodicity.

## 7.2 Computational environment

The analysis pipeline described in section 6.2 is entirely implemented in Python. For these tests, we use Python 3.5, with Numpy 1.15.1. The library Dionysus 2.0.7 [46] is used for persistent homology, zigzag persistence, and bottleneck distance. Networks are handled by igraph 0.7.1, and machine-learning algorithms are provided by Scikit-Learn 0.19.2 [54].

The program runs on a shared-memory system with 32 cores of 3.6 GHz, 756 GB of RAM, and 1.6 TB of storage. It runs Ubuntu Linux 16.04.5. Dionysus was compiled from the latest development version using GCC 5.4.0 with the optimization level -O3.

## 7.3 Results

### 7.3.1 Generative model

For this study, random networks have been generated with the following parameters (keeping the notations from subsection 7.1.1):

- the base graph $G$ is an Erdős-Rényi graph with 40 nodes and an edge probability of 90%,
- the total time range $T$ for the sequence of times is 200,
- the frequency $f$ is 15/200.

Figure 7.3 shows the density and sample times for a single edge. A series of presence times like this is generated for each edge in the base graph.

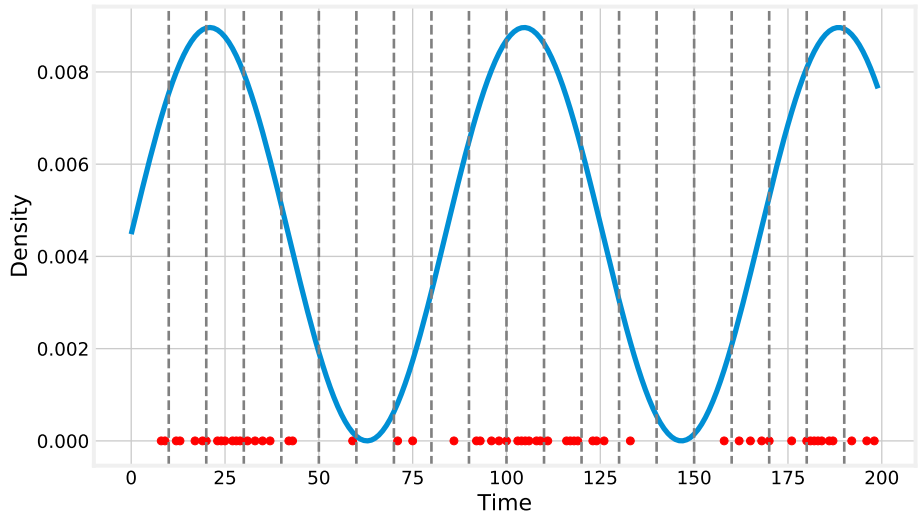



Figure 7.3: Example of periodic density for edge times generation (blue), with random edge times (red), and the sliding windows (grey).

The generated temporal network is then subdivided into 20 subnetworks. The sliding windows are also represented on Figure 7.3.

From these subnetworks, persistence is computed, in the form of persistence diagrams. An example can be found in Figure 7.4.

Figure 7.5 represents the output of hierarchical clustering for a random network, with zigzag and WRCF persistence, and the sliced Wasserstein kernel and the bottleneck distance. This clustering is representative of what is obtained by applying the same pipelines to many temporal networks generated by the random model of subsection 7.1.1.

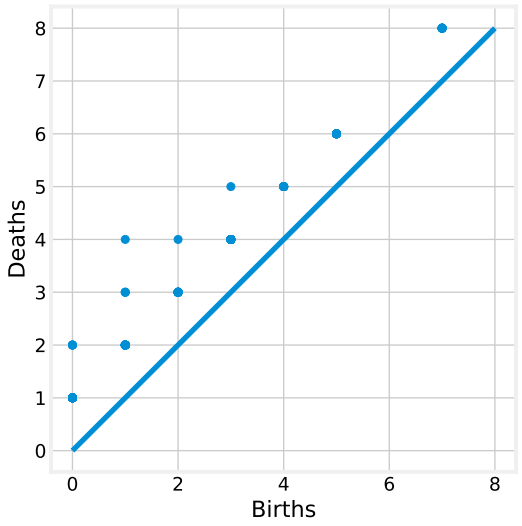
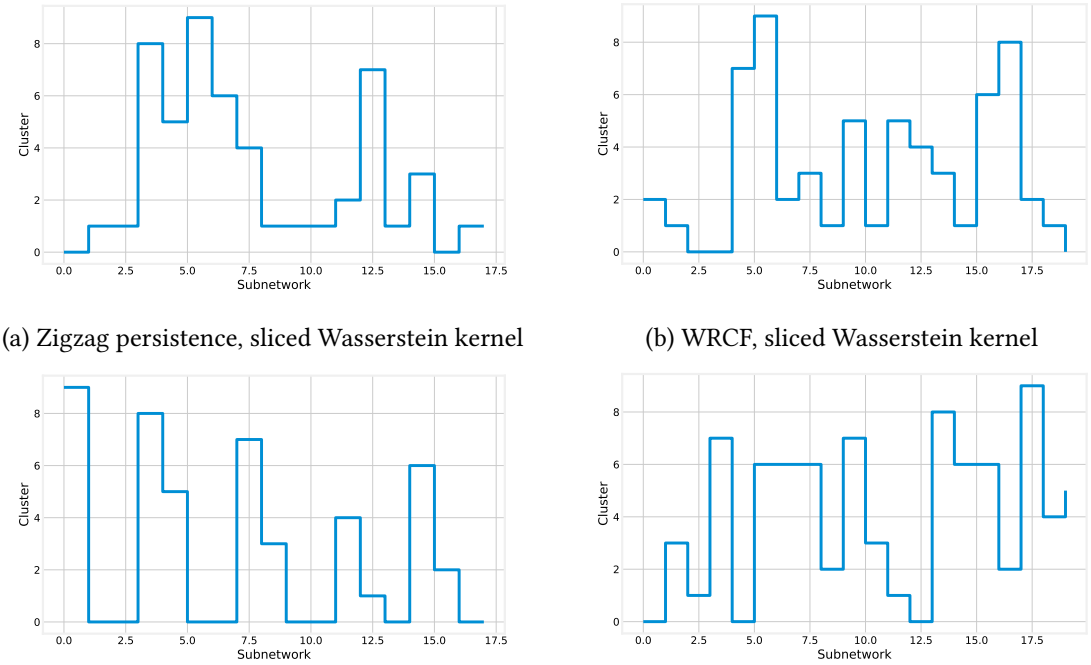



Figure 7.4: Example persistence diagram.




(a) Zigzag persistence, sliced Wasserstein kernel

(b) WRCF, sliced Wasserstein kernel

(c) Zigzag persistence, bottleneck distance

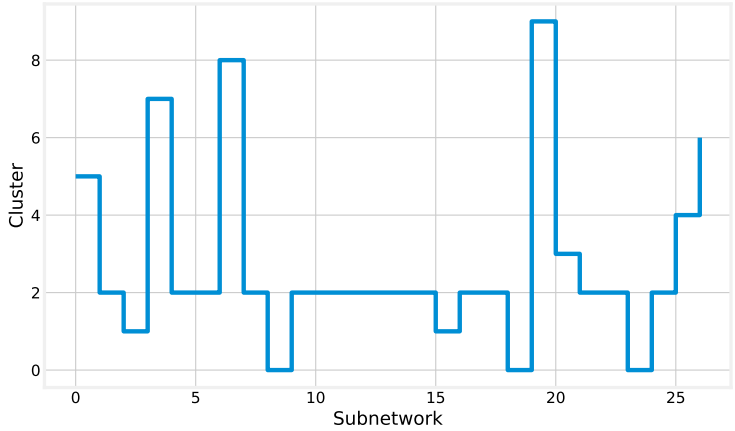
(d) WRCF, bottleneck distance

Figure 7.5: Hierarchical clustering with 10 clusters of a random temporal network.

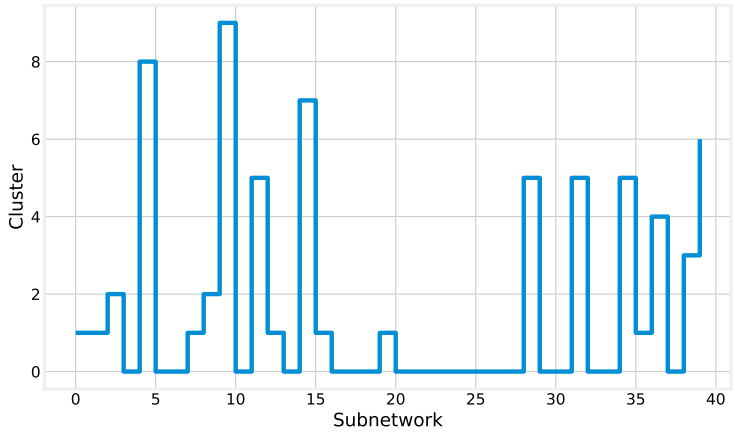
As we can see on the figure, the hierarchical clustering algorithm is able to determine the periodicity of the temporal networks when using the sliced Wasserstein kernel. However, with the simple bottleneck distance, the periodicity is not correctly detected. The periodicity detection can be confirmed by moving further up in the dendrogram of the clustering algorithm. With only 2 or 3 clusters, the low and high sections of the density (Figure 7.3) are still accurately classified with the sliced Wasserstein kernel, while the subnetworks are distributed randomly among the clusters with the bottleneck distance.

Somewhat less clear is the comparison between zigzag persistence and WRCF persistence. When generating many samples from the random temporal network model, WRCF and sliced Wasserstein kernel clustering is noisier and less consistent in its periodicity detection than its zigzag persistence counterpart. This indicates that the aggregation of the temporal subnetworks lead to the creation of artificial topological features that introduce noise in the dataset. (See subsubsection 6.2.4.2 for details on why aggregation introduce artificial simplices in the temporal network.)

### 7.3.2 SocioPatterns dataset




(a) Zigzag persistence, sliced Wasserstein kernel

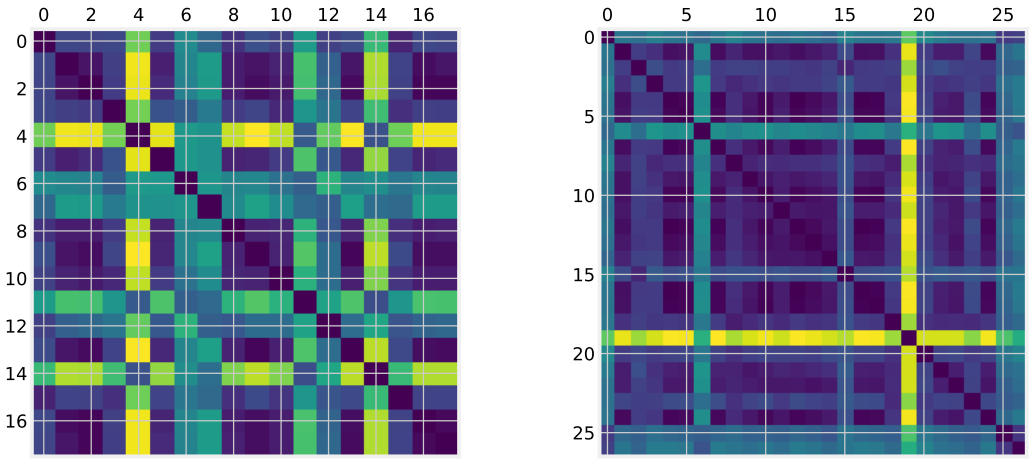



(b) WRCF, sliced Wasserstein kernel

Figure 7.6: Hierarchical clustering with 10 clusters of the SocioPatterns dataset.

In the study of the SocioPatterns dataset, we expect to uncover a periodicity on a scale of a day. Therefore, we would like to partition the time range of the dataset into windows approximately the length of a day. However, this leads to very sparse subnetworks, which do not exhibit enough topological features for a complete study. Since the main periodicity in the dataset is expected to be a weekday/weekend succession, we choose a resolution of two days.

The previous section has demonstrated that the sliced Wasserstein kernel was the most suitable to uncover periodicity in a temporal network. The results, with zigzag persistence and WRCF, of the hierarchical clustering algorithm are shown in Figure 7.6.




(a) Generative model

(b) SocioPatterns dataset

Figure 7.7: Gram matrices of the sliced Wasserstein kernel with zigzag persistence.

However, the subnetworks do not cluster periodically in either case. This is confirmed by visualizing the Gram matrix of the sliced Wasserstein kernel (Figure 7.7). The Gram matrix obtained with the generative model exhibit a cyclical pattern, while the one from the SocioPatterns dataset do not show enough distinctions between the subnetworks.

It is unclear whether this is due to the analysis pipeline, or to the temporal network itself not exhibiting enough periodicity on a topological level. To confirm this, one would need to compare our analysis with one using traditional network statistics, such as the ones in [3, 4, 29, 37]. Other empirical networks, such as telecommunication networks, may also exhibit more obvious cyclical patterns, where topological features might be useful.

# 8 Conclusions

## 8.1 Topological data analysis of temporal networks

Periodicity detection on our generative model has proven successful. More importantly, topological features and persistent homology seem to play an important part in the classification task. The general idea of partitioning the time range of a temporal network into sliding windows, and running an unsupervised clustering algorithm, works in the context of periodicity detection.

More generally, we have introduced persistent homology and topological data analysis methods for the study of temporal networks. Building on previous work clustering different temporal network generative models with persistent homology [61], we have expanded both the methods used and the applications, solving the real-world problem of periodicity detection. All in all, it is clear that persistent homology is a promising new direction for the study of temporal networks.

Topological data analysis is a recent field, with new methods and approaches being constantly developed and improved. In this project, we have compared different approaches. In the context of periodicity detection, zigzag persistence is a small improvement over the topological analysis of aggregated graphs using weight rank clique filtration. If this result was confirmed by other studies, it would be an interesting development, as it would imply that the temporal aspect is essential and cannot be discarded easily when studying temporal networks.

One of the most active research areas of topological data analysis has been its applications in machine learning. Considerable efforts have been deployed in the development of various vectorization techniques to embed topological information into a feature space suitable for statistical learning algorithms. In this project, a few of these methods have been compared for their theoretical guarantees, and their practical applications in periodicity detection. From a mathematical point of view, kernels seem the most promising approach, by offering strong stability properties and algebraic structures on the feature space. This development leads to a broader class of applications in machine learning where topological analysis can be useful. These theoretical advances have translated into much better results for periodicity detection. The simple bottleneck distance in the space of diagram (with a structure of metric space) was not able to determine any kind of periodicity in the random networks, whereas the sliced Wasserstein kernel (embedding persistence diagrams in its RKHS, with an metric equivalent to the distance in the space of persistence diagrams) picked up the period accurately. This confirms previous work on shape classification, where kernels on persistence diagrams significantly outperformed other feature embeddings [15, 42, 63].

Finally, we have tried to apply the same analysis to detect periodicity on real-world data, the SocioPatterns dataset. Our model was not able to detect a periodicity with a change of patterns between the weekdays and the weekends. This is unclear whether it is due to the limits in some part of our analysis pipeline, or to the periodicity in the network being non-topological in nature. A future study might focus on combining topological features with traditional network statistics.

## 8.2 Future work

Further study of topological features of temporal networks is needed. We could imagine other applications than periodicity detection, such as community detection [26]. Many standard methods are difficult to adapt to temporal network models, and computational topology could bring an additional perspective in these tasks, by complementing traditional network statistics.

In the specific context of periodicity detection, this analysis can be expanded by varying the parameters such as the resolution and the overlap. It could be especially useful for inferring the period in a temporal network.

One should also explore the other vectorization methods in the context of periodicity detection. It would be interesting to know how persistence images, or the other kernels, perform in this task. Last but not least, it is essential to compare the performance of the topological features with more traditional network statistics. It would also be interesting to combine both aspects and use both set of features in machine-learning tasks.

Finally, temporal networks seem to be the ideal context to apply multidimensional persistence. For instance, the weight rank clique filtration adds a "weight" dimension to the existing time dimension. In theory, it would be possible to use this by constructing a 2-parameter filtration on the network, and computing persistence on it.

# A Topology

In the following chapter, we recall a few essential definitions in topology. This is in large part taken from [27].

In this chapter, all vector spaces will be over a field $\mathbb{K}$, which is either the field of real numbers $\mathbb{R}$ or the field of complex numbers $\mathbb{C}$.

## A.1 Metric spaces

**Definition A.1** (Distance, metric space)**.** An application $d : X \times X \mapsto \mathbb{R}^+$ is a *distance* over $X$ if
(i) $\forall x, y \in X,\ d(x, y) = 0 \Leftrightarrow x = y$ (separation),
(ii) $\forall x, y \in X,\ d(x, y) = d(y, x)$ (symmetry),
(iii) $\forall x, y, z \in X,\ d(x, y) + d(y, z) \geq d(x, z)$ (triangle inequality).
In this case, $(X, d)$ is called a *metric space*.

If $Y$ is a subset of $X$ and $X$ is a metric space (with the distance $d$), then $(Y, d)$ is immediately a metric space itself. $d_Y$ is called the *induced metric* on $Y$.

If $(X, d)$ is metric space, then for all $x \in X$ and $r > 0$, the set

$$B(x, r) := \{y \in X : d(x, y) < r\}$$

is called the *open ball* centered at $x$ and of radius $r$. The *closed ball* centered at $x$ and of radius $r$ is defined by

$$B_c(x, r) := \{y \in X : d(x, y) \leq r\}.$$

An important class of metric spaces is the one where the set $X$ is itself a normed vector space.

**Definition A.2** (Norm)**.** Let $V$ be a vector space over $\mathbb{K}$. An application $N : V \mapsto \mathbb{R}^+$ is a *norm* over $V$ if
(i) $\forall x \in V,\ N(x) = 0 \Leftrightarrow x = 0$,
(ii) $\forall x \in V, \forall \lambda \in \mathbb{K},\ N(\lambda x) = |\lambda| N(x)$,
(iii) $\forall x, y \in V,\ N(x) + N(y) \geq N(x + Y)$.

Let $(V, N)$ be a normed vector space. For every subset $X$ of $V$, one can define $d(x, y) := N(x - y)$ for all $x, y \in X$. Using the properties of the norm $N$, one can check easily that $d$ is a distance, and therefore $(X, d)$ is a metric space.

There are many norms possible on a vector space. This brings the need to compare these various norms.

**Definition A.3** (Norm equivalence)**.** Let $V$ be a vector space. Two norms $N_1$ and $N_2$ on $V$ are said to be *equivalent* if there are two constants $C_1$ and $C_2$ such that

$$\forall x \in V, \quad N_1(x) \leq C_1 N_2(x) \quad \text{and} \quad N_2(x) \leq C_2 N_1(x).$$

Geometrically speaking, two norms are equivalent if the unit ball for the norm $N_1$ contains a non-empty ball centred at 0 for the norm $N_2$, and vice-versa.

## A.2 Completeness

**Definition A.4** (Convergence)**.** A sequence $(x_n)_{n\in\mathbb{N}}$ of elements of a metric space $(X, d)$ *converges* to a limit $x$ if

$$\lim_{n\to\infty} d(x_n, x) = 0.$$

**Definition A.5** (Cauchy sequence)**.** A sequence $(x_n)_{n\in\mathbb{N}}$ of elements of a metric space $(X, d)$ is a *Cauchy sequence* if

$$\forall \varepsilon > 0, \exists n_0 \in \mathbb{N}, \text{ such that: } \forall n, m \geq n_0,\ d(x_n, x_m) < \varepsilon.$$

Note that every convergent sequence is a Cauchy sequence, but the opposite is not true in general.

**Definition A.6** (Completeness)**.** A metric space $(X, d)$ is *complete* if, and only if, every Cauchy sequence converges to an element of $X$.

**Definition A.7** (Banach space)**.** A *Banach space* is a complete normed vector space.

## A.3 Hilbert spaces

In this section, vector spaces are defined over $\mathbb{C}$. The theory extends easily to vector spaces over $\mathbb{R}$.

An application $L$ between two $\mathbb{C}$-vector spaces $V$ and $W$ is said to be *anti-linear* if

$$\forall \lambda, \mu \in \mathbb{C}, \forall x, y \in V,\ L(\lambda x + \mu y) = \bar{\lambda} L(x) + \bar{\mu} y.$$

**Definition A.8** (Hermitian product)**.** An application $\langle \cdot, \cdot \rangle : V \times V \mapsto \mathbb{C}$ is

(i) a *sesquilinear form* if $x \mapsto \langle x, y\rangle$ is linear and $y \mapsto \langle x, y\rangle$ is anti-linear,
(ii) a *Hermitian form* if it is sesquilinear and $\langle x, y\rangle = \overline{\langle y, x\rangle}$ for all $x, y \in V$,
(iii) a *Hermitian product* if it is a Hermitian form positive definite, i.e. if $\langle x, x\rangle > 0$ for all $x \neq 0$.

*Remark.* In the case of vector spaces over $\mathbb{R}$, sesquilinear forms are simply bilinear, Hermitian forms are symmetric bilinear, and Hermitian products are inner products.

**Proposition A.1** (Cauchy-Schwartz inequality)**.** *Let $\langle \cdot, \cdot \rangle$ be a Hermitian product over $V$. Then, for all $x, y \in V$,*

$$|\langle x, y\rangle| \leq \sqrt{\langle x, x\rangle}\sqrt{\langle y, y\rangle},$$

*where the two sides are equal if and only if $x$ and $y$ are linearly dependent.*

*Proof.* Suppose that $x \neq 0$ and $y \neq 0$ (otherwise the proposition is obvious). For all $t > 0$, we compute

$$\langle x - ty, x - ty\rangle = \langle x, x\rangle - 2t\mathrm{Re}\langle x, y\rangle + t^2\langle y, y\rangle \geq 0.$$

Thus, for all $t > 0$,

$$2\mathrm{Re}\langle x, y\rangle \leq \frac{1}{t}\langle x, x\rangle + t\langle y, y\rangle.$$

We minimize the right-hand side by choosing

$$t = \sqrt{\frac{\langle x, x\rangle}{\langle y, y\rangle}},$$

thus

$$\mathrm{Re}\langle x, y\rangle \leq \sqrt{\langle x, x\rangle}\sqrt{\langle y, y\rangle}.$$

The inequality follows by replacing $x$ by $e^{i\theta}x$ and using the fact that

$$\forall z \in \mathbb{C},\ |z| = \sup_{\theta\in\mathbb{R}} \mathrm{Re}\left(e^{i\theta}z\right).$$

The equality case follows from setting $\langle x - ty, x - ty\rangle = 0$. □

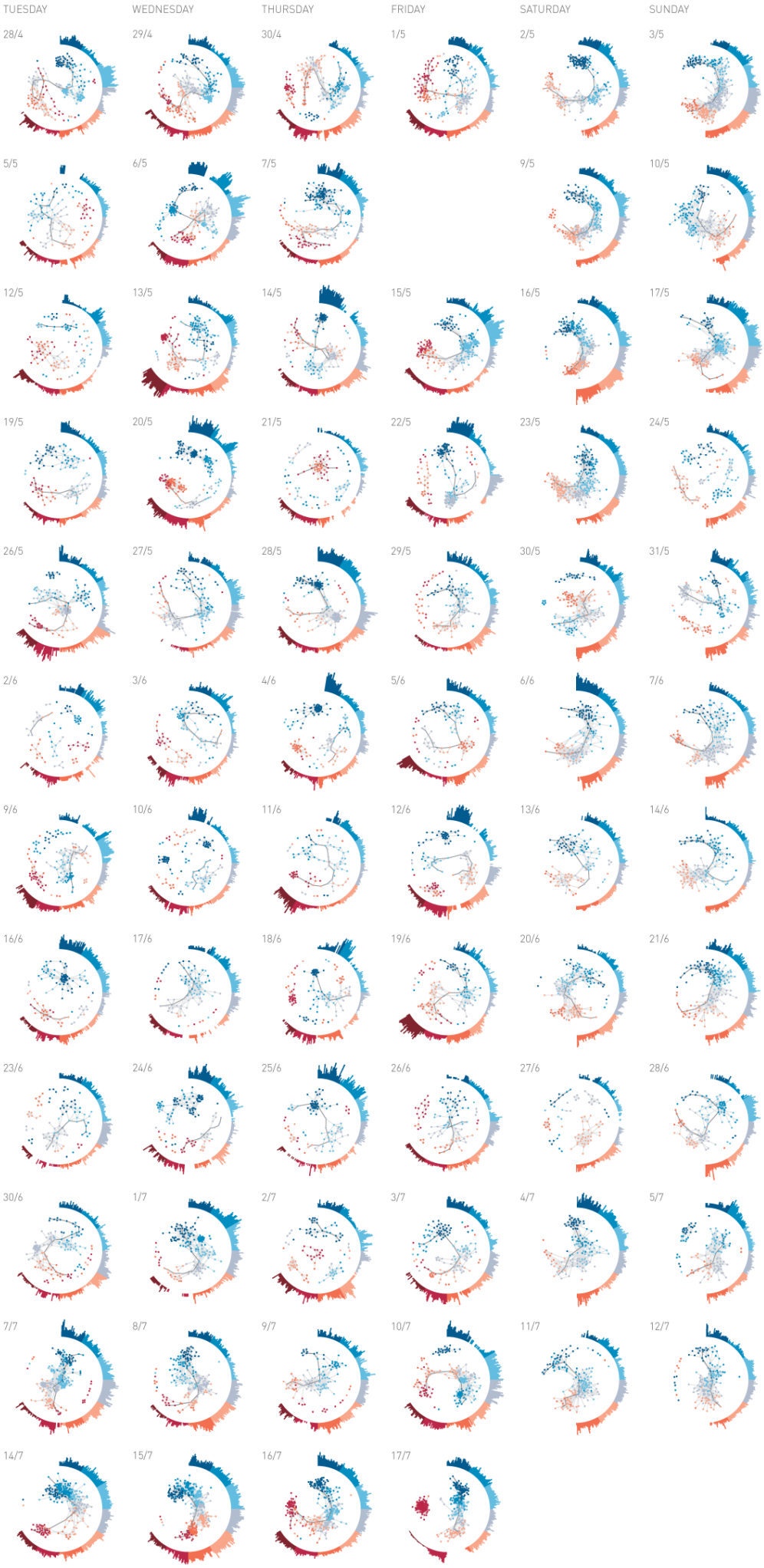
If $\langle\cdot,\cdot\rangle$ is a Hermitian product over $V$, it can be verified easily that the form

$$\|\cdot\| : x \mapsto \sqrt{\langle x, x\rangle}$$

is a norm over $V$. The triangle inequality comes from the Cauchy-Schwartz formula.

**Definition A.9** (Pre-Hilbert space)**.** A *pre-Hilbert space* is a vector space $V$ with a Hermitian product $\langle\cdot,\cdot\rangle$ and the associated norm $\|\cdot\|$. It is a metric space for the distance $d(x, y) := \|x - y\|$.

**Definition A.10** (Hilbert space)**.** A pre-Hilbert space $H$ is a *Hilbert space* if $(H, \|\cdot\|)$ is a Banach space.




# INFECTIOUS SOCIOPATTERNS

## SIXTY-NINE DAYS OF CLOSE ENCOUNTERS AT THE SCIENCE GALLERY

In the spring of 2009, the Science Gallery in Dublin, Ireland, held the artscience exhibition INFECTIOUS: STAY AWAY, which explored mechanisms of contagion and strategies of containment. The visitors of this exhibition were invited to participate in INFECTIOUS SOCIOPATTERNS, a simulated epidemic of an electronic infectious agent that spreads through close-range proximity of individuals. The simulation system was built on top of the SocioPatterns sensing platform. This platform uses wearable electronic badges to sense sustained face-to-face proximity –a proxy observable for social contact– among the participants.

More than 30,000 visitors participated in INFECTIOUS SOCIOPATTERNS over the three-month course of the exhibition. All sensory data generated by the system during this period was gathered and stored for use in the scientific research of the SocioPatterns project.

The collected data also served as input for SIXTY-NINE DAYS OF CLOSE ENCOUNTERS AT THE SCIENCE GALLERY. This visualization is structured in a six by twelve grid of daily diagrams. The columns span Tuesdays to Sundays –Monday is the closing day of the venue– while the rows span twelve weeks of INFECTIOUS: STAY AWAY


For each day, a cumulative contact graph is shown in which nodes represent visitors and edges connect individuals who spent time in face-to-face proximity. Nodes are color-coded according to their time of arrival at the venue, as shown in panel S1. The diameter of the contact graph, i.e., the maximal shortest path between node pairs, is highlighted as shown in panel S2.

The contact graph is wrapped by a circular bar chart that displays the recorded number of social encounters over two-minute intervals, as shown in panel S3. The position and angle of the bars are those of the hour hand of a 12-hour clock at the corresponding time. The colors of the bars match those used for the arrival time of nodes in the graph, so that the bar chart also serves as the legend for the color-coding of time.

More details on the data collection process and the properties of the contact graphs shown here can be found in the Journal of Theoretical Biology [ "What's in a crowd? Analysis of face-to-face behavioral networks", by Lorenzo Isella, Juliette Stehlé, Alain Barrat, Ciro Cattuto, Jean-François Pinton, and Wouter Van den Broeck, Journal of Theoretical Biology 271, 166-180 (2011) ]

This visualization was created by Wouter Van den Broeck and Marco Quaggiotto, in collaboration with Lorenzo Isella, Ciro Cattuto and Alain Barrat. This work is part of the SocioPatterns project (www.sociopatterns.org), with support from the ISI Foundation in Turin, Italy (www.isi.it), and the Science Gallery in Dublin, Ireland (www.sciencegallery.com). © 2011, sociopatterns.org

ISI INSTITUTE FOR SCIENTIFIC INTERCHANGE FOUNDATION

SCIENCE GALLERY

# C Code

## C.1 zigzag.py

```
#!/usr/bin/env python3

import numpy as np
import igraph as ig
import dionysus as d


def sliding_windows(g, res=0.1, overlap=0):
    """Compute subnetworks of a temporal network based on temporal
    partitioning of the time range.

    :param g: igraph Graph
    :param res: resolution
    :param overlap: overlap

    :return: a list of temporal networks.
    """
    times = np.array(g.es["time"])
    duration = res * (times.max() - times.min())
    windows = []
    for i in range(int(1/res)):
        edges = g.es.select(time_gt=times.min() + duration*i,
                            time_lt=times.min() + duration*(i+1))
        windows.append(g.subgraph_edges(edges))
    return windows


def max_simplicial_complex(g):
    """Return the maximal simplicial complex of a network g.
    """
    return d.Filtration(g.maximal_cliques())


def find_transitions(a):
    """Find the transition times in an array of presence times.
    """
    res = []
    prev = False
    for i, cur in enumerate(a):
        if cur != prev:
            res.append(i)
        prev = cur
    return res


def presence_times(g):
    """Compute the data required to compute zigzag persistence:
    simplicial complex and transition times.
```

```
    :param g: igraph Graph

    :return: a tuple with the maximum simplicial complex and the
             transition times of each simplex.
    """
    max_simplicial_complex = d.Filtration(g.cliques())
    filts = []
    for t in np.sort(np.unique(g.es["time"])):
        edges = g.es.select(time_eq=t)
        cliques = g.subgraph_edges(edges).cliques()
        filts.append(d.Filtration(cliques))
    presences = [[s in filt for filt in filts] for s in max_simplicial_complex]
    presences = [find_transitions(p) for p in presences]
    return (max_simplicial_complex, presences)


def zigzag_network(g):
    """Compute zigzag persistence on a temporal network.

    :param g: igraph Graph

    :return: a list of persistence diagrams.
    """
    (f, t) = presence_times(g)
    _, dgms, _ = d.zigzag_homology_persistence(f, t)
    return dgms
```

## C.2 wrcf.py

```
#!/usr/bin/env python3

import numpy as np
import igraph as ig
import dionysus as d


def wrcf(G, weight="weight"):
    """Compute the weight-rank clique filtration (WRCF) of a graph.

    :param G: igraph Graph
    :param weight: name of the weight attribute

    :return: a Dionysus filtration.
    """
    # Define filtration step 0 as the set of all nodes
    filt = d.Filtration()
    for v in G.vs:
        filt.append(d.Simplex([v.index], 0))
    # Rank all edge weights
    distinct_weights = np.unique(G.es[weight])[::-1]
    for t, w in enumerate(distinct_weights):
        # At filtration step t, threshold the graph at weight[t]
        subg = G.subgraph_edges(G.es(lambda e: e[weight] >= w))
        # Find all maximal cliques and define them to be simplices
        for clique in subg.maximal_cliques():
            for s in d.closure([d.Simplex(clique)], len(clique)):
                filt.append(d.Simplex(s, t+1))
    filt.sort()
    return(filt)
```

```
def wrcf_diagram(graph, weight="weight"):
    """Compute persistence diagrams of a graph using WRCF.

    :param graph: igraph Graph
    :param weight: name of the weight attribute

    :return: a list of persistence diagrams.
    """
    filt = wrcf(graph, weight=weight)
    pers = d.homology_persistence(filt)
    dgms = d.init_diagrams(pers, filt)
    return dgms
```

## C.3 sliced_wasserstein.py

```
import numpy as np
import dionysus as d


def diagram_array(dgm):
    """Convert a Dionysus diagram to a Numpy array.

    :param dgm: Dionysus Diagram

    :return: a Numpy array of tuples representing the points in the
             diagram.
    """
    res = []
    for p in dgm:
        if p.death != np.inf:
            res.append([p.birth, p.death])
    return np.array(res)


def SW_approx(dgm1, dgm2, M):
    """Approximate computation of the Sliced Wasserstein kernel.

    :param dgm1: first Diagram
    :param dgm2: second Diagram
    :param M int: number of directions

    :return: The approximate value of the Sliced Wasserstein kernel of
             dgm1 and dgm2, sampled over M dimensions.
    """
    dgm1 = diagram_array(dgm1)
    dgm2 = diagram_array(dgm2)
    if dgm1.size == 0 or dgm2.size == 0:
        return 0
    # Add \pi_\delta(dgm1) to dgm2 and vice-versa
    proj1 = dgm1.dot([1, 1])/np.sqrt(2)
    proj2 = dgm2.dot([1, 1])/np.sqrt(2)
    dgm1 = np.vstack((dgm1, np.vstack((proj2, proj2)).T))
    dgm2 = np.vstack((dgm2, np.vstack((proj1, proj1)).T))
    SW = 0
    theta = -np.pi/2
    s = np.pi/M
    for i in range(M):
        # Project each diagram on the direction theta
```

```
            vec = [1, np.arctan(theta)]
            vec = vec / np.linalg.norm(vec)
            V1 = dgm1.dot(vec)
            V2 = dgm2.dot(vec)
            # Sort the projections
            V1.sort()
            V2.sort()
            # l1-distance between the projections
            SW = SW + s * np.sum(np.abs(V1 - V2))
            theta = theta + s
        return 1/np.pi * SW
```

## C.4 generative.py

```
#!/usr/bin/env python3

import numpy as np
import igraph as ig
import dionysus as d

import multiprocessing
# from dask.distributed import Client

from zigzag import sliding_windows, zigzag_network
from wrcf import wrcf_diagram
from sliced_wasserstein import diagram_array, SW_approx

import dill


def random_edge_presences(T, f):
    """Generate random times sampled over a periodic distribution.

    :param T: time range
    :param f: frequency

    :return: an array of times.
    """
    density = np.sin(f * np.arange(T)) + 1
    density /= np.sum(density)
    samplesize = np.random.randint(T//2)
    times = np.random.choice(np.arange(T), size=samplesize, replace=False, p=density)
    times = np.sort(times)
    return times


def remove_inf(dgm):
    """Remove infinite points in a persistence diagram.

    :param dgm: Diagram

    :return: the same diagram without the infinite points.
    """
    res = d.Diagram()
    for p in dgm:
        if p.death != np.inf:
            res.append(p)
    return res
```

```
## Global parameters
NODES = 40
EDGE_PROB = 0.9
TIME_RANGE = 200
FREQ = 15/TIME_RANGE
N_WINDOWS = 20

## Computations
ZIGZAG_PERS = True
WRCF_PERS = True
SW_KERNEL = True
BOTTLENECK_DIST = True


if __name__=="__main__":
    print("Generating random temporal network...", end="", flush=True)
    basegraph = ig.Graph.Erdos_Renyi(NODES, EDGE_PROB)
    g = ig.Graph()
    g.add_vertices(len(basegraph.vs))
    for e in basegraph.es:
        times = random_edge_presences(TIME_RANGE, FREQ)
        for t in times:
            g.add_edge(e.source, e.target, time=t)
    print("done.")

    print("Temporal partitioning...", end="", flush=True)
    wins = sliding_windows(g, 1/N_WINDOWS)
    print("done.")

    pool = multiprocessing.Pool(processes=multiprocessing.cpu_count())

    if ZIGZAG_PERS:
        print("Zigzag persistence...", end="", flush=True)
        zz_dgms = pool.map(zigzag_network, wins)
        dill.dump(zz_dgms, open("generative/zz_dgms.dill", "wb"))
        print("done, saved.")

    if WRCF_PERS:
        print("WRCF...", end="", flush=True)
        ## Collapse each subnetwork into a static graph: the weight is the
        ## number of appearances of each edge
        for w in wins:
            w.es["time"] = np.repeat(1, len(w.es["time"]))
            w.simplify(combine_edges="sum")
            w.es["weight"] = w.es["time"]
            del w.es["time"]
        wrcf_dgms = pool.map(wrcf_diagram, wins)
        dill.dump(wrcf_dgms, open("generative/wrcf_dgms.dill", "wb"))
        print("done.")

    pool.terminate()

    if ZIGZAG_PERS and SW_KERNEL:
        print("Sliced Wasserstein Kernel (zigzag)...", end="", flush=True)
        zz_dgms1 = [dgm[1] for dgm in zz_dgms if len(dgm) > 1]
        zz_gram1 = np.array([[SW_approx(zz_dgms1[i], zz_dgms1[j], 10)
                              for i in range(len(zz_dgms1))] for j in range(len(zz_dgms1))])
        dill.dump(zz_gram1, open("generative/zz_gram1.dill", "wb"))
        print("done, saved.")
    if WRCF_PERS and SW_KERNEL:
        print("Sliced Wasserstein Kernel (WRCF)...", end="", flush=True)
```

```
        wrcf_dgms1 = [dgm[1] for dgm in wrcf_dgms if len(dgm) > 1]
        wrcf_gram1 = np.array([[SW_approx(wrcf_dgms1[i], wrcf_dgms1[j], 10)
                                for i in range(len(wrcf_dgms1))] for j in range(len(wrcf_dgms1))])
        dill.dump(wrcf_gram1, open("generative/wrcf_gram1.dill", "wb"))
        print("done, saved.")

    if ZIGZAG_PERS and BOTTLENECK_DIST:
        print("Bottleneck distance (zigzag)...", end="", flush=True)
        zz_dgms1 = list(map(remove_inf, zz_dgms1))
        zz_distmat = np.array([[d.bottleneck_distance(zz_dgms1[i], zz_dgms1[j])
                                for i in range(len(zz_dgms1))] for j in range(len(zz_dgms1))])
        dill.dump(zz_distmat, open("generative/zz_distmat.dill", "wb"))
        print("done, saved.")
    if WRCF_PERS and BOTTLENECK_DIST:
        print("Bottleneck distance (WRCF)...", end="", flush=True)
        wrcf_dgms1 = list(map(remove_inf, wrcf_dgms1))
        wrcf_distmat = np.array([[d.bottleneck_distance(wrcf_dgms1[i], wrcf_dgms1[j])
                                  for i in range(len(wrcf_dgms1))] for j in range(len(wrcf_dgms1))])
        dill.dump(wrcf_distmat, open("generative/wrcf_distmat.dill", "wb"))
        print("done, saved.")
```

## C.5 sociopatterns.py

```
#!/usr/bin/env python3

import numpy as np
import igraph as ig
import dionysus as d

import multiprocessing
# from dask.distributed import Client

from zigzag import sliding_windows, zigzag_network
from wrcf import wrcf_diagram
from sliced_wasserstein import diagram_array, SW_approx

import dill


def remove_inf(dgm):
    """Remove infinite points in a persistence diagram.

    :param dgm: Diagram

    :return: the same diagram without the infinite points.
    """
    res = d.Diagram()
    for p in dgm:
        if p.death != np.inf:
            res.append(p)
    return res


## Global parameters
N_WINDOWS = 40

## Computations
ZIGZAG_PERS = True
WRCF_PERS = True
SW_KERNEL = True
```

```
BOTTLENECK_DIST = True


if __name__=="__main__":
    print("Loading SocioPatterns dataset...", end="", flush=True)
    g = ig.read("data/sociopatterns/infectious/infectious.graphml")
    del g.es["id"]
    # print(g.summary())
    print("done.")

    print("Temporal partitioning...", end="", flush=True)
    wins = sliding_windows(g, 1/N_WINDOWS)
    print("done.")

    pool = multiprocessing.Pool(processes=multiprocessing.cpu_count())

    if ZIGZAG_PERS:
        print("Zigzag persistence...", end="", flush=True)
        zz_dgms = pool.map(zigzag_network, wins)
        dill.dump(zz_dgms, open("sociopatterns/zz_dgms.dill", "wb"))
        print("done, saved.")

    if WRCF_PERS:
        print("WRCF...", end="", flush=True)
        ## Collapse each subnetwork into a static graph: the weight is the
        ## number of appearances of each edge
        for w in wins:
            w.es["time"] = np.repeat(1, len(w.es["time"]))
            w.simplify(combine_edges="sum")
            w.es["weight"] = w.es["time"]
            del w.es["time"]
        wrcf_dgms = pool.map(wrcf_diagram, wins)
        dill.dump(wrcf_dgms, open("sociopatterns/wrcf_dgms.dill", "wb"))
        print("done.")

    pool.terminate()

    if ZIGZAG_PERS and SW_KERNEL:
        print("Sliced Wasserstein Kernel (zigzag)...", end="", flush=True)
        zz_dgms1 = [dgm[1] for dgm in zz_dgms if len(dgm) > 1]
        zz_gram1 = np.array([[SW_approx(zz_dgms1[i], zz_dgms1[j], 10)
                              for i in range(len(zz_dgms1))] for j in range(len(zz_dgms1))])
        dill.dump(zz_gram1, open("sociopatterns/zz_gram1.dill", "wb"))
        print("done, saved.")
    if WRCF_PERS and SW_KERNEL:
        print("Sliced Wasserstein Kernel (WRCF)...", end="", flush=True)
        wrcf_dgms1 = [dgm[1] for dgm in wrcf_dgms if len(dgm) > 1]
        wrcf_gram1 = np.array([[SW_approx(wrcf_dgms1[i], wrcf_dgms1[j], 10)
                                for i in range(len(wrcf_dgms1))] for j in range(len(wrcf_dgms1))])
        dill.dump(wrcf_gram1, open("sociopatterns/wrcf_gram1.dill", "wb"))
        print("done, saved.")

    if ZIGZAG_PERS and BOTTLENECK_DIST:
        print("Bottleneck distance (zigzag)...", end="", flush=True)
        zz_dgms1 = list(map(remove_inf, zz_dgms1))
        zz_distmat = np.array([[d.bottleneck_distance(zz_dgms1[i], zz_dgms1[j])
                                for i in range(len(zz_dgms1))] for j in range(len(zz_dgms1))])
        dill.dump(zz_distmat, open("sociopatterns/zz_distmat.dill", "wb"))
        print("done, saved.")
    if WRCF_PERS and BOTTLENECK_DIST:
        print("Bottleneck distance (WRCF)...", end="", flush=True)
```

```
        wrcf_dgms1 = list(map(remove_inf, wrcf_dgms1))
        wrcf_distmat = np.array([[d.bottleneck_distance(wrcf_dgms1[i], wrcf_dgms1[j])
                                  for i in range(len(wrcf_dgms1))] for j in range(len(wrcf_dgms1))])
        dill.dump(wrcf_distmat, open("sociopatterns/wrcf_distmat.dill", "wb"))
        print("done, saved.")
```

## C.6 clustering.py

```
#!/usr/bin/env python3

import numpy as np

from sklearn.cluster import AgglomerativeClustering
from sklearn.svm import OneClassSVM

import dill

import matplotlib
matplotlib.use("PDF")
import matplotlib.pyplot as plt
plt.style.use("fivethirtyeight")
plt.rcParams["figure.figsize"] = (10, 6)


N_CLUSTERS = 10

GENERATIVE = True
SOCIOPATTERNS = True

if __name__=="__main__":
    if GENERATIVE:
        print("==== Generative model ====")
        zz_dgms = dill.load(open("generative/zz_dgms.dill", "rb"))
        wrcf_dgms = dill.load(open("generative/wrcf_dgms.dill", "rb"))
        zz_gram1 = dill.load(open("generative/zz_gram1.dill", "rb"))
        wrcf_gram1 = dill.load(open("generative/wrcf_gram1.dill", "rb"))
        zz_distmat = dill.load(open("generative/zz_distmat.dill", "rb"))
        wrcf_distmat = dill.load(open("generative/wrcf_distmat.dill", "rb"))

        print("Zigzag + kernel")
        clf = AgglomerativeClustering(
            n_clusters=N_CLUSTERS, affinity='precomputed', linkage='average')
        clf.fit(zz_gram1)
        fig, ax = plt.subplots()
        ax.step(range(len(clf.labels_)), clf.labels_, where='post')
        ax.set_xlabel("Subnetwork")
        ax.set_ylabel("Cluster")
        fig.savefig("fig/gen_zz_k.pdf", transparent=True,
                    pad_inches=0.3, bbox_inches="tight")

        print("WRCF + kernel")
        clf = AgglomerativeClustering(
            n_clusters=N_CLUSTERS, affinity='precomputed', linkage='average')
        clf.fit(wrcf_gram1)
        fig, ax = plt.subplots()
        ax.step(range(len(clf.labels_)), clf.labels_, where='post')
        ax.set_xlabel("Subnetwork")
        ax.set_ylabel("Cluster")
        fig.savefig("fig/gen_wrcf_k.pdf", transparent=True,
                    pad_inches=0.3, bbox_inches="tight")
```

```
        print("Zigzag + bottleneck")
        clf = AgglomerativeClustering(
            n_clusters=N_CLUSTERS, affinity='precomputed', linkage='average')
        clf.fit(zz_distmat)
        fig, ax = plt.subplots()
        ax.step(range(len(clf.labels_)), clf.labels_, where='post')
        ax.set_xlabel("Subnetwork")
        ax.set_ylabel("Cluster")
        fig.savefig("fig/gen_zz_b.pdf", transparent=True,
                    pad_inches=0.3, bbox_inches="tight")

        print("WRCF + bottleneck")
        clf = AgglomerativeClustering(
            n_clusters=N_CLUSTERS, affinity='precomputed', linkage='average')
        clf.fit(wrcf_distmat)
        fig, ax = plt.subplots()
        ax.step(range(len(clf.labels_)), clf.labels_, where='post')
        ax.set_xlabel("Subnetwork")
        ax.set_ylabel("Cluster")
        fig.savefig("fig/gen_wrcf_b.pdf", transparent=True,
                    pad_inches=0.3, bbox_inches="tight")

    if SOCIOPATTERNS:
        print("==== SocioPatterns dataset ====")
        zz_dgms = dill.load(open("sociopatterns/zz_dgms.dill", "rb"))
        wrcf_dgms = dill.load(open("sociopatterns/wrcf_dgms.dill", "rb"))
        zz_gram1 = dill.load(open("sociopatterns/zz_gram1.dill", "rb"))
        wrcf_gram1 = dill.load(open("sociopatterns/wrcf_gram1.dill", "rb"))
        zz_distmat = dill.load(open("sociopatterns/zz_distmat.dill", "rb"))
        wrcf_distmat = dill.load(open("sociopatterns/wrcf_distmat.dill", "rb"))

        print("Zigzag + kernel")
        clf = AgglomerativeClustering(n_clusters=N_CLUSTERS, affinity='precomputed', linkage='average')
        clf.fit(zz_gram1)
        fig, ax = plt.subplots()
        ax.step(range(len(clf.labels_)), clf.labels_, where='post')
        ax.set_xlabel("Subnetwork")
        ax.set_ylabel("Cluster")
        fig.savefig("fig/sp_zz_k.pdf", transparent=True, pad_inches=0.3, bbox_inches="tight")

        print("WRCF + kernel")
        clf = AgglomerativeClustering(n_clusters=N_CLUSTERS, affinity='precomputed', linkage='average')
        clf.fit(wrcf_gram1)
        fig, ax = plt.subplots()
        ax.step(range(len(clf.labels_)), clf.labels_, where='post')
        ax.set_xlabel("Subnetwork")
        ax.set_ylabel("Cluster")
        fig.savefig("fig/sp_wrcf_k.pdf", transparent=True, pad_inches=0.3, bbox_inches="tight")

        print("Zigzag + bottleneck")
        clf = AgglomerativeClustering(n_clusters=N_CLUSTERS, affinity='precomputed', linkage='average')
        clf.fit(zz_distmat)
        fig, ax = plt.subplots()
        ax.step(range(len(clf.labels_)), clf.labels_, where='post')
        ax.set_xlabel("Subnetwork")
        ax.set_ylabel("Cluster")
        fig.savefig("fig/sp_zz_b.pdf", transparent=True, pad_inches=0.3, bbox_inches="tight")

        print("WRCF + bottleneck")
        clf = AgglomerativeClustering(n_clusters=N_CLUSTERS, affinity='precomputed', linkage='average')
```

```
clf.fit(wrcf_distmat)
fig, ax = plt.subplots()
ax.step(range(len(clf.labels_)), clf.labels_, where='post')
ax.set_xlabel("Subnetwork")
ax.set_ylabel("Cluster")
fig.savefig("fig/sp_wrcf_b.pdf", transparent=True, pad_inches=0.3, bbox_inches="tight")
```

# *Bibliography*


[1] Henry Adams, Tegan Emerson, Michael Kirby, et al. "Persistence Images: A Stable Vector Representation of Persistent Homology". In: *Journal of Machine Learning Research* 18.8 (2017), pp. 1–35 (cit. on pp. 14, 15).

[2] Robert J. Adler, Omer Bobrowski, Matthew S. Borman, et al. *Persistent homology for random fields and complexes*. Institute of Mathematical Statistics, 2010. ISBN: 978-0-940600-79-9. DOI: 10.1214/10-IMSCOLL609 (cit. on p. 14).

[3] Talayeh Aledavood, Sune Lehmann, and Jari Saramäki. "Digital daily cycles of individuals". In: *Frontiers in Physics* 3 (2015). ISSN: 2296-424X. DOI: 10.3389/fphy.2015.00073 (cit. on pp. 28, 32).

[4] Talayeh Aledavood, Eduardo López, Sam G. B. Roberts, et al. "Daily Rhythms in Mobile Telephone Communication". In: *PLOS ONE* 10.9 (Sept. 21, 2015), e0138098. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0138098 (cit. on pp. 28, 32).

[5] Danielle S. Bassett and Olaf Sporns. "Network neuroscience". In: *Nature Neuroscience* 20.3 (Mar. 2017), pp. 353–364. ISSN: 1546-1726. DOI: 10.1038/nn.4502 (cit. on p. 1).

[6] Ulrich Bauer. *ripser: Ripser: a lean C++ code for the computation of Vietoris–Rips persistence barcodes*. original-date: 2015-10-27T21:43:59Z. Apr. 3, 2018 (cit. on p. 11).

[7] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. "Distributed Computation of Persistent Homology". In: *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. Ed. by Catherine C. McGeoch and Ulrich Meyer. Philadelphia, PA: Society for Industrial and Applied Mathematics, May 2014, pp. 31–38. ISBN: 978-1-61197-319-8. DOI: 10.1137/1.9781611973198.4 (cit. on p. 11).

[8] Marya Bazzi, Lucas G. S. Jeub, Alex Arenas, et al. "Generative Benchmark Models for Mesoscale Structure in Multilayer Networks". In: *arXiv:1608.06196 [cond-mat, physics:nlin, physics:physics, stat]* (Aug. 22, 2016). arXiv: 1608.06196 (cit. on p. 1).

[9] Alain Berlinet and Christine Thomas-Agnan. *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Google-Books-ID: bX3TBwAAQBAJ. Springer Science & Business Media, June 28, 2011. 369 pp. ISBN: 978-1-4419-9096-9 (cit. on p. 16).

[10] Peter Bubenik. "Statistical Topological Data Analysis using Persistence Landscapes". In: *Journal of Machine Learning Research* 16 (2015), pp. 77–102 (cit. on pp. 14, 15).

[11] Gunnar Carlsson. "Topology and data". In: *Bulletin of the American Mathematical Society* 46.2 (Jan. 29, 2009), pp. 255–308. ISSN: 0273-0979. DOI: 10.1090/S0273-0979-09-01249-X (cit. on pp. 1, 9).

[12] Gunnar Carlsson and Vin de Silva. "Zigzag Persistence". In: *arXiv:0812.0197 [cs]* (Nov. 30, 2008). arXiv: 0812.0197 (cit. on p. 13).

[13] Gunnar Carlsson, Vin de Silva, and Dmitriy Morozov. "Zigzag Persistent Homology and Real-valued Functions". In: *Proceedings of the Twenty-fifth Annual Symposium on Computational Geometry*. SCG '09. New York, NY, USA: ACM, 2009, pp. 247–256. ISBN: 978-1-60558-501-7. DOI: 10.1145/1542362.1542408 (cit. on pp. 13, 24).

[14] Gunnar Carlsson and Afra Zomorodian. "The Theory of Multidimensional Persistence". In: *Discrete & Computational Geometry* 42.1 (July 1, 2009), pp. 71–93. ISSN: 0179-5376, 1432-0444. DOI: 10.1007/s00454-009-9176-0 (cit. on p. 13).

[15] Mathieu Carrière, Marco Cuturi, and Steve Oudot. "Sliced Wasserstein Kernel for Persistence Diagrams". In: *International Conference on Machine Learning*. International Conference on Machine Learning. July 17, 2017, pp. 664–673 (cit. on pp. 14, 16, 17, 21, 25, 33).

[16] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, et al. "Time-varying graphs and dynamic networks". In: *International Journal of Parallel, Emergent and Distributed Systems* 27.5 (Oct. 1, 2012), pp. 387–408. ISSN: 1744-5760. DOI: 10.1080/17445760.2012.668546 (cit. on p. 3).

[17] Ciro Cattuto, Wouter Van den Broeck, Alain Barrat, et al. "Dynamics of Person-to-Person Interactions from Distributed RFID Sensor Networks". In: *PLOS ONE* 5.7 (July 15, 2010), e11596. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0011596 (cit. on p. 28).

[18] Frédéric Chazal and Bertrand Michel. "An introduction to Topological Data Analysis: fundamental and practical aspects for data scientists". In: *arXiv preprint arXiv:1710.04019* (2017) (cit. on p. 6).

[19] Frédéric Chazal, Vin de Silva, and Steve Oudot. "Persistence stability for geometric complexes". In: *Geometriae Dedicata* 173.1 (Dec. 1, 2014), pp. 193–214. ISSN: 0046-5755, 1572-9168. DOI: 10.1007/s10711-013-9937-z (cit. on pp. 10, 11, 14).

[20] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. "Stability of Persistence Diagrams". In: *Discrete & Computational Geometry* 37.1 (Jan. 1, 2007), pp. 103–120. ISSN: 0179-5376, 1432-0444. DOI: 10.1007/s00454-006-1276-5 (cit. on pp. 10, 11).

[21] Tamal K. Dey, Fengtao Fan, and Yusu Wang. "Computing Topological Persistence for Simplicial Maps". In: *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*. SOCG'14. New York, NY, USA: ACM, 2014, 345:345–345:354. ISBN: 978-1-4503-2594-3. DOI: 10.1145/2582112.2582165 (cit. on p. 13).

[22] H. Edelsbrunner, D. Letscher, and A. Zomorodian. "Topological persistence and simplification". In: *Proceedings 41st Annual Symposium on Foundations of Computer Science*. Proceedings 41st Annual Symposium on Foundations of Computer Science. Nov. 2000, pp. 454–463. DOI: 10.1109/SFCS.2000.892133 (cit. on p. 11).

[23] Herbert Edelsbrunner and J. Harer. *Computational topology: an introduction*. OCLC: ocn427757156. Providence, R.I: American Mathematical Society, 2010. 241 pp. ISBN: 978-0-8218-4925-5 (cit. on pp. 6, 7, 9).

[24] François Fouss, Marco Saerens, and Masashi Shimbo. *Algorithms and Models for Network Data and Link Analysis*. Google-Books-ID: AUJfDAAAQBAJ. Cambridge University Press, July 12, 2016. 549 pp. ISBN: 978-1-107-12577-3 (cit. on p. 12).

[25] Laetitia Gauvin, Mathieu Génois, Márton Karsai, et al. "Randomized reference models for temporal networks". In: *arXiv:1806.04032 [physics, q-bio]* (June 11, 2018). arXiv: 1806.04032 (cit. on p. 1).

[26] M. Girvan and M. E. J Newman. "Community structure in social and biological networks". In: *Proceedings of the National Academy of Sciences of United States of America* 99.12 (2002), pp. 7821–7826. ISSN: 0027-8424. DOI: 10.1073/pnas. (cit. on pp. 28, 34).

[27] François Golse, Yves Laszlo, Frank Pacard, et al. *MAT321 Analyse réelle*. 2015 (cit. on p. 35).

[28] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Vol. 1. 2009. 1–694. ISBN: 978-0-387-84857-0. DOI: 10.1007/b94608 (cit. on pp. 21, 25).

[29] P. Holme. "Network dynamics of ongoing social relationships". In: *EPL (Europhysics Letters)* 64.3 (Nov. 2003), p. 427. ISSN: 0295-5075. DOI: 10.1209/epl/i2003-00505-4 (cit. on pp. 28, 32).

[30] Petter Holme. "Modern temporal network theory: a colloquium". In: *The European Physical Journal B* 88.9 (Sept. 1, 2015), p. 234. ISSN: 1434-6028, 1434-6036. DOI: 10.1140/epjb/e2015-60657-4 (cit. on pp. 1, 28).

[31] Petter Holme, Beom Jun Kim, Chang No Yoon, et al. "Attack vulnerability of complex networks". In: *Physical Review E* 65.5 (May 7, 2002), p. 056109. DOI: 10.1103/PhysRevE.65.056109 (cit. on p. 28).

[32] Petter Holme and Jari Saramäki. "Temporal networks". In: *Physics Reports*. Temporal Networks 519.3 (Oct. 1, 2012), pp. 97–125. ISSN: 0370-1573. DOI: 10.1016/j.physrep.2012.03.001 (cit. on pp. 1, 20).

[33] Danijela Horak, Slobodan Maletić, and Milan Rajković. "Persistent homology of complex networks". In: *Journal of Statistical Mechanics: Theory and Experiment* 2009.3 (2009), P03034. ISSN: 1742-5468. DOI: 10.1088/1742-5468/2009/03/P03034 (cit. on p. 1).

[34] *Infectious SocioPatterns*. SocioPatterns.org. Mar. 31, 2011. URL: http://www.sociopatterns.org/datasets/infectious-sociopatterns/ (cit. on p. 28).

[35] *Infectious SocioPatterns dynamic contact networks*. SocioPatterns.org. Nov. 28, 2011. URL: http://www.sociopatterns.org/datasets/infectious-sociopatterns-dynamic-contact-networks/ (cit. on p. 28).

[36] Lorenzo Isella, Juliette Stehlé, Alain Barrat, et al. "What's in a crowd? Analysis of face-to-face behavioral networks". In: *Journal of Theoretical Biology* 271.1 (Feb. 21, 2011), pp. 166–180. ISSN: 0022-5193. DOI: 10.1016/j.jtbi.2010.11.033 (cit. on p. 28).

[37] Hang-Hyun Jo, Márton Karsai, János Kertész, et al. "Circadian pattern and burstiness in mobile phone communication". In: *New Journal of Physics* 14.1 (Jan. 25, 2012), p. 013055. ISSN: 1367-2630. DOI: 10.1088/1367-2630/14/1/013055 (cit. on pp. 28, 32).

[38] Jakob Jonsson. *Simplicial complexes of graphs*. Lecture Notes in Mathematics. Berlin: Springer, 2008. ISBN: 978-3-540-75859-4. DOI: 10.1007/978-3-540-75859-4,10.1007/978-3-540-75859-4 (cit. on p. 1).

[39] Richard M. Karp. "Reducibility among combinatorial problems". In: *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Chapter 9. 2010, pp. 219–241. ISBN: 978-3-540-68274-5. DOI: 10.1007/978-3-540-68279-0_8 (cit. on p. 24).

[40] Mikko Kivelä, Alexandre Arenas, Marc Barthelemy, et al. "Multilayer Networks". In: *Journal of Complex Networks* 2.3 (Sept. 1, 2014), pp. 203–271. ISSN: 2051-1310, 2051-1329. DOI: 10.1093/comnet/cnu016. arXiv: 1309.7233 (cit. on pp. 1, 3).

[41] Gautier Krings, Márton Karsai, Sebastian Bernhardsson, et al. "Effects of time window size and placement on the structure of an aggregated communication network". In: *EPJ Data Science* 1.1 (Dec. 2012), p. 4. ISSN: 2193-1127. DOI: 10.1140/epjds4 (cit. on p. 21).

[42] Genki Kusano, Kenji Fukumizu, and Yasuaki Hiraoka. "Kernel method for persistence diagrams via kernel embedding and weight factor". In: *arXiv:1706.03472 [physics, stat]* (June 12, 2017). arXiv: 1706.03472 (cit. on pp. 14, 18, 19, 33).

[43] Roland Kwitt, Stefan Huber, Marc Niethammer, et al. "Statistical Topological Data Analysis - A Kernel Perspective". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, et al. Curran Associates, Inc., 2015, pp. 3070–3078 (cit. on pp. 14, 17).

[44] Clément Maria, Jean-Daniel Boissonnat, Marc Glisse, et al. "The Gudhi Library: Simplicial Complexes and Persistent Homology". In: *Mathematical Software – ICMS 2014*. International Congress on Mathematical Software. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Aug. 5, 2014, pp. 167–174. ISBN: 978-3-662-44198-5. DOI: 10.1007/978-3-662-44199-2_28 (cit. on p. 11).

[45] Clément Maria and Steve Oudot. "Computing Zigzag Persistent Cohomology". In: *arXiv:1608.06039 [cs]* (Aug. 21, 2016). arXiv: 1608.06039 (cit. on p. 13).

[46] Dimitriy Morozov. *dionysus: Library for computing persistent homology*. original-date: 2017-07-14T19:02:35Z. Apr. 11, 2018 (cit. on pp. 11, 13, 29).

[47] Dmitriy Morozov. "Persistence algorithm takes cubic time in worst case". In: *BioGeometry News, Dept. Comput. Sci., Duke Univ* (2005) (cit. on p. 11).

[48] Krikamol Muandet, Kenji Fukumizu, Bharath Sriperumbudur, et al. "Kernel Mean Embedding of Distributions: A Review and Beyond". In: *Foundations and Trends® in Machine Learning* 10.1 (June 28, 2017), pp. 1–141. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000060 (cit. on p. 16).

[49] Elizabeth Munch, Katharine Turner, Paul Bendich, et al. "Probabilistic Fréchet means for time varying persistence diagrams". In: *Electronic Journal of Statistics* 9.1 (2015), pp. 1173–1204. ISSN: 1935-7524. DOI: 10.1214/15-EJS1030 (cit. on p. 14).

[50] M. E. J. Newman. "Network structure from rich but noisy data". In: *Nature Physics* 14.6 (June 2018), pp. 542–545. ISSN: 1745-2481. DOI: 10.1038/s41567-018-0076-1 (cit. on p. 20).

[51] M. E. J. Newman. *Networks: an introduction*. OCLC: ocn456837194. Oxford ; New York: Oxford University Press, 2010. 772 pp. ISBN: 978-0-19-920665-0 (cit. on pp. 1, 4).

[52] Nina Otter, Mason A. Porter, Ulrike Tillmann, et al. "A roadmap for the computation of persistent homology". In: *EPJ Data Science* 6.1 (Dec. 1, 2017), p. 17. ISSN: 2193-1127. DOI: 10.1140/epjds/s13688-017-0109-5 (cit. on pp. 1, 11–13).

[53] Steve Y. Oudot. *Persistence theory: from quiver representations to data analysis*. Mathematical surveys and monographs volume 209. Providence, Rhode Island: American Mathematical Society, 2015. 218 pp. ISBN: 978-1-4704-2545-6 (cit. on p. 10).

[54] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (Oct. 2011), 2825–2830. ISSN: 1533-7928 (cit. on pp. 26, 29).

[55] Leto Peel and Aaron Clauset. "Detecting change points in the large-scale structure of evolving networks". In: *arXiv:1403.0989 [physics, stat]* (Mar. 4, 2014). arXiv: 1403.0989 (cit. on p. 21).

[56] Jose A. Perea, Anastasia Deckard, Steve B. Haase, et al. "SW1PerS: Sliding windows and 1-persistence scoring; discovering periodicity in gene expression time series data". In: *BMC Bioinformatics* 16 (Aug. 16, 2015), p. 257. ISSN: 1471-2105. DOI: 10.1186/s12859-015-0645-6 (cit. on p. 2).

[57] Jose Perea and Chris Traile. "Sliding windows and persistence". In: *The Journal of the Acoustical Society of America* 141.5 (May 1, 2017), pp. 3585–3585. ISSN: 0001-4966. DOI: 10.1121/1.4987655 (cit. on p. 2).

[58] Giovanni Petri and Alain Barrat. "Simplicial Activity Driven Model". In: *arXiv:1805.06740 [physics]* (May 17, 2018). arXiv: 1805.06740 (cit. on p. 1).

[59] Giovanni Petri, Martina Scolamiero, Irene Donato, et al. "Topological Strata of Weighted Complex Networks". In: *PLoS ONE* 8 (June 1, 2013), e66506. DOI: 10.1371/journal.pone.0066506 (cit. on pp. 1, 12).

[60] Mason A. Porter and James P. Gleeson. "Dynamical Systems on Networks: A Tutorial". In: *arXiv:1403.7663 [cond-mat, physics:nlin, physics:physics]* (Mar. 29, 2014). arXiv: 1403.7663 (cit. on p. 1).

[61] Erin Price-Wright. "A Topological Approach to Temporal Networks". MSc dissertation in Mathematics and Foundations of Computer Science. University of Oxford, 2015 (cit. on pp. 1, 2, 27, 33).

[62] Jan Reininghaus. *DIPHA (A Distributed Persistent Homology Algorithm)*. original-date: 2015-12-25T17:23:32Z. Apr. 3, 2018 (cit. on p. 11).

[63] Jan Reininghaus, Stefan Huber, Ulrich Bauer, et al. "A stable multi-scale kernel for topological machine learning". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 4741–4748 (cit. on pp. 17–19, 33).

[64] Bruno Ribeiro, Nicola Perra, and Andrea Baronchelli. "Quantifying the effect of temporal resolution on time-varying networks". In: *Scientific Reports* 3 (Oct. 21, 2013), p. 3006. ISSN: 2045-2322. DOI: 10.1038/srep03006 (cit. on p. 21).

[65] Dino Sejdinovic. *Advanced Topics in Statistical Machine Learning*. Feb. 10, 2018 (cit. on p. 16).

[66] Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. "Dualities in persistent (co)homology". In: *Inverse Problems* 27.12 (Dec. 1, 2011), p. 124003. ISSN: 0266-5611, 1361-6420. DOI: 10.1088/0266-5611/27/12/124003 (cit. on p. 11).

[67] Vin de Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. "Persistent Cohomology and Circular Coordinates". In: *Discrete & Computational Geometry* 45.4 (June 1, 2011), pp. 737–759. ISSN: 1432-0444. DOI: 10.1007/s00454-011-9344-x (cit. on p. 11).

[68] Bernadette J. Stolz, Heather A. Harrington, and Mason A. Porter. "Persistent homology of time-dependent functional networks constructed from coupled time series". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 27.4 (Apr. 1, 2017), p. 047410. ISSN: 1054-1500. DOI: 10.1063/1.4978997 (cit. on p. 1).

[69] Rajmonda Sulo, Tanya Berger-Wolf, and Robert Grossman. "Meaningful Selection of Temporal Resolution for Dynamic Networks". In: *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*. MLG '10. New York, NY, USA: ACM, 2010, pp. 127–136. ISBN: 978-1-4503-0214-2. DOI: 10.1145/1830252.1830269 (cit. on pp. 20, 21).

[70] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. "The worst-case time complexity for generating all maximal cliques and computational experiments". In: *Theoretical Computer Science*. Computing and Combinatorics 363.1 (Oct. 25, 2006), pp. 28–42. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2006.06.015 (cit. on p. 12).

[71] Katharine Turner, Yuriy Mileyko, Sayan Mukherjee, et al. "Fréchet Means for Distributions of Persistence Diagrams". In: *Discrete & Computational Geometry* 52.1 (July 1, 2014), pp. 44–70. ISSN: 0179-5376, 1432-0444. DOI: 10.1007/s00454-014-9604-7 (cit. on p. 14).

[72] Matthias Zeppelzauer, Bartosz Zieliński, Mateusz Juda, et al. "Topological Descriptors for 3D Surface Analysis". In: *Computational Topology in Image Context*. International Workshop on Computational Topology in Image Context. Lecture Notes in Computer Science. Springer, Cham, June 15, 2016, pp. 77–87. ISBN: 978-3-319-39440-4 978-3-319-39441-1. DOI: 10.1007/978-3-319-39441-1_8 (cit. on p. 15).

[73] Afra Zomorodian. “The Tidy Set: A Minimal Simplicial Set for Computing Homology of Clique Complexes”. In: *Proceedings of the Twenty-sixth Annual Symposium on Computational Geometry*. SoCG ’10. New York, NY, USA: ACM, 2010, pp. 257–266. ISBN: 978-1-4503-0016-2. DOI: 10.1145/1810959.1811004 (cit. on p. 12).

[74] Afra Zomorodian and Gunnar Carlsson. “Computing Persistent Homology”. In: *Discrete & Computational Geometry* 33.2 (Feb. 1, 2005), pp. 249–274. ISSN: 0179-5376, 1432-0444. DOI: 10.1007/s00454-004-1146-y (cit. on pp. 7, 9, 11).