State-of-the-art methods in topological data analysis

Ben Napier

March 11, 2022

Abstract

To be written.

Contents

Chapter 1

Introduction

To be written.

Chapter 2

Background in topology and combinatorics

2.1 Persistence modules

This section serves to build a single algebraic structure called a *persistence module*, for which we can describe the concept of *persistence*, but to do so we require tools from *category theory* and *module theory*.

We move to form a bijection between the isomorphism classes of *persistence modules* of finite type over a field and the finite sets of $\mathcal{P}$-intervals.

2.1.1 A brief touch on category theory

Category theory has the ambitious goal to generalize all of mathematics in terms of *categories*, irrespective of the underlying structure of such objects.

Definition 2.1.1 (Category)**.** A *category* C consists of

(i) a class $\mathrm{ob}(C)$ of *objects*;

(ii) a class $\hom(C)$ of *morphisms* between objects;

(iii) a *domain* class function $\mathrm{dom} : \hom(C) \to \mathrm{ob}(C)$;

(iv) a *codomain* class function $\mathrm{cod} : \hom(C) \to \mathrm{ob}(C)$;

(v) for every $f, g \in \hom(C)$ such that $\mathrm{cod}(f) = \mathrm{dom}(g)$, a morphism denoted $g \circ f$ (or gf) called their *composite*; and

such that for all $f, g, h \in \hom(C)$ and $x, y \in \mathrm{ob}(C)$,

(i) if $\mathrm{cod}(f) = \mathrm{dom}(g)$, then $\mathrm{dom}(g \circ f) = \mathrm{dom}(f)$ and $\mathrm{cod}(g \circ f) = \mathrm{cod}(g)$;

(ii) there exists a morphism $\mathrm{id}_x \in \hom(C)$ (called the *identity morphism*) such that $\mathrm{dom}(\mathrm{id}_x) = x$ and $\mathrm{cod}(\mathrm{id}_x) = x$;

(iii) if $\mathrm{cod}(f) = \mathrm{dom}(g)$ and $\mathrm{cod}(g) = \mathrm{dom}(h)$, then $(h \circ g) \circ f = h \circ (g \circ f)$; and

(iv) if $\mathrm{dom}(f) = x$ and $\mathrm{cod}(f) = y$, then $\mathrm{id}_y \circ f = f \circ \mathrm{id}_x$.

Note here that there is little in the way of description of what is meant by an *object* and a *morphisms*, which is the purpose at this level of abstraction. We now define a mapping *between* categories. For a category C and $x, y \in \mathrm{ob}(C)$, $\hom(x, y)$ denotes a subclass of of $\hom(C)$ such that if $f \in \hom(x, y)$ then $\mathrm{dom}(f) = x$ and $\mathrm{cod}(f) = y$. Such a morphism may be denoted $f : x \to y$.

Definition 2.1.2 (Isomorphism)**.** Let C be a category and $x, y \in \mathrm{ob}(C)$. A morphism $f : \hom(x, y)$ is an *isomorphism* if there exists a morphism $f^{-1} \in \hom(y, x)$ such that $f \circ f^{-1} = \mathrm{id}_y$ and $f^{-1} \circ f = \mathrm{id}_x$. We call f^{-1} is *inverse* of f.

Definition 2.1.3 (Functor)**.** Let C and D be categories. A *functor* F from C to D is a mapping that

(i) associates each $x \in \mathrm{ob}(C)$ to a $F(x) \in \mathrm{ob}(C)$;

(ii) associates each $f \in \hom(C)$ to a $F(f) \in \hom(D)$ such that $\mathrm{dom}(F(f)) = F(\mathrm{dom}(f))$ and $\mathrm{cod}(F(f)) = F(\mathrm{cod}(f))$ such that

  (a) for every $x \in \mathrm{ob}(C)$, we have $F(\mathrm{id}_x) = \mathrm{id}_{F(x)}$; and

  (b) for all morphism $f, g \in \hom(C)$ such that $\mathrm{cod}(f) = \mathrm{dom}(g)$, we have $F(g \circ f) = F(g) \circ F(f)$.

That is, functors between categories much preserve identity morphisms and composition of morphisms. We may write $F : C \to D$ to denote a functor from category C to category D.

Example 2.1.4 (Examples of functors)**.**

(i) For any category C, we may define the *identity endofunctor* (an endofunctor is functor that maps from a category to itself) that maps all objects to itself and all morphisms to itself, we denote this functor id_C.

(ii) We may alternatively define *homology* as a *contravariant* functor from the category of chain complexes to the category of abelian groups (or modules, which we will later see). A contravariant functor is defined in the same way as a regular functor, except for the fact that they *reverse composition*; that is $F(g \circ f) = F(f) \circ F(g)$ (in (ii)(b) above).

(iii) Another interesting example (unrelated to this work) is the abelianization functor mapping from the category of groups to the category

of abelian groups. The abelianization of some group G is the quotient of G by the relation $xy = yx$ for all $x, y \in G$, and the construction of this functor is as one would expect (using the canonical quotient map).

Proposition 2.1.5. *Functors preserve isomorphisms.*

Proof. Let C and D be categories and $F : C \to D$ a functor. Let $f \in \hom(C)$ be an isomorphism with inverse f^{-1}. Then

$$F(f) \circ F(f^{-1}) = F(f \circ f^{-1}) = F(\mathrm{id}_y) = \mathrm{id}_{F(y)}$$

and

$$F(f^{-1}) \circ F(f) = F(f^{-1} \circ f) = F(\mathrm{id}_x) = \mathrm{id}_{F(x)},$$

thus $F(f) \in \hom(D)$ is an isomorphism. □

Definition 2.1.6 (Equivalence of categories)**.** Let C and D be categories. An *equivalence* of two categories is a pair of functors $F : C \to D$ and $G : D \to C$ alongside two natural isomorphisms $F \circ G \simeq \mathrm{id}_D$ and $G \circ F \simeq \mathrm{id}_C$.

If such an equivalence of categories exists between categories C and D, then C and D are said to be *equivalent*, denoted $C \simeq D$, and in the definition above $F : C \to D$ (or $G : D \to C$) are said to define an equivalence of categories. Equivalence of categories is more analogous to *homotopy equivalence* from homotopy theory then it is to homeomorphic; however, this is not a perfect analogy.

Equivalence of categories do not present particularly strong correspondence between objects within each category; however, it does provide a bijection between the isomorphisms within each category.

Corollary 2.1.7. *Let F be a functor which defines an equivalence between categories C and D. Then $f \in \hom(C)$ is an isomorphism if and only if $F(f) \in \hom(D)$ is an isomorphism.*

This subsection will potentially be extended to describe the details of the *natural isomorphism* above; however, it may take up another page (or so) and add little to the readers understanding when applied to persistence modules.

2.1.2 On modules and their structure

Here we briefly touch on *module theory* and a fundamental result from abstract algebra on the structure of finitely generated modules over principal ideal domain.

A module is a generalization of a vector spaces, but we replace the field of scalars with a ring.

Definition 2.1.8 (Module)**.** Let R be a commutative ring with multiplicative identity 1. A *R-module* M is a set with operations $+$ and $\cdot$ such that

(i) $(M, +)$ is an abelian group;

(ii) for all $r, s \in R$ and $m, n \in M$:

(a) $(r + s) \cdot m = r \cdot m + s \cdot m$;

(b) $(rs) \cdot m = r \cdot (s \cdot m)$;

(c) $r \cdot (m + n) = r \cdot m + r \cdot n$; and

(d) $1 \cdot m = m$.

The use of studying modules is vast; we may study some object by making it into a module over some ring with *well-behaved* properties to give us insight of the original object.

Example 2.1.9 (Examples of modules)**.**

(i) Every abelian group is a $\mathbb{Z}$-module, and in fact this correspondence is bijective.

(ii) Every ideal of a commutative ring A is an A-module.

(iii) Every quotient ring of a commutative ring A is an A-module.

We define the morphisms in the category of modules as one would expect.

Definition 2.1.10 (Module homomorphism)**.** Let R be a commutative ring and M and N be R-modules. A function $f : M \to N$ is a *R-module homomorphism* (or *R-linear map*) if for all $m, n \in M$ and $r \in R$, $f(m+n) = f(m) + f(n)$ and $f(rx) = rf(x)$.

A module homomorphism is called a *module isomorphism* if it is a bijection, and if there is such a homomorphism between two modules then they are said to be *isomorphic* (denoted with the typical $\cong$).

We borrow the notion of *finitely generated* from linear algebra and apply this to modules in the way you would expect, but we benefit from the following algebraic definition.

Definition 2.1.11 (Finitely generated module)**.** Let R be a commutative ring and M be a R-module. M is *finitely generated* if there is an exact sequence $R^p \to M \to 0$ for some $p \in \mathbb{N}_0$. Additionally, M is *finitely presented* if there exists an exact sequence $R^q \to R^p \to M \to 0$ for some $p, q \in \mathbb{N}_0$.

We now introduce the notion of *gradation*, which is a decomposition of a ring or module.

Definition 2.1.12 (Graded ring)**.** A *graded ring* is a ring R equipped with a direct sum decomposition of abelian groups $R \cong \bigoplus_{i \in \mathbb{N}_0} R_i$ such that $R_m R_n \subset R_{m+n}$ for all $m, n \in \mathbb{N}_0$.

We similarly define the gradation of a module.

Definition 2.1.13 (Graded module)**.** Let $R \cong \bigoplus_{n\in\mathbb{N}_0} R_n$ be a graded commutative ring and M a R-module. M is a *graded module* if there is a direct sum of abelian groups $M \cong \bigoplus_{i\in\mathbb{N}_0} M_i$ such that $R_m M_n \subset M_{m+n}$ for all $m, n \in \mathbb{N}_0$.

We note here that our gradation is a decomposition over $\mathbb{N}_0$, which may be referred to as a $\mathbb{N}_0$-*grading*. Other texts may choose $\mathbb{Z}$-gradation as the standard, for which we would call the above a *non-negatively graded ring or module*.

We now present the standard structure theorem for finitely generated modules over a principal ideal domain.

Theorem 2.1.14. *Let R be a principal ideal domain and M a finitely generated R-module. Then there is a unique decomposition*

$$M \cong R^{\beta} \oplus \bigoplus_{i=1}^{m} R/(d_i) \tag{2.1}$$

where $d_i \in R$ such that $d_1 \mid d_2 \mid \ldots \mid d_m$ and $\beta \in \mathbb{Z}$.

Sketch of proof. As R is a principal ideal domain, it is a Noetherian ring in which the concepts of finitely generated and finitely presented coincide. Thus, M is finitely presented. That is, there exists an exact sequence $R^q \to R^p \to M \to 0$ where $p, q \in \mathbb{N}$. We take A as a presentation matrix isomorphic to the morphism $R^q \to R^p$. Then SNF A yields the required decomposition, where diagonal entries correspond the to d_i's. □

A similar result holds for the graded case.

Theorem 2.1.15. *Let R be a graded principal ideal domain and M a finitely generated graded R-module. Then there is a unique decomposition*

$$M \cong \left(\bigoplus_{i=1}^{n} \Sigma^{\alpha_i} R\right) \oplus \left(\bigoplus_{i=1}^{m} \Sigma^{\gamma_i} R/(d_i)\right) \tag{2.2}$$

where $d_i \in R$ such that $d_1 \mid d_2 \mid \ldots \mid d_m$, $\alpha_i, \gamma_i \in \mathbb{Z}$, and Σ^k denotes a k-shift upward in grading.

A proof for this mimics that for Theorem 2.1.14, for which there exists a variant of the Smith normal form algorithm for graded principal ideal domains [1].

We have thus established a resemblance of the structure of finitely generated modules and finitely generated graded modules to that of vector spaces, but with some finite size *torsional* portion.

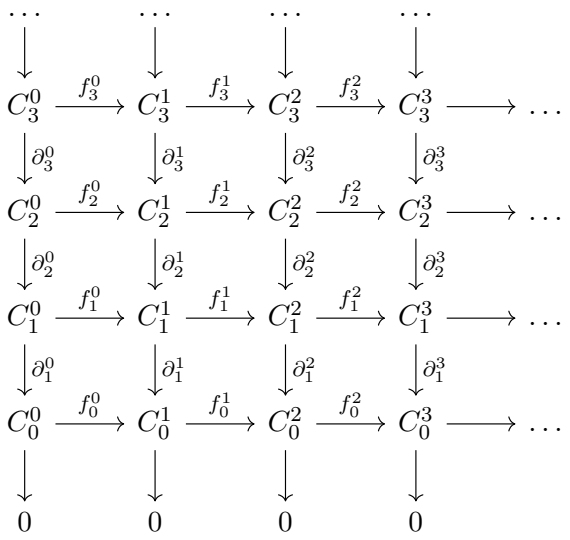


Figure 2.1: A portion of a persistence complex.

2.1.3 Persistence modules

We move to describe an underlying algebraic structure to describe all complexes in a filtration. First, we generalize chain complexes.

Definition 2.1.16 (Chain complex)**.** Let R be a commutative ring. A chain complex $C = (C_*, \partial_*)$ is a sequence of R-modules $\{C_i\}_{i\in\mathbb{N}_0}$ connected by module homomorphism $\{\partial_i\}_{i\in\mathbb{Z}}$ with $\partial_i : C_i \to C_{i-1}$ such that $\partial_i \circ \partial_{i+1} = 0$ for all $i \in \mathbb{Z}$.

In our scope, we consider *non-negative* chain complexes; that is, $C_i = 0$ for $i < 0$. We assure the reader that the standard constructions built on sequences of abelian group (such as homology, chain maps, exact sequences, etc.) still hold in the generalization to modules.

Definition 2.1.17 (Persistence complex)**.** A *persistence complex* $\mathcal{C} = \{(C^i, f^i)\}_{i\in\mathbb{N}_0}$ over a commutative ring R is a family of chain complexes $C^i = (C^i_*, \partial^i_*)$ and chain maps $f^i : C^i_* \to C^{i+1}_*$.

To illustrate this definition, one may draw a commutative diagram akin to Figure 2.1. For a persistence complex $\mathcal{C} = \{(C^i, f^i)\}_{i\in\mathbb{N}_0}$, we define its homology as one may expect, $H_*(\mathcal{C}) = \{(H_*(C^i), f^i_*)\}_{i\in\mathbb{N}_0}$ where f^i_* is the standard induced map on homology.

Definition 2.1.18 (Persistence module)**.** A *persistence module* $\mathcal{M} = \{(M^i, \varphi^i)\}_{i\in\mathbb{N}_0}$ over a commutative ring R is a family of R-modules M^i with homomorphisms $\varphi^i : M^i \to M^{i+1}$.

Lemma 2.1.19. *The homology of a persistence complex is a persistence*

module.

Proof. Let $\mathcal{C} = \{(C^i, f^i)\}_{i\in\mathbb{N}_0}$ be a persistent complex over a commutative ring R. Each $C^i = (C^i_*, \partial^i_*)$ is a R-module, as each C^i_n is a R-module. Thus each $H_*(C^i)$ is a R-module. We also have that

$$f^i_*([c] + [d]) = [f^i(c + d)] = [f^i(c) + f^i(d)] = f^i_*([c]) + f^i_*([d])$$

and

$$f^i_*(r[c]) = f^i_*([rc]) = [f^i(rc)] = [rf^i(c)] = r[f^i(c)] = rf^i_*([c])$$

and so $f^i_* : H_*(C^i) \to H_*(C^{i+1})$ is a module homomorphism. Therefore, $H^*(\mathcal{C}) = \{(H_*(C^i), f^i_*)\}_{i\in\mathbb{N}_0}$ is a persistence module. □

Definition 2.1.20 (Finite type persistence complex)**.** A persistence complex $\mathcal{C} = \{(C^i, f^i)\}_{i\in\mathbb{N}_0}$ over a commutative ring R is of *finite type* if each C^i is finitely generated and there is some $m \in \mathbb{N}_0$ such that f^i is an isomorphism for all $i \geq m$.

We similarly define persistence modules of finite type in the same way.

We aim to apply the decomposition in Equation 2.2 to our persistence module, to motivate the following construction.

Definition 2.1.21 (Graded construction of a persistence module)**.** Let $\mathcal{M} = \{(M^i, \varphi^i)\}_{i\in\mathbb{N}_0}$ be a persistence module over a commutative ring R. We equip $R[t]$ with the standard degree grading and define a graded module over $R[t]$ by

$$\alpha(\mathcal{M}) = \bigoplus_{i\in\mathbb{N}_0} M^i$$

where the R-module structure is the sum of the individual components and the action of t is given by $tm^i = \varphi^i(m^i)$.

Here, the standard degree gradation of $R[t]$ is the direct sum of the homogeneous polynomials; that is,

$$R[t] = \bigoplus_{i\in\mathbb{N}_0} \{rt^i : r \in R\}.$$

Multiplication by t can be thought of as an upwards shift in the gradation.

For a commutative ring R, α may be considered a functor between the category of persistent modules of finite type over R (the objects and morphisms are clear here) and the category of finitely generated graded modules over $R[t]$ (where our morphisms all correspond to multiplication by t), and in fact gives us an *equivalence* of these categories.

Theorem 2.1.22 (ZC-Representation Theorem [2])**.** *Let R be a commutative ring. α defines an equivalence of categories between the category of persistence modules of finite type over R and the category of finitely generated graded modules over $R[t]$.*

Proof as in [3]. `todo` □

By Corollary 2.1.7, we have established a bijective correspondence between the isomorphism classes of persistence modules of finite type over R and isomorphism classes of finitely generated graded $R[t]$-modules.

This correspondence does not provide particular assistance in decomposing persistence modules when R is not a field; the classification of modules over $\mathbb{Z}[t]$ is complicated. Although, by setting $R = F$ for some field F, we get a particularly simple decomposition. $F[t]$ is a graded principal ideal domain and its only graded ideals are of the form (t^n), for $n \in \mathbb{N}_0$. Thus, applying Theorem 2.1.15 we get

$$M \cong \left(\bigoplus_{i=1}^{n} \Sigma^{\alpha_i} F[t]\right) \oplus \left(\bigoplus_{i=1}^{m} \Sigma^{\gamma_i} F[t]/(t^{n_i})\right) \tag{2.3}$$

where $\alpha_i, \gamma_i \in \mathbb{Z}$, and Σ^k denotes a k-shift upward in grading (note that coincides with multiplication by t^k in our case), for some finitely generated graded $F[t]$-module M.

Definition 2.1.23 ($\mathcal{P}$-interval)**.** A $\mathcal{P}$-interval is a tuple $(i, j) \in \mathbb{N}_0 \times \mathbb{N}^\infty$ such that $i < j$.

For the above definition, we define $\mathbb{N}^\infty = \mathbb{N} \cup \{+\infty\}$.

Proposition 2.1.24. *Let $\mathcal{S}$ be the set of $\mathcal{P}$-intervals and $\mathcal{F}$ the set of isomorphism classes of finitely generated graded $F[t]$-modules. Define $Q' : \mathcal{S} \to \mathcal{F}$ by $(i, j) \mapsto \Sigma^i F[t]/(t^{j-i})$ and $(i, +\infty) \mapsto \Sigma^i F[t]$. Then the map $Q : \mathcal{P}_{<\infty}(\mathcal{S}) \to \mathcal{F}$ defined by*

$$Q(S) = \bigoplus_{l=1}^{n} Q(i_l, j_l),$$

where $S = \{(i_l, j_l)\}_{l=1}^{n}$ is a finite set of $\mathcal{P}$-intervals, is a bijection.

Proof. Let M be a graded $F[t]$-module. Then by Equation 2.3,

$$M \cong \left(\bigoplus_{i=1}^{n} \Sigma^{\alpha_i} F[t]\right) \oplus \left(\bigoplus_{i=1}^{m} \Sigma^{\gamma_i} F[t]/(t^{n_i})\right)$$

where $\alpha_i, \gamma_i \in \mathbb{Z}$, and Σ^k denotes a k-shift upward in grading. We then define $Q^{-1} : \mathcal{F} \to \mathcal{P}_{<\infty}(\mathcal{S})$ by

$$Q^{-1}(M) = \{(\alpha_i, \infty) : i = 1, \dots, n\} \cup \{(\gamma_i, n_i + \gamma_i) : i = 1, \dots, m\}.$$

It is then straight forward to check that $Q \circ Q^{-1} = \mathrm{id}_{\mathcal{F}}$ and $Q^{-1} \circ Q = \mathrm{id}_{\mathcal{P}_{<\infty}(\mathcal{S})}$. □

Corollary 2.1.25. *There is a bijection between the isomorphism classes of persistence modules of finite type over a field F and the finite sets of $\mathcal{P}$-intervals.*

We have now established that every (isomorphism class of a) persistence module has a one-to-one correspondence to a set of $\mathcal{P}$-intervals. We now move to see how we may interpret these $\mathcal{P}$-intervals.

2.2 Persistent homology

An important note about this section: this was written a few weeks ago before I started forming my report. I (on the day this draft is due!) forgot that I had infact not introduced persistent homology anywhere in the report! I have thus copied this section across, but I will be going back through this to connect it to the proceeding section.

We first discuss some intuition about persistent homology, in effort to motivate a following definition, and a (more traditional) alternative definition. Consider a filtration of some simplicial complex. Persistent homology aims to describe how *features* (homology classes) are *born* at a certain part of the filtration, and then *die* at a later date (this terminology will be shortly formalised).

Consider a filtration $\{K_i\}_{i=0}^n$ of some simplicial complex K. For each $i \leq j$, we have an inclusion map $\iota_{i,j} : K_i \hookrightarrow K_j$ from which we form an induced homomorphism on the homology groups $f_p^{i,j} : H_p(K_i) \to H_p(K_j)$, $[c] \mapsto [\iota_{i,j}(c)]$ for each $p \in \mathbb{Z}_{\geq 0}$. Thus the filtration corresponds to a sequence of homology groups as follows.

An example of the above note, instead of considering a simplicial complex here, we can instead just use the already established persistence complex.

$$H_p(K_0) \xrightarrow{f_p^{0,1}} H_p(K_1) \xrightarrow{f_p^{1,2}} \dots \xrightarrow{f_p^{n-1,n}} H_p(K_n)$$

As we move from K_i to K_{i+1}, we may gain new homology classes, and we may also lose classes as they become trivial or merge with others.

Definition 2.2.1 (Persistent homology)**.** Let $\{K_i\}_{i=0}^n$ be a filtration of some simplicial complex K. For $p \in \mathbb{Z}_{\geq 0}$ and $i, j \in \mathbb{Z}_{\geq 0}$ with $i \leq j$, we define the *pth persistent homology* as

$$H_p^{i,j}(K) = \frac{Z_p(K_i)}{Z_p(K_i) \cap B_p(K_j)}.$$

That is, the cycles of K_i that modulo the boundaries of K_j (sans any boundaries that are not cycles of K_i). This definition is easier to accept, but the following is the more traditional definition and is easier to get around with.

Definition 2.2.2 (Persistent homology, alt.)**.** Let $\{K_n\}_{i=0}^n$ be a filtration of some simplicial complex K. Let $f_p^{i,j} : H_p(K_i) \to H_p(K_j)$ be the homomorphism on homology groups induced by the inclusion map $K_i \hookrightarrow K_j$ as previously discussed. We define the *pth persistent homology* as the image of this homomorphism, $H_p^{i,j}(K) = \operatorname{im} f_p^{i,j}$.

The *pth persistent Betti numbers* are defined as one may expect: $\beta_p^{i,j}$ is the rank of the corresponding pth persistent homology group. Now that we have some formal groundwork, we will introduce some terminology. Let $\{K_i\}_{i=0}^n$ be some filtration of a simplicial complex K, let $p \in \mathbb{Z}_{\geq 0}$, and let $i, j \in \mathbb{Z}_{\geq 0}$ with $i \leq j$.

- Let $p \in \mathbb{Z}_{\geq 0}$. A *$p$-hole* is another word for a p-dimensional homology class; that is, $[c] \in H_p(K)$.
- A p-hole $[c]$ is *born* at K_i if $[c] \in H_p(K_i) = H_p^{i,i}(K)$ but $[c] \notin H_p^{i-1,i}(K)$.
- A p-hole $[c]$ that is born at K_i *dies entering* K_j if it merges with an older class from K_{j-1} to K_j; that is, $f_p^{i,j-1}([c]) \notin H_p^{i-1,j-1}(K)$ but $f_p^{i,j}([c]) \in H_p^{i-1,j}(K)$.
- Suppose $[c]$ is a p-hole that is born at K_i and dies entering K_j. The *index persistence* of $[c]$ is $\operatorname{ipers}([c]) = j - i$.
- Denote $\mu_p^{i,j}$ as the number of p-holes that are born at K_i and dies entering K_j; that is, $\mu_p^{i,j} = (\beta_p^{i,j-1} - \beta_p^{i,j}) - (\beta_p^{i-1,j-1} - \beta_p^{i-1,j})$.

We move to visualise persistent Betti numbers in $\mathbb{R}^2$, but to do so we need the following property.

Lemma 2.2.3. *Let $\{K_i\}_{i=0}^n$ be a filtration of some simplicial complex K. For each $i, j \in \{0, 1, \ldots, n\}$ with $i \leq j$ and $p \in \mathbb{Z}_{\geq 0}$ we have*

$$\beta_p^{i,j} = \sum_{k \in \{0,\ldots,i\}} \sum_{l \in \{j+1,\ldots,n\}} \mu_p^{k,l}.$$

This fits with our intuition, $\beta_p^{i,j}$ is the number of p-holes that are alive from K_i to K_j; that is, the number of p-holes that are born at K_i or earlier, and die entering K_{j+1} or later.

Up to now, we have only needed a simplicial complex with a filtration defined upon it, but to continue with our visualisation of persistent Betti numbers, we need more. Let $f : K \to \mathbb{R}$ be some map defined on a simplicial complex K such that f is non-decreasing on increasing sequences of faces (that is, if $\sigma \subset \tau \in K \implies f(\sigma) \leq f(\tau)$). Then we let $\{K_i\}_{i=0}^n$ be the induced filtration of K such that $K_0 = \varnothing$, $K_n = K$, and for each $i \in \{1, \ldots, n-1\}$: $K_i = f^{-1}(-\infty, a]$ for some $a \in \mathbb{R}$.

Now, we let $\{a_i\}_{i=0}^n$ be such that $K_i = f^{-1}(-\infty, a_i]$. Suppose $[c]$ is a p-hole that is born at K_i and dies entering K_j. The *persistence* of $[c]$ is $\text{pers}([c]) = a_j - a_i$.

We define the *pth persistence diagram* of f as the multiset

$$\text{dgm}_p(f) = \{(a_i, a_j)^{\mu_p^{i,j}} : \mu_p^{i,j} > 0, 0 \leq i < j \leq n\},$$

where $(a_i, a_j)^{\mu_p^{i,j}}$ denotes the element (a_i, a_j) with multiplicity $\mu_p^{i,j}$. We observe that $\mu_p^{i,i} = 0$.

This subsection will conclude with the interpration of $\mathcal{P}$-intervals as a persistence diagram.

2.3 Complexes

The typical input when calculating persistent homology is a *filtered simplicial complex*. This section serves to define this, and present some standard methods for constructing such a complex from different types of data.

2.3.1 Simplicial complexes

A simplicial complex may be considered as an embedding of points, line segments, triangles, and n-dimensional counterparts in Euclidean space; however, it is often beneficial to look at simplicial complexes as a purely combinatorial object; we omit details on how it will be embedded into Euclidean space. Strictly, this combinatorial view would be referred to as an *abstract simplicial complex*, but we will refer to it as a *simplicial complex*.

Definition 2.3.1 (Simplicial complex)**.** An *simplicial complex* is a family of sets that is closed under taking subsets.

Let K be a simplicial complex. We call a member of this family $\sigma \in K$ a *p-simplex*, where $p = |\sigma| - 1$. We define K_p as the set of p-simplices in K and define the p-skeleton of K as $\bigcup_{i\in\{0,\dots,p\}} K_p$. The dimension is the greatest p such that K_p is non-empty. A simplicial complex K' is said to be a *subcomplex* of K if $K' \subset K$. K. K is said to be finite if $|K| < \infty$. We will be working exclusively with finite simplicial complex, so we assume all simplicial complexes are finite unless otherwise stated.

An *orientation* of a k-simplex $\sigma = \{v_0, \dots, v_k\}$ is an equivalence clas of the orderings of the vertices of σ. We let $(v_0, \dots, v_k) \sim (v_{\tau(0)}, \dots, v_{\tau(k)})$ if the sign of the permutation τ is 1. We may denote an oriented simplex by $[\sigma]$. We denote the set of oriented p-simplices of K by $[K_p]$.

Definition 2.3.2 (Filtration)**.** A *filtration* is a family $(S_i)_{i\in\mathcal{I}}$ of subobjects of a given structure S, where $\mathcal{I}$ is some totally ordered set, and for $i, j \in I$ we have

$$i \leq j \implies S_i \subset S_j.$$

Definition 2.3.3 (Filtered simplicial complex)**.** A *filtered simplicial complex* is a filtration of simplicial complexes $(K_i)_{i\in\mathcal{I}}$.

We note that a maximal filtration of simplicial complexes induces an ordering on the simplices. For example, consider the filtered simplicial complex

$$\{\{1\}\} \subset \{\{1\},\{2\}\} \subset \{\{1\},\{2\},\{1,2\}\}.$$

This filtration corresponds to the ordering $(\{1\},\{2\},\{1,2\})$.

For a non-maximal filtration of simplicial complexes, there are multiple induced orderings. For example, consider the filtered simplicial complex

$$\{\{1\},\{2\}\} \subset \{\{1\},\{2\},\{1,2\}\}.$$

This may correspond to the ordering above or $(\{2\},\{1\},\{1,2\})$. We call such a corresponding ordering a *compatible ordering of simplices* from the filtration.

Definition 2.3.4 (Weight function)**.** Let K be a simplicial complex. $w : K \to \mathbb{R}$ is a *weight function* if $\{w^{-1}(-\infty,\varepsilon]\}_{\varepsilon\in\mathbb{R}}$ is a filtered simplicial complex.

By definition, we see that a weight function on a simplicial complex defines a filtered simplicial complex. We call a simplicial complex with a weight function (K,w) a *weight-filtered simplicial complex.*

Lemma 2.3.5. *Any filtered simplicial complex is a persistence complex.*

Proof. Let $(K^i)_{i\in\mathcal{I}}$ be a filtration of a simplicial complex K. As K is finite, we may reindex as $\{K^i\}_{i=0}^n$. For each $i \in \{0,\dots,n\}$ and $k \in \mathbb{N}_0$, we set $C_k^i = \mathbb{Z}\langle[K_k^i]\rangle$ and define the standard boundary map

$$\partial_k^i([v_0,\dots,v_k]) = \sum_{j=0}^{k}(-1)^j[v_0,\dots,v_{i-1},v_{i+1},\dots,v_k].$$

We set the chain maps as the inclusions up to filtration. For $i > n$, we set $C_k^i = C_k^n$, $\partial_k^i = \partial_k^n$, and the chain maps as the identity map. □

We will now move to introduce some constructions of filtered simplicial complexes, first on point cloud data (that is, subsets of a metric space).

2.3.2 Vietoris-Rips complex

Definition 2.3.6 (Vietoris-Rips complex)**.** Let (M,d) be a metric space, $S \subset M$ be finite, and $\varepsilon > 0$. The *Vietoris-Rips complex* of S at scale ε, denoted $\mathrm{VR}(S;\varepsilon)$, is a simplicial complex defined as

$$\mathrm{VR}(S;\varepsilon) = \{\sigma \subset S : d(u,v) \le \varepsilon \ \ \forall u,v \in \sigma\}.$$

Lemma 2.3.7. *Any Vietoris-Rips complex is a simplicial complex.*

Proof. Let (M, d) be a metric space, $S \subset M$ be finite, $\varepsilon \geq 0$, and $K = \mathrm{VR}(S; \varepsilon)$. Let $\sigma \in K$ and $\sigma' \subset \sigma$. Trivially, $\varnothing \in K$. Let $u, v \in \sigma'$ be (not necessarily distinct) points. Then $u, v \in \sigma$, so $d(u, v) \leq \varepsilon$. Thus $\sigma' \in K$. □

We now use the Vietoris-Rips complex to form a filtered simplicial complex.

Definition 2.3.8 (Filtered Vietoris-Rips complex)**.** Let (M, d) be a metric space and $S \subset M$ be finite. We define the filtered Vietoris-Rips complex as $(\mathrm{VR}(S; \varepsilon))_{\varepsilon \in \mathbb{R}_{\geq 0}}$.

Corollary 2.3.9 (of Lemma 2.3.7)**.** *A filtered Vietoris-Rips complex is a filtered simplicial complex.*

We now introduce an alternative definition of a filtered Vietoris-Rips complex, by defining a weight function. This will be useful when we consider fast constructions in Section 4.1.

Definition 2.3.10 (Filtered Vietoris-Rips complex, by weight function)**.** Let (M, d) be a metric space and $S \subset M$ be finite. We define the filtered Vietoris-Rips complex as $(w(-\infty, \varepsilon))_{\varepsilon \in \mathbb{R}_{\geq 0}}$ where we define the weight function as

$$w(\sigma) = \begin{cases} 0 & \text{if } \dim \sigma \leq 0 \\ d(u, v) & \text{if } \sigma = \{u, v\}, \\ \max_{\tau \subset \sigma} w(\tau) & \text{otherwise.} \end{cases}$$

Lemma 2.3.11. *Definition 2.3.8 and Definition 2.3.10 are equivalent.*

Proof. Let (M, d) be a metric space, $S \subset M$ be finite, and $\varepsilon \in \mathbb{R}_{\geq 0}$. We will show that $w^{-1}(-\infty, \varepsilon] = \mathrm{VR}(S; \varepsilon)$. Let $\sigma \subset S$. Trivially, if $\dim \sigma \leq 0$ then $\sigma \in w^{-1}(-\infty, \varepsilon]$ and $\sigma \in \mathrm{VR}(S; \varepsilon)$. Now suppose $\dim \sigma = 1$, so $\sigma = \{u, v\}$ (distinct points). Then $w(\sigma) = d(u, v)$ and so $\sigma \in w^{-1}(-\infty, \varepsilon]$ if and only if $\sigma \in \mathrm{VR}(S; \varepsilon)$. Now suppose $\dim \sigma \geq 2$. Then

$$w(\sigma) = \max_{\sigma' \subset \sigma} w(\sigma') = \max_{\{u,v\} \subset \sigma} d(u, v)$$

and so $w(\sigma) \leq \varepsilon$ if and only if $d(u, v) \leq \varepsilon$ for all $\{u, v\} \subset \sigma$. □

2.3.3 Čech complex

Definition 2.3.12 (Čech complex)**.** Let (M, d) be a metric space, $F \subset M$ be a finite set of points in M, and $r > 0$. The *Čech complex* of F with radius r, denoted by $\check{\mathrm{C}}(F; r)$, is an abstract simplicial complex (F, S) where

$$\left\{ \sigma \subset F : \bigcap_{x \in \sigma} \overline{B}(x; r) \neq \varnothing \right\}.$$

This subsection needs to be extended to describe Čech complex and build the required theory to understand the algorithms for a fast construction. There is also differing notations used here, but it will be fixed!

2.3.4 A novel complex built on graphs

[4]

The typical *road-map* of persistent homology is to take a point cloud in some metric space, construct a filtered Vietoris-Rips complex, and then compute the persistent homology of the complex. We now explore an alternative route to persistent homology, by constructing a filtered complex from a graph.

This section is dedicated to the theory my novel method of constructing simplicial complexes from graphs.

2.4 Matroid theory

[5]

Definition 2.4.1 (Matroid)**.** A *matroid* is an abstract simplicial complex (V, S) with the additional property that for $\sigma, \tau \in S$, if $|\sigma| < |\tau|$ then there is $v \in \tau \setminus \sigma$ such that $\sigma \cup \{v\} \in S$.

Although this is a simple property, the structure imposed induces many useful properties.

This section serves as a brief introduction into matroid theory, though as discussed this may be moved to an appendix.

Chapter 3

Computational complexity theory

3.1 Modelling computation

This section serves as an introduction to formalizations of our intuitive notions of *problems* and *algorithms*. We will look at how one may precisely define a problem, and use the abstract framework of Turing machines to explore the computability of such problems. We then introduce some tools that we can use to study the resources needed by algorithms, allowing us to classify algorithms based on their *difficulty*.

3.1.1 Computational problems

In order to study computational problems, we *encode* them so we may study them as objects derived from a finite set.

Definition 3.1.1 (Kleene star)**.** Given a set X, define $X^0 = \{\varepsilon\}$ (the set containing only the empty string) and $X^1 = X$. For each $i \in \{2, 3, \ldots\}$, we recursively define the set

$$X^{i+1} = \{wv : w \in X^i, v \in X\}.$$

Definition 3.1.2 (Formal language)**.** A *formal language* (or just *language*) $\mathcal{L}$ over an *alphabet* (some none empty set of symbols, which are called *letters*) Σ is some subset of Σ^*. A *word* $w \in \mathcal{L}$ is an element of a language.

Simple encodings can be used to represent general mathematical objects as strings of bits (that is, words of a language over $\{0, 1\}$), but we avoid dealing with low level representation details. We use $\langle x \rangle$ to denote some canonical

binary representation of an object x, but we will typically omit this notation and simple use x to refer to the object and its representation.

Formally, a *computational problem* is a problem that we may expect a computer to be able to solve. We first consider a simple type of computational problem.

Problem 3.1.3 (PRIMALITY).
Instance: let $n \in \mathbb{N}$.
Question: is n prime?

Problem 3.1.4 (REACHABILITY).
Instance: Let $G = (V, E)$ be a graph and $v, w \in V$ two vertices.
Question: is there a path from v to w in G?

Both PRIMALITY and REACHABILITY expect a *yes* or *no* answer. Problems of this form are called *decision problems*, and we may formally define them as below.

Definition 3.1.5 (Decision problem)**.** A *decision problem* is a yes-or-no question on an infinite set of inputs. Formally, it is a tuple (I, Y) where $I \subset \{0,1\}^*$ is an alphabet of *inputs* and $Y \subset I$ is a language of inputs for which the answer is *yes*.

We may rewrite these problems as follows:

$$\begin{aligned} \text{PRIMALITY} &= (\mathbb{N}, \{n \in \mathbb{N} : n \text{ is prime}\}), \\ \text{REACHABILITY} &= (I, Y) \end{aligned}$$

where $I = \{(G, u, v) : G = (V, E) \text{ is a graph and } u, v \in V\}$ and $(G, u, v) \in Y$ if and only if there is a path from u to v in G.

Definition 3.1.6 (Function problem)**.** A *function problem* is a relation $P \subset \{0,1\}^* \times \{0,1\}^*$. An algorithm solves P if for every $x \in \{0,1\}^*$, the algorithm produces $y \in \{0,1\}^*$ such that $(x, y) \in P$ (if such a y exists).

Problem 3.1.7 (TSP).
Instance: let H be an undirected weighted graph.
Question: find the shortest possible Hamiltonian cycle (that is, a path that visits each vertex once and starts and ends at the same vertex).

3.1.2 Turing machines

One may informally define an algorithm as a collection of simple instructions for carrying out some task. Turing [6] introduced the now ubiquitous model of computation. We will not bother ourselves with the granular detail of the definition of a Turing machine, as many variants exist that are all equivalent.

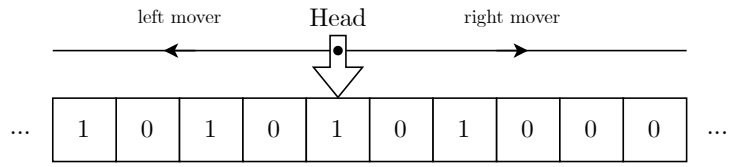


Figure 3.1: Schematic of a Turing machine.

Theorem 3.1.8 (Turing [6])**.** *The intuitive notion of an algorithm is equivalent to the mathematical concept of an algorithm defined by Turing machines.*

A Turing machine M consists of the following components:

- a finite set of states;
- an infinite tape with storage cells, with cells containing a single symbol from some alphabet Π;
- a device called the head that can read and write on a cell, and move along the tape; and
- a transition function.

M will start its computation at an initial state, with an input on the tape, and the head at the start of the input. The head will read the contents of the cell, and by also considering the state, use the transition function to decide what to write on the cell, whether to move right or left along the tape, and what state to enter. The Turing machine will run until its enters a halt state (although it may also run forever), and the output of the machine will be the contents of the cells on the tape when the machine halts. We denote the output of a Turing machine on input $x \in \Sigma^*$ ($\Sigma \subset \Pi$ is the *input alphabet*) as $M(x)$ (note this may be undefined if the Turing machine does not halt). See Figure 3.1 for a schematic of a Turing machine.

For decision problems, we disregard the machine's output but introduce two new halt states: an accept state and a reject state. If M halts on input x in the accept state, we say that M *accepts* x and similarly for M rejecting x. We denote $L(M)$ as the language of strings that M accepts. If there is a Turing machine M such that $\mathcal{L} = L(M)$, then the language L is said to be *semidecidable*. We say a language is *decidable* if it is semidecidable and rejects all strings outside of the language.

Decision problems offer a simple introduction to modelling computation, but most of the problems we will look at expect an answer more than just *yes* or *no*.

We take a brief moment to comment on the existence on *undecidable* problems; that is, a problem for which no algorithm decides it.

Problem 3.1.9 (HaltingProblem).
Instance: let M be a Turing machine and $x \in \Sigma^*$ an input.
Question: does M halt on x?

Theorem 3.1.10. *There is no Turing machine that decides* HaltingProblem.

Sketch of proof. We omit the details of the proof, as it requires some constructs that are of little use in our context, but we provide a rough outline. We construct an adversary Turing machine D that takes as input a Turing machine M. D will accept if M rejects with itself (or rather, an encoding of itself) as input. If M accepts itself, then D loops indefinitely. By considering how D runs on the encoding of itself, we find a contradiction. □

3.2 Computational complexity

3.2.1 Defining complexity

The *complexity* of an algorithm is the amount of resources required to run it, but we focus on the *time* (time complexity) and *memory* (space complexity) requirements.

Definition 3.2.1 (Time complexity)**.** Let M be a Turing machine that halts on all inputs. The *time complexity* (or *running time*) of M is a function $T_M : \mathbb{N} \to \mathbb{N}$ where $T_M(n)$ denotes the maximum number of steps that M uses on an input of length n.

We have already discussed variations of Turing machines that are equivalent that may lead to subtle differences to running times of algorithm, and we need a tool to deal with this.

Definition 3.2.2 (Big O notation)**.** Let $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$. Then $f(n) = O(g(n))$ if there is $k \in \mathbb{R}$ and $N \in \mathbb{N}$ such that for all $n \geq N$, $f(n) \leq kg(n)$.

Time complexities of algorithms are given in terms of Big O notation, which has the analogue of $\leq$ on the reals. We also have a similar notation for an analogue of $\geq$ on the reals.

Definition 3.2.3 (Big Ω notation)**.** Let $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$. Then $f(n) = \Omega(g(n))$ if there is $k \in \mathbb{R}$ and $N \in \mathbb{N}$ such that for all $n \geq N$, $f(n) \geq kg(n)$.

And finally we give an analogue of equality.

Definition 3.2.4 (Big Θ notation)**.** Let $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$. Then $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Table 3.1 shows some common time complexity and algorithms with that complexity.

Table 3.1: A list of common time complexities.

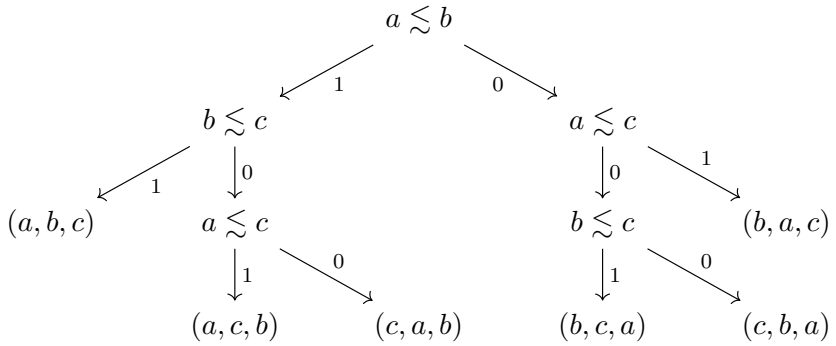
Notation	Name	Example
$O(1)$	Constant	Search in hash table
$O(\log n)$	Logarithmic	Binary search
$O(n)$	Linear	Searching an unsorted list
$O(n \log n)$	Linearithmic	Fastest comparison-based sorting
$O(n^c)$, $c \geq 1$	Polynomial	Reachability
$O(2^n)$	Exponential	Dynamic programming solution to TSP
$O(n!)$	Factorial	Brute-force solution to TSP

Problem 3.2.5 (Sorting).
Instance: let $(A, \lesssim)$ be a totally preordered set.
Question: find an enumeration $\{a_i\}_{i=1}^{|A|}$ of A, so $a_i \lesssim a_j$ for all $i < j$.

We informally introduce the *comparison model* for sorting algorithms. A *comparison sort* is a sorting algorithm that may only read the elements of a totally preordered set $(A, \lesssim)$ through the operation $\lesssim$.

Theorem 3.2.6. *Any deterministic comparison sort algorithm has time complexity* $\Omega(n \log n)$.

Proof. We recall that our definition of time complexity considers the *worst-case performance* of an algorithm. Let M be a sorting algorithm (Turing machine) with time complexity $T_M(n)$. We now fix a set A of $n \in \mathbb{N}$ elements. We may construct a tree G where the leaves correspond to all possible outputs of our algorithm depending on the total preorder given on A, and the internal nodes correspond to a comparison. For example, below we see such a decision tree for $A = \{a, b, c\}$.



The number of leaves for such a decision tree is the number of permutations of the input set; that is, $n!$. It is understood this balanced binary tree has

a height $h = \log(n!)$. It follows that

$$h = \log(n!) = \log\left(\prod_{i=1}^{n} i\right) = \sum_{i=1}^{n} \log i \leq \sum_{i=1}^{n} \log n = n \log n$$

and so, as h corresponds to the maximum number of operations we may perform, $T_M(n) = \Omega(n \log n)$. □

`potential a section here on a lower bound for matrix multiplication`

Definition 3.2.7 (Space complexity)**.** Let M be a Turing machine that halts on all inputs. The *space complexity* of M is a function $S_M : \mathbb{N} \to \mathbb{N}$ where $S_M(n)$ denotes the maximum number of distinct tape cells that M visits on an input of length n.

Remark 3.2.8. One may see that, for any Turing machine M, that $S_M(n) = O(n)$ as we have to store the input at the start of the computation. In order to allow for meaningful analysis into algorithms that use sublinear space, we consider two tapes on our Turing machine: a read-only tape in which the input sits and a read/write tape which we consider to be the *working space*.

3.2.2 Complexity classes

I am considering removing this section completely, the study of complexity classes of function problems is amist with nuances and constructions that I may struggle to introduce in 55 pages.

Complexity classes are sets of related problems, defined in terms of their required resources, such as time complexity and space complexity. A majority of complexity classes are defined for decision problems, but here we are focusing on certain function problems. Thus, we omit what one may concern as standard material and move to the classes of interest.

Definition 3.2.9 (FP)**.** FP is the class of function problems that are polynomial-time solvable.

We note that a function problem is defined as a relation, and does not capture the totality of a function. We can work around this as follows: let $P \subset \Sigma^* \times \Sigma^*$ be a function problem. We pick a special string $y_0 \in \Sigma^*$ such that for all $x \in \Sigma^*$, $(x, y_0) \notin P$ (if this is not possible, we extend Σ to contain such a character). Then for each x such that $(x, y) \notin P$ for all $y \in \Sigma^*$, we output y_0. We note that this does not change the complexity.

Definition 3.2.10 (TFNP)**.** A binary relation $P \subset \Sigma^* \times \Sigma^*$ is in TFNP if and only if there is a deterministic polynomial-time that, given $x, y \in \Sigma^*$, can determine whether $(x, y) \in P$ and for every $x \in \Sigma^*$, there is a $y \in \Sigma^*$ which is at most polynomially longer than x such that $P(x, y)$ holds.

Chapter 4

Problems in topological data analysis

I aim to fill this space with an introduction to the persistent homology roadmap, and to identify key bottlenecks in the computation process. Hopefully to aptly introduce the proceeding sections...

4.1 Vietoris-Rips complexes

Problem 4.1.1 (VietorisRipsFiltration)**.**
Instance: let (M, d) be a metric space and $S \subset M$ be a finite set of points.
Question: compute a compatible ordering of simplices from the filtered Vietoris-Rips complex $(\mathrm{VR}(S;\varepsilon))_{\varepsilon\in\mathbb{R}_{\geq 0}}$.

For a compatible ordering $\{K^i\}_{i=0}^n$, we note that K^n is the simplicial complex containing a single $|S|$-simplex and all of its faces, so the total count of faces is the sum of the first n triangular numbers (including the zeroth triangular number), which is not computationally feasible. It is for this reason that we upper bound ε in our construction, giving the following derived problem.

Problem 4.1.2 ($\hat{\varepsilon}$-VietorisRipsFiltration)**.**
Instance: let (M, d) be a metric space, $S \subset M$ be a finite set of points, and $\hat{\varepsilon} \in \mathbb{R}_{\geq 0}$.
Question: compute a compatible ordering of simplices from the filtered Vietoris-Rips complex $(\mathrm{VR}(S;\varepsilon))_{\varepsilon\in[0,\hat{\varepsilon})}$.

We now follow the approach given by Zomorodian [7]. Let (M, d) be a metric space, $S \subset M$ be a finite set of points, and $\hat{\varepsilon} \in \mathbb{R}_{\geq 0}$. We may split the computation of the Vietoris-Rips complex into the following phases:

(i) compute the weighted graph (G, w) where $G = (V, E)$ is the 1-skeleton of $\mathrm{VR}(S;\hat{\varepsilon})$ and $w : E \to \mathbb{R}_{\geq 0}$ is defined by $w(u, v) = d(u, v)$;

(ii) compute the *expansion* of (G, w) to the weight-filtered simplicial com-

plex $(\mathrm{VR}(S;\hat{\varepsilon}), w)$; then

(iii) obtain the simplex ordering by sorting the simplices according to their weights.

Let n be the number of simplices in $\mathrm{VR}(S;\hat{\varepsilon})$ and we assume that we can compute $d(u,v)$ for any $u,v \in S$ in $O(1)$ time. It is clear (iii) reduces to SORTING, which we can achieve in time $\Theta(n \log n)$. We first look at (i), and it is clear that it reduces to the following problem.

Problem 4.1.3 (ε-NNs).
Instance: let (M,d) be a metric space, $S \subset M$ be a finite set of points, and $\varepsilon \in \mathbb{R}_{\geq 0}$.
Question: for each $x \in S$, compute $\{y \in S \setminus \{x\} : d(x,y) \leq \varepsilon\}$.

We now present an approximate version of ε-NNs.

Problem 4.1.4 ($(1+\delta,\varepsilon)$-ANNs).
Instance: let (M,d) be a metric space, $S \subset M$ be a finite set of points, and $\varepsilon \in \mathbb{R}_{\geq 0}$.
Question: for each $x \in S$, compute $\{y \in S \setminus \{x\} : d(x,y) \leq (1+\delta)\varepsilon\}$.

4.1.1 An algorithm for ε-NNs [8]

We first recognise the brute-force solution, which may run in $O(n^2)$ time and has the benefit of being exact.

Arya et al. [8] made use of a particular tree structure (called BBD-trees) to achieve a solution in the case when $M = \mathbb{R}^d$, with query time of $O(c(d) \log n)$ and $O(dn)$ space, with $O(dn \log n)$ processing time. The function $c(d)$ is some function dependent on the dimension of M, there is no information on the asymptotic behaviour of this function presented in the literature, but empirical evidence suggests that it depends heavily on the underlying distribution of $S \subset M$.

4.1.2 An algorithm for $(1+\delta,\varepsilon)$-ANNs [8]

The exact algorithm provided by Arya et al. [8] is in fact an approximation algorithm that allows the value of δ to be set (precisely to $\delta = 0$ for the exact case). In the case $\delta > 0$, the function c given above also depends on this δ but all other complexities remain the same. A lower bound of c is given as

$$c(d,\delta) \leq d\lceil 1 + 6d/\delta \rceil^d.$$

4.1.3 The state-of-the-art for $(1+\delta,\varepsilon)$-ANNs [9]

A recent benchmark study [10] found the fastest current algorithm to solve $(1+\delta,\varepsilon)$-ANN as one based on *navigable small-world graphs with controllable*

hierarchy [9]. We briefly touch on some underlying theory before introducing this algorithm.

Algorithm 1 A greedy routing algorithm for ε-NNs.

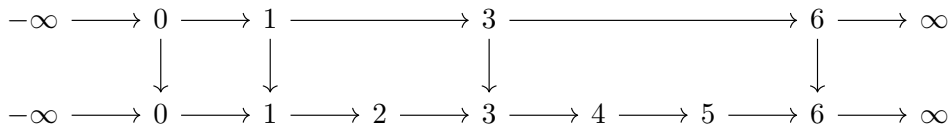
```
1:  function GreedyRouting(graph G, heuristic h, start node v)
2:      best ← v
3:      while true do
4:          bestNeighbour ← arg max_{n∈N_G(best)} h(n)
5:          if h(bestNeighbour) > h(best) then
6:              best ← bestNeighbour
7:          else
8:              return best
9:          end if
10:     end while
11: end function
```

A graph is a *navigable small-world graph* if the greedy graph routing (shown in Algorithm 1) runs in $O(\log^k n)$ time, for $k > 1$. We combine such a graph with the concept of *skip lists*.

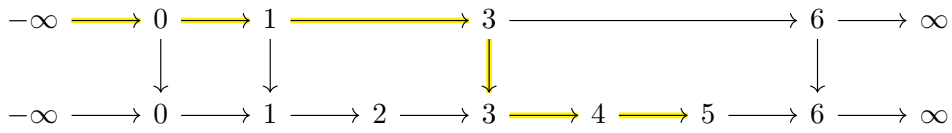
A *skip list* is a probabilistic data structure that allows $O(\log n)$ search complexity and insertion complexity within an ordered sequence of n elements.

There is an argument to include in an earlier section covering other theory in computer science integral to the description of speed-ups: hash tables, skip lists, etc.

The initial idea: for a sorted list, we create a duplicate list where a given element of the initial list is duplicated with probability 0.5. For every duplicated item, we store a pointer at the item in the second list back to the first list. We also add *sentinel nodes* for safety. Take the sorted list $(0, 1, 2, 3, 4, 5, 6)$ as follows.

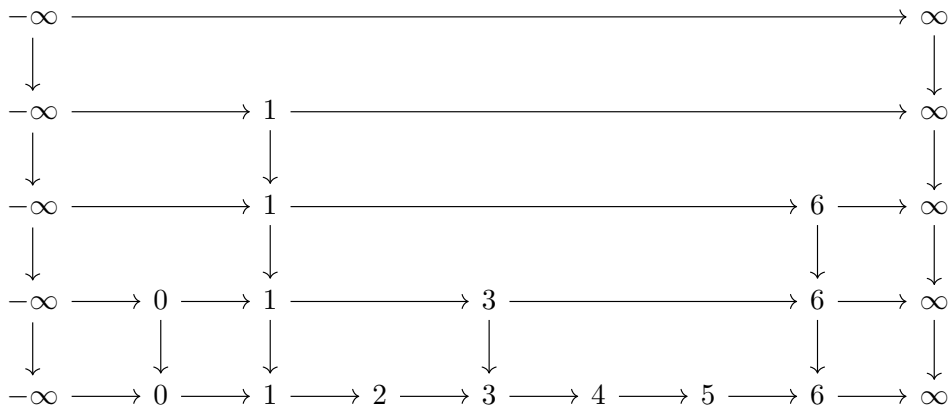


Searching for 5 is shown below.



The probability a given duplicated node is followed by k non-duplicated nodes is $\frac{1}{2^k}$. Thus, the expected number of nodes examined when searching the initial list is $\sum_{k=1}^{\infty} \frac{1}{2^k} = 2$. So by adding this *shortcut list*, we have reduced the time complexity from n to $\frac{n}{2} + O(1)$.

An improvement: keep adding shortcut lists of the top shortcut list until we are out of elements.



The expected number of levels is $\log n$, which is easy to prove. At each level, we cut search time in half (excluding overhead). This gives us a search time of $O(\log n)$.

To combine this with navigable small-world networks: we build layers of graphs up sequentially. The bottom layer (as in skip lists) is the graph where the vertices are our points and two points have an edge if they are an ε-neighbour. Vertices have a fixed probability of moving up to the next layer, and the way we choose in higher layers is by using a heuristic that favours diverse connections.

I had hoped to have added more to this algorithm description before the draft report due date, I do indent to discuss this a bit more.

4.1.4 Inductive algorithm for (ii) [7]

We fix some ordering on the vertices of our 1-skeleton.

function LowerNghbrs(graph G, vertex $u \in G$)
  return $\{v \in V(G) : v < u, \{u, v\} \in E(G)\}$
end function
function InductiveExpansion(graph G, level $k \in \{2, 3, \ldots\}$)
  $K \leftarrow V(G) \cup E(G)$
  for $i \leftarrow 1$ to k **do**
    for each i-simplex $\tau \in K$ **do**
      $N \leftarrow \cap_{u \in \tau}$LowerNghbrs$(G, u)$
      for each $v \in N$ **do**
        $K \leftarrow K \cup \{\tau \cup \{u\}\}$
      end for
    end for
  end for
end function

There is missing discussion about this algorithm, including running time, space complexity, etc.

4.1.5 Incremental algorithm for (ii) [7]

function AddCofaces(graph G, level k, simplex τ, N, K)
  if $\dim \tau \geq k$ **then**
    return
  end if
  for each $v \in N$ **do**
    $\sigma \leftarrow \tau \cup \{v\}$
    $M \leftarrow N \cap \text{LowerNghbrs}(G, v)$
    AddCofaces(G, k, σ, M, K)
  end for
end function
function IncrementalExpansion(G, k)
  $K \leftarrow \varnothing$
  for each $u \in V(G)$ **do**
    $N \leftarrow \text{LowerNghbrs}(G, u)$
    AddCofaces$(G, k\{u\}, N, K)$
  end for
end function

There is missing discussion about this algorithm, including running time, space complexity, etc.

4.1.6 Comparison of methods

This section intends to give a comparison of the various methods for (i) and (ii) we have introduced, implemented by myself in Python and then running times and space used plotted with input size.

4.2 Čech complexes `todo`

4.3 Integral simplicial homology

There is a key paper here that I am planning following quite closely, the structure of this section hopes to mimic the last: introduce various methods (including state-of-the-art), discuss their time complexities, and the compare runs.

4.3.1 Regular Smith normal form approach

This subsection will mimic writing you have already seen from me in first term, and will include content from my poster and presentation.

4.3.2 Lattice basis reduction approach

This seems to be the standard approach to integral homology, or at least to finding the Smith normal form of an integer matrix. It avoids the coefficient blow-up which the regular algorithm suffers from.

4.4 Computing persistent homology

4.4.1 Standard algorithm

This seciton was written a few weeks ago and not properly attached to the rest of this report. Expect this to be rewritten to fit together with the rest of the content.

Definition 4.4.1 (Codimension)**.** Let K be a n-simplex and L be an m-simplex such that L is a face of K ($m < n$). The *codimension* of L in K is the difference between the dimensions; that is, $\operatorname{codim}(L) = \dim(K) - \dim(L)$.

Like before, we consider a filtration $\{K_i\}_{i=0}^{n}$ for a simplicial complex K and impart that $K_0 = \varnothing$ and $K_n = K$. We now place a total ordering on the simplices of K, denoted $\sigma_1, \ldots, \sigma_m$, in a way that it agrees with our filtration:

- a face of a simplex precedes the simplex; and
- a simplex in K_i precedes the simplices in K_j which are not in K_i for all $j > i$.

We now construct a square matrix $\partial \in M_m(\mathbb{F}_2)$ such that

$$\partial[i,j] = \begin{cases} 1 & \sigma_i \text{ is a face of } \sigma_j \text{ with codimension 1,} \\ 0 & \text{else.} \end{cases}$$

We now introduce a standard algorithm for the reduction of this boundary matrix to barcodes.

```
1  def add_col_to_col(col_in: int, col_out: int) -> int: ...
2  def low(col: int) -> int: ...
3  def is_col_reduced(col: int) -> bool: ...
4
5  for i in range(len(bmat)):
6      while not is_col_reduced(i):
7          for j in range(0, i):
8              if low(i) == low(j):
9                  add_col_to_col(j, i)
10                 break
```

The implementations of the top three functions are omitted, but their actions and return values are described. First, we understand this script expects a matrix `bmat`, a list of lists representing the square matrix ∂. `low` returns the largest row index of an element in a given column with a non-zero entry, and if the column is all zero it returns -1. `is_col_reduced(col)` evaluates true if `col` is the column with smallest index with its value of `low` or if `low(col)` $= -1$; otherwise, it evaluates false. We precisely define these functions below.

- `add_col_to_col` performs the expected column operation on `bmat`.
- $\texttt{low}(\texttt{col}) = \begin{cases} -1 & \text{if } \partial[i+1, \texttt{col}+1] = 0 \text{ for all } i, \\ \min\{i : \partial[i+1, \texttt{col}+1] \neq 0\} & \text{else.} \end{cases}$
- $\texttt{is\_col\_reduced}(\texttt{col}) = \begin{cases} 1 & \text{if } \texttt{low}(\texttt{i}) = \texttt{low}(\texttt{col}) \neq -1 \text{ for all } i < \texttt{col}, \\ 0 & \text{else.} \end{cases}$

We note that each of these functions can be implemented in $O(m)$ time. The exact implementation above runs in $O(m^3)$ time. The reason for the above implementation is to be clear on the exactness of the algorithm.

On to the main loop (line `5` onwards), we first claim that this terminates. Although not immediately obvious, our assumptions upon $\sigma_1, \ldots, \sigma_m$ from which we build ∂ (which `bmat` represents) assert that $\texttt{is\_col\_reduced}(i)$ will evaluate true by repeating `7` to `10`. We have that $\texttt{is\_col\_reduced}(j)$ is true for each $j \in \{0, \ldots i-1\}$. As $\texttt{is\_col\_reduced}(i)$ is false, there is $i_1 < i$ such that $\texttt{low}(i_1) = \texttt{low}(i)$. So we add the i_1th column to the ith column, making the new value of $\texttt{low}(i)$ strictly less than $\texttt{low}(i_1)$ (as, by definition, every entry below $\texttt{low}(i_1)$ is zero). To appreciate this, we recall that we are in $\mathbb{F}_2$. If $\texttt{is\_col\_reduced}(i)$ is true, we are done; otherwise, there is $i_2 < i$ such that $\texttt{low}(i_2) = \texttt{low}(i) < \texttt{low}(i_1)$. Again, we perform the appropriate column operation. We continue this operation, and as there are only a finite number of rows (that is, a finite number of unique values for `low`), then we must reach a point at which $\texttt{is\_col\_reduced}(i)$ is false.

First, we learn to read the ranks of the homology groups of K. Let $R \in M_m(\mathbb{F}_2)$ be the resulting matrix from running the standard algorithm described above on ∂. Define

$$\begin{aligned} \text{zeros}_p(R) &= \{i : \text{column } i \text{ in } R \text{ is zero and } \dim \sigma_i = p\}, \\ \text{lows}_p(R) &= \{i : \text{low}(j) = i \text{ for some } j \text{ and } \dim \sigma_i = p\}. \end{aligned}$$

If $K_i = \{\sigma_j\}_{j=1}^i$, we claim that $\text{zeros}_p(R) - \text{lows}_p(R) = \beta_p$, which can be seen by observing that $\text{zeros}_p(R)$ is the rank of the p-cycles, and $\text{lows}_p(R)$ is the rank of the p-boundaries.

Next, we move to persistent homology. Let $f : K \to \mathbb{R}$ be defined on our simplicial complex as in the last section which induces our filtration, with associated sequence $\{a_i\}_{i=0}^m$. If $K_i = \{\sigma_j\}_{j=1}^i$ then $(a_i, a_j) \in \text{dgm}_p(f)$ if and only if $i = \text{low}(j)$ and $\dim \sigma_i = p$. Now we assume otherwise, so there is a step in the filtration in which more than one simplex is added. We simply construct the filtration $\{K'_i\}_{i=0}^m$ such that $K'_i = \{\sigma_j\}_{j=1}^i$ and let $k' : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ be the sequence such that $K_i = K'_{k'(i)}$ for all i. Now define $k : \mathbb{Z}_{\geq 0} \to \mathbb{Z}_{\geq 0}$ as $i \mapsto \min\{i' : k(i') \geq i\}$. This is a mapping between our filtration and a filtration which only adds one simplex at a time when

we move up. Let (i, j) be the indices of some point in the pth persistence diagram for K'. Then $(a_{k(i)}, a_{k(j)}) \in \mathrm{dgm}_p(f)$

[11]

4.4.2 Sparse matrices

Again this section was written a few weeks ago and needs to be fit to the report.

So we have a cubic algorithm for calculating persistent homology in the number of simplices. Let us consider a triangulation of S^n: $K = \mathcal{P}(\{v_1, \ldots, v_{n+2}\}) \setminus \{\{v_1, \ldots, v_{n+2}\}\}$ and consider the ordering of simplices $\sigma_1, \ldots, \sigma_{2^{n+2}-2}$ as discussed, inducing the filtration $\{K_i\}_{i=0}^{2^{n+2}-2}$. From a computational perspective, we say that there is an exponential count of simplices, which is size of our boundary matrix. So, using the method above, we may calculate the persistent homology of the n-sphere in $O((2^{n+2}-2)^3) = 2^{O(n)}$ time.

In practice, we may want to form our filtration from spaces with some notion of characterisation between data points. Precisely, let (M, d) be a metric space and we consider the Vietoris-Rips complex of some finite subset $S \subset M$ with diameter l,

$$\mathrm{VR}(S; l) = \{\sigma \subset S : \mathrm{diam}(\sigma) \leq l\}$$

where $\mathrm{diam} : \mathcal{P}(M) \to \mathbb{R}$ such that $S \mapsto \sup_{x,y \in S} d(x, y)$. This induces an ordering on the subcomplexes (consider Vietoris-Rips complex as we let l vary from 0 to ∞), which defines a filtration $\{K_i\}_{i=0}^{n}$ (where $K_0 = \varnothing$ and $K_n = K = \mathrm{VR}(S; \infty)$). We have

$$|K| = |\mathrm{VR}(S; \infty)| = |\mathcal{P}(S)| = 2^{|S|},$$

so we can calculate the persistence diagram of such a filtration in $2^{O(n)}$ time where $n = |S|$.

See below the boundary matrix for $\partial\Delta_2$ (homotopy equivalent to S^1).

$$\begin{pmatrix}
0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

This is a sparse matrix; that is, almost all of the entries are 0. This allows us to make a number of computational speed-ups. Recall the operations used in the implementation earlier, as we keep them in mind when considering a data structure.

We now show the sparse matrix representation from Edelsbrunner and Harer [11], and present an alternative.

4.4.3 Reduction by killing

This subsection and the succeeing subsections focus on specific speed-ups utilised by Ripser.

[12]

4.4.4 Compute persistent cohomology

[13]

4.4.5 Apparent and emergent pairs

[14]

4.4.6 Parallelism

[15]

4.4.7 Matroid filtrations

[16]

Chapter 5

Applications of computational topology

5.1 Contagion maps

This section aims to build on the novel simplicial complex construction from graphs applied to contagion maps.

[17]

5.2 Cyclomatic complexity

Cyclomatic complexity indicates the complexity of a program by the number number of linearly independent paths it can take through the source code, the link to homology is clear. The use of persistent homology here appears to be novel (maybe for a good reason...).

5.3 Simplicial homology global optimisation

Simplicial homology has apparently found use in optimisation problems, though I have not read far into this.

Bibliography

[1] Primoz Skraba and Mikael Vejdemo-Johansson. "Persistence modules: algebra and algorithms". In: *arXiv preprint arXiv:1302.2015* (2013).

[2] Afra Zomorodian and Gunnar Carlsson. "Computing persistent homology". In: *Discrete & Computational Geometry* 33.2 (2005), pp. 249–274.

[3] René Corbet and Michael Kerber. "The representation theorem of persistence revisited and generalized". In: *Journal of Applied and Computational Topology* 2.1 (2018), pp. 1–31.

[4] Massimo Ferri, Dott Mattia G Bergomi, and Lorenzo Zu. "Simplicial complexes from graphs towards graph persistence". In: *arXiv preprint arXiv:1805.10716* (2018).

[5] James Oxley. "What is a matroid". In: *Cubo* 5.179 (2003), p. 780.

[6] Alan Mathison Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". In: *J. of Math* 58.345-363 (1936), p. 5.

[7] Afra Zomorodian. "Fast construction of the Vietoris-Rips complex". In: *Computers & Graphics* 34.3 (2010), pp. 263–271.

[8] Sunil Arya et al. "An optimal algorithm for approximate nearest neighbor searching fixed dimensions". In: *Journal of the ACM (JACM)* 45.6 (1998), pp. 891–923.

[9] Yu A Malkov and Dmitry A Yashunin. "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs". In: *IEEE transactions on pattern analysis and machine intelligence* 42.4 (2018), pp. 824–836.

[10] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. "ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms". In: *Information Systems* 87 (2020), p. 101374.

[11] Herbert Edelsbrunner and John Harer. *Computational topology: an introduction*. American Mathematical Soc., 2010.

[12] Chao Chen and Michael Kerber. "Persistent homology computation with a twist". In: *Proceedings 27th European workshop on computational geometry*. Vol. 11. 2011, pp. 197–200.

[13] Vin De Silva, Dmitriy Morozov, and Mikael Vejdemo-Johansson. "Dualities in persistent (co) homology". In: *Inverse Problems* 27.12 (2011), p. 124003.

[14] Ulrich Bauer. "Ripser: efficient computation of Vietoris–Rips persistence barcodes". In: *Journal of Applied and Computational Topology* 5.3 (2021), pp. 391–423.

[15] Simon Zhang, Mengbai Xiao, and Hao Wang. "GPU-accelerated computation of Vietoris-Rips persistence barcodes". In: *arXiv preprint arXiv:2003.07989* (2020).

[16] Gregory Henselman and Robert Ghrist. "Matroid filtrations and computational persistent homology". In: *arXiv preprint arXiv:1606.00199* (2016).

[17] Dane Taylor et al. "Topological data analysis of contagion maps for examining spreading processes on networks". In: *Nature communications* 6.1 (2015), pp. 1–11.